EsgynDB SQL 参考手册

适用于 EsgynDB 2.8.0 及更高版本

2021 年 03 月

版权
© Copyright 2015-2021 贵州易鲸捷信息技术有限公司

公告
本文档包含的信息如有更改，恕不另行通知。
保留所有权利。除非版权法允许，否则在未经易鲸捷预先书面许可的情况下，严禁改编或翻译本手册的内容。易鲸捷对于本文中所包含的技术或编辑错误、遗漏概不负责。
易鲸捷产品和服务附带的正式担保声明中规定的担保是该产品和服务享有的唯一担保。本文中的任何信息均不构成额外的保修条款。

声明
Microsoft® 和 Windows® 是美国微软公司的注册商标。Java® 和 MySQL® 是 Oracle 及其子公司的注册商标。Bosun 是 Stack Exchange 的商标。Apache®、Hadoop®、HBase®、Hive®、openTSDB®、Sqoop® 和 Trafodion® 是 Apache 软件基金会的商标。Esgyn，EsgynDB 和 QianBase 是易鲸捷的商标。

目录

目录 ..... **iii**

关于本文档 **31**

目标读者 31

新增和更改的信息 32

文档结构 34

符号约定 34

1. 介绍 **38**

1.1 SQL 语言 39

1.2 使用 EsgynDB SQL 访问 HBase 表 40

1.2.1 初始化 EsgynDB 元数据 40

1.2.2 HBase 表的访问方式 40

1.2.3 对比：EsgynDB SQL 表与原生 HBase 表 42

1.2.4 HBase 表支持的 SQL 语句 42

1.3 使用 EsgynDB SQL 访问 Hive 表 43

1.3.1 Hive 表的 ANSI 名称 43

1.3.2 Hive 到 EsgynDB SQL 的数据类型映射 43

1.3.3 Hive 表支持的 SQL 语句 45

1.4 数据一致性和访问选项 46

1.4.1 READ COMMITTED 46

1.5 事务管理 46

1.5.1 用户定义事务和系统定义事务 47

1.5.2 DML 语句规则 47

1.5.3 AUTOCOMMIT 选项效果 47

1.5.4 并发性 48

1.5.5 事务隔离级别 48

1.6 ANSI 合规性和 EsgynDB SQL 扩展语句 48

1.6.1 符合 ANSI 标准的语句 48

1.6.2 EsgynDB SQL 扩展语句 49

1.6.3 符合 ANSI 标准的函数 50

1.7 EsgynDB SQL 错误消息 51

2. SQL 语句 52
2.1 类别 54
2.1.1 数据定义语言（DDL）语句 54
2.1.2 数据操纵语言（DML）语句 55
2.1.3 事务控制语句 55
2.1.4 数据控制和安全性语句 56
2.1.5 存储过程和用户定义函数语句 56
2.1.6 预处理语句 57
2.1.7 控制语句 57
2.1.8 对象命名语句 57
2.1.9 SHOW, GET 和 EXPLAIN 语句 58
2.2 ALTER LIBRARY 语句 58
2.2.1 ALTER LIBRARY 语法 59
2.2.2 ALTER LIBRARY 注意事项 59
2.2.3 ALTER LIBRARY 例子 59
2.3 ALTER SEQUENCE 语句 60
2.3.1 ALTER SEQUENCE 语法 60
2.3.2 ALTER SEQUENCE 注意事项 61
2.3.3 ALTER SEQUENCE 例子 61
2.4 ALTER TABLE 语句 62
2.4.1 ALTER TABLE 语法 64
2.4.2 ALTER TABLE 注意事项 70
2.4.3 ALTER TABLE 例子 71
2.5 ALTER USER 语句 79
2.5.1 ALTER USER 语法 79
2.5.2 ALTER USER 注意事项 79
2.5.3 ALTER USER 例子 79
2.6 BEGIN WORK 语句 81
2.6.1 BEGIN WORK 注意事项 81
2.6.2 BEGIN WORK 例子 81
2.7 CALL 语句 82

2.7.1 CALL 语法 82
2.7.2 CALL 注意事项 82
2.7.3 CALL 例子 84
2.8 COMMIT WORK 语句 86
2.8.1 COMMIT WORK 注意事项 86
2.8.2 COMMIT WORK 例子 86
2.9 CONTROL QUERY CANCEL 语句 88
2.9.1 CONTROL QUERY CANCEL 语法 88
2.9.2 CONTROL QUERY CANCEL 注意事项 88
2.9.3 CONTROL QUERY CANCEL 例子 89
2.10 CONTROL QUERY DEFAULT 语句 90
2.10.1 CONTROL QUERY DEFAULT 语法 90
2.10.2 CONTROL QUERY DEFAULT 注意事项 90
2.10.3 CONTROL QUERY DEFAULT 例子 91
2.11 CREATE FUNCTION 语句 92
2.11.1 CREATE FUNCTION 语法 93
2.11.2 CREATE FUNCTION 注意事项 95
2.11.3 CREATE FUNCTION 例子 95
2.12 CREATE INDEX 语句 96
2.12.1 CREATE INDEX 语法 96
2.12.2 CREATE INDEX 注意事项 99
2.12.3 CREATE INDEX 例子 100
2.13 CREATE LIBRARY 语句 102
2.13.1 CREATE LIBRARY 语法 102
2.13.2 CREATE LIBRARY 注意事项 102
2.13.3 CREATE LIBRARY 例子 103
2.14 CREATE PROCEDURE 语句 104
2.14.1 CREATE PROCEDURE 语法 105
2.14.2 CREATE PROCEDURE 注意事项 109
2.14.3 CREATE PROCEDURE 例子 111
2.15 CREATE ROLE 语句 115

2.15.1 CREATE ROLE 语法 115
2.15.2 CREATE ROLE 注意事项 115
2.15.3 CREATE ROLE 例子 116
2.16 CREATE SCHEMA 语句 117
2.16.1 CREATE SCHEMA 语法 117
2.16.2 CREATE SCHEMA 注意事项 118
2.16.3 CREATE SCHEMA 例子 118
2.17 CREATE SEQUENCE 语句 121
2.17.1 CREATE SEQUENCE 语法 122
2.17.2 CREATE SEQUENCE 注意事项 125
2.17.3 CREATE SEQUENCE 例子 125
2.18 CREATE TABLE 语句 127
2.18.1 CREATE TABLE 语法 129
2.18.2 CREATE TABLE 注意事项 137
2.18.3 CREATE TABLE 的 EsgynDB SQL 扩展语句 143
2.18.4 CREATE TABLE 例子 144
2.19 CREATE TRIGGER 语句 152
2.19.1 CREATE TRIGGER 语法 152
2.19.2 CREATE TRIGGER 注意事项 153
2.19.3 CREATE TRIGGER 例子 153
2.20 CREATE VIEW 语句 155
2.20.1 CREATE VIEW 语法 155
2.20.2 CREATE VIEW 注意事项 156
2.20.3 CREATE VIEW 例子 159
2.21 DELETE 语句 162
2.21.1 DELETE 语法 162
2.21.2 DELETE 注意事项 163
2.21.3 DELETE 例子 164
2.22 DROP FUNCTION 语句 166
2.22.1 DROP FUNCTION 语法 166
2.22.2 DROP FUNCTION 注意事项 166

2.22.3 DROP FUNCTION 例子 166
2.23 DROP INDEX 语句 167
2.23.1 DROP INDEX 语法 167
2.23.2 DROP INDEX 注意事项 167
2.24 DROP LIBRARY 语句 168
2.24.1 DROP LIBRARY 语法 168
2.24.2 DROP LIBRARY 注意事项 168
2.24.3 DROP LIBRARY 例子 169
2.25 DROP PROCEDURE 语句 170
2.25.1 DROP PROCEDURE 语法 170
2.25.2 DROP PROCEDURE 注意事项 170
2.25.3 DROP PROCEDURE 例子 170
2.26 DROP ROLE 语句 171
2.26.1 DROP ROLE 语法 171
2.26.2 DROP ROLE 注意事项 171
2.26.3 DROP ROLE 例子 172
2.27 DROP SCHEMA 语句 173
2.27.1 DROP SCHEMA 语法 173
2.27.2 DROP SCHEMA 注意事项 173
2.27.3 DROP SCHEMA 例子 173
2.28 DROP SEQUENCE 语句 174
2.28.1 DROP SEQUENCE 语法 174
2.28.2 DROP SEQUENCE 注意事项 174
2.28.3 DROP SEQUENCE 例子 174
2.29 DROP TABLE 语句 175
2.29.1 DROP TABLE 语法 175
2.29.2 DROP TABLE 注意事项 175
2.29.3 DROP TABLE 例子 176
2.30 DROP TRIGGER 语句 177
2.30.1 DROP TRIGGER 语法 177
2.30.2 DROP TRIGGER 注意事项 177

2.30.3 DROP TRIGGER 例子 177
2.31 DROP VIEW 语句 178
2.31.1 DROP VIEW 语法 178
2.31.2 DROP VIEW 注意事项 178
2.31.3 DROP VIEW 例子 178
2.32 EXECUTE 语句 179
2.32.1 EXECUTE 语法 179
2.32.2 EXECUTE 注意事项 180
2.32.3 EXECUTE 例子 180
2.33 EXPLAIN 语句 183
2.33.1 EXPLAIN 语法 183
2.33.2 EXPLAIN 注意事项 184
2.34 GET 语句 189
2.34.1 GET 语法 189
2.34.2 GET 注意事项 192
2.34.3 GET 例子 193
2.35 GET HBASE OBJECTS 语句 197
2.35.1 GET HBASE OBJECTS 语法 197
2.35.2 GET HBASE OBJECTS 例子 198
2.36 GET VERSION OF METADATA 语句 199
2.36.1 GET VERSION OF METADATA 注意事项 199
2.36.2 GET VERSION OF METADATA 例子 199
2.37 GET VERSION OF SOFTWARE 语句 201
2.37.1 GET VERSION OF SOFTWARE 注意事项 201
2.37.2 GET VERSION OF SOFTWARE 例子 201
2.38 GRANT 语句 203
2.38.1 GRANT 语法 204
2.38.2 GRANT 注意事项 206
2.38.3 GRANT 例子 208
2.39 GRANT COMPONENT PRIVILEGE 语句 210
2.39.1 GRANT COMPONENT PRIVILEGE 语法 210

2.39.2 GRANT COMPONENT PRIVILEGE 注意事项 215
2.39.3 GRANT COMPONENT PRIVILEGE 例子.....215
2.40 GRANT ROLE 语句 216
2.40.1 GRANT ROLE 语法.....216
2.40.2 GRANT ROLE 注意事项.....216
2.40.3 GRANT ROLE 例子 216
2.41 INSERT 语句 217
2.41.1 INSERT 语法.....217
2.41.2 INSERT 注意事项 218
2.41.3 INSERT 例子.....220
2.42 INVOKE 语句 223
2.42.1 INVOKE 语法 223
2.42.2 INVOKE 注意事项 223
2.42.3 INVOKE 例子 223
2.43 MERGE 语句.....224
2.43.1 MERGE 语法.....224
2.43.2 MERGE 注意事项.....224
2.43.3 MERGE 例子.....227
2.44 PREPARE 语句.....229
2.44.1 PREPARE 语法.....229
2.44.2 PREPARE 注意事项.....229
2.44.3 PREPARE 例子.....229
2.45 REGISTER USER 语句 231
2.45.1 REGISTER USER 语法 231
2.45.2 REGISTER USER 注意事项 231
2.45.3 REGISTER USER 例子 232
2.46 REVOKE 语句 233
2.46.1 REVOKE 语法 233
2.46.2 REVOKE 注意事项 235
2.46.3 REVOKE 例子 236
2.47 REVOKE COMPONENT PRIVILEGE 语句 238

2.47.1 REVOKE COMPONENT PRIVILEGE 语法 238
2.47.2 REVOKE COMPONENT PRIVILEGE 注意事项 239
2.47.3 REVOKE COMPONENT PRIVILEGE 例子 240
2.48 REVOKE ROLE 语句 241
2.48.1 REVOKE ROLE 语法 241
2.48.2 REVOKE ROLE 注意事项 241
2.48.3 REVOKE ROLE 例子 242
2.49 ROLLBACK WORK 语句 243
2.49.1 ROLLBACK WORK 语法 243
2.49.2 ROLLBACK WORK 注意事项 243
2.49.3 ROLLBACK WORK 例子 243
2.50 SAVEPOINT 语句 244
2.50.1 SAVEPONT 语法 244
2.50.2 SAVEPOINT 注意事项 244
2.50.3 SAVEPOINT 例子 244
2.51 SELECT 语句 245
2.51.1 SELECT 语法 247
2.51.2 SELECT 注意事项 260
2.51.3 SELECT 例子 267
2.52 SET SCHEMA 语句 278
2.52.1 SET SCHEMA 语法 278
2.52.2 SET SCHEMA 注意事项 278
2.52.3 SET SCHEMA 例子 278
2.53 SET TRANSACTION 语句 279
2.53.1 SET TRANSACTION 语法 279
2.53.2 SET TRANSACTION 注意事项 279
2.53.3 SET TRANSACTION 例子 280
2.54 SHOWCONTROL 语句 281
2.54.1 SHOWCONTROL 语法 281
2.54.2 SHOWCONTROL 例子 282
2.55 SHOWDDL 语句 283

2.55.1 SHOWDDL 语法 283
2.55.2 SHOWDDL 注意事项 284
2.55.3 SHOWDDL 例子 285
2.56 SHOWDDL SCHEMA 语句 287
2.56.1 SHOWDDL SCHEMA 语法 287
2.56.2 SHOWDDL SCHEMA 注意事项 287
2.56.3 SHOWDDL SCHEMA 例子 288
2.57 SHOWSTATS 语句 289
2.57.1 SHOWSTATS 语法 289
2.57.2 SHOWSTATS 注意事项 291
2.57.3 SHOWSTATS 例子 291
2.58 TABLE 语句 293
2.58.1 TABLE 语法 293
2.58.2 TABLE 注意事项 293
2.58.3 TABLE 例子 293
2.59 UNREGISTER USER 语句 294
2.59.1 UNREGISTER USER 语法 294
2.59.2 UNREGISTER USER 注意事项 294
2.59.3 UNREGISTER USER 例子 295
2.60 UPDATE 语句 296
2.60.1 UPDATE 语法 296
2.60.2 UPDATE 注意事项 298
2.60.3 UPDATE 例子 301
2.61 UPDATE USING UPSERT 语句 302
2.61.1 UPDATE USING UPSERT 注意事项 302
2.61.2 UPDATE USING UPSERT 例子 302
2.62 UPSERT 语句 308
2.62.1 UPSERT 语法 308
2.62.2 UPSERT 例子 309
2.63 VALUES 语句 312
2.63.1 VALUES 语法 312

2.63.2 VALUES 注意事项 312
2.63.3 VALUES 例子 312
3. SQL 实用工具 314
3.1 CLEANUP 语句 315
3.1.1 CLEANUP 语法 315
3.1.2 CLEANUP 注意事项 317
3.1.3 CLEANUP 例子 317
3.2 DUMP 语句 322
3.2.1 DUMP 语法 322
3.2.2 DUMP 注意事项 322
3.2.3 DUMP 例子 323
3.3 LOAD 语句 324
3.3.1 LOAD 语法 325
3.3.2 LOAD 注意事项 328
3.3.3 LOAD 例子 331
3.4 POPULATE INDEX 语句 354
3.4.1 POPULATE INDEX 语法 354
3.4.2 POPULATE INDEX 注意事项 355
3.4.3 POPULATE INDEX 例子 355
3.5 PURGEDATA 语句 356
3.5.1 PURGEDATA 语法 356
3.5.2 PURGEDATA 注意事项 356
3.5.3 PURGEDATA 例子 356
3.6 TRUNCATE 语句 357
3.6.1 TRUNCATE 语法 357
3.6.2 TRUNCATE 注意事项 357
3.6.3 TRUNCATE 例子 358
3.7 UNLOAD 语句 359
3.7.1 UNLOAD 语法 359
3.7.2 UNLOAD 注意事项 361
3.7.3 UNLOAD 例子 361

3.8 UPDATE STATISTICS 语句 363
3.8.1 UPDATE STATISTICS 语法 364
3.8.2 UPDATE STATISTICS 注意事项 368
3.8.3 UPDATE STATISTICS 例子 372
4. SQL 语言元素 375
4.1 授权 ID 376
4.2 字符集 376
4.3 列 376
4.3.1 列引用 376
4.3.2 派生列名称 377
4.3.3 派生列名称例子 377
4.3.4 列的默认设置 378
4.4 约束 378
4.4.1 在 SQL 表上创建或添加约束 378
4.4.2 约束名 378
4.5 相互关系名称 379
4.5.1 显式相互关系名称 379
4.5.2 隐式相互关系名称 380
4.5.3 相互关系名称例子 380
4.6 数据库对象 380
4.6.1 所有权 380
4.7 数据库对象名称 380
4.7.1 SQL 对象的逻辑名称 381
4.7.2 SQL 对象的命名空间 381
4.8 数据类型 381
4.8.1 可比和兼容的数据类型 387
4.8.2 字符串数据类型 391
4.8.3 日期时间数据类型 396
4.8.4 时间间隔数据类型 399
4.8.5 数字型数据类型 402
4.8.6 数据类型的隐式转换 405

4.9 表达式.....405
4.9.1 字符值表达式.....406
4.9.2 日期时间值表达式.....407
4.9.3 日期间隔值表达式.....411
4.9.4 数字值表达式.....416
4.10 标识符.....419
4.10.1 常规标识符.....419
4.10.2 限定标识符.....419
4.10.3 不区分大小写字母的限定标识符.....420
4.10.4 标识符例子.....420
4.11 标识列.....421
4.11.1 标识列语法.....421
4.11.2 标识列例子.....421
4.12 索引.....423
4.12.1 SQL 索引.....423
4.13 键.....424
4.13.1 聚集键.....424
4.13.2 SYSKEY.....424
4.13.3 索引键.....424
4.13.4 主键.....425
4.14 字面值.....425
4.14.1 字符串字面值.....425
4.14.2 日期时间字面值.....428
4.14.3 时间间隔字面值.....429
4.14.4 数字字面值.....432
4.15 Null.....434
4.15.1 空值和默认值的使用情况对比.....434
4.15.2 定义允许或禁止出现空值的列.....434
4.15.3 Null 对 Oracle 的兼容支持.....435
4.16 谓词.....437
4.16.1 BETWEEN 谓词.....438

4.16.2 比较谓词 440
4.16.3 EXISTS 谓词 444
4.16.4 IN 谓词 446
4.16.5 LIKE 谓词 449
4.16.6 REGEXP 谓词 451
4.16.7 NULL 谓词 452
4.16.8 OVERLAPS 谓词 453
4.16.9 量化比较谓词 460
4.17 权限 464
4.18 角色 465
4.19 Schemas 466
4.19.1 创建和删除 schema 466
4.20 搜索条件 466
4.20.1 搜索条件注意事项 467
4.20.2 搜索条件例举例 468
4.21 子查询 468
4.21.1 SELECT 形式的子查询 469
4.21.2 使用子查询提供比较值 469
4.21.3 提供比较值的嵌套子查询 469
4.21.4 提供比较值时出现的相关子查询 470
4.22 表 470
4.22.1 基表 471
4.23 视图 471
4.23.1 SQL 视图 471
4.23.2 视图例子 472
4.24 ROWID 472
4.24.1 ROWID 语法 472
4.24.2 ROWID 例子 472
4.25 ROWNUM 473
4.25.1 ROWNUM 语法 473
4.25.2 ROWNUM 例子 473

5. SQL 子句 474
5.1 DEFAULT 子句 475
5.1.1 DEFAULT 子句语法 475
5.1.2 DEFAULT 子句例子 476
5.2 FORMAT 子句 477
5.2.1 FORMAT 子句语法 478
5.2.2 日期格式注意事项 478
5.2.3 其他格式注意事项 479
5.2.4 FORMAT 子句例子 479
5.3 SAMPLE 子句 480
5.3.1 SAMPLE 子句语法 481
5.3.2 SAMPLE 子句注意事项 482
5.3.3 SAMPLE 子句例子 482
5.4 TRANSPOSE 子句 488
5.4.1 TRANSPOSE 子句语法 488
5.4.2 TRANSPOSE 子句注意事项 490
5.4.3 TRANSPOSE 子句例子 492
6. SQL 函数和表达式 497
6.1 类别 501
6.2 标准规范化 501
6.2.1 聚合（集合）函数 501
6.2.2 字符串函数 503
6.2.3 日期时间函数 504
6.2.4 LOB 函数 507
6.2.5 数学函数 507
6.2.6 加密函数 509
6.2.7 舍入函数 509
6.2.8 其他函数和表达式 513
6.2.9 标准规范化 514
6.3 ABS 函数 515
6.3.1 ABS 函数例子 515

6.4 AES_ENCRYPT 函数 516
6.4.1 AES_ENCRYPT 语法 516
6.4.2 AES_ENCRYPT 注意事项 518
6.4.3 AES_ENCRYPT 例子 518
6.5 ACOS 函数 519
6.5.1 ACOS 函数例子 519
6.6 ADD_MONTHS 函数 520
6.6.1 ADD_MONTHS 函数例子 520
6.7 ASCII 函数 521
6.7.1 ASCII 函数注意事项 521
6.7.2 ASCII 函数例子 524
6.8 ASIN 函数 526
6.8.1 ASIN 函数例子 526
6.9 ATAN 函数 527
6.9.1 ATAN 函数例子 527
6.10 ATAN2 函数 528
6.10.1 ATAN2 函数例子 528
6.11 AUTHNAME 函数 529
6.11.1 AUTHNAME 函数注意事项 529
6.11.2 AUTHNAME 函数例子 529
6.12 AVG 函数 530
6.12.1 AVG 函数注意事项 530
6.13 BITAND 函数 532
6.13.1 BITAND 函数注意事项 532
6.13.2 BITAND 函数例子 532
6.14 BUFFERTOLOB 函数 534
6.14.1 BUFFERTOLOB 函数语法 534
6.14.2 BUFFERTOLOB 函数例子 534
6.15 CASE（条件）表达式 535
6.15.1 CASE（条件）表达式注意事项 536
6.15.2 CASE（条件）表达式例子 537

6.16 CAST 表达式 539
6.16.1 CAST 表达式注意事项 539
6.16.2 CAST 表达式例子 540
6.17 CEILING 函数 541
6.17.1 CEIL 语法 541
6.17.2 CEIL 注意事项 542
6.17.3 CEIL 例子 542
6.18 CHAR 函数 544
6.18.1 CHAR 函数注意事项 544
6.18.2 CHAR 函数例子 544
6.19 CHAR_LENGTH 函数 545
6.19.1 CHAR_LENGTH 函数注意事项 545
6.19.2 CHAR_LENGTH 函数例子 545
6.20 COALESCE 函数 547
6.20.1 COALESCE 函数例子 547
6.21 CODE_VALUE 函数 548
6.21.1 CODE_VALUE 函数例子 548
6.22 CONCAT 函数 549
6.22.1 串联运算符 (||) 549
6.22.2 CONCAT 函数注意事项 549
6.22.3 CONCAT 函数例子 550
6.23 CONVERTTOHEX 函数 551
6.23.1 CONVERTTOHEX 函数注意事项 551
6.23.2 CONVERTTOHEX 函数例子 551
6.24 CONVERTTIMESTAMP 函数 553
6.24.1 CONVERTTIMESTAMP 函数注意事项 553
6.24.2 CONVERTTIMESTAMP 函数例子 553
6.25 COS 函数 555
6.25.1 COS 函数例子 555
6.26 COSH 函数 556
6.26.1 COSH 函数例子 556

6.27 COUNT 函数 557
6.27.1 COUNT 函数注意事项 557
6.27.2 COUNT 函数例子 557
6.28 CRC32 函数 559
6.28.1 CRC32 函数例子 559
6.29 CURRENT 函数 560
6.29.1 CURRENT 函数例子 560
6.30 CURRENT_DATE 函数 561
6.30.1 CURRENT_DATE 函数例子 561
6.31 CURRENT_TIME 函数 562
6.31.1 CURRENT_TIME 函数例子 562
6.32 CURRENT_TIMESTAMP 函数 563
6.32.1 CURRENT_TIMESTAMP 函数例子 563
6.33 CURRENT_USER 函数 564
6.33.1 CURRENT_USER 函数注意事项 564
6.33.2 CURRENT_USER 函数例子 564
6.34 DATE_ADD 函数 565
6.34.1 DATE_ADD 函数例子 565
6.35 DATE_SUB 函数 566
6.35.1 DATE_SUB 函数例子 566
6.36 DATEADD 函数 567
6.36.1 DATEADD 函数例子 567
6.37 DATEDIFF 函数 569
6.37.1 DATEDIFF 函数语法 569
6.37.2 DATEDIFF 注意事项 570
6.37.3 DATEDIFF 函数例子 570
6.38 DATEFORMAT 函数 573
6.38.1 DATEFORMAT 函数注意事项 573
6.38.2 DATEFORMAT 函数例子 573
6.39 DATE_PART 函数（时间间隔） 574
6.39.1 DATE_PART（时间间隔）函数例子 574

6.40 DATE_PART 函数（时间戳）.....575
6.40.1 DATE_PART 函数例子.....575
6.41 DATE_TRUNC 函数.....577
6.41.1 DATE_TRUNC 函数例子.....577
6.42 DAY 函数.....578
6.42.1 DAY 函数例子.....578
6.43 DAYNAME 函数.....579
6.43.1 DAYNAME 函数注意事项.....579
6.43.2 DAYNAME 函数例子.....579
6.44 DAYOFMONTH 函数.....580
6.44.1 DAYOFMONTH 函数例子.....580
6.45 DAYOFWEEK 函数.....581
6.45.1 DAYOFWEEK 函数例子.....581
6.46 DAYOFYEAR 函数.....582
6.46.1 DAYOFYEAR 函数例子.....582
6.47 DECODE 函数.....583
6.47.1 DECODE 函数注意事项.....583
6.47.2 DECODE 函数例子.....585
6.48 DEGREES 函数.....587
6.48.1 DEGREES 函数例子.....587
6.49 EMPTY_BLOB 函数.....588
6.49.1 EMPTY_BLOB 函数语法.....588
6.49.2 EMPTY_BLOB 函数注意事项.....588
6.49.3 EMPTY_BLOB 函数例子.....588
6.50 EMPTY_CLOB 函数.....589
6.50.1 EMPTY_CLOB 函数语法.....589
6.50.2 EMPTY 注意事项.....589
6.50.3 EMPTY_CLOB 函数例子.....589
6.51 EXP 函数.....590
6.51.1 EXP 函数例子.....590
6.52 EXPLAIN 函数.....591

6.52.1 EXPLAIN 函数注意事项.....591
6.52.2 EXPLAIN 函数例子.....594
6.53 EXTERNALTOLOB 函数.....595
6.53.1 EXTERNALTOLOB 函数语法.....595
6.53.2 EXTERNALTOLOB 函数注意事项.....595
6.53.3 EXTERNALTOLOB 函数例子.....595
6.54 EXTRACT 函数.....596
6.54.2 EXTRACT 函数例子.....598
6.55 FILETOLOB 函数.....605
6.55.1 FILETOLOB 函数例子.....606
6.56 FIND_IN_SET 函数.....607
6.56.1 FIND_IN_SET 函数语法.....607
6.56.2 FIND_IN_SET 函数例子.....607
6.57 FLOOR 函数.....608
6.57.1 FLOOR 注意事项.....609
6.57.2 FLOOR 例子.....609
6.58 FROM_UNIXTIME 函数.....611
6.58.1 FROM_UNIXTIME 例子.....611
6.59 GROUP_CONCAT 函数.....611
6.59.1 GROUP_CONCAT 例子.....611
6.60 HOUR 函数.....613
6.60.1 HOUR 函数例子.....613
6.61 INSERT 函数.....614
6.61.1 INSERT 函数例子.....614
6.62 IS_IPV4 函数.....615
6.62.1 IS_IPV4 例子.....615
6.63 IS_IPV6 函数.....616
6.63.1 IS_IPV6 例子.....616
6.64 INET_ATON 函数.....617
6.64.1 INET_ATON 例子.....617
6.65 INET_NTOA 函数.....618

6.65.1 INET_NTOA 例子 618
6.66 ISNULL 函数 619
6.66.1 ISNULL 函数例子 619
6.67 JULIANTIMESTAMP 函数 620
6.67.1 JULIANTIMESTAMP 函数注意事项 620
6.67.2 JULIANTIMESTAMP 函数例子 620
6.68 LCASE 函数 622
6.68.1 LCASE 函数例子 622
6.69 LEFT 函数 623
6.69.1 LEFT 函数例子 623
6.70 LISTAGG 函数 625
6.70.1 LISTAGG 函数的例子 625
6.71 LOCATE 函数 627
6.71.1 LOCATE 函数注意事项 627
6.71.2 LOCATE 函数例子 627
6.72 LOG 函数 629
6.72.1 LOG 函数例子 629
6.73 LOG10 函数 630
6.73.1 LOG10 函数例子 630
6.74 LOWER 函数 631
6.74.1 LOWER 函数注意事项 631
6.74.2 LOWER 函数例子 631
6.75 LPAD 函数 633
6.75.1 LPAD 函数例子 633
6.76 LTRIM 函数 634
6.76.1 LTRIM 函数注意事项 634
6.77 MAX/MAXIMUM 函数 635
6.77.1 MAX/MAXIMUM 函数注意事项 635
6.77.2 MAX/MAXIMUM 函数例子 635
6.78 MD5 函数 636
6.78.1 MD5 例子 636

6.79 MIN 函数 637
6.79.1 MIN 函数注意事项 637
6.79.2 MIN 函数例子 637
6.80 MINUTE 函数 638
6.80.1 MINUTE 函数例子 638
6.81 MOD 函数 639
6.81.1 MOD 函数例子 639
6.82 MONTH 函数 640
6.82.1 MONTH 函数例子 640
6.83 MONTHNAME 函数 641
6.83.1 MONTHNAME 函数注意事项 641
6.83.2 MONTHNAME 函数例子 641
6.84 NULLIF 函数 642
6.84.1 NULLIF 函数例子 642
6.85 NULLIFZERO 函数 643
6.85.1 NULLIFZERO 函数例子 643
6.86 NVL 函数 644
6.86.1 NVL 函数例子 644
6.87 NVL2 函数 646
6.87.1 NVL2 函数例子 646
6.88 OCTET_LENGTH 函数 647
6.88.1 OCTET_LENGTH 函数注意事项 647
6.88.2 OCTET_LENGTH 函数例子 647
6.89 OVERLAY 函数 648
6.89.1 OVERLAY 语法 648
6.89.2 OVERLAY 例子 649
6.90 PI 函数 652
6.90.1 PI 函数例子 652
6.91 PIVOT 函数 653
6.91.1 PIVOT 语法 653
6.91.2 PIVOT 例子 654

6.92 POSITION 函数 656
6.92.1 POSITION 函数注意事项 656
6.92.2 POSITION 函数例子 657
6.93 POWER 函数 658
6.93.1 POWER 函数例子 658
6.94 QUARTER 函数 659
6.94.1 QUARTER 函数例子 659
6.95 RADIANS 函数 660
6.95.1 RADIANS 函数例子 660
6.96 RAND 函数 661
6.96.1 RAND 函数语法 661
6.96.2 RAND 函数注意事项 661
6.96.3 RAND 函数例子 662
6.97 REGEXP_COUNT 函数 665
6.97.1 REGEXP_COUNT 函数例子 665
6.98 REGEXP_REPLACE 函数 666
6.98.1 REGEXP_REPLACE 函数例子 666
6.99 REGEXP_SUBSTR 函数 667
6.99.1 REGEXP_SUBSTR 函数例子 667
6.100 REPEAT 函数 669
6.100.1 REPEAT 函数例子 669
6.101 REPLACE 函数 670
6.101.1 REPLACE 函数例子 670
6.102 REVERSE 函数 671
6.102.1 REVERSE 语法 671
6.102.2 REVERSE 例子 671
6.103 RIGHT 函数 673
6.103.1 RIGHT 函数例子 673
6.104 ROLLUP 函数 674
6.104.1 ROLLUP 注意事项 674
6.104.2 ROLLUP 例子 674

6.105 ROUND 函数 680
6.105.1 ROUND 函数例子 680
6.106 RPAD 函数 682
6.106.1 RPAD 函数例子 682
6.107 RTRIM 函数 684
6.107.1 RTRIM 函数注意事项 684
6.107.2 RTRIM 函数例子 684
6.108 SECOND 函数 685
6.108.1 SECOND 函数例子 685
6.109 SHA 函数 686
6.109.1 SHA 例子 686
6.110 SHA2 函数 687
6.110.1 SHA2 例子 687
6.111 SIGN 函数 688
6.111.1 SIGN 函数例子 688
6.112 SIN 函数 689
6.112.1 SIN 函数例子 689
6.113 SINH 函数 690
6.113.1 SINH 函数例子 690
6.114 SLEEP 函数 691
6.114.1 SLEEP 例子 691
6.115 SPACE 函数 692
6.115.1 SPACE 函数例子 692
6.116 SPLIT_PART 函数 693
6.116.1 SPLIT_PART 语法 693
6.116.2 SPLIT_PART 例子 694
6.117 SQRT 函数 697
6.117.1 SQRT 函数例子 697
6.118 STDDEV 函数 698
6.118.1 STDDEV 函数注意事项 698
6.118.2 STDDEV 函数例子 699

6.119 STRINGTOLOB 函数 700
6.119.1 STRINGTOLOB 函数语法 700
6.119.2 STRINGTOLOB 函数例子 700
6.120 SUBSTRING/SUBSTR 函数 701
6.120.1 替代形式 701
6.120.2 SUBSTRING/SUBSTR 函数注意事项 702
6.120.3 SUBSTRING/SUBSTR 函数例子 702
6.120.1 SUBSTRING/SUBSTR 函数兼容 Oracle 703
6.121 SUM 函数 704
6.121.1 SUM 函数注意事项 704
6.121.2 SUM 函数例子 704
6.122 SYSDATE 函数 706
6.122.1 SYSDATE 例子 706
6.123 SYSTIMESTAMP 函数 709
6.123.1 SYSTIMESTAMP 例子 709
6.124 SYS_GUID 函数 710
6.124.1 结果的数据类型 710
6.124.2 SYS_GUID 例子 710
6.125 TAN 函数 711
6.125.1 TAN 函数例子 711
6.126 TANH 函数 712
6.126.1 TANH 函数例子 712
6.127 TIMESTAMPADD 函数 713
6.127.1 TIMESTAMPADD 函数例子 713
6.128 TIMESTAMPDIFF 函数 715
6.128.1 TIMESTAMPDIFF 函数例子 715
6.129 TO_CHAR 函数 717
6.129.1 TO_CHAR 语法 717
6.129.2 TO_CHAR 注意事项 722
6.129.3 TO_CHAR 例子 723
6.130 TO_DATE 函数 732

6.130.1 TO_DATE 注意事项 733
6.130.2 TO_DATE 例子 733
6.131 TO_TIME 函数 734
6.131.1 TO_TIME 例子 734
6.132 TO_TIMESTAMP 函数 735
6.132.1 TO_TIMESTAMP 例子 735
6.133 TRANSLATE 函数 736
6.134 TRIM 函数 737
6.134.1 TRIM 函数注意事项 737
6.134.2 TRIM 函数例子 737
6.135 TRUNC 函数 739
6.135.1 TRUNC 语法 739
6.135.2 TRUNC 注意事项 740
6.135.3 TRUNC 例子 740
6.136 UCASE 函数 741
6.136.1 UCASE 函数注意事项 741
6.136.2 UCASE 函数例子 741
6.137 UPPER 函数 743
6.137.1 UPPER 函数例子 743
6.138 UPSHIFT 函数 744
6.138.1 UPSHIFT 函数例子 744
6.139 USER 函数 746
6.139.1 USER 函数注意事项 746
6.139.2 USER 函数例子 746
6.140 UUID 函数 747
6.140.1 结果的数据类型 747
6.140.2 UUID 例子 747
6.141 UUID_SHORT 函数 748
6.141.1 结果的数据类型 748
6.141.2 UUID_SHORT 例子 748
6.142 UNIX_TIMESTAMP 函数 749

6.142.1 UNIX_TIMESTAMP 例子 750
6.143 VARIANCE 函数 751
6.143.1 VARIANCE 函数注意事项 751
6.143.2 VARIANCE 函数例子 752
6.144 WEEK 函数 754
6.144.1 WEEK 函数例子 754
6.145 YEAR 函数 755
6.145.1 YEAR 函数例子 755
6.146 ZEROIFNULL 函数 756
6.146.1 ZEROIFNULL 函数例子 756
7. OLAP 函数 757
7.1 窗口函数注意事项 757
7.1.1 空值 758
7.2 ORDER BY 子句支持 OLAP 函数表达式 759
7.3 窗口函数的使用限制 761
7.4 AVG 窗口函数 763
7.4.1 AVG 窗口函数例子 764
7.5 COUNT 窗口函数 765
7.5.1 COUNT 窗口函数例子 766
7.6 DENSE_RANK 窗口函数 767
7.6.1 DENSE_RANK 窗口函数例子 767
7.7 FIRST_VALUE 窗口函数 768
7.7.1 FIRST_VALUE 窗口函数例子 768
7.8 LAG 窗口函数 769
7.8.1 LAG 窗口函数例子 769
7.9 LAST_VALUE 窗口函数 770
7.10 LEAD 窗口函数 771
7.10.1 LEAD 窗口函数例子 771
7.11 LISTAGG 窗口函数 772
7.11.1 LISTAGG 窗口函数的例子 772
7.12 MAX 窗口函数 774

7.12.1 MAX 窗口函数例子 775
7.13 MIN 窗口函数 776
7.13.1 MIN 窗口函数例子 777
7.14 RANK 窗口函数 778
7.14.1 RANK 窗口函数例子 778
7.15 ROW_NUMBER 窗口函数 779
7.15.1 ROW_NUMBER 窗口函数例子 779
7.16 STDDEV 窗口函数 780
7.16.1 STDDEV 函数例子 781
7.17 STDDEV_POP 窗口函数 782
7.17.1 STDDEV_POP 窗口函数例子 783
7.18 SUM 窗口函数 784
7.18.1 SUM 窗口函数例子 785
7.19 SYS_CONNECT_BY_PATH 窗口函数 786
7.19.1 SYS_CONNECT_BY_PATH 函数例子 786
7.20 VARIANCE 窗口函数 787
7.20.1 VARIANCE 窗口函数例子 788
8. SQL 运行时统计信息 789
8.1 PERTABLE 和 OPERATOR 统计 790
8.2 适应性统计集合 790
8.3 检索 SQL 运行时统计信息 791
8.3.1 使用 GET STATISTICS 命令 791
8.3.2 GET STATISTICS 命令语法 791
8.3.3 GET STATISTICS 命令例子 793
8.4 显示 SQL 运行时统计信息 799
8.4.1 显示 SQL 运行时统计信息的例子 805
8.4.2 一条正在执行中的语句的 PERTABLE 统计信息 806
8.4.3 一条正在执行中的语句的 ACCUMULATED 统计信息 808
8.4.4 一条正在执行中的语句的 PROGRESS 统计信息 809
8.4.5 一条正在执行中的语句的 DEFAULT 统计信息 811
8.4.6 使用父查询 ID 812

8.4.7 子查询 ID 813
8.5 收集 RMS 的相关统计信息 815
8.6 使用 QUERYID_EXTRACT 函数 817
8.6.1 QUERYID_EXTRACT 函数语法 818
8.6.2 QUERYID_EXTRACT 函数例子 818
8.7 一个查询语句的每个碎片实例的统计信息.....819
8.7.1 STATISTICS 表值函数语法 819
8.7.2 获取查询语句的每个碎片实例的统计信息时的注意事项.....820
附录 A 保留字.....822
附录 B 控制查询默认（CQD）属性.....827
1. HBase 环境 CQD.....827
2. Hive 环境 CQD 827
3. 管理直方图.....827
4. 优化器.....831
5. 管理 Schema.....833
6. 事务控制和锁定.....834
附录 C 使用限制.....836

关于本文档

本手册描述的参考信息涉及到基于 Trafodion 数据库引擎的 EsgynDB 产品所支持的 SQL 语句、函数和其他 SQL 语言元素的语法。

EsgynDB SQL 语句和实用工具命令既可以交互输入，也可以使用诸如 EsgynDB 命令界面（TrafCI）之类的基于客户端的工具从脚本文件输入。

更多由于安装和配置连接和使用 EsgynDB 数据库的客户端应用程序的信息，请参阅《EsgynDB 安装部署指南》。

在本手册中，SQL 语言元素、语句、以及语句内部的子句都基于 ANSI SQL:1999 标准而定。

目标读者

本手册适用于数据库管理员和应用程序员，以使用 SQL 读取、更新和创建映射到 HBase 表的 EsgynDB SQL 表，并访问原生 Hbase 表和 Hive 表。

您应该熟悉结构化查询语言（SQL）和美国国家数据库语言标准 SQL:1999。

新增和更改的信息

此版本包含以下新功能的更新:

版本	日期	说明
2.8.0	2021 年 3 月	删除 sequence by 语句以及相关窗口函数和其他信息 UPDATE STATISTICS 增加 FOR TABLE LIKE 语法 补充一些 EsgynDB SQL 语法指南 2.7.0 中的信息 4.8.2.1 中增加 VARCHAR 和 VARCHAR2 的区别 MOD 函数增加%用法 新增 6.56 FIND_IN_SET 函数 CREATE TABLE 和 CREATE INDEX 中增加 VOLATILE 和 TEMPORARY 等同说明 2.18.2.3 中增加新的临时表机制说明 2.62.1UPSERT 语法小节中添加 USING LOAD 注意事项
2.7.0	2020 年 4 月	GRANT COMPONENT PRIVILEGE 语法中的组件权限列表更新 alter sequence 语句增加 RESTART WITH 选项 添加 NVL2 函数 添加数据类型的隐式转换 添加 Select 语句中 for update 子句 create schema/create table/alter table 中添加 stored desc/incremental backup 属性 添加 update using upsert 添加 ROWID 和 ROWNUM 添加 REGEXP_COUNT，REGEXP_SUBSTR 以及 REGEXP_REPLACE 函数 添加 LISTAGG 函数和 LISTAGG 窗口函数 添加 SYS_CONNECT_BY_PATH 函数 添加 SAVEPOINT 使用方法 添加 Null 支持四则运算和逻辑运算 添加 QianBase hint 语法 修改 CREATE TABLE 语句中 like 部分 添加以下语法说明: • VARCHAR*VARCHAR

		• 支持 systemstamp(n) • order by nulls first/last • 兼容 oracle 的 • to_char(xxx,‘99999999999999.99’) • to_char 和 to_date 函数新增格式 • connect by 递归语法增强 • Oracle Number 兼容，自动转换成 Numeric • 语法支持事务中使用 TRUNCATE • 兼容 Oracle 语法支持 hint • Sequence 的 ORDER|NO ORDER • to_char 返回类型由原来的 char 改成 varchar • Order by 语句中支持使用函数进行排序 • SUBSTR 的起始位置为负数的行为实现与 Oracle 兼容 • 兼容 Oracle，在聚合函数中也支持字符串到数值的隐式转换 • 其他。 添加 Global Sequence 添加 DUMP 命令 添加 TRUNCATE 命令
2.6.0	2020 年 1 月	第一版

文档结构

章或附录	说明
1 介绍	介绍 EsgynDB SQL，并包括数据一致性、事务管理 ANSI 合规性之类的主题。
2 SQL 语句	介绍 EsgynDB SQL 所支持的 SQL 语句。
3 SQL 实用工具	介绍 EsgynDB SQL 所支持的 SQL 实用工具。
4 SQL 语言元素	介绍部分 SQL 语言，例如 EsgynDB SQL 语句语法里出现的数据库对象、数据类型、表达式、标识符、字面值和谓词。
5 SQL 子句	介绍 EsgynDB SQL 语句所使用的子句。
6 SQL 函数和表达式	介绍能在 EsgynDB SQL 语句里使用的特定函数和表达式。
7 OLAP 函数	介绍特定的 OLAP（联机分析处理）函数。
8 SQL 运行时统计信息	介绍如何收集查询语句或运行时管理系统（RMS）的统计信息和返回的 RMS 计数器。
附录 A 保留字	列出 EsgynDBSQL 里保留的文字。
附录 B 控制查询默认（CQD）属性	介绍 EsgynDB SQL 环境所支持的控制查询默认（CQD）属性。
附录 C 使用限制	介绍 EsgynDB SQL 的使用限制。

符号约定

此列表总结了本手册中用于语法表现的符号约定。

- 大写字母

  大写字母表示关键字和保留字。按如下所示准确键入这些项目。未包含在括号内的项目是必需的。

  ```
  SELECT
  ```

- 小写字母
  小写字母(无论字体如何)表示所提供的变量项目。未包含在括号内的项目是必需的。

  ```
  file-name
  ```

- [ ]方括号
  方括号括住可选语法项。

  ```
  DATETIME [start-field TO] end-field
  ```

  方括号内的一组项目是一个列表，您可以从中选择一个项目或不选择任何项目。

  列表中的项目可以垂直排列，在列表的每一侧都有对齐的方括号，也可以水平排列，用一对方括号括起来，并用垂直线隔开。

  例如:

  ```
  DROP SCHEMA schema [CASCADE]
  DROP SCHEMA schema [ CASCADE | RESTRICT ]
  ```

- { } 大括号
  大括号括住所需的语法项目。

  ```
  FROM { grantee [, grantee ] ... }
  ```

  大括号内的一组项目是一个列表，您需要从中选择一个项目。

  列表中的项目可以垂直排列，在列表的每一侧都有对齐的大括号，也可以水平排列，用一对大括号括起来，并用垂直线隔开。

  例如:

  ```
  INTERVAL { start-field TO end-field }
  { single-field }
  INTERVAL { start-field TO end-field | single-field }
  ```

- |垂直线
  垂直线将水平列表中括在方括号或大括号中的选项分隔开。

  ```
  {expression | NULL}
  ```

- ...省略号

紧跟在一对方括号或大括号后面的省略号表示可以多次重复所包含的语法项序列。

```
ATTRIBUTE[S] attribute [, attribute] ...
{, sql-expression } ...
```

紧跟在单个语法项后面的省略号表示可以多次重复该语法项。

例如:

```
expression-n ...
```

- 标点符号
  圆括号、逗号、分号和其他之前未描述的符号必须按如下所式键入。

```
DAY (datetime-expression)
@script-file
```

符号（如方括号或大括号）周围的引号表示该符号是一个必须按如下所示键入的字符。

例如:

```
"{" module-name [, module-name] ... "}"
```

- 项目间距
  除非其中一个项目是一个标点符号，如圆括号或逗号，否则项目之间必需显示空格。

```
DAY (datetime-expression) DAY(datetime-expression)
```

如果两个项目之间没有空格，则不允许使用空格。在该例中，句号和任何其他项目之间不允许使用空格:

```
myfile.sh
```

- 行间距
  如果命令的语法太长而无法放在一行中，则每一个延续行都缩进三个空格，并用空行与前一行分隔开。
  此间距将连续行中的项目与垂直选择列表中的项目区分开来。

```
match-value [NOT] LIKE _pattern
[ESCAPE esc-char-expression]
```

1. 介绍

EsgynDB SQL 数据库软件允许您使用符合 ANSI SQL:1999 标准的 SQL 语句访问映射到 HBase 表的 EsgynDB SQL 表里的数据，并访问原生的 HBase 表和 Hive 表。

本章目录:

1.1 SQL 语言
1.2 使用 EsgynDB SQL 访问 HBase 表
1.3 使用 EsgynDB SQL 访问 Hive 表
1.4 数据一致性和访问选项
1.5 事务管理
1.6 ANSI 合规性和 EsgynDB SQL 扩展语句
1.7 EsgynDB SQL 错误消息

本手册的其他章节描述了单个语句、命令和语言元素的语法和语义。

1.1 SQL 语言

SQL 语言包含可用来访问 SQL 数据库的语句和其他语言元素。
更多关于 SQL 语句的信息，请参阅 SQL 语句。

SQL 语言元素是语句和命令的一部分，它包括数据类型、表达式、函数、标识符、字面值和谓词。更多信息，请参阅 SQL 语言元素 和 SQL 子句。

更多关于特定函数和表达式的详细信息， 请参阅 SQL 函数和表达式和 OLAP 函数。

1.2 使用 EsgynDB SQL 访问 HBase 表

- 可以使用 EsgynDB SQL 语句读取、更新和创建 HBase 表。
- 初始化 EsgynDB 元数据
- HBase 表的访问方式
- EsgynDB SQL 表与原生 HBase 表的对比
- HBase 表支持的 SQL 语句

更多关于 HBase 环境控制查询默认（CQD）设置列表的信息，请参阅附录 B.1 HBase 环境 CQD。

1.2.1 初始化 EsgynDB 元数据

首次使用 SQL 语句访问 HBase 表之前，需要初始化 EsgynDB 元数据。

运行以下命令初始化 EsgynDB 元数据:

1.2.2 HBase 表的访问方式

EsgynDB SQL 支持以下 HBase 表访问方式:

- 访问 EsgynDB SQL 表
- 通过 Cell-Per-Row 方式访问 HBase 表
- 逐行访问 HBase 表

1.2.2.1 访问 EsgynDB SQL 表

EsgynDB SQL 表是由 CREATE TABLE 语句生成、并映射到 HBase 表的一种关系型 SQL 表。

EsgynDB SQL 表在 TRAFODION 编目里都有常规的 ANSI 名称。

EsgynDB SQL 表可以具有一个完全限定的 ANSI 名称，具体形式为 TRAFODION.schema-name.object-name。

为了访问一个 EsgynDB SQL 表，请在 EsgynDB SQL 语句里指定其 ANSI 表名，这类似于在关系型数据库里运行 SQL 语句时指定 ANSI 表名。例如:

```
CREATE TABLE trafodion.sales.odetail
( ordernum NUMERIC (6) UNSIGNED NO DEFAULT NOT NULL,
partnum NUMERIC (4) UNSIGNED NO DEFAULT NOT NULL,
unit_price NUMERIC (8,2) NO DEFAULT NOT NULL,
qty_ordered NUMERIC (5) UNSIGNED NO DEFAULT NOT NULL,
PRIMARY KEY (ordernum, partnum) );
```

```
INSERT INTO trafodion.sales.odetail VALUES ( 900000, 7301,
425.00, 100 );

SET SCHEMA trafodion.sales;
SELECT * FROM odetail;
```

EsgynDB SQL 表的详细信息，请参阅对比：EsgynDB SQL 表与原生 HBase 表。

1.2.2.2 通过 Cell-Per-Row 方式访问 HBase 表

为了通过 Cell-Per-Row 方式访问 HBase 数据，请指定 schema HBASE."_CELL_"，并且该表的 ANSI 完整名称为分隔的表名。

您可以指定任何 HBase 表名，无论其是否通过 EsgynDB SQL 创建。例如：

```
select * from hbase."_CELL_"."TRAFODION.MYSCH.MYTAB";
select * from hbase."_CELL_"."table_created_in_HBase";
```

通过该 schema 访问的所有表都具有相同的列布局：

```
>>invoke hbase."_CELL_"."table_created_in_HBase";
      (

        ROW_ID          VARCHAR(100) ...
        , COL_FAMILY    VARCHAR(100) ...
        , COL_NAME      VARCHAR(100) ...
        , COL_TIMESTAMP LARGEINT ...
        , COL_VALUE     VARCHAR(1000) ...
        )
        PRIMARY KEY (ROW_ID)

>>select * from hbase."_CELL_"."mytab";
```

1.2.2.3 逐行访问 HBase 表

为了使用逐行模式访问 HBase 数据，需指定 schema HBASE."_ROW_"，并且该表的 ANSI 完整名称为分隔的表名。

您可以指定任何 HBase 表名，无论其是否通过 EsgynDB SQL 创建。例如：

```
select * from hbase."_ROW_"."TRAFODION.MYSCH.MYTAB";
select * from hbase."_ROW_"."table_created_in_HBase";
```

该行的所有列值都将作为 big varchar 而被返回：

```
>>invoke hbase."_ROW_"."mytab";
(

        ROW_ID VARCHAR(100)  ...
         , COLUMN_DETAILS VARCHAR(10000)  ...
         )
         PRIMARY KEY (ROW_ID)

>>select * from hbase."_ROW_"."mytab";
```

1.2.3 对比：EsgynDB SQL 表与原生 HBase 表

与常规 HBase 表相比，EsgynDB SQL 表具有以下优势：

- 看起来是带有固定列的常规的结构化 SQL 表。
- 支持关系型数据库里常见的 SQL 数据类型。
- 支持复合键，不像 HBase 表只有一个行键（字符串）。
- 支持索引。
- 支持 salting，它会添加行键的哈希值作为前缀，以防出现序列键热点。更多信息，请参阅 CREATE TABLE 语句。

EsgynDB SQL 表的问题在于它们使用一种固定格式表示列值，这使原生 HBase 应用程序难以访问。此外，它们的一种固定结构使 HBase 自带动态列不灵活。

1.2.4 HBase 表支持的 SQL 语句

您可以在 HBase 表上使用这些 SQL 语句：

SELECT 语句	CREATE INDEX 语句
INSERT 语句	CREATE TABLE 语句
UPDATE 语句	CREATE VIEW 语句
DELETE 语句	DROP INDEX 语句
MERGE 语句	DROP TABLE 语句
GET 语句	DROP VIEW 语句
INVOKE 语句	GRANT 语句
ALTER TABLE 语句	REVOKE 语句

1.3 使用 EsgynDB SQL 访问 Hive 表

您可以使用 EsgynDB SQL 语句访问 Hive 表。

- Hive 表的 ANSI 名称
- Type Hive 到 EsgynDB SQL 的类型映射
- Hive 表支持的 SQL 语句

更多关于 Hive 环境控制查询默认（CQD）设置列表的信息，请参阅附录 B.2 Hive 环境 CQD。

1.3.1 Hive 表的 ANSI 名称

Hive 表出现在一个名为 HIVE.HIVE 的特殊 catalog 和 schema 的 EsgynDB Hive ANSI 命名空间中。

若要对一个名为 T 的 Hive 表进行操作，需在 EsgynDB SQL 语句里指定一个隐式或显式名称，例如：HIVE.HIVE.T。如果已定义一个名为 T 的 Hive 表，则以下例子会生效：

```
set schema hive.hive;
cqd hive_max_string_length '20';    -- 产生更具可读性的显示内容
select * from t;                    -- 隐式表名
set schema trafodion.seabase;
select * from hive.hive.t;          -- 显式表名
```

1.3.2 Hive 到 EsgynDB SQL 的数据类型映射

EsgynDB 执行以下数据类型映射：

从 Hive 到 EsgynDB 的数字数据类型映射

从 Hive 到 EsgynDB 的 Numeric 数据类型映射

Hive [1]			EsgynDB		
数据类型	范围	字节	数据类型	范围	字节
微整型(有符号)	-128 ~ +127	1	微整型(有符号)	-128 ~ +127	1
小整型(有符号)	-32,768 ~ +32,767	2	小整型(有符号)	-32,768 ~ +32,767	2
整型，(有符号)	-2,147,483,648 ~ +2,147,483,647	4	整型(有符号)	-2,147,483,648 ~ +2,147,483,647	4

大整型(有符号)	-9,223,372,036,854,775,808~+9,223,372,036,854,775,807	8	大整型(有符号)	-9,223,372,036,854,775,808 ~ +9,223,372,036,854,775,807	8

1. Hive 只支持**有符号的数值数据类型**。

更多有关 Hive 和 EsgynDB 的数值数据类型的信息，请参阅下表。

Hive 和 EsgynDB 的数字数据类型对比

Hive			EsgynDB		
数据类型	**范围**	**字节**	**数据类型**	**范围**	**字节**
微整型(有符号)	-128 ~ +127	1	微整型（有符号)	-128 ~ +127	1
			微整形（无符号)	0 ~ 255	
小整型(有符号)	-32,768 ~ +32,767	2	小整型(有符号)	-32,768 ~ +32,767	2
			小整型，（无符号)	0 ~ 65,535	
整型(有符号)	-2,147,483,648 ~ +2,147,483,647	4	整型(有符号)	-2,147,483,648 ~ +2,147,483,647	4
			整形(无符号)	0 ~ 4,294,967,295	
大整型(有符号)	-9,223,372,036,854,775,808~+9,223,372,036,854,775,807	8	大整型，（有符号)	-9,223,372,036,854,775,808 ~ +9,223,372,036,854,775,807	8
/	/	/	大整型(无符号)	0 ~ 18,446,744,073,709,551,615	

从 Hive 到 EsgynDB 的字符串数据类型映射

从 Hive 到 EsgynDB 的 Character String 数据类型映射

Hive	EsgynDB
string	varchar(*n* 字节) 字符集 utf8[1]
varchar	varchar
char	char

1. 数值 n 由 CQD HIVE_MAX_STRING_LENGTH 决定。更多信息，请参阅附录 B.2 Hive 环境 CQD。

从 Hive 到 EsgynDB 的数字数据类型映射

从 Hive 到 EsgynDB 的 Numeric 数据类型映射

Hive	EsgynDB
float(浮点型)	实型
decimal (精度, 标度)	decimal (precision, scale)[1] numeric (precision, scale)[2]
double（双精度型）	float

1. 如果 p 小于或等于 18, decimal (precision, scale)被映射到 decimal (precision, scale)。
2. 如果 p 大于 18，decimal(precision, scale)被映射到 numeric(precision, scale)。

从 Hive 到 EsgynDB 的日期时间数据类型映射

从 Hive 到 EsgynDB 的 Datetime 数据类型映射

Hive	EsgynDB
timestamp	timestamp(6)[1]
date	date

1. Hive支持纳秒级分辨率（精度为9）的时间戳。EsgynDB SQL仅支持微秒级分辨率（精度为9）的时间戳。

1.3.3 Hive 表支持的 SQL 语句

您可以在 Hive 表上使用这些 SQL 语句：

- SELECT 语句
- LOAD 语句
- GET TABLES (更多信息，请参阅 GET 语句)
- INVOKE 语句

1.4 数据一致性和访问选项

DML 语句的访问选项会影响查询访问数据的一致性。

任何 DML 语句都能使用 FOR option ACCESS 子句指定访问选项。对于一条 SELECT 语句而言， 通过使用相同的子句，您能为 FROM 子句里引用的单个表和视图指定访问选项。

DML 语句里 option 的可能设置包括:

READ COMMITTED：规定 DML 语句所访问的数据必须来自已经提交的行。

DML 语句的 SQL 默认访问选项是 READ COMMITTED。更多信息，请参阅事务隔离级别。

1.4.1 READ COMMITTED

该选项仅允许您访问已提交的数据。

为了实现该选项，必须对 DML 语句请求的数据加锁，但实际上并不会锁定该数据，因此能减少锁请求的冲突。

如果无法授予一个锁（暗示该行包含未提交的数据），则 DML 语句请求将继续等待，直到目前持有的锁被释放为止。

与 READ UNCOMMITTED 相比，READ COMMITTED 提供了更高层次的数据一致性级别。使用该访问选项执行的语句不允许出现脏读，但允许出现不可重复读和幻读。

READ COMMITTED 为任何不需要可重复读能力的进程都提供了足够的一致性。READ COMMITTED 是默认的隔离级别。

1.5 事务管理

事务 (必须整体完成的一组数据库更改操作)是出现故障或中断时基本的可恢复单元。客户端工具通过使用 ODBC 或 JDBC 与数据库进行交互操作，从而控制事务。典型的事件顺序是:

1) 启动事务。
2) 进行数据库更改。
3) 提交事务。

然而，如果无法更改或不希望完成事务，您可以中止事务，这样数据库就可回滚到初始状态。

以下为事务管理的注意事项：

- 用户定义事务和系统定义事务
- DML 语句规则
- AUTOCOMMIT 选项效果
- 并发性
- 事务隔离级别

1.5.1 用户定义事务和系统定义事务

1.5.1.1 用户定义事务

用户定义事务是用户自己定义的事务。

为确定一系列语句执行成功与否，您能使用 BEGIN WORK 语句和 COMMIT WORK 语句定义一个包含这些语句的事务，还能使用 ROLLBACK WORK 语句中断事务。

若已开启 AUTOCOMMIT 选项，则不必显式结束事务，因为 EsgynDB SQL 将会自动结束事务。

有时会出现一个错误，要求中止用户定义的事务。

EsgynDB SQL 将会自动中止事务，并返回一个错误，指明事务已经回滚。

1.5.1.2 系统定义事务

系统定义的事务是在某些情况下，EsgynDB SQL 定义的事务。

大多数 DML 语句会在开始执行时隐式地发起事务。更多信息，请参阅隐式事务。然而，即使一个事务被隐式地发起，您仍然必须使用 COMMIT WORK 语句或 ROLLBACK WORK 语句显式地结束它。 若已开启 AUTOCOMMIT 选项，则不必显式结束事务。

1.5.2 DML 语句规则

若出现死锁，DML 语句将超时并收到一个错误。

1.5.3 AUTOCOMMIT 选项效果

您能在 SET TRANSACTION 语句中设置 AUTOCOMMI 选项。

若在语句执行结束时出现错误，该选项指定 EsgynDB SQL 自动提交或回滚事务。该选项适用于系统发起事务的任何语句。更多信息，请参阅 SET TRANSACTION 语句。

若开启该选项，那么 EsgynDB SQL 将在语句执行结束时自动提交或回滚针对数据库所做的任何更改。

1.5.4 并发性

并发性是两个或多个进程同时访问相同的数据。

可用的并发度——无论请求访问已被访问数据的进程已获准、还是被置于等待队列中——取决于访问模式的目的（读取或更新）和隔离级别。目前，仅有的隔离级别为 READ COMMITTED。

EsgynDB SQL 为大多数操作提交数据库并发访问，并通过并发控制、表的打开和关闭机制控制数据库访问。

对于 DML 操作而言，访问选项可以影响并发度。更多信息，请参阅数据一致性和访问选项。

1.5.5 事务隔离级别

事务的隔离级别为 READ COMMITTED。

1.5.5.1 READ COMMITTED

该选项符合ANSI标准，它使您的事务只能访问已提交的数据。如果指定的隔离级别为 READ COMMITTED，则无法获得行锁。

READ COMMITTED 提供更高级别的数据一致性。采用这种隔离级别执行的事务禁止出现脏读，但允许出现不可重复读和幻读。

READ COMMITTED 为任何无需可重复读能力的事务都提供了足够的一致性。

默认的隔离级别为 READ COMMITTED。

1.6 ANSI 合规性和 EsgynDB SQL 扩展语句

EsgynDB SQL 最符合 Core SQL 99 标准。此外，EsgynDB SQL 还包括 SQL 99 标准的一些特征和部分 SQL 2003 标准，以及特殊的 EsgynDB SQL 扩展语句。本手册里的语句和 SQL 元素都符合 ANSI 标准（不包括指定的 EsgynDB SQL 扩展语句）。

1.6.1 符合 ANSI 标准的语句

以下为符合 ANSI 标准、但某些可能包含 EsgynDB SQL 扩展语句：

- ALTER TABLE 语句
- CALL 语句
- COMMIT WORK 语句
- CREATE FUNCTION 语句
- CREATE PROCEDURE 语句
- CREATE ROLE 语句
- CREATE SCHEMA 语句

- CREATE TABLE 语句
- CREATE VIEW 语句
- DELETE 语句
- DROP FUNCTION 语句
- DROP PROCEDURE 语句
- DROP ROLE 语句
- DROP SCHEMA 语句
- DROP TABLE 语句
- DROP VIEW 语句
- EXECUTE 语句
- GRANT 语句
- GRANT ROLE 语句
- INSERT 语句
- MERGE 语句
- PREPARE 语句
- REVOKE 语句
- REVOKE ROLE 语句
- ROLLBACK WORK 语句
- SELECT 语句
- SET SCHEMA 语句
- SET TRANSACTION 语句
- TABLE 语句
- UPDATE 语句
- VALUES 语句

1.6.2 EsgynDB SQL 扩展语句

以下为符合 ANSI 标准的 EsgynDB SQL 扩展语句：

- ALTER LIBRARY 语句
- ALTER USER 语句
- BEGIN WORK 语句
- CONTROL QUERY CANCEL 语句
- CONTROL QUERY DEFAULT 语句
- CREATE INDEX 语句
- CREATE LIBRARY 语句
- DROP INDEX 语句

- DROP LIBRARY 语句
- EXPLAIN 语句
- GET 语句
- GET HBASE OBJECTS 语句
- GET VERSION OF METADATA 语句
- GET VERSION OF SOFTWARE 语句
- GRANT COMPONENT PRIVILEGE 语句
- INVOKE 语句
- LOAD 语句
- REGISTER USER 语句
- REVOKE COMPONENT PRIVILEGE 语句
- SHOWCONTROL 语句
- SHOWDDL 语句
- SHOWDDL SCHEMA 语句
- SHOWSTATS 语句
- UNLOAD 语句
- UNREGISTER USER 语句
- UPDATE STATISTICS 语句
- UPSERT 语句

1.6.3 符合 ANSI 标准的函数

以下为符合 ANSI 标准、但某一些可能包含 EsgynDB SQL 扩展语句：

- AVG 函数
- CASE（条件）表达式
- CAST 表达式
- CHAR_LENGTH 函数
- COALESCE 函数
- COUNT 函数
- CURRENT 函数
- CURRENT_DATE 函数
- CURRENT_TIME 函数
- CURRENT_TIMESTAMP 函数
- CURRENT_USER 函数
- EXTRACT 函数
- LOWER 函数

- MAX/MAXIMUM 函数
- MIN 函数
- NULLIF 函数
- OCTET_LENGTH 函数
- POSITION 函数
- SESSION_USER
- SUBSTRING/SUBSTR 函数
- SUM 函数
- TRIM 函数
- UPPER 函数

所有其他函数都是 EsgynDB SQL 扩展语句。

1.7 EsgynDB SQL 错误消息

EsgynDB SQL 会报告错误消息和异常情况。出现错误情况时，EsgynDB SQL 会返回一个错误编号和错误情况的简述。

例如：EsgynDB SQL 可能显示以下错误消息：

```
*** ERROR[1000] A syntax error occurred.
```

消息编号为 SQLCODE 值（不含符号），本例中 SQLCODE 值为 1000。

2. SQL 语句

本章介绍 EsgynDB SQL 语句的语法和语义。

EsgynDB SQL 语句既能交互输入，也能从脚本文件输入（使用基于客户端的工具，例如 EsgynDB 命令行界面）。
安装和配置能连接和使用 EsgynDB 数据库的客户端应用程序，请参阅《EsgynDB 安装部署指南》。

本章目录:

2.1 类别
2.2 ALTER LIBRARY 语句
2.3 ALTER SEQUENCE 语句
2.4 ALTER TABLE 语句
2.5 ALTER USER 语句
2.6 BEGIN WORK 语句
2.7 CALL 语句
2.8 COMMIT WORK 语句
2.9 CONTROL QUERY CANCEL 语句
2.10 CONTROL QUERY DEFAULT 语句
2.11 CREATE FUNCTION 语句
2.12 CREATE INDEX 语句
2.13 CREATE LIBRARY 语句
2.14 CREATE PROCEDURE 语句
2.15 CREATE ROLE 语句
2.16 CREATE SCHEMA 语句
2.17 CREATE SEQUENCE 语句
2.18 CREATE TABLE 语句
2.19 CREATE TRIGGER 语句
2.20 CREATE VIEW 语句
2.21 DELETE 语句
2.22 DROP FUNCTION 语句
2.23 DROP INDEX 语句
2.24 DROP LIBRARY 语句
2.25 DROP PROCEDURE 语句

2.26 DROP ROLE 语句
2.27 DROP SCHEMA 语句
2.28 DROP SEQUENCE 语句
2.29 DROP TABLE 语句
2.30 DROP TRIGGER 语句
2.31 DROP VIEW 语句
2.32 EXECUTE 语句
2.33 EXPLAIN 语句
2.34 GET 语句
2.35 GET HBASE OBJECTS 语句
2.36 GET VERSION OF METADATA 语句
2.37 GET VERSION OF SOFTWARE 语句
2.38 GRANT 语句
2.39 GRANT COMPONENT PRIVILEGE 语句
2.40 GRANT ROLE 语句
2.41 INSERT 语句
2.42 INVOKE 语句
2.43 MERGE 语句
2.44 PREPARE 语句
2.45 REGISTER USER 语句
2.46 REVOKE 语句
2.47 REVOKE COMPONENT PRIVILEGE 语句
2.48 REVOKE ROLE 语句
2.49 ROLLBACK WORK 语句
2.50 SAVEPOINT 语句
2.51 SELECT 语句
2.52 SET SCHEMA 语句
2.53 SET TRANSACTION 语句
2.54 SHOWCONTROL 语句
2.55 SHOWDDL 语句
2.56 SHOWDDL SCHEMA 语句
2.57 SHOWSTATS 语句
2.58 TABLE 语句
2.59 UNREGISTER USER 语句

2.60 UPDATE 语句
2.61 UPDATE USING UPSERT 语句
2.62 UPSERT 语句
2.63 VALUES 语句

2.1 类别

这些语句按照各自的功能分类如下：

- 数据定义语言（DDL）语句
- 数据操纵语言（DML）语句
- 事务控制语句
- 数据控制和安全性语句
- 存储过程和用户定义函数语句
- 预处理语句
- 控制语句
- 对象命名语句
- SHOW, GET 和 EXPLAIN 语句

2.1.1 数据定义语言（DDL）语句

这些语句创建、删除或修改 EsgynDB SQL Schema 或对象的定义。

语句	作用
ALTER LIBRARY 语句	更新 EsgynDB 数据库里的库对象的物理文件名。
ALTER TABLE 语句	修改表属性。
ALTER USER 语句	修改用户属性。
CREATE FUNCTION 语句	将采用 C 语言编写的用户定义函数（UDF）注册为 EsgynDB 数据库里的函数。
CREATE INDEX 语句	在表上创建索引。
CREATE LIBRARY 语句	在 EsgynDB 数据库里注册库对象。
CREATE PROCEDURE 语句	在 EsgynDB 数据库里注册 Java 方法作为 Java 存储过程（SPJ）。
CREATE ROLE 语句	创建角色。
CREATE SCHEMA 语句	在数据库里创建 schema。
CREATE TABLE 语句	创建表。
CREATE VIEW 语句	创建视图。
DROP FUNCTION 语句	从 EsgynDB 数据库里删除用户定义函数（UDF）。
DROP INDEX 语句	删除索引。

DROP LIBRARY 语句	从 EsgynDB 数据库里删除库对象。
DROP PROCEDURE 语句	从 EsgynDB 数据库里删除 Java 存储过程（SPJ）。
DROP ROLE 语句	删除角色。
DROP SCHEMA 语句	从数据库里删除 schema。
DROP TABLE 语句	删除表。
DROP VIEW 语句	删除视图。
REGISTER USER 语句	在 SQL 数据库里注册用户，关联用户登录名和数据库用户名。
UNREGISTER USER 语句	从 SQL 数据库里删除数据库用户名。

2.1.2 数据操纵语言（DML）语句

这些语句删除、插入、选择或更新一个表或多个表中的行。

语句	作用
DELETE 语句	从一个表或视图里删除行。
INSERT 语句	将数据插入表或视图里。
MERGE 语句	执行 UPSERT 操作（若此行存在则更新表，否则，将该行插入表里），或将匹配行从一个表更新（合并）到另一个表。
SELECT 语句	从表和视图里检索数据。
TABLE 语句	等效于查询规范 SELECT * FROM table。
UPDATE 语句	更新一个表或视图的列值。
UPDATE USING UPSERT 语句	用于大批量更新表数据。
UPSERT 语句	若该行存在则更新表，否则，将该行插入表里。
VALUES 语句	显示行值构造函数内的表达式求值结果和行子查询的结果。

2.1.3 事务控制语句

这些语句指定用户定义的事务，并设置下一个事务的属性。

语句	作用
BEGIN WORK 语句	启动一个事务。
COMMIT WORK 语句	事务执行期间提交所做的更改，并结

	束事务。
ROLLBACK WORK 语句	事务执行期间撤销所做的更改，并结束事务。
SET TRANSACTION 语句	设置下一个 SQL 事务的属性——无论是否自动提交数据库更改。

2.1.4 数据控制和安全性语句

这些语句注册用户、创建角色、授予权限和撤销授权。

语句	作用
ALTER USER 语句	更改数据库里注册用户的关联属性。
CREATE ROLE 语句	创建 SQL 角色。
DROP ROLE 语句	删除 SQL 角色。
GRANT 语句	将 SQL 对象的访问权限授予特定的用户或角色。
GRANT COMPONENT PRIVILEGE 语句	将一个或多个组件的权限授予一个用户或角色。
GRANT ROLE 语句	将一个或多个角色授予一个用户。
REGISTER USER 语句	在 SQL 数据库里注册用户，关联用户登录名和数据库用户名。
REVOKE 语句	撤销特定用户或角色的 SQL 对象访问权限。
REVOKE COMPONENT PRIVILEGE 语句	删除用户或角色的一个或多个组件的权限。
REVOKE ROLE 语句	删除用户的一个或多个角色。
UNREGISTER USER 语句	从 SQL 数据库里删除数据库用户名。

2.1.5 存储过程和用户定义函数语句

这些语句创建和执行 Java 存储过程（SPJ）或创建用户定义函数（UDF），以修改库的访问权限、执行 SPJ 或 UDF：

语句	作用
ALTER LIBRARY 语句	更新 EsgynDB 数据库里的库对象的物理文件名。
CALL 语句	初始化 EsgynDB SQL 数据库里的 Java

	存储过程（SPJ）。
CREATE FUNCTION 语句	将采用 C 语言编写的用户定义函数（UDF）注册为 EsgynDB 数据库里的函数。
CREATE LIBRARY 语句	在 EsgynDB 数据库里注册库对象。
CREATE PROCEDURE 语句	将 Java 方法注册为 EsgynDB 数据库里的 Java 存储过程（SPJ）。
DROP FUNCTION 语句	从 EsgynDB 数据库里删除用户定义函数（UDF）。
DROP LIBRARY 语句	从 EsgynDB 数据库里删除库对象。
DROP PROCEDURE 语句	从 EsgynDB 数据库里删除 Java 存储过程（SPJ）。
GRANT 语句	授权特定用户访问库对象或执行 SPJ 或 UDF。
REVOKE 语句	撤销特定用户访问库对象或执行 SPJ 或 UDF 的权限。

2.1.6 预处理语句

这些语句预处理并执行一条 SQL 语句。

语句	作用
EXECUTE 语句	执行已通过 PREPARE 语句预处理的 SQL 语句。
PREPARE 语句	编译 SQL 语句，以便以后在同一个会话里和 EXECUTE 语句配套使用。

2.1.7 控制语句

这些语句控制 DML 语句的执行、默认选项、查询执行计划和性能。

语句	作用
CONTROL QUERY CANCEL 语句	取消带查询 ID 且正在执行的查询。
CONTROL QUERY DEFAULT 语句	更改影响查询计划的默认属性。

2.1.8 对象命名语句

该语句指定 schema 默认 ANSI 名称。

语句	作用

SET SCHEMA 语句	为当前会话的非限定对象名称设置默认 ANSI schema。

2.1.9 SHOW, GET 和 EXPLAIN 语句

这些语句显示数据库对象或查询执行计划的相关信息。

语句	作用
EXPLAIN 语句	显示查询执行计划里包含的信息。
GET 语句	显示数据库对象名称、组件、组件权限、角色或 EsgynDB 里存在的用户。
GET HBASE OBJECTS 语句	通过 SQL 接口显示 HBase 对象列表。
GET VERSION OF METADATA 语句	显示 EsgynDB 实例中的元数据版本，并标明元数据是否为当前版本。
GET VERSION OF SOFTWARE 语句	显示系统上安装的 EsgynDB 软件版本，并标明其是否为当前版本。
INVOKE 语句	生成一条记录描述，它对应指定表或视图里的行。
SHOWCONTROL 语句	显示已生效的 CONTROL QUERY DEFAULT 属性。
SHOWDDL 语句	显示元数据里的对象在创建时使用的 DDL 语法，或以 GRANT 语句形式返回一条用户、角色或组件的描述。
SHOWDDL SCHEMA 语句	显示元数据里的 schema 在创建时使用的 DDL 语法，并显示该 schema 所有者的授权 ID。
SHOWSTATS 语句	显示表中一组或多组列的直方图统计信息，这些统计信息可优化的访问计划。

2.2 ALTER LIBRARY 语句

该语句更新 EsgynDB 数据库里的库对象的物理文件名。

一个库对象可以是 SPJ 的 JAR 文件或 UDF 的库文件。

ALTER LIBRARY 是 EsgynDB SQL 扩展语句。

```
ALTER LIBRARY [[catalog-name.]schema-name.]library-name
FILE library-filename
[HOST NAME host-name]
[LOCAL FILE host-filename]
```

2.2.1 ALTER LIBRARY 语法

- `[[catalog-name.]schema-name.]library-name`
  指定库对象的 ANSI 逻辑名称，其中每部分都是一个有效的 SQL 标识符，最长 128 个字符。
  指定 schema 里已注册的库对象的名称。
  如果您没有完全限定一个库名，EsgynDB SQL 将根据当前会话的 schema 限定它。更多信息，请参阅标识符和数据库对象名称。

- FILE `library-filename`
  指定已重新部署的库文件（SPJ 的 JAR 文件或 UDF 的库文件）的完整路径。

- HOST NAME `host-name`
  指定已部署文件所驻留的客户端主机的名称。

- LOCAL FILE `host-filename`
  指定已部署文件在客户端主机上的存储路径。

2.2.2 ALTER LIBRARY 注意事项

HOST NAME 和 LOCAL FILE 都与位置相关。

必需的权限

发出一条 ALTER LIBRARY 语句，必须满足以下条件之一：

- 您是 DB_ROOT（数据库根用户）。
- 您是库的所有者。
- 您拥有 SQL_OPERATIONS 组件的 ALTER 或 ALTER_LIBRARY 权限。

2.2.3 ALTER LIBRARY 例子

- 该语句为 SALES schema 中名为 SALESLIB 的库更新 JAR 文件（SPJ）：
  `ALTER LIBRARY sales.saleslib FILE Sales2.jar;`

- 该语句为默认 schema 中名为 MYUDFS 的库更新库文件（UDF）：
  ALTER LIBRARY myudfs FILE $TMUDFLIB;

2.3 ALTER SEQUENCE 语句

该语句更改现有序列的未来行为，包括 INCREMENT、MAXVALUE、CACHE 和 CYCLE。
可以同时指定多个选项，并用空格分隔。任何未更改的选项都将保留其以前的设置。
ALTER SEQUENCE 是 EsgynDB SQL 扩展语句。
更多信息，请参阅 CREATE SEQUENCE 语句或 DROP SEQUENCE 语句。

```
ALTER SEQUENCE [[catalog-name.] schema-name.] sequence-
name
[INCREMENT BY integer]
[MAXVALUE integer | NO MAXVALUE]
[CACHE integer]
[CYCLE | NO CYCLE]
[RESTART WITH integer]
```

2.3.1 ALTER SEQUENCE 语法

上述选项的作用与创建序列时的作用相同。更多信息，请参阅 CREATE SEQUENCE 语句。

注意：

- START WITH, MINVALUE, NO CACHE 和 DATA TYPE 无法更改。而要使用所需的选项删除并重新创建序列。
  可以通过 RESTART WITH 来修改序列的起始值。

- 如果更改序列，则未使用的预分配值将丢失。

- 如果在第一次引用SEQNUM (*sequence-name*, NEXT)之前更改 INCREMENT BY，则会跳过某些序列值。

例如，
- 创建 *sequence1*.

```
SQL>CREATE SEQUENCE sequence1
START WITH 1
INCREMENT BY 1;
```

```
--- SQL operation complete.
```

- 将 INCREMENT BY 更改为 6.

```
SQL>ALTER SEQUENCE sequence1
INCREMENT BY 6;

  --- SQL operation complete.
```

- 第一次是通过调用SEQNUM (sequence-name, CURRENT)或 SEQNUM(sequence-name, NEXT)来返回序列的初始值，因此对 SEQNUM(sequence1, NEXT)的首次引用为1。

```
SQL>SELECT SEQNUM(sequence1, NEXT) FROM DUAL;

(EXPR)
--------------------
                   1

--- 1 row(s) selected.
```

- 随着后续对SEQNUM（sequencename，NEXT）的每次引用，它都将返回序列的一个新增量值。因为INCREMENT BY在首次引用SEQNUM（sequence1，NEXT）之前被更改为6，所以跳过2、3、4、5和6。

```
SQL>SELECT SEQNUM(sequence1, NEXT) FROM DUAL;

(EXPR)
--------------------
                   7
--- 1 row(s) selected.
```

2.3.2 ALTER SEQUENCE 注意事项

执行 ALTER SEQUENCE 语句，必须满足以下条件之一:

- 您是 DB_ROOT（数据库根用户）。
- 您拥有 SQL_OPERATIONS 组件的 ALTER 或 ALTER_SEQUENCE 权限。

2.3.3 ALTER SEQUENCE 例子

- 以下语句更改了序列*employee_seq*的MAXVALUE和CACHE，该序列在CRAETE SEQUENCE例子中创建。

```
ALTER SEQUENCE employee_seq
MAXVALUE 20000
CACHE 30
```

- 以下例子将序列seq1的起始值更改更改为100。

```
SQL>create sequence seq1;

--- SQL operation complete.

SQL>alter sequence seq1 restart with 100;

--- SQL operation complete.

SQL>showddl sequence seq1;


CREATE SEQUENCE TRAFODION.SEABASE.SEQ1
  START WITH 100
  INCREMENT BY 1
  MAXVALUE 9223372036854775806
  MINVALUE 1
  CACHE 25
  NOORDER
  NO CYCLE
  LARGEINT
;
```

2.4 ALTER TABLE 语句

该语句更改 EsgynDB SQL 表。更多信息，请参阅表。

```
ALTER TABLE name alter-action

alter-action is:
ADD [IF NOT EXISTS][COLUMN] column-definition
| ADD [CONSTRAINT constraint-name] table-constraint
| ADD PRIMARY KEY (column-name1[,column-name2])
| ENABLE ALL [UNIQUE] INDEXES
| ENABLE INDEX index-name
| DISABLE ALL [UNIQUE] INDEXES
| DISABLE INDEX index-name
| DROP CONSTRAINT constraint-name [RESTRICT]
| RENAME TO new-name
| DROP [COLUMN] [IF EXISTS] column-name
| ALTER COLUMN column-definition
| ATTRIBUTES (synchronous | no) replication
| GENERATE stored desc
| ATTRIBUTE incremental backup


column-definition is:
column-name data-type
([DEFAULT default]
[[CONSTRAINT constraint-name] column-constraint]...)

data-type is:
CHAR[ACTER] [(length)[CHARACTERS]]
[CHARACTER SET char-set-name]
[UPSHIFT] [[NOT] CASESPECIFIC]
| char[acter] varying (length [characters])
[CHARACTER SET char-set-name]
[UPSHIFT] [[NOT] CASESPECIFIC]
| varchar (length [characters])
[CHARACTER SET char-set-name]
[UPSHIFT] [[NOT] CASESPECIFIC]
| varchar2 (length [characters])
[CHARACTER SET char-set-name]
[UPSHIFT] [[NOT] CASESPECIFIC]
| nchar [(length [characters])]
[UPSHIFT] [[NOT] CASESPECIFIC]
| nchar varying (length [characters])
[UPSHIFT] [[NOT] CASESPECIFIC]
| numeric [(precision [,scale])] [signed|unsigned]
| tinyint [signed|unsigned]
| smallint [signed|unsigned]
| int[eger] [signed|unsigned]
| largeint
| dec[imal] [(precision [,scale])] [signed|unsigned]
| float [(precision)]
```

```
| real
| double precision
| date
| time [(time-precision)]
| timestamp [(timestamp-precision)]
| interval { start-field to end-field | single-field }
| blob [({numeric literal} [unit])] [STORAGE 'storage
literal']
| clob [({numeric literal} [unit])] [STORAGE 'storage
literal']
default is:
literal
| NULL
| CURRENT_DATE
| CURRENT_TIME
| CURRENT_TIMESTAMP }

column-constraint is:
NOT NULL
| UNIQUE
| PRIMARY KEY [ASC[ENDING] | DESC[ENDING]]
| CHECK (condition)
| REFERENCES ref-spec

table-constraint is:
UNIQUE (column-list)
| CHECK (condition)
| FOREIGN KEY (column-list) REFERENCES ref-spec

ref-spec is:
referenced-table [(column-list)]
column-list is:
     column-name[, column-name]...
```

2.4.1 ALTER TABLE 语法

- *table-name*

  指定当前对象名称。更多信息，请参阅数据库对象名称。

- ADD [COLUMN] column-definition

  添加一列到表中。

一个列定义的子句包括:

- column-name

  指定表的新列名。column-name 是一种 SQL 标识符。

  column-name 在表的列名里必须是唯一的。如果列名是 EsgynDB SQL 保留字，那么必须用双引号分隔。例如: "sql".myview.

- `data-type`
  指定列名里可存储值的数据类型。若无指定默认值，则使用 NULL。

- DEFAULT `default`
  指定一个列的默认值或指定该列不具有默认值。
  您既能通过 DEFAULT 子句显式地声明默认值，也能通过省略 DEFAULT 和 NOT NULL 子句启用 null 作为默认值。若省略 DEFAULT 子句，并且指定 NOT NULL，则 EsgynDB SQL 会返回一个错误。
  对于表中现有的行而言，添加的列都将取默认值。如果您将默认值设置为日期时间值-CURRENT_DATE、CURRENT_TIME 或 CURRENT_TIMESTAMP，那么 EsgynDB SQL 将使用 January 1, 1 A.D.12:00:00.000000 作为现有行的默认日期和时间。
  对于已添加列之后再添加的任何行而言，若没有为该列指定任何数值（作为添加行操作的一部分），那么在添加行时，该列将基于当前的时间戳接收一个默认值。

- [[CONSTRAINT `constraint-name`] `column-constraint`]
  指定一个列名或表约束名。
  `constraint-name` 必须与 `table` 具有相同的 schema，并且在其 schema 约束名中是唯一的。
  若您省略在 `constraint-name` 里指定名称的 schema 部分，则 EsgynDB SQL 将使用 table 的 schema 扩展该约束名。更多信息，请参阅数据库对象名称。
  如果您没有指定约束名，则 EsgynDB SQL 会构建一个 SQL 标识符作为该表 schema 里的约束名。该标识符包括一个完全限定的表名和系统生成的唯一标识符。例如：表约束 A.B.C 可能被分配 A.B.C_123..._01....的名称。

- `column-constraint` 选项：

  - NOT NULL
    这是一个列约束，它指定该列不能包含空值。如果您省略 NOT NULL，那么列中可以出现空值。如果您同时指定 NOT NULL 和 NO DEFAULT，那么插入表里的每行都必须包含一个该列的值。更多信息，请参阅 Null。

- UNIQUE

  这是一个列约束，它指定该列不能多次出现同一个数值。如果您省略 UNIQUE，那么允许出现重复数值，除非该列是 PRIMARY KEY 的一部分。您定义为 UNIQUE 的列必须被指定为 NOT NULL（非空）。

- `PRIMARY KEY [ASC[ENDING] | DESC[ENDING]]`

  这是一个列约束，它指定一列作为表的主键。
  ASCENDING 和 DESCENDING 指定键内一列中的条目的方向。默认值为 ASCENDING。

  表中每行 PRIMARY KEY 的值必须是唯一的。在任何 CREATE TABLE 语句中都只能指定一次 PRIMARY KEY。
  EsgynDB SQL 使用主键作为表的聚集键，以避免创建单独的唯一索引来实现主键约束。

- CHECK (`condition`)

  这是一个约束，它指定了表里的各行都必须满足的条件。更多详情，请参阅搜索条件。
  您不能在 CHECK 约束里引用 CURRENT_DATE、CURRENT_TIME 或 CURRENT_TIMESTAMP，也不能在 CHECK 约束里使用子查询。

- REFERENCES `ref-spec`

  指定一个 REFERENCES 列约束。采用 REFERENCES 约束的各列的最大组合长度为 2048 字节。

  `ref-spec` 是:
  `referenced-table [(column-list)]`

  `referenced-table` 是引用约束里的外键所引用的表。
  `referenced-table` 不能是一个视图。`referenced-table` 不能与 table 相同。

  `referenced-table` 对应于表的外键。

  `column-list`（列列表）指定 `referenced-table` 的一列或多列，`referenced-table` 对应于 `table` 的外键。

与 REFERENCES 相关联的列表、与 FOREIGN KEY 相关联的列表里的列必须具备相同的顺序。

若省略 column-list，那么被引用表的 PRIMARY KEY 列将是被引用的列。

一个表可以有无数的引用约束，您也可以在多个引用约束里指定相同的外键，但您必须单独定义每个引用约束。您无法创建自引用的外键约束。

- ADD [CONSTRAINT constraint-name] table-constraint
  向表中添加一个约束，并可以指定 constraint-name 作为该约束的名称。新的约束必须与表里现有的任何数据都保持一致。

- CONSTRAINT constraint-name
  指定列名或表约束名。
  constraint-name 必须与表具有相同的 schema，并且在其 schema 约束名中是唯一的。
  如果您省略在 constraint-name 里指定名称的 schema 部分，则 EsgynDB SQL 将使用 table 的 schema 扩展该约束名。更多信息，请参阅数据库对象名称。
  如果您没有指定约束名，则 EsgynDB SQL 会构建一个 SQL 标识符作为该表 schema 的约束名。该标识符包括一个完全限定的表名和系统生成的唯一标识符。例如：表约束 A.B.C 可能被分配 A.B.C_123..._01....的名称。

- table-constraint 选项：
  - UNIQUE (column-list)
    这是一种表约束，它指定该列或多列不得多次出现同一个数值或同一组数值。
    column-list 里不得多次出现相同的列。此外，在 UNIQUE 约束上指定的一组列不能与表的任何其他 UNIQUE 约束制定的一组列或表的 PRIMARY KEY 相匹配。所有被定义为 UNIQUE 的列都必须被指定为 NOT NULL。唯一性索引强制实施 UNIQUE 约束。
    如果 column-list 里已有一个唯一性索引，那么 EsgynDB SQL 将使用该索引。如果没有唯一性索引，那么系统会创建一个此类索引。

  - CHECK (condition)
    这是一种约束，它指定了表里的各行都必须满足的条件。更多信息，请

参阅搜索条件。

您既无法在 CHECK 约束里引用 CURRENT_DATE、CURRENT_TIME 或 CURRENT_TIMESTAMP，也无法在 CHECK 约束里使用子查询。

- FOREIGN KEY (column-list) REFERENCES ref-spec NOT ENFORCED

  这是一种表约束，它指定表的引用约束，并声明表里的一列或多列（外键）可以仅包含那些匹配 REFERENCES 子句指定表的一列或多列的数值。然而，由于已经指定了 NOT ENFORCED，所以不会检查这种关系。

  两列或一组列必须具备相同的特性（数据类型、长度、尺度、精度）。如果不存在 FOREIGN KEY 子句，则 `table` 的外键是已定义的列；如果存在 FOREIGN KEY 子句，则外键是 FOREIGN KEY 子句里指定的一列或多列。

- ENABLE ALL [UNIQUE] INDEXES
  - ENABLE ALL UNIQUE INDEXES

    启用表上的所有唯一索引。

  - ENABLE ALL INDEXES

    启用表上的所有索引(包括非唯一索引和唯一索引)。

- *index-name*

  是一个 SQL 标识符，它指定索引的简单名称。您无法使用其 schema 名限定 index-name。索引在 schema 中有自己的名称空间，因此索引名可能与表或约束名相同。但是，一个 schema 中的任何两个索引都不能具有相同的名称。

- ENABLE INDEX *index-name*

  启用表上的索引

- DISABLE ALL [UNIQUE] INDEXES
  - DISABLE ALL UNIQUE INDEXES

    禁用表上的所有唯一索引。

  - DISABLE ALL INDEXES

    禁用表上的所有索引(包括非唯一索引和唯一索引)。

- DISABLE INDEX `index-name`

  禁用表上的索引。索引仍然存在于数据中，但是优化器不会选择它作为访问路径。

- DROP CONSTRAINT **`constraint-name`** [RESTRICT]

  删除表中约束。

  删除约束时，如果 EsgynDB SQL 最初创建了相同的索引，那么 EsgynDB

SQL 将删除相关索引；如果该约束使用一个现有索引，那么该索引将不会被删除。

- CONSTRAINT `constraint-name`
  指定列名或表约束名。
  `constraint-name` 必须与 `table` 具有相同的 schema，并在其 schema 中拥有唯一约束名。
  如果省略在 `constraint-name` 里指定名称的 schema 部分，则 EsgynDB SQL 将使用表所在的 schema 扩展该约束名。更多信息，请参阅数据库对象名称。
  如果没有指定约束名，则 EsgynDB SQL 会构建一个 SQL 标识符作为该表 schema 的约束名。 该标识符包括一个完全限定的表名，以及系统生成的唯一标识符。 例如：表约束 A.B.C 可能被分配 A.B.C_123..._01...的名称。

- RENAME TO new-name [CASCADE]
  更改相同 schema 中对象的逻辑名称。

  `new-name`
  执行 RENAME TO 操作后，指定新的对象名称。CASCADE 规定重命名对象上的索引和约束都将被重新命名。

- ADD IF NOT EXISTS column-definition
  如果 `table` 中没有列，添加一列至 `table`。

  `column-definition` 的子句与 ADD [COLUMN] `column-definition`. 中描述的情况相同。

- DROP COLUMN [IF EXISTS]column-name
  从 `table` 中删除指定的列和该列的数据，但无法删除一个主键列。

- ALTER COLUMN column-name data-type
  更改现有 EsgynDB 表列的数据类型必须满足以下条件，否则将会返回 SQL 错误 1404。
  - 旧列（现有的列）和新列的数据类型都必须为 VARCHAR（变长字符串数据类型）。
  - 新列的长度必须大于或等于旧列的长度。
  - 新旧字符集必须相同。

- 改后的列不能为主键或二级索引键的一部分。
- 表也不能是 VOLATILE 表。

- ATTRIBUTES (synchronous | no) replication
  启用同步复制（和相关索引）或禁用复制更改表的复制属性。

- GENERATE stored desc
  更改 table，使其有 stored desc 属性，同时生成 descriptor。

- ATTRIBUTE incremental backup
  更改 table，使其有 incremental backup 属性。

2.4.2 ALTER TABLE 注意事项

2.4.2.1 添加一列对视图定义的影响

在表里添加一列对现有的视图定义没有任何影响。定义子句初始求值时，将使用显式列引用替换视图定义里 SELECT *指定的隐式列引用。

2.4.2.2 授权和可用性要求

ALTER TABLE 语句只能运行在用户创建的表上。

必需的权限

发出一条 ALTER TABLE 语句，必须满足以下条件之一：

- 您是 DB_ROOT（数据库根用户）。
- 您是库的所有者。
- 您拥有 SQL_OPERATIONS 组件的 ALTER 或 ALTER_TABLE 权限。

创建引用完整性约束所需要的权限

为了创建一个引用完整性约束（一个表约束，它引用另一个表的某列），必须满足下列条件之一：

- 您是 DB_ROOT（数据库根用户）。
- 您是引用表和被引用表的所有者。
- 您拥有引用表和被引用表以下权限：
  - 对于引用表，您拥有 SQL_OPERATIONS 组件的 ALTER 或 ALTER_TABLE 权限。
  - 对于被引用表，您能通过个人用户名或授予的角色拥有该表的 REFERENCES（或 ALL）权限。

如果该约束在查询表达式里引用另一个表，那么您必须拥有此表的 SELECT 权

限。

ALTER TABLE 语句是非事务性的。如果同时对某一对象执行 DML 操作（例如，INSERT/UPDATE/DELETE/UPSERT/MERGE 操作）与 ALTER TABLE 操作，则系统可能报错。

为了避免发生此类错误，在执行 ALTER TABLE 语句前，应停止所有执行 DML 操作的应用程序；在执行 ALTER TABLE 语句后，重启应用程序。

2.4.3 ALTER TABLE 例子

- 该例子可以添加一列：

```
ALTER TABLE persnl.project
     ADD COLUMN projlead
          NUMERIC (4) UNSIGNED
```

- 该例子可以更改现有表的列—— 正面和负面案例：

```
>>INVOKE T;

-- Definition of Trafodion table TRAFODION.SEABASE.T
-- Definition current Wed Jul 8 01:28:40 2015

(

SYSKEY  LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
, A          INT NO DEFAULT NOT NULL NOT DROPPABLE
, B          VARCHAR(30) CHARACTER SET ISO88591 COLLATE

     DEFAULT DEFAULT NULL

)

--- SQL operation complete.
>>ALTER TABLE T ALTER COLUMN B VARCHAR(40);

--- SQL operation complete.
>>INVOKE T;

-- Definition of Trafodion table TRAFODION.SEABASE.T
-- Definition current Wed Jul 8 01:29:03 2015

   (
  SYSKEY     LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
, A          INT NO DEFAULT NOT NULL NOT DROPPABLE
, B          VARCHAR(40) CHARACTER SET ISO88591 COLLATE
     DEFAULT DEFAULT NULL
```

```
)

--- SQL operation complete.
>>ALTER TABLE T ALTER COLUMN B VARCHAR(30);

*** ERROR[1404] Datatype for column B cannot be altered.

--- SQL operation failed with errors.
```

- 该例子可以更改列名。

```
Alter TABLE PRODUCT
ALTER COLUMN vend_id RENAME TO cstm_id;
```

- 该例子将列 c2 添加到表 t1，将 c2 设置为主键，并为主键指定 pk_t1。

  条目的方向是降序。

```
SQL>ALTER TABLE t1 ADD c2 int CONSTRAINT pk_t1 PRIMARY
KEY DESC;
--- SQL operation complete.


SQL>SHOWDDL t1;
CREATE TABLE TRAFODION.SEABASE.T1
(
C1 INT DEFAULT NULL NOT SERIALIZED
, C2 INT DEFAULT NULL NOT NULL NOT DROPPABLE
NOT SERIALIZED
, CONSTRAINT TRAFODION.SEABASE.PK_T1 PRIMARY KEY (C2
DESC)
)
ATTRIBUTES ALIGNED FORMAT
;
-- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON
TRAFODION.SEABASE.T1 TO
DB__ROOT WITH GRANT OPTION;
   --- SQL operation complete.
```

- NATIONAL CHAR（或 NCHAR）是 CHARACTER SET UCS2 的缩写，因此在使用 NCHAR 时无法更改 CHARACTER SET，如以下例子所示。

注意：无论字符集在安装时被指定为 ISO88591 还是 UTF8，NCHAR 或 NATIONAL CHAR 都隐式地假设字符集是 **UCS2**，并且 NCHAR 或 NATIONAL CHAR 的字符集（UCS2）不能被指定或更改。对于 ALTER 和

CREATE 语句都是如此。

```
SQL>CREATE TABLE test1 (c1 nchar);
--- SQL operation complete.
SQL>SHOWDDL test1;
CREATE TABLE TRAFODION.SEABASE.TEST1
(
C1 CHAR(1) CHARACTER SET UCS2 COLLATE DEFAULT
DEFAULT NULL NOT SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT
;
-- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON
TRAFODION.SEABASE.TEST1 TO
DB__ROOT WITH GRANT OPTION;
--- SQL operation complete.
SQL>ALTER TABLE test1 ALTER COLUMN c1 NCHAR CHARACTER
SET ISO88591;
*** ERROR[15001] A syntax error occurred at or before:
alter table test1 alter column c1 nchar character set
iso88591;
^ (47 characters from start of SQL
   statement) [2018-07-13 10:24:12]
```

- 该例子展示了如何更改 c1 NCHAR 列的数据类型。

```
SQL>CREATE TABLE test1 (c1 nchar);
--- SQL operation complete.
SQL>SHOWDDL test1;
CREATE TABLE TRAFODION.SEABASE.TEST1
(
C1 CHAR(1) CHARACTER SET UCS2 COLLATE DEFAULT
DEFAULT NULL NOT SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT
;
-- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON
TRAFODION.SEABASE.TEST1 TO
DB__ROOT WITH GRANT OPTION;
--- SQL operation complete.
SQL>ALTER TABLE test1 ALTER COLUMN c1 varchar(10)
CHARACTER SET ISO88591;
--- SQL operation complete.
SQL>SHOWDDL test1;
CREATE TABLE TRAFODION.SEABASE.TEST1
(
C1 VARCHAR(10) CHARACTER SET ISO88591 COLLATE
DEFAULT DEFAULT NULL NOT SERIALIZED /*altered_col*/
)
```

```
 ATTRIBUTES ALIGNED FORMAT
 ;
 -- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON
 TRAFODION.SEABASE.TEST1 TO
 DB__ROOT WITH GRANT OPTION;
 --- SQL operation complete.
```

- 下表 orders 有两个非唯一索引(*index_orders1* 和 *index_orders2*)和两个唯一索引(unique_index_orders1 和 *unique_index_orders2*)。

```
SQL>SHOWDDL orders;
CREATE TABLE TRAFODION.SEABASE.ORDERS
(
  ORDERNUM     NUMERIC(6, 0) DEFAULT NULL NOT SERIALIZED
, ORDER_DATE   DATE DEFAULT NULL NOT SERIALIZED
, DELIV_DATE   DATE DEFAULT NULL NOT SERIALIZED
, SALESREP     NUMERIC(4, 0) DEFAULT NULL NOT
SERIALIZED
, CUSTNUM      NUMERIC(4, 0) DEFAULT NULL NOT
SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT
;
CREATE INDEX INDEX_ORDERS1 ON TRAFODION.SEABASE.ORDERS
(
ORDER_DATE ASC
)
ATTRIBUTES ALIGNED FORMAT
;
CREATE INDEX INDEX_ORDERS2 ON TRAFODION.SEABASE.ORDERS
(
DELIV_DATE ASC
)
ATTRIBUTES ALIGNED FORMAT
;
CREATE UNIQUE INDEX UNIQUE_INDEX_ORDERS1 ON
TRAFODION.SEABASE.ORDERS
(
SALESREP ASC
)
ATTRIBUTES ALIGNED FORMAT
;
CREATE UNIQUE INDEX UNIQUE_INDEX_ORDERS2 ON
TRAFODION.SEABASE.ORDERS
(
CUSTNUM ASC
)
ATTRIBUTES ALIGNED FORMAT
;
-- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON
```

```
TRAFODION.SEABASE.ORDERS TO
DB__ROOT WITH GRANT OPTION;
--- SQL operation complete.
```

- 禁用一个唯一索引(*unique_index_orders1*)，并验证所有非唯一索引(*index_orders1 和 index_orders2*)和其余唯一索引(*unique_index_orders2*)仍然被启用。

```
SQL>ALTER TABLE ORDERS DISABLE INDEX
unique_index_orders1;
--- SQL operation complete.
SQL>SHOWDDL orders;
CREATE TABLE TRAFODION.SEABASE.ORDERS
(
  ORDERNUM     NUMERIC(6, 0) DEFAULT NULL NOT
SERIALIZED
, ORDER_DATE   DATE DEFAULT NULL NOT SERIALIZED
, DELIV_DATE   DATE DEFAULT NULL NOT SERIALIZED
, SALESREP     NUMERIC(4, 0) DEFAULT NULL NOT
SERIALIZED
, CUSTNUM      NUMERIC(4, 0) DEFAULT NULL NOT
SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT
;
CREATE INDEX INDEX_ORDERS1 ON TRAFODION.SEABASE.ORDERS
(
ORDER_DATE ASC
)
ATTRIBUTES ALIGNED FORMAT
;
CREATE INDEX INDEX_ORDERS2 ON TRAFODION.SEABASE.ORDERS
(
DELIV_DATE ASC
)
ATTRIBUTES ALIGNED FORMAT
;
CREATE UNIQUE INDEX UNIQUE_INDEX_ORDERS2 ON
TRAFODION.SEABASE.ORDERS
(
CUSTNUM ASC
)
ATTRIBUTES ALIGNED FORMAT
;
-- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON
TRAFODION.SEABASE.ORDERS TO
DB__ROOT WITH GRANT OPTION;
--- SQL operation complete.
```

- 禁用一个非唯一索引(*index_orders1*)，并验证其余非唯一索引(*index_orders2*)

和其余唯一索引(*unique_index_orders2*)仍然被启用。

```
SQL>ALTER TABLE ORDERS DISABLE INDEX index_orders1;
--- SQL operation complete.
SQL>SHOWDDL orders;
CREATE TABLE TRAFODION.SEABASE.ORDERS
(
  ORDERNUM    NUMERIC(6, 0) DEFAULT NULL NOT SERIALIZED
, ORDER_DATE  DATE DEFAULT NULL NOT SERIALIZED
, DELIV_DATE  DATE DEFAULT NULL NOT SERIALIZED
, SALESREP    NUMERIC(4, 0) DEFAULT NULL NOT SERIALIZED
, CUSTNUM     NUMERIC(4, 0) DEFAULT NULL NOT SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT
;
CREATE INDEX INDEX_ORDERS2 ON TRAFODION.SEABASE.ORDERS
(
DELIV_DATE ASC
)
ATTRIBUTES ALIGNED FORMAT
;
CREATE UNIQUE INDEX UNIQUE_INDEX_ORDERS2 ON
TRAFODION.SEABASE.ORDERS
(
CUSTNUM ASC
)
ATTRIBUTES ALIGNED FORMAT
;
-- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON
TRAFODION.SEABASE.ORDERS TO
DB__ROOT WITH GRANT OPTION;
--- SQL operation complete.
```

- 禁用所有索引，并验证所有索引(包括唯一索引和唯一索引)都已被禁用。

```
SQL>ALTER TABLE ORDERS DISABLE ALL INDEXES;
--- SQL operation complete.
SQL>SHOWDDL orders;
CREATE TABLE TRAFODION.SEABASE.ORDERS
(
ORDERNUM NUMERIC(6, 0) DEFAULT NULL NOT SERIALIZED
, ORDER_DATE DATE DEFAULT NULL NOT SERIALIZED
, DELIV_DATE DATE DEFAULT NULL NOT SERIALIZED
, SALESREP NUMERIC(4, 0) DEFAULT NULL NOT SERIALIZED
, CUSTNUM NUMERIC(4, 0) DEFAULT NULL NOT SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT
;
-- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON
TRAFODION.SEABASE.ORDERS TO
```

```
DB__ROOT WITH GRANT OPTION;
--- SQL operation complete.
```

- 启用所有索引并验证结果。

```
SQL>ALTER TABLE orders ENABLE ALL INDEXES;
--- SQL operation complete.
SQL>SHOWDDL orders;
CREATE TABLE TRAFODION.SEABASE.ORDERS
(
ORDERNUM NUMERIC(6, 0) DEFAULT NULL NOT SERIALIZED
, ORDER_DATE DATE DEFAULT NULL NOT SERIALIZED
, DELIV_DATE DATE DEFAULT NULL NOT SERIALIZED
, SALESREP NUMERIC(4, 0) DEFAULT NULL NOT SERIALIZED
, CUSTNUM NUMERIC(4, 0) DEFAULT NULL NOT SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT
;
CREATE INDEX INDEX_ORDERS1 ON TRAFODION.SEABASE.ORDERS
(
ORDER_DATE ASC
)
ATTRIBUTES ALIGNED FORMAT
;
CREATE INDEX INDEX_ORDERS2 ON TRAFODION.SEABASE.ORDERS
(
DELIV_DATE ASC
)
ATTRIBUTES ALIGNED FORMAT
;
CREATE UNIQUE INDEX UNIQUE_INDEX_ORDERS1 ON
TRAFODION.SEABASE.ORDERS
(
SALESREP ASC
)
ATTRIBUTES ALIGNED FORMAT
;
CREATE UNIQUE INDEX UNIQUE_INDEX_ORDERS2 ON
TRAFODION.SEABASE.ORDERS
(
CUSTNUM ASC
)
ATTRIBUTES ALIGNED FORMAT
;
-- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON
TRAFODION.SEABASE.ORDERS TO
DB__ROOT WITH GRANT OPTION;
--- SQL operation complete.
```

- 该列子给已存在的表赋予 stored desc 属性，同时生成 descriptor

```
SQL>create table table2(a int);

--- SQL operation complete.

SQL>alter table table2 generate stored desc;

--- SQL operation complete.


SQL>showddl table2;

CREATE TABLE TRAFODION.SEABASE.TABLE2
  (
    A      INT DEFAULT NULL NOT SERIALIZED
  )
 ATTRIBUTES ALIGNED FORMAT NAMESPACE 'TRAF_RSRVD_3'
STORED DESC
;

-- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES,
ALTER, DROP ON TRAFODION.SEABASE.TABLE2 TO DB__ROOT
WITH GRANT OPTION;

--- SQL operation complete.
```

- 该例子给已存在的表赋予 Incremental Backup 属性。

```
SQL>alter table table1 attribute incremental backup;

--- SQL operation complete.
```

2.5 ALTER USER 语句

该语句更改数据库里注册用户的关联属性，它是 EsgynDB SQL 扩展语句。

```
ALTER USER database-username alter-action[, alter-action]
alter-action is :
SET EXTERNAL NAME directory-service-username
| SET { ONLINE | OFFLINE }
```

2.5.1 ALTER USER 语法

- *database-username*
  这是当前注册的数据库用户名。

- SET EXTERNAL NAME
  更改目录服务里的用户标识名，这也是用户连接数据库时所指定的名称。
  - *directory-service-username*
    指定目录服务里的新用户名。
    它是一种不区分大小写的常规或分隔标识符。
    更多信息，请参阅不区分大小写字母的分隔标识符。

- SET { ONLINE | OFFLINE }
  更改控制用户连接至数据库的属性。
  - ONLINE
    允许一个用户连接到数据库。
  - OFFLINE
    禁止一个用户连接到数据库。

2.5.2 ALTER USER 注意事项

只有具有用户管理权限的用户（已被授予 MANAGE_USERS 组件权限的用户）能执行以下操作：

- 为任何用户设置 EXTERNAL NAME
- 为任何用户设置 ONLINE | OFFLINE 属性

最初只有 DB__ROOT（数据库根用户）被授予 MANAGE_USERS 组件权限。

2.5.3 ALTER USER 例子

- 更改一个用户的外部名称：
  ```
  ALTER USER ajones SET EXTERNAL NAME
  "Americas\ArturoJones";
  ```

- 更改一个用户的属性以使其连接到数据库：
  ```
  ALTER USER ajones SET ONLINE;
  ```

2.6 BEGIN WORK 语句

该语句启动显示事务。这里的事务包括 SQL 语句序列定义的操作集，它们在输入 BEGIN WORK 语句后立即开始执行,并在输入下一条 COMMIT 或 ROLLBACK 语句后结束。更多信息，请参阅事务管理。

如果一个事务目前仍处于活动状态，那么输入 BEGIN WORK 语句后会抛出错误。

BEGIN WORK 是 EsgynDB SQL 扩展语句。

2.6.1 BEGIN WORK 注意事项

以 BEGIN WORK 语句启动一个事务，以 COMMIT WORK 或 ROLLBACK WORK 语句结束一个事务。

2.6.2 BEGIN WORK 例子

将三条独立的语句（两条 INSERT 语句和一条 UPDATE 语句）作为一组，即可在一个事务里更新数据库:

- 该语句启动一个事务。

```
BEGIN WORK;
```

--- SQL operation complete.

```
INSERT INTO sales.orders VALUES (125, DATE '2008-03-23',
DATE '2008-03-30', 75, 7654);
```

--- 1 row(s) inserted.

```
INSERT INTO sales.odetail VALUES (125, 4102, 25000, 2);
```

--- 1 row(s) inserted.

```
UPDATE invent.partloc SET qty_on_hand = qty_on_hand - 2
     WHERE partnum = 4102 AND loc_code = 'G45';
```

--- 1 row(s) updated.

- 该语句结束一个事务。

```
COMMIT WORK;
```

--- SQL operation complete.

2.7 CALL 语句

该语句调用 EsgynDB SQL 数据库里的 Java 存储过程（SPJ）。

```
CALL procedure-ref ([argument-list])
procedure-ref is :
[[catalog-name.]schema-name.]procedure-name
argument-list is :
SQL-expression[, SQL-expression]...
```

2.7.1 CALL 语法

- *procedure-ref*
  指定了一个表单 的 ANSI 逻辑名称：
- *[[catalog-name.]schema-name.]procedure-name*
  该名称的每部分都是一个有效的 SQL 标识符，最长 128 个字符。更多信息，请参阅标识符和数据库对象名称。
  如果没有完全限定一个存储过程名称，EsgynDB SQL 将根据当前会话的 schema 限定它。

- *argument-list*
  接收 IN、INOUT 或 OUT 形参所用的实参。这些实参都由 SQL 表达式组成，包括以逗号分隔的动态参数。

- *SQL-expression[{, SQL-expression}...]*
  每个表达式都必须按照下列数据类型之一进行求值：
  - Character 数值
  - Date-time 数值
  - Numeric 数值

  SPJ 里不得出现 Interval 数值表达式。更多信息，请参阅输入参数和输出参数。不得在参数列表里指定结果集。

2.7.2 CALL 注意事项

2.7.2.1 CALL 使用限制

您可以在应用程序或例如 TrafCI 的命令行界面中使用 CALL 语句作为独立的 SQL 语句。您不能在一条复合语句内部或行集里使用 CALL 语句。

2.7.2.2 使用 CALL 的必要权限

发出一条 CALL 语句，必须满足以下条件之一：

- 您是 DB_ROOT（数据库根用户）。
- 您是该存储过程的所有者。
- 您已经通过用户名或授予的角色获得了 EXECUTE（或 ALL）权限。更多信息，请参阅 GRANT 语句。

存储过程由当前 EsgynDB ID 执行。

2.7.2.3 输入参数

通过 IN 或 INOUT 参数将数据传递给 SPJ。IN 参数使用以下 SQL 表达式之一：

- Literal
- SQL 函数（包括 CASE 和 CAST 表达式）
- 算术或串联运算
- 标量子查询
- 一个应用程序里的动态参数
- TrafCI 里的命名参数或未命名参数

您仅可使用动态参数、命名参数或未命名参数作为 INOUT 参数。更多信息，请参阅表达式。

2.7.2.4 输出参数

SPJ 将数值返回到 OUT 和 INOUT 参数里。

输出参数在应用程序里必须是动态参数，在 HPDCI 里则必须是命名参数或未命名参数。每个调用应用程序都必须在其环境里定义 OUT 和 INOUT 参数的语义。

2.7.2.5 参数的数据转换

当参数数据类型与存储过程的正式数据类型兼容但并不匹配时，EsgynDB SQL 将执行隐式数据转换。

- 对于存储过程的输入值而言，实际实参值将被转换为正式参数类型。
- 对于存储过程的输出值而言，具有正式参数数据类型的实际输出值将被转换为已声明的动态参数类型。

2.7.2.6 空值输入和输出

如果该参数对应的 Java 数据类型支持空值，那么您既可以将空值作为输入传给 SPJ，也可以将空值从 SPJ 传出。

如果空值作为输入或输出用于一个并不支持空值的参数，则 EsgynDB SQL 将会返回一个错误。

2.7.2.7 事务语义

如果目前不存在处于活动状态的事务，那么 CALL 语句将会自动发起一个事务。然而，CALL 语句执行失败并不总会自动中止事务。

2.7.3 CALL 例子

- 在 TrafCI 中，执行一个名为 MONTHLYORDERS 的 SPJ，其中字面值为 IN 参数，未命名的参数为 OUT 参数:

```
CALL sales.monthlyorders(3,?);
```

- 该 CALL 语句执行一个存储过程，它接收一个 IN 参数（日期字面值）、返回一个 OUT 参数（NUM_ORDERS 列里的一行）和两个结果集:

```
CALL sales.ordersummary('01/01/2001', ?);

NUM_ORDERS
-------------------
13

ORDERNUM      NUM_PARTS  AMOUNT     ORDER_DATE LAST_NAME
--------      ---------  ---------  ---------- ------------

100210        4          19020.00   2006-04-10 HUGHES
100250        4          22625.00   2006-01-23 HUGHES
101220        4          45525.00   2006-07-21 SCHNABL
200300        3          52000.00   2006-02-06 SCHAEFFER
200320        4           9195.00   2006-02-17 KARAJAN
200490        2           1065.00   2006-03-19 WEIGL

.
.
.
--- 13 row(s) selected.
ORDERNUM      PARTNUM    UNIT_PRICE QTY_ORDERED     PARTDESC
--------      --------   ---------- ---------- -------------
---

100210        2001       1100.00          3    GRAPHIC
PRINTER,M1
100210        2403        620.00          6    DAISY
PRINTER,T2
100210         244       3500.00          3    PC GOLD, 30 MB
100210        5100        150.00         10    MONITOR    BW,
TYPE 1
100250        6500         95.00         10    DISK
CONTROLLER
100250        6301        245.00         15    GRAPHIC  CARD,
HR
```

```
    .
    .
    .
--- 70 row(s) selected.
--- SQL operation complete.
```

2.8 COMMIT WORK 语句

该语句在当前事务执行期间提交任何对象更改或结束一个事务。 更多详情，请参阅事务管理。

```
COMMIT [WORK]
```

WORK 是一个没有效果的可选关键字。

活动状态事务之外发布 COMMIT WORK 语句时将生成错误 8605。

2.8.1 COMMIT WORK 注意事项

BEGIN WORK 语句启动一个事务，COMMIT WORK 或 ROLLBACK WORK 结束一个事务。

2.8.2 COMMIT WORK 例子

假定您的应用程序会向库存清单里添加信息。您已经从新供应商处收到了 24 个终端，希望添加该供应商并更新库存量。这些终端的部件号为 5100，分配给供应商的编号为 17。每个终端的成本费用为 800 美元。

该事务必须将终端订单添加到 PARTSUP，将供应商添加到 SUPPLIER 表，并更新 PARTLOC 里的 QTY_ON_HAND。成功执行 INSERT 和 UPDATE 语句后，您可以提交该事务，如下所示：

```
-- This statement initiates a transaction.
BEGIN WORK;
--- SQL operation complete.

-- This statement inserts a new entry into PARTSUPP. INSERT
INTO invent.partsupp
VALUES (5100, 17, 800.00, 24);
--- 1 row(s) inserted.

-- This statement inserts a new entry into SUPPLIER. INSERT
INTO invent.supplier
VALUES (17, 'Super Peripherals','751 Sanborn Way', 'Santa
Rosa', 'California', '95405');
--- 1 row(s) inserted.

-- This statement updates the quantity in PARTLOC. UPDATE
invent.partloc
SET qty_on_hand = qty_on_hand + 24
WHERE partnum = 5100 AND loc_code = 'G43';
--- 1 row(s) updated.
```

```
-- This statement ends a transaction.
COMMIT WORK;
--- SQL operation complete
```

2.9 CONTROL QUERY CANCEL 语句

该语句取消一条标有查询 ID 且正在执行中的查询。

您可以在基于客户端的工具（例如 TrafCI）中或通过任何 ODBC 或 JDBC 应用程序执行 CONTROL QUERY CANCEL 语句。

此语句是 EsgynDB SQL 扩展语句。

```
CONTROL QUERY CANCEL QID query-id [COMMENT 'comment-text']
```

2.9.1 CONTROL QUERY CANCEL 语法

- *query-id*
  指定一个正在执行的查询的 ID，此为 SQL 编译器生成的唯一标识符。

- *'comment-text'*
  指定在取消查询的错误消息上显示的可选注释。

2.9.2 CONTROL QUERY CANCEL 注意事项

2.9.2.1 CONTROL QUERY CANCEL 优势

该语句在不停止主执行进程（MXOSRVR）的情况下终止多个查询。

与标准 ODBC/JDBC 取消方法相比，这种取消类型具有以下优势：

- ANSI 定义的错误消息将被返回给客户端会话，SQLSTATE 将被设置为 HY008。
- 取消查询后，重要的缓存对象将被持久化，包括主执行进程及其编译程序、缓存在主服务器的编译语句、编译程序的查询缓存、以及其缓存的元数据和直方图。
- 客户端不需要重新建立连接，它的预处理语句也被保存。
- 即使多个客户端使用一台中间层应用服务器共享连接，取消一个客户端正在执行的查询不会影响共享相同连接的其他客户端。

2.9.2.2 CONTROL QUERY CANCEL 使用限制

一些正在执行的查询可能无法在 60 秒间隔内响应 CONTROL QUERY CANCEL 语句。对于这些查询，EsgynDB SQL 将停止它们的 ESP 进程。如果这个动作可以取消查询，您将会看到上述所有优点。

如果发出 CONTROL QUERY CANCEL 语句后 120 秒内都没有终止一个正在执行的查询，EsgynDB SQL 将会停止主执行进程、终止该查询、并产生连接丢失错误。这种情况下，您将无法看到上述任何优点。相反，您会丢失连接、需要

重新建立连接、并重新准备查询。这种情况经常出现在 CALL、DDL 和实用工具语句的使用场合中，很少出现在其他语句的使用场合里。

该语句不能与以下语句同时使用：

- 在单行和单个分区上运行的唯一性查询
- 处于非执行状态的查询，例如：一条正在编译中的查询
- CONTROL QUERY DEFAULT、BEGIN WORK、COMMIT WORK、ROLLBACK WORK 和 EXPLAIN 语句
- 静态编译的元数据查询
- 在异常情况下执行的查询，例如：没有运行时统计信息或没有 ID 的查询

2.9.2.3 使用 CONTROL QUERY CANCEL 的必要权限

为了发出一条 CONTROL QUERY CANCEL 语句，必须满足以下条件之一：

- 您是 DB_ROOT（数据库根用户）。
- 您拥有（发出了）该查询。
- 您拥有 SQL_OPERATIONS 组件的 QUERY_CANCEL 权限。

2.9.3 CONTROL QUERY CANCEL 例子

该语句取消一条指定的查询，并提供一条关于取消操作的注释：

```
control query cancel qid
MXID11000010941212288634364991407000000003806U3333300_156
016_S1
comment 'Query is consuming too many resources.';
```

在一个单独的会话中，发出查询的客户端将会看到一条关于该查询被取消的错误消息。

```
>>execute s1;

*** ERROR[8007] The operation has been canceled.Query is
consuming too
many resources.
```

2.10 CONTROL QUERY DEFAULT 语句

该语句更改当前进程的默认设置。您可以在基于客户端的工具（例如 TrafCI）中、或通过任何 ODBC 或 JDBC 应用程序执行该语句。

该语句是 EsgynDB SQL 扩展语句。

```
{ CONTROL QUERY DEFAULT | CQD } control-

default-option control-default-option is:

attribute {'attr-value' | RESET}
```

2.10.1 CONTROL QUERY DEFAULT 语法

- `attribute`
  这是一个代表属性名称的字符串。更多信息，请参阅附录 B 控制查询默认（CQD）属性。

- `attr-value`
  这是一个代表属性值的字符串。即使该值为数字，您也必须指定 attr-value 作为引用字符串。

- RESET
  指定在当前会话中使用该语句设置的属性，该属性将被重置为当前会话开始时生效的数值。

2.10.2 CONTROL QUERY DEFAULT 注意事项

2.10.2.1 CONTROL QUERY DEFAULT 作用域

该语句的执行结果将持续有效，直到当前进程终止或执行其他相同属性的语句覆盖该结果为止。CQD 将在编译时被应用，因此它们无法影响目前已经预处理的任何语句。例如：

```
PREPARE x FROM SELECT * FROM t;
CONTROL QUERY DEFAULT SCHEMA 'myschema';
EXECUTE x;                    -- 使用默认 schema SEABASE
SELECT * FROM t2;             -- 使用 MYSCHEMA;
PREPARE y FROM SELECT * FROM t3; CONTROL QUERY DEFAULT
SCHEMA 'seabase';
EXECUTE y; EMA;               -- 使用 MYSCHEMA;
```

2.10.3 CONTROL QUERY DEFAULT 例子

- 该例子将查询的最大并行度更改为 2。该值必须小于集群中 CPU 的数量。

```
SQL>CONTROL QUERY DEFAULT PARALLEL_NUM_ESPS '2';
--- SQL operation complete.
```

- 该例子将当前进程里的 PARALLEL_NUM_ESPS 属性重置为其系统值。在本例中，编译器计算要使用的 ESP 数量。

```
SQL>CONTROL QUERY DEFAULT PARALLEL_NUM_ESPS 'system';
--- SQL operation complete.
```

- 该例子允许优化器使用散列连接生成执行。

```
SQL>CONTROL QUERY DEFAULT HASH_JOIN 'ON';
--- SQL operation complete.
```

- 将当前进程里列名的最大支持长度改为 200:

```
CONTROL QUERY DEFAULT HBASE_MAX_COLUMN_NAME_LENGTH '200';
```

- 将当前进程里的 HBASE_MAX_COLUMN_NAME_LENGTH 属性重置为其初始值:

```
CONTROL QUERY DEFAULT HBASE_MAX_COLUMN_NAME_LENGTH RESET;
```

2.11 CREATE FUNCTION 语句

该语句将一个采用 C 语言编写的用户定义函数（UDF）注册为 EsgynDB 数据库里的函数。

目前，EsgynDB 支持创建标量 UDF，被调用时它会返回单个数值或一行数值。在 SELECT 语句的 SELECT list 或 WHERE 子句里，标量 UDF 被调用作为 SQL 表达式。

注意：目前的事务不支持 DDL 语句。这意味着您既无法在用户定义的事务（BEGIN WORK...COMMIT WORK）中运行该语句，也无法在 AUTOCOMMIT 选项关闭时运行该语句。为了运行该语句，必须开启该会话的 AUTOCOMMIT（自动提交）选项（默认）。

```
CREATE FUNCTION function-ref ([parameter-declaration[,
parameter-declaration]...])
     {RETURN | RETURNS} (return-parameter-declaration[,
     return-parameter-
     declaration]...)
     EXTERNAL NAME 'character-string-literal'
     LIBRARY[[catalog-name.]schema-name.]library-name
     [LANGUAGE C]
     [PARAMETER STYLE SQL]
     [NO SQL]
     [NOT DETERMINISTIC | DETERMINISTIC]
     [FINAL CALL | NO FINAL CALL]
     [NO STATE AREA | STATE AREA size]
     [NO PARALLELISM | ALLOW ANY PARALLELISM]

function-ref is:
[[catalog-name.]schema-name.]function-name

parameter-declaration is:
[IN] [sql-parameter-name]sql-datatype

return-parameter-declaration is:
[OUT] [sql-parameter-name] sql-datatype
```

2.11.1 CREATE FUNCTION 语法

- *function-ref ([parameter-declaration[,parameter-declaration]...])*

  指定函数的名称、以及任何与外部函数相对应的 SQL 参数。

- *function-ref*

  指定一个表单的 ANSI 逻辑名称:

  - *[[catalog-name.]schema-name.]function-name*

    该名称的每部分都是一个有效的 SQL 标识符，最长 128 个字符。更多信息，请参阅标识符和数据库对象名称。

    指定一个尚未被相同 schema 里的存储过程或函数所采用的唯一的名称。如果您没有完全限定一个函数名称，EsgynDB SQL 将根据当前会话的 schema 限定它。

- *parameter-declaration*

  指定一个与外部函数签名相对应的 SQL 参数。

  - *[IN] [sql-parameter-name] sql-datatype*

    IN

    指定将数据传递给函数的参数。

    sql-parameter-name

    指定该参数的 SQL 标识符。更多信息，请参阅标识符。

    sql-datatype

    指定一种与外部函数签名里的参数数据类型相对应的 SQL 数据类型。sql-datatype 是 EsgynDB 支持的一种 SQL 数据类型。更多信息，请参阅数据类型。

- {RETURN | RETURNS} *(return-parameter-declaration[,return-parameter-declaration]...)*

  指定该函数的输出类型。

  - *return-parameter-declaration*

    指定输出值所用的 SQL 参数:

    [OUT] *[sql-parameter-name]sql-datatype*

    OUT

    指定一个从该函数接收数据的参数。

    sql-parameter-name

    指定返回参数的 SQL 标识符。更多信息，请参阅标识符。

sql-datatype
指定返回参数的 SQL 数据类型。sql-datatype 是 EsgynDB 里支持的一种 SQL 数据类型。更多信息，请参阅数据类型。

- EXTERNAL NAME *'method-name'*
  指定一个外部函数方法的大小写敏感性名称。

- LIBRARY *[[catalog-name.]schema-name.]library-name*
  指定一个包含外部函数的库的 ANSI 逻辑名称。如果您没有完全限定一个库名，EsgynDB SQL 将根据当前会话的 schema 限定它。

- LANGUAGE C
  指定外部函数采用 C 语言编写。该子句是可选的。

- PARAMETER STYLE SQL
  指定传递给外部函数的参数运行时约定必须采用 SQL 语言编写。该子句是可选的。

- NO SQL
  指定该函数不能执行 SQL 操作。该子句是可选的。

- DETERMINISTIC | NOT DETERMINISTIC
  指定该函数对于给定的一组实参数值的 OUT 形参是否始终返回相同的数值（DETERMINISTIC，默认行为）或不返回相同的数值（NOT DETERMINISTIC）。
  如果该函数是确定性函数，则 EsgynDB SQL 无需每次都执行它，以产生结果；如果该函数不是确定性函数，则 EsgynDB SQL 会将结果缓存，并在后续执行期间重新使用它们，从而优化执行效果。

- FINAL CALL | NO FINAL CALL
  指定是否对函数进行最终调用。最终调用能够使该函数释放系统资源。默认值为 FINAL CALL。

- NO STATE AREA | STATE AREA size
  指定是否将状态区域分配给该函数。size 是一个整数，指的是以字节为单位的内存。可接受的数值范围是 0 到 16000。默认值为 NO STATE AREA。

- NO PARALLELISM | ALLOW ANY PARALLELISM

指定调用函数时是否采用并行方式。默认值为 ALLOW ANY PARALLELISM。

2.11.2 CREATE FUNCTION 注意事项

2.11.2.1 CREATE FUNCTION 必需的权限

发出一条 CREATE FUNCTION 语句，必须满足以下条件之一：

- 您是 DB_ROOT（数据库根用户）。
- 您正在一个共享的 schema 中创建函数，并且您对于创建函数期间使用的库拥有 USAGE（或 ALL）权限。USAGE 权限使您能够读取该库的底层 JAR 文件。
- 您是私有 schema 的所有者，并且您对于创建函数期间使用的库拥有 USAGE（或 ALL）权限。USAGE 权限使您能够读取该库的底层 JAR 文件。
- 您不但对于 SQL_OPERATIONS 组件拥有 CREATE 或 CREATE_ROUTINE 组件级权限，而且对于创建函数期间使用的库拥有 USAGE（或 ALL）权限。USAGE 权限使您能够读取该库的底层 JAR 文件。

注意：在这种情况下，如果您在私有 schema 中创建函数，您必须是 schema 的所有者。

2.11.3 CREATE FUNCTION 例子

- 该 CREATE FUNCTION 语句创建一个函数，将两个整数相加求和：

```
create function add2 (int, int)
   returns (total_value int)
   external name 'add2'
   library myudflib;
```

- 该 CREATE FUNCTION 语句创建一个函数，它返回五个输入整数的最小值、最大值和平均值：

```
create function mma5 (int, int, int, int, int)
   returns (min_value int, max_value int, avg_value int)
   external name 'mma5'
   library myudflib;
```

- 该 CREATE FUNCTION 语句创建一个函数，它最多能将 32 个字符的输入字符串反转：

```
create function reverse (varchar(32))
   returns (reversed_string varchar(32))
   external name 'reverse'
   library myudflib;
```

2.12 CREATE INDEX 语句

该语句基于一个表或表状对象创建一个 SQL 索引。CREATE VOLATILE INDEX 语句创建一个 SQL 索引，其只能存在于创建该索引的 SQL 会话期间。会话结束时，临时索引将被自动删除。更多信息，请参阅索引。

注意： 目前的事务不支持 DDL 语句。这意味着您既无法在用户定义的事务（BEGIN WORK...COMMIT WORK）中运行该语句，也无法在 AUTOCOMMIT 选项关闭时运行该语句。为了运行该语句，必须开启该会话的 AUTOCOMMIT（自动提交）选项（默认）。

CREATE INDEX 是 EsgynDB SQL 扩展语句。

```
CREATE [VOLATILE|TEMPORARY] INDEX index ON table
   (column-name [ASC[ENDING] | DESC[ENDING]]
   [,column-name [ASC[ENDING] | DESC[ENDING]]]...)
   [populate-option]
   [HBASE_OPTIONS (hbase-options-list)]
   [SALT LIKE TABLE]

populate-option is:
   POPULATE
| NO POPULATE

hbase-options-list is:
   hbase-option = 'value'[, hbase-option = 'value']...
```

2.12.1 CREATE INDEX 语法

- `VOLATILE|TEMPORARY`
  CREATE VOLATILE INDEX 语句创建一个 SQL 索引，其生命周期仅限于创建索引的 SQL 会话。临时索引在会话结束时自动删除。更多信息，请参阅索引。TEMPORARY 与 VOLATILE 等同。

- `UNIQUE`
  CREATE UNIQUE INDEX 语句在表上创建唯一索引，并强制索引字段的唯一性，即，它强制规定不允许任何重复的列值，并且必须在创建列上的唯一索引之前消除这些值。
  唯一索引不仅提高了查询性能，还提供了数据完整性检查。当插入或更新表的行时，将检查约束。
  注意：对于只能包含一个 NULL 值的列，多个 NULL 值则被视为重复值。

- `Index-name`
  这是一种 SQL 标识符，它指定新索引的简易名称。您不能用该索引所属

schema 的名称来限定它。一个 schema 内的索引都有各自的命名空间，因此索引名可能与表名或约束名相同。然而，一个 schema 内的索引名必须各不相同。

- table
  这是用于创建索引的表名。更多信息，请参阅数据库对象名称。

- *column-name* [ASC[ENDING] | DESC[ENDING]] [,*column-name* [ASC[ENDING] | DESC[ENDING]]]...
  指定索引里包含的表列。索引里的列序不必对应于表里的列序。
  ASCENDING 或 DESCENDING 指定索引里的行的存储和检索顺序。默认值为 ASCENDING。
  这些索引行将根据指定的索引首列数值进行排序。如果多个索引行共享相同的首列数值，则第二列将被用来对这些索引行进行排序，依此类推。
  如果一个非唯一索引里出现了重复的索引行，则它们的顺序将基于底层表键列的指定序列。为了进行排序（并非其他目的），空值应大于其他值。

- populate-option
  - NO POPULATE

  指定在创建索引时不填充该索引。创建了索引的分区，但没有向索引写入数据，并将其标记为"脱机"。您可以使用 DROP INDEX 语句删除脱机索引。DROP TABLE 语句还会删除指定表的脱机索引。DML 语句对脱机索引没有影响。
  如果创建索引的目的是将其用于约束，则必须在创建约束之前填充该索引。您可以使用 POPULATE INDEX 语句填充脱机索引并删除其脱机名称。

  - POPULATE

  指定要创建和填充的索引。

  默认为 POPULATE。

- HBASE_OPTIONS (hbase-option= 'value'[,hbase-option = 'value']...)
  为该索引设置的 HBase 选项列表这些选项的应用与索引基表设置的任何 HBase 选项都无关。

- *hbase-option = 'value'*

  这是此类 HBase 选项之一，其分配数值如下：

HBase 选项	**接受值**	**默认值**
BLOCKCACHE	'true' | 'false'	'true'
BLOCKSIZE	'1024' - '64*1024*1024'	'64*1024'
BLOOMFILTER	'NONE' | 'ROW' | 'ROWCOL'	'ROW'
CACHE_BLOOMS_ON_WRITE	'true' | 'false'	'false'
CACHE_DATA_ON_WRITE	'true' | 'false'	'false'
CACHE_INDEXES_ON_WRITE	'true' | 'false'	'false'
COMPACT	'true' | 'false'	'true'
COMPACT_COMPRESSION	'GZ' | 'LZ4' | 'LZO' | 'NONE' | 'SNAPPY'	'NONE'
COMPRESSION	'GZ' | 'LZ4' | 'LZO' | 'NONE' | 'SNAPPY'	'NONE'
DATA_BLOCK_ENCODING	'DIFF' | 'FAST_DIFF' | 'NONE' | 'PREFIX'	'NONE'
DURABILITY	'USE_DEFAULT' | 'SKIP_WAL' | 'ASYNC_WAL' | 'SYNC_WAL' | 'FSYNC_WAL'	'SYNC_WAL'
EVICT_BLOCKS_ON_CLOSE	'true' | 'false'	'false'
IN_MEMORY	'true' | 'false'	'false'
KEEP_DELETED_CELLS	'true' | 'false'	'false'
MAX_FILESIZE	'2*1024*1024' - '2^{63}-1'	'10*1024*1024*1024'
MAX_VERSIONS	'1' | *'positive-integer'*	'1'
MEMSTORE_FLUSH_SIZE	'1024*1024' - '2^{63}-1'	'128*1024*1024'
MIN_VERSIONS	'0' | *'positive-integer'*	'0'
PREFIX_LENGTH_KEY	*'positive-integer'*，应小于表键的最大长度，只有当 SPLIT_POLICY 为 KeyPrefixRegionSplitPolicy 时才适用。	'2'
REPLICATION_SCOPE	'0' | '1'	'1'
SPLIT_POLICY	'org.apache.hadoop.hbase.regionserver.ConstantSizeRegionSplitPolicy'|'org.apache.hadoop.hbase.regionserver.IncreasingToUpperBoundRegion	'org.apache.hadoop.hbase.regionserver.IncreasingToUpperBoundRegionSplit

	SplitPolicy'| 'org.apache.hadoop.hbase.regionserver.KeyPrefixRegionSplitPolicy'	Policy'
TTL	'-1' (forever) | *'positive-integer'*	'2147483647' (forever)

- SALT LIKE TABLE
  这将导致索引使用相同的“加盐”方案（即 SALT USING num PARTITIONS [ON *(column[, column]*...)]）作为其基表。

2.12.2 CREATE INDEX 注意事项

2.12.2.1 创建索引步骤

在一个事务下创建索引，步骤如下：

1) 事务（用户启动的事务或系统启动的事务）开始执行。
2) 行被写入元数据中。
3) 创建物理标签保存索引（未经审计）。
4) 锁定基表，以便进行共享读访问，从而防止其他人对基表进行插入、更新和删除操作。
5) 使用侧插树读取基表，以实现未提交的读访问，由此可以装载索引。

注意：由于这里的分区是未经审查的空分区，侧插树可以快速地装载执行专门优化的数据。

---

6) 索引装载完后，索引审查属性将被打开、并被连接至基表（使索引上线）。
7) 该事务既可以由系统提交、也可以由请求者稍后提交。

即使该事务由用户启动，如果执行基本语义检查后操作仍失败，则索引将不复存在、并且整个事务都将被回滚。

如果要在已填充的大型 EsgynDB 表上创建索引，则应使用 NO POPULATE 选项，然后运行 POPULATE INDEX 实用工具来加载索引。因为 CREATE INDEX 只能在单个事务中执行，所以如果要移动大量数据，它可能会遇到事务限制，比如事务超时。

2.12.2.2 授权和可用性要求

一个索引和被其编入索引的表具有相同的安全性。

CREATE INDEX 防止对一个正在编制索引的表执行 INSERT、DELETE 和

UPDATE 操作。

如果其他进程在操作开始时已经锁定了表里的行，则 CREATE INDEX 语句将会继续等待、直到它的锁请求得到授权或出现超时为止。

您无法直接访问一个索引。

2.12.2.3 必需的权限

发出一条 CREATE INDEX 语句，必须满足以下条件之一：

- 您是 DB_ROOT（数据库根用户）。
- 您正在一个共享 schema 里创建表。
- 您是私有 schema 的所有者。
- 您是该表的所有者。
- 您拥有 SQL_OPERATIONS 组件的 ALTER、ALTER_TABLE、CREATE 或 CREATE_INDEX 权限。

注意：在这种情况下，如果您在私有 schema 中创建函数，您必须是 schema 的所有者。

2.12.2.4 索引的限制

非唯一索引里的列的长度总和加上底层表的聚集键的长度总和不得超过 2048 个字节。

每个表的索引数量没有限制。

2.12.3 CREATE INDEX 例子

- 该例子在表的两列上创建一个索引：

```
CREATE INDEX xempname
   ON persnl.employee (last_name, first_name);
```

- 该例子显示只能在临时表上创建临时索引。

```
SQL>create volatile table test2(c1 int);
--- SQL operation complete.
SQL>create volatile index idx_test2 on test2(c1);
--- SQL operation complete.
SQL>create table test1(c1 int);
--- SQL operation complete.
SQL>create volatile index idx_test1 on test1 (c1);
*** ERROR[4082] Object TEST1 does not exist or is
inaccessible. [2017-01-13
11:35:26]
```

- 该例子显示只能在具有一个 NULL 值的表上创建唯一索引。

```
SQL>select * from t25;
A B
----------- ----------
1 NULL
2 aaa
--- 2 row(s) selected.
SQL>create unique index idx_t25 on t25(b);
--- SQL operation complete.
SQL>showddl t25;
CREATE TABLE TRAFODION.SEABASE.T25
(
A INT DEFAULT NULL NOT SERIALIZED,
B CHAR(10) CHARACTER SET ISO88591 COLLATE DEFAULT
DEFAULT NULL NOT SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT;
CREATE UNIQUE INDEX IDX_T25 ON TRAFODION.SEABASE.T25
(
B ASC
)
ATTRIBUTES ALIGNED FORMAT;
--- SQL operation complete.
```

- 该例子显示不能在具有多个 NULL 值的表上创建唯一索引。

SQL 参考手册

```
SQL>select * from t25;
A B
----------- ----------
1 NULL
2 aaa
--- 2 row(s) selected.
SQL>create unique index idx_t25 on t25(b);
*** ERROR[8110] Duplicate rows detected. [2017-01-12
17:17:51]
*** ERROR[1053] Unique index TRAFODION.SEABASE.IDX_T25
could not be created because
the specified column(s) contain duplicate data. [2017-
01-12 17:17:51]
```

- 该例子创建了索引 *index_test*，该索引不会被填充到表 *test* 的 *c1* 列上。

```
SQL>create index index_test on test (c1) no populate;
--- SQL operation complete.
```

2.13 CREATE LIBRARY 语句

CREATE LIBRARY 在 EsgynDB 数据库里注册一个库对象。一个库对象可以是 SPJ 的 JAR 文件、也可以是 UDF 的库文件。

CREATE LIBRARY 是 EsgynDB SQL 扩展语句。

注意： 目前的事务不支持 DDL 语句。这意味着您既无法在用户定义的事务（BEGIN WORK...COMMIT WORK）中运行该语句，也无法在 AUTOCOMMIT 选项关闭时运行该语句。为了运行该语句，必须开启该会话的 AUTOCOMMIT（自动提交）选项（默认）。

```
CREATE LIBRARY [[catalog-name.]schema-name.]library-name
FILE 'library-filename'
[HOST NAME 'host-name']
[LOCAL FILE 'host-filename']
```

2.13.1 CREATE LIBRARY 语法

- `[[catalog-name.]schema-name.]library-name`
  指定库对象的 ANSI 逻辑名称，其中每部分都是一个有效的 SQL 标识符，最长 128 个字符。指定一个尚未被相同 schema 里的存储过程或函数所采用的唯一的名称。
  如果您没有完全限定一个库名，EsgynDB SQL 将根据当前会话的 schema 限定它。更多信息，请参阅标识符和数据库对象名称。

- FILE *'library-filename'*
  指定已部署的库文件（SPJ 的 JAR 文件或 UDF 的库文件）的完整路径。

注意： 确保将库文件上载到 EsgynDB 集群，然后将库文件复制到集群中所有节点上的同一目录，然后再运行 CREATE LIBRARY 语句。否则，您将看到一条错误消息，指出未找到 JAR 或 DLL 文件。

- HOST NAME *'host-name'*
  指定已部署文件所驻留的客户端主机的名称。

- LOCAL FILE *'host-filename'*
  指定已部署文件在客户端主机上的存储路径。

2.13.2 CREATE LIBRARY 注意事项

- 一个库对象不能引用一个其他库对象已经引用的库文件。如果其他库对象正

在使用 library-filename，则 CREATE LIBRARY 命令将运行失败。

- library-filename 必须指定一个现有的文件。否则 CREATE LIBRARY 命令将运行失败。

- CREATE LIBRARY 命令无法验证一个指定的 library-filename 是否为有效的可执行文件。

- HOST NAME 和 LOCAL FILE 都与位置相关。

2.13.2.1 必需的权限

发出一条 CREATE LIBRARY 语句，必须满足以下条件之一：

- 您是 DB_ROOT（数据库根用户）。
- 您正在共享的 schema 里创建库，并且拥有 MANAGE_LIBRARY 权限。
- 您是私有 schema 的所有者，并且拥有 MANAGE_LIBRARY 权限。
- 您拥有 SQL_OPERATIONS 组件的 CREATE 或 CREATE_LIBRARY 权限，并且拥有 MANAGE_LIBRARY 权限。

注意：在这种情况下，如果您在私有 schema 中创建函数，您必须是该 schema 的所有者。

2.13.3 CREATE LIBRARY 例子

- 该 CREATE LIBRARY 语句为 JAR 文件（SPJ）在 SALES schema 里注册一个名为 SALESLIB 的库。

```
CREATE LIBRARY sales.saleslib
 FILE '/opt/home/trafodion/spjjars/Sales.jar';
```

- 该 CREATE LIBRARY 语句在默认的库文件（UDF）schema 里注册一个名为 MYUDFS 的库。

```
CREATE LIBRARY myudfs FILE $UDFLIB;
```

2.14 CREATE PROCEDURE 语句

CREATE PROCEDURE 语句可以在 EsgynDB 数据库里注册一个 Java 方法作为 Java 存储过程（SPJ）。

注意： 目前的事务不支持 DDL 语句。这意味着您既无法在用户定义的事务（BEGIN WORK...COMMIT WORK）中运行该语句，也无法在 AUTOCOMMIT 选项关闭时运行该语句。为了运行该语句，必须开启该会话的 AUTOCOMMIT（自动提交）选项（默认）。

```
CREATE PROCEDURE procedure-ref([sql-parameter-list])
     EXTERNAL NAME 'java-method-name [java-signature]'
     LIBRARY [[catalog-name.]schema-name.]library-name [
     EXTERNAL SECURITY external-security-type]
     LANGUAGE JAVA
     PARAMETER STYLE JAVA
     [NO SQL | CONTAINS SQL | MODIFIES SQL DATA | READS SQL
     DATA]
     [DYNAMIC RESULT SETS integer]
     [TRANSACTION REQUIRED | NO TRANSACTION REQUIRED]
     [DETERMINISTIC | NOT DETERMINISTIC]
     [NO ISOLATE | ISOLATE]

procedure-ref is:
     [[catalog-name.]schema-name.]procedure-name

sql-parameter-list is:
     sql-parameter[, sql-parameter]...

sql-parameter is:
     [parameter-mode] [sql-identifier] sql-datatype

parameter-mode is:
     IN
     | OUT
     | INOUT

java-method-name is:
     [package-name.]class-name.method-name

java-signature is:
     ([java-parameter-list])

java-parameter-list is:
     java-datatype[, java-datatype]...
```

```
external-security-type is:
      DEFINER
      | INVOKER
```

注意： 本语法图里的分隔变量须区分大小写。大小写敏感性变量包括 *java-method-name* 、*java-signature*、*class-file-path* 和 procedure-ref 的任何分隔部分。剩下的语法则不区分大小写。

2.14.1 CREATE PROCEDURE 语法

- `procedure-ref([sql-parameter[,sql-parameter]...])`
  指定 Java 存储过程（SPJ）的名称和任何与 SPJ 方法签名相对应的 SQL 参数。

  - *procedure-ref*
    指定一个表单的 ANSI 逻辑名称：
    [[catalog-name.]schema-name.]procedure-name
    其中的每部分都是一个有效的 SQL 标识符，最长 128 个字符。 更多信息，请参阅标识符和数据库对象名称。
    指定一个尚未被相同 schema 里的存储过程或函数所采用的唯一的名称。EsgynDB SQL 不支持存储过程名称的重载，即您无法在底层 SPJ 方法里多次注册同一个存储过程名称。
    如果您没有完全限定一个存储过程名称，EsgynDB SQL 将根据当前会话的 schema 限定它。

  - *sql-parameter*
    指定一个与 SPJ 方法签名相对应的 SQL 参数：
    [parameter-mode] [sql-identifier] sql-datatype

  - *parameter-mode*
    指定一个参数的模式：IN、OUT 或 INOUT。 默认模式为 IN。
    IN
    指定一个将数据传递给 SPJ 的参数。
    OUT
    指定一个从 SPJ 接收数据的参数。该参数必须是一个数组。
    INOUT
    指定一个将数据传递给 SPJ、并从 SPJ 接收数据的参数。该参数必须是一个数组。

- *sql-identifier*

  指定该参数的 SQL 标识符。更多信息，请参阅标识符。

- *sql-datatype*

  指定一个与 SPJ 方法的 Java 参数相对应的 SQL 数据类型。更多信息，请参阅数据类型。

  sql-datatype 可以是：

SQL 数据类型	**映射到 Java 数据类型...**
CHAR[ACTER] CHAR[ACTER] VARYING VARCHAR VARCHAR2 PIC[TURE] X[1] NCHAR NCHAR VARYING NATIONAL CHAR[ACTER] NATIONAL CHAR[ACTER] VARYING	java.lang.String
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
dec[2] pic[ture] s9[3] numeric (包括精度大于 18 的 NUMERIC)[2]	java.math.BigDecimal
smallint[2]	short
int[2]	int 或 java.lang.Integer[4]
largeint[2]	long 或 java.lang.Long[4]
NUMERIC (包括精度大于 18 的 NUMERIC)[2] DEC[IMAL] PIC[TURE] S9[3]	java.math.BigDecimal
SMALLINT	short
INT[EGER]	int 或 java.lang.Integer[4]
LARGEINT	long 或 java.lang.Long
FLOAT	double 或 java.lang.Double[4]
REAL	float 或 java.lang.Float[4]
DOUBLE PRECISION （双精度型）	double 或 java.lang.Double[4]

更多信息，请参阅数据类型。

---

[1] Trafodion 数据库将 PIC X 存储为 CHAR 数据类型。

[2] SQL 参数的 Numeric 数据类型都必须带有签名，这是 Trafodion 数据库的默认配置。

[3] Trafodion 数据库存储 PIC S9 作为 DECIMAL 或 NUMERIC 数据类型。

[4] 默认情况下，SQL 数据类型将被映射到 Java 原始类型。只有当您指定 EXTERNAL NAME 子句的 Java 签名里的包装类时，SQL 数据类型才会被映射到 Java 包装类。

- EXTERNAL NAME'*java-method-name [java-signature]*'
  - *java-method-name*
    指定一个表单的 SPJ 方法的大小写敏感性名称:
    [package-name.]class-name.method-name
    Java 方法必须存在于数据库注册库内的一个 Java 类文件（class-name.class）里。Java 方法必须被定义为公共（public）静态（static）方法，且返回类型为 void。
    如果包含 SPJ 方法的类文件是 Java 包的一部分，则您还必须指定 Java 包名。另外，如果您未指定 Java 包名，则 CREATE PROCEDURE 语句将无法注册 SPJ。

  - *java-signature*
    指定 SPJ 方法签名。其由以下部分组成:
    ([java-datatype[, java-datatype]...])
    只有当您希望指定 Java 包装类（例如: *java.lang.Integer*）、而非 Java 原始数据类型（例如: *int*）时，您才必须提供 Java 签名。
    一种 SQL 数据类型默认映射到一种 Java 原始数据类型。
    Java 签名为大小写敏感性，它必须置于括号里面，例如:（java.lang.Integer, java.lang.Integer）。该签名必须按照各种参数数据类型在类文件的 Java 方法定义里出现的先后顺序指定它们。每种对应于 OUT 或 INOUT 形参的 Java 数据类型的后部都必须跟有一个空心方括号（[]），例如: java.lang.Integer[]。

  - *java-datatype*
    指定一种可映射的 Java 数据类型。更多有关将 Java 数据类型映射到 SQL 数据类型，请参阅 sql-datatype。

- LIBRARY*[[catalog-name.]schema-name.]library-name*
  指定一个包含 SPJ 方法的库的 ANSI 逻辑名称。如果您没有完全限定一个库名，EsgynDB SQL 将根据当前会话的 schema 限定它。

- EXTERNAL SECURITY *external-security-type*
  执行（或调用）SPJ 时可以确定用户权限或权利。SPJ 可以具有以下外部安全类型之一:
  - INVOKER 确定每个用户是否可以使用存储过程调用者的权限执行或调用该存储过程。该行为被称作调用者权利，在未指定 EXTERNAL

SECURITY 的情况下，这是默认行为。

调用者权利允许一个拥有 SPJ 执行权限的的用户使用其现有的权限去调用 SPJ。这种情况下，用户必须被授权访问 SPJ 运行所依托的底层数据库对象。

注意：授权用户访问底层数据库对象能使用户直接访问这些对象，由此可能导致用户了解到一些不应知晓的更为敏感或重要的数据，从而构成风险。例如：一个 SPJ 可以运行在底层数据库对象的数据子集上，但该数据库对象还可能包含用户不应知晓的其他更为敏感或关键的数据。

- DEFINER 确定用户是否能使用存储过程创建者的权限执行或调用存储过程，该行为被称作定义者权利。

  定义者权利的优势在于用户可以在未被授予底层数据库对象权限的情况下调用存储过程，从而操作数据。这样，用户在直接访问或操作数据库里更为敏感或重要的数据时将会受限。

  然而，在授权用户执行具有定义者外部安全性的 SPJ 时您还是必须谨慎，因为这些用户今后无需底层数据库对象权限即可执行 SPJ。

- LANGUAGE JAVA

  指定外部用户定义的例程必须以 Java 语言编写。

- PARAMETER STYLE JAVA

  指定传递给外部用户定义例程的参数运行时约定必须采用 Java 语言编写。

- NO SQL

  指定 SPJ 不能执行 SQL 操作。

- CONTAINS SQL | MODIFIES SQL DATA | READS SQL DATA

  指定 SPJ 可以执行 SQL 操作。所有这些选项的行为都与 CONTAINS SQL 相同，即：SPJ 能够读取和修改 SQL。

  使用这些选项之一注册一个含有 SQL 语句的方法。如果您未指定一种 SQL 访问模式，则默认模式将是 CONTAINS SQL。

- DYNAMIC RESULT SETS *integer*

  指定 SPJ 能返回的结果集的最大数量。

  该选项只有在方法签名包含一个 `java.sql.ResultSet[]`对象时才适。

  如果方法包含一个结果集对象，则其有效数量范围介于 1 和 255 之间。

  SPJ 方法返回的结果集的实际数量可以小于或等于该数量。

如果您未指定该选项，则默认值将为 0（零），即 SPJ 不会返回任何结果集。

- TRANSACTION REQUIRED | NO TRANSACTION REQUIRED
  确定 SPJ 是否必须运行在一个从调用应用程序中继承的事务（TRANSACTION REQUIRED，默认选项）里或 SPJ 是否无需继承调用应用程序的事务（NO TRANSACTION REQUIRED）即可运行。
  通常情况下，您希望一个存储过程能够从调用应用程序里继承事务。然而，如果 SPJ 方法无法访问数据库或如果您希望一个存储过程管理自身事务，您应将该存储过程的事务属性设置为 NO TRANSACTION REQUIRED。更多信息，请参阅事务属性对于 SPJ 的影响。

- DETERMINISTIC | NOT DETERMINISTIC
  指定 SPJ 对于给定的一组实参数值的 OUT 和 INOUT 形参是否始终返回相同的数值（DETERMINISTIC）或不返回相同的数值（NOT DETERMINISTIC，默认选项）。
  如果您指定 DETERMINISTIC，则 EsgynDB SQL 无需每次都调用 SPJ，以便产生结果。相反，EsgynDB SQL 会缓存这些结果，并在后续调用时重新使用它们，因此可以优化 CALL 语句的效果。

- NO ISOLATE | ISOLATE
  指定在数据库服务器环境（NO ISOLATE）里执行 SPJ 还是在隔离环境里（ISOLATE）里执行 SPJ。EsgynDB SQL 允许这两种选项，但始终在 UDR 服务器进程（ISOLATE）里执行 SPJ。

2.14.2 CREATE PROCEDURE 注意事项

2.14.2.1 必需的权限

发出一条 CREATE PROCEDURE 语句，必须满足以下条件之一：

- 您是 DB_ROOT（数据库根用户）。
- 您正在一个共享的 schema 中创建存储过程，并且您对于创建存储过程期间使用的库拥有 USAGE（或 ALL）权限。USAGE 权限使您能读取该库的底层 JAR 文件，其包含 SPJ Java 方法。
- 您是私有 schema 的所有者，并且您对于创建存储过程期间使用的库拥有 USAGE（或 ALL）权限。USAGE 权限使您能读取该库的底层 JAR 文件，其包含 SPJ Java 方法。
- 您不但对于 SQL_OPERATIONS 组件拥有 CREATE 或 CREATE_ROUTINE 组件级权限，而且对于创建存储过程期间使用的库拥有 USAGE（或 ALL）权

限。USAGE 权限使您能够读取该库的底层 JAR 文件，其包含 SPJ Java 方法。

注意：在这种情况下，如果您在私有 schema 中创建函数，您必须是该 schema 的所有者。

2.14.2.2 事务属性对于 SPJ 的影响

- Transaction Required
  - 使用事务控制语句或方法
    - Δ 如果您指定 TRANSACTION REQUIRED（默认选项），并且目前暂无处于活动状态的事务，则 CALL 语句会自动发起一个事务。这种情况下，您不应在 SPJ 方法里使用事务控制语句（或等效的 JDBC 事务方法）。事务控制语句包括 COMMIT WORK 和 ROLLBACK WORK，等效的 JDBC 事务方法包括 Connection.commit()和 Connection.rollback()。
    - Δ 当一个存储过程的事务属性被设置为 TRANSACTION REQUIRED 时，如果您尝试在 SPJ 方法里使用事务控制语句或方法，则 SPJ 方法里的事务控制语句或方法将被忽略，并且 Java 虚拟机（JVM)不会报告任何错误或警告。当一个存储过程的事务属性被设置为 TRANSACTION REQUIRED 时，您应该依靠事务控制语句或方法在应用程序里调用该存储过程，并允许调用应用程序管理事务。
  - 提交或回滚一个事务
    - Δ 如果您不在调用应用程序里使用事务控制语句，则 CALL 语句所发起的事务可能不会将提交或回滚数据库更改。如果开启 AUTOCOMMIT 选项（默认设置），则数据库引擎会在 CALL 语句执行结束时自动提交或回滚任何数据库更改。然而，如果关闭 AUTOCOMMIT 选项，则当前事务将保持活动状态，直到客户端会话结束或您显式地提交或回滚该事务为止。
    - Δ 在调用一个 SPJ 时为了保证实现原子工作单元，请在 CALL 语句执行成功后在调用应用程序里使用 COMMIT WORK 语句提交该事务、或在 CALL 语句执行失败后在调用应用程序里使用 ROLLBACK WORK 语句回滚该事务。

- No Transaction Required

  一些情况下，您不希望 SPJ 方法从调用应用程序里继承事务。相反，您可能希望一个存储过程能够管理其自身的事务或在没有事务的情况下运行。

  不继承调用应用程序的事务在以下情况下有效：
  - 一个存储过程执行多个长时间运行的操作，例如：数据库里的多个 DDL

操作或表维护操作。在这种情况下，您可能希望在 SPJ 方法内定期提交这些操作，以防表被长时间锁定。

- 一个存储过程执行的某些 SQL 操作只能运行在没有活动状态事务的情况下。例如：当一个事务已经处于活动状态时，带有 WITH NO ROLLBACK 选项的 INSERT、UPDATE 和 DELETE 语句将被拒绝执行。同样，当一个存储过程从调用应用程序里继承一个事务时，该操作也将被拒绝执行。如果一个事务已经处于活动状态，则 PURGEDATA 语句也将被拒绝执行。
- 存储过程无法访问数据库。这种情况下，该存储过程无需从调用应用程序里继承事务。通过将一个存储过程的事务属性设置为 NO TRANSACTION REQUIRED，您可以避免出现传播到存储过程的调用应用程序事务的开销。

在这些情况下，您应该在创建一个存储过程时将其事务属性设置为 NO TRANSACTION REQUIRED。

如果您指定 NO TRANSACTION REQUIRED，并且如果 SPJ 方法创建了一条 JDBC 默认连接，则该连接默认情况下将启用自动提交选项。您既可以使用自动提交事务、也可以禁用自动提交（`conn.setAutoCommit(false);`）；必要时可以使用 JDBC 事务方法，`Connection.commit()` 和 `Connection.rollback()` 提交或回滚工作。

2.14.3 CREATE PROCEDURE 例子

- 该 CREATE PROCEDURE 语句可以注册一个名为 LOWERPRICE 的 SPJ，其不接收任何参数：

```
SET SCHEMA SALES;
CREATE PROCEDURE lowerprice()
     EXTERNAL NAME 'Sales.lowerPrice'
     LIBRARY saleslib
     LANGUAGE JAVA
     PARAMETER STYLE JAVA
     MODIFIES SQL DATA;
```

由于没有通过 catalog 和 schema 限定存储过程名称，因此 EsgynDB SQL 根据当前的会话设置限定它，这里的默认编目为 TRAFODION，schema 则被设置为 SALES。

由于存储过程需要能够读取和修改任何 SQL 数据，因此需在 CREATE PROCEDURE 语句里指定 MODIFIES SQL DATA。

请使用以下 CALL 语句调用该 SPJ:

```
CALL lowerprice();
```

该存储过程会将数据库里订单数量小于或等于 50 件的商品的价格削减 10%.

- CREATE PROCEDURE 语句注册一个名为 TOTALPRICE 的 SPJ，该 SPJ 可以接收三个输入参数、并将总价数值返回给 INOUT 形参:

```
CREATE PROCEDURE trafodion.sales.totalprice
        (IN qty NUMERIC (18),
        IN rate VARCHAR (10),
        INOUT price NUMERIC (18,2))
   EXTERNAL NAME 'Sales.totalPrice'
   LIBRARY sales.saleslib
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   NO SQL;
```

为了在 TrafCI 里调用 SPJ，请使用以下语句:

```
SET PARAM ?p 10.00;
CALL sales.totalprice(23, 'standard', ?p);
p
--------------------
253.97

--- SQL operation complete.
```

由于存储过程不能读取和修改任何 SQL 数据，因此在 CREATE PROCEDURE 语句里指定 NO SQL。

- 该 CREATE PROCEDURE 语句注册一个名为 MONTHLYORDERS 的 SPJ，该 SPJ 将接收某个月份的整数值，并返回订单数量:

```
CREATE PROCEDURE sales.monthlyorders
                       (IN INT,
                       OUT number INT)
EXTERNAL NAME 'Sales.numMonthlyOrders (int,
java.lang.Integer[])'
LIBRARY sales.saleslib
LANGUAGE JAVA
PARAMETER STYLE JAVA
READS SQL DATA;
```

由于 OUT 形参应该映射到 Java 包装类 `java.lang.Integer`，所以您必须指定 EXTERNAL NAME 子句里的 Java 签名。请使用以下 CALL 语句调用该 SPJ：

```
CALL sales.monthlyorders(3, ?);

ORDERNUM
-----------
4
--- SQL operation complete.
```

- 该 CREATE PROCEDURE 语句注册一个名为 ORDERSUMMARY 的 SPJ，该 SPJ 将接收某个格式化为字符串的日期， 并返回该日期或此后的订单的相关信息。

```
CREATE PROCEDURE sales.ordersummary
                      (IN on_or_after_date VARCHAR (20),
                      OUT num_orders LARGEINT)
  EXTERNAL NAME 'Sales.orderSummary (int, long[])'
  LIBRARY sales.saleslib
  EXTERNAL SECURITY invoker
  LANGUAGE JAVA
  PARAMETER STYLE JAVA
  READS SQL DATA
  DYNAMIC RESULT SETS 2;
```

请使用以下 CALL 语句调用该 SPJ：

```
CALL neo.sales.ordersummary('01-01-2014', ?);
```

存储过程 ORDERSUMMARY 会返回指定日期（01-01-2014）或此后订单的相关信息：

```
NUM_ORDERS
--------------
13

ORDERNUM NUM_PARTS AMOUNT ORDER_DATE  LAST_NAME
-------- --------- -----  ---------- ---------
100210    4       19020.00 2014-04-10 HUGHES
100250    4       22625.00 2014-01-23 HUGHES
101220    4       45525.00 2014-07-21 SCHNABL
```

…
--- 13 row(s) selected.

```
ORDERNUM PARTNUM UNIT_PRICE QTY_ORDERED PARTDESC
-------- ------- ---------- ----------- --------
100210    244   3500.00     3         PC GOLD, 30 MB
100210    2001  1100.00     3     GRAPHIC PRINTER,M1
100210    2403   620.00     6       DAISY PRINTER,T2
…
--- 70 row(s) selected.
--- SQL operation complete.
```

2.15 CREATE ROLE 语句

CREATE ROLE 语句创建一个 SQL 角色。更多信息，请参阅角色。

```
CREATE ROLE role-name [ WITH ADMIN grantor ]
grantor is:
     database-username
```

2.15.1 CREATE ROLE 语法

- *role-name*
  是一种指定新角色的 SQL 标识符、不区分大小写的常规或分隔标识符。
  更多信息，请参阅不区分大小写字母的分隔标识符。
  role-name 既不能是一个现有的角色名称、也不能是一个已注册的数据库用户名。然而，role-name 可以是一个已配置的 directory-service 用户名。

- WITH ADMIN *grantor*
  指定一个并非当前用户的角色所有者。这是一条可选子句。

- *grantor*
  指定一个您希望分配角色所有者的已注册数据库用户名。

2.15.2 CREATE ROLE 注意事项

- 为了创建一个角色，您必须为 DB_ROOT（数据库根用户）或已被授予 SQL_OPERATIONS 的 MANAGE_ROLES 组件权限。
- PUBLIC、_SYSTEM、NONE 和以 DB_ 开头的数据库用户名都被保留。您无法使用任何此类名称指定角色名。

2.15.2.1 角色所有权

通过在 WITH ADMIN grantor 子句中指定授权者为用户，您可以将角色所有权赋予用户。角色所有者可以执行这些操作：

- 授予和撤销用户角色。
- 删除角色。

角色所有权是永久性的。您在创建一个角色后，不得变更该角色的所有权或将其分配给其他用户。

2.15.3 CREATE ROLE 例子

- 创建一个角色、并指定当前用户作为角色所有者:
  ```
  CREATE ROLE clerks;
  ```

- 创建一个角色、并指定其他用户作为角色所有者:
  ```
  CREATE ROLE sales WITH ADMIN cmiller;
  ```

2.16 CREATE SCHEMA 语句

该语句在数据库里创建一个 schema。更多信息，请参阅 Schemas。

```
CREATE [schema-class] SCHEMA [IF NOT EXISTS] schema-clause
[stored desc]
[incremental backup]


schema-class is:
     [ PRIVATE | SHARED ]

schema-clause is:
     { schema-name [AUTHORIZATION authid] | AUTHORIZATION
     authid }
```

2.16.1 CREATE SCHEMA 语法

- *schema-class*
  标明默认情况下是否只有授权 ID 才可访问 schema（PRIVATE）、以及是否任何数据库用户都可以将对象添加到 schema（SHARED）。默认类为 PRIVATE（私有类）。

注意: 在早期版本中创建的 schema 是 SHARED schema。

- IF NOT EXISTS
  如果 schema 不存在，则创建 schema。如果省略，并且 schema 已经存在则会引发错误。

- *schema-name*
  这既是新的 schema 名称，也是一个 SQL 标识符，它指定一个当前并非是 schema 名称的唯一名称。此参数是可选的。然而，如果您未指定 schema 名称，则必须指定授权子句。如果没有提供 schema 名称，则使用授权 ID 作为 schema 名称。如果授权 ID 名称匹配现有的 schema，则 CREATE SCHEMA 命令将会运行失败。

- *authid*
  此为拥有和管理 schema 此的数据库用户或角色的名称。如果该子句不存在，则当前用户将成为 schema 所有者。

- *stored desc*

创建 schema 时直接带有 stored desc 属性，后续 schema 下新建的表会自动带有 stored desc 属性，同时生成 descriptor。

- `Incremental Backup`
  创建 schema 时直接带有 Incremental Backup 属性，后续 schema 下新建的表会自动带有 incremental Backup 属性，都可以在上面做增量备份

2.16.2 CREATE SCHEMA 注意事项

2.16.2.1 保留的 schema 名称

以前导下划线(_)开头的 schema 名称将被保留，以供将来使用。

2.16.2.2 AUTHORIZATION 子句

AUTHORIZATION 子句是可选的。如果您省略该子句，则当前用户将成为 schema 所有者。

注意：即使未对 EsgynDB 数据库启用授权，也会为模式名称分配授权标识。但是，除非启用授权，否则不会强制执行。

schema 所有者可以执行 schema 的操作以及 schema 内部对象的操作。例如：

- 改变对象的 DDL
- 删除 schema
- 删除对象
- 使用诸如 UPDATE STATISTICS 和 PURGEDATA 之类的实用工具命令管理对象

2.16.2.3 谁可以创建 schema?

创建 schema 的权限受控于 SQL_OPERATIONS 组件的 CREATE_SCHEMA 权限。默认情况下，该权限将被授予 PUBLIC、但也可被 DB ROOT（数据库根用户）撤销。

当授权被初始化后，所有授权 ID 都将被授予 CREATE_SCHEMA 权限：

- PUBLIC
- DB__ROOT
- DB__ROOTROLE

DB__ROOT 或被授予 DB__ROOTROLE 角色的任何人都可以授予 CREATE SCHEMA 权限。

2.16.3 CREATE SCHEMA 例子

- 该例子创建一个名为 MYSCHEMA、且归属当前用户的私有 schema:
  ```
  CREATE SCHEMA myschema;
  ```

- 该例子创建了一个共享 schema，并指定 CliffG 作为所有者:
  ```
  CREATE SHARED SCHEMA hockey_league AUTHORIZATION
  "CliffG";
  ```

- 该例子创建了一个私有 schema，并指定 DBA 角色作为 schema 所有者:
  ```
  CREATE PRIVATE SCHEMA contracts AUTHORIZATION DBA;
  ```
  被授予 DBA 角色的用户可以授权其他用户和角色访问 CONTRACTS schema 里的对象。

- 该例子创建了一个名为 JSMITH 的 schema:
  ```
  CREATE PRIVATE SCHEMA AUTHORIZATION JSmith;
  ```

- 该例子创建一个带有 stored desc 属性的 schema
  ```
  SQL>create schema sch1 stored desc;

  --- SQL operation complete.

  SQL>showddl schema sch1;

  CREATE PRIVATE SCHEMA TRAFODION.SCH1 AUTHORIZATION
  DB__ROOT STORED DESC ;
  -- GRANT SELECT, INSERT, DELETE, UPDATE, USAGE,
  REFERENCES, EXECUTE, CREATE, ALTER, DROP ON SCHEMA
  TRAFODION.SCH1 TO DB__ROOT WITH GRANT OPTION;

  --- SQL operation complete.。
  ```

- 该例子创建一个带有 Incremental Backup 属性的 schema
  ```
  SQL>create schema test_my incremental backup;

  --- SQL operation complete.
  ```

注意:

- 如果您想在已经创建的 schema 上添加 stored desc 属性，可以通过以下语句来实现:
  ```
  alter schema schema-name generate stored desc
  ```
  举例:
  ```
  SQL>alter schema sch2 generate stored desc;
  ```

```
--- SQL operation complete.

SQL>showddl schema sch2;


CREATE PRIVATE SCHEMA TRAFODION.SCH2 AUTHORIZATION
DB__ROOT STORED DESC ;
-- GRANT SELECT, INSERT, DELETE, UPDATE, USAGE,
REFERENCES, EXECUTE, CREATE, ALTER, DROP ON SCHEMA
TRAFODION.SCH2 TO DB__ROOT WITH GRANT OPTION;

--- SQL operation complete.
```

- 不支持通过 alter schema 的方式来添加这个属性
- stored desc 属性和 incremental backup 属性连用时,
  - ◊ 顺序必须是 stored desc 在前, incremental backup 在后
  - ◊ 两个属性连用时, create schema 中使用 if not exists 会报错, 因此创建时不能加上 if not exists 这三个关键字。 这是一个缺陷, 尚未解决

2.17 CREATE SEQUENCE 语句

该语句生成一个自动递增的数值序列，可供多个用户用作主键值。例如，要在序列中只生成奇数，可以创建一个以 START WITH=1, INCREMENT BY=2 定义的序列。

可以同时指定多个选项，并用空格分隔。

有两个表达式可以与序列一起使用:

- SEQNUM(*sequence-name*, CURRENT)：获取当前会话序列返回的当前值，而不是序列的当前值。

  例如，一个名为 *employee_seq* 的序列对象，它以 1 开头，最多缓存 25 个值，仅使用 3 个用户。

  如果用户 A 调用 SEQNUM(employee_seq, CURRENT)，则返回 1，并保留 1 到 25，用户 B 立即调用 SEQNUM(*sequence-name*, CURRENT)，返回 26，并保留 26 到 51，然后用户 C 立即调用 SEQNUM(sequence-name, CURRENT，返回 51，并保留 51 到 75。

  现在，如果用户 A 立即调用 SEQNUM(*employee_seq*, NEXT)，则返回 2 而不是 76。如果用户 A 在达到 25 之后立即调用 SEQNUM(*employee_seq*, NEXT)，则返回 76，此时用户 A 的缓存已耗尽。
- SEQNUM(*employee_seq*, NEXT)：推进序列并检索下一个值。对于递增序列，该值是增加的。

注意:

- 第一次是通过调用 SEQNUM(*sequence-name*, CURRENT) 或 SEQNUM(*sequence-name*, NEXT) 来返回序列的初始值。

- 随着后续对 SEQNUM（sequencename，NEXT）的每次引用，它都返回序列的一个新增量值。

- 为了兼容 Oracle，可以使用以下语句引用 SQL 语句中的序列值:
  CURRVAL：返回序列的当前值
  NEXTVAL：递增序列并返回下一个值
  必须使用序列名限定 CURRVAL 和 NEXTVAL:
  sequence.CURRVAL
  sequence.NEXTVAL

CREATE SEQUENCE 是 EsgynDB SQL 扩展语句。

更多信息，请参阅 DROP SEQUENCE 语句或 ALTER SEQUENCE 语句。

```
CREATE SEQUENCE [[catalog-name.]schema-name.]sequence-name
[START WITH integer]
[INCREMENT BY integer]
[MAXVALUE integer | NO MAXVALUE]
[MINVALUE integer]
[CACHE integer | NO CACHE]
[CYCLE | NO CYCLE]
[ORDER | NO ORDER]
[DATA TYPE]
[GLOBAL [TIMEOUT long integer]]
```

默认情况下，它是一个递增序列，以 1 开头，增量为 1，缓存值为 25，最大值为 2^{63}-2，不循环，属于 LARGEINT 数据类型。

2.17.1 CREATE SEQUENCE 语法

- [[catalog-name.]schema-name.]sequence-name

  指定要创建的序列的名称，以及可选的目录和 schema 名称，其中每部分都是一个有效的 sql 标识符，最长 128 个字符。
  该名称应该是唯一的，并且对于同一 schema 中的任何会话都不存在。EsgynDB 不支持会话名称的重载。也就是说，不能多次注册相同的会话名称。如果您没有完全限定会话名称，那么 EsgynDB 将根据当前会话的 schema 对其进行限定。更多信息，请参阅标识符和数据库对象名称。

- START WITH

  指定初始值。该值只能是正数，并且必须大于或等于 MINVALUE(如果指定 NO CACHE)且小于 MAXVALUE。

  注意:
  如果初始值超出了 MINVALUE 或 MAXVALUE 指定的范围，则会引发错误。
  如果未指定，则 START WITH 默认为 1。对于升序序列，此值为 MINVALUE。

- INCREMENT BY

  指定连续序列值之间的增量值。
  该值只能是正数，因此 EsgynDB 仅支持递增序列。该值必须小于或等于 MAXVALUE 和 MINVALUE 之间的差值。

如果未指定，则 INCREMENT BY 默认为 1。

- MAXVALUE
  指定升序序列停止生成值或循环的最大值。默认值为 2^{63}-2。
  序列的 MAXVALUE 取决于数据类型支持的最大值，您还可以在该范围内指定 MAXVALUE。

注意：

- 如果 MAXVALUE 超出支持范围，则返回一个错误；或者在超过 MAXVALUE 且指定了 NO CYCLE 的情况下，如果为升序序列生成一个后续值，也返回一个错误。
- 如果 INCREAMENT BY 的值不是 1，则 MAXVALUE 可能与序列生成的实际最大值不同。例如，当用 START with 3 指定序列时，INCREMENT 4 和 MAXVALUE 12 将返回最大值 11。

- NO MAXVALUE
  如果为升序序列指定 NO MAXVALUE，则 MAXVALUE 是您指定的值或数据类型支持的最大值。

- MINVALUE
  指定升序序列在达到最大极限后循环的最小值。默认值为 1。
  序列的 MINVALUE 取决于数据类型支持的最小值，您还可以在该范围内指定 MINVALUE。
  如果未指定 MINVALUE 和 START WITH，则对于升序序列，MINVALUE 默认值为 1。

- CYCLE
  指定序列耗在达到 MAXVALUE 后，用尽其范围或进行换行。
  如果指定，对于升序序列，它将在达到 MAXVALUE 后从 MINVALUE 重新启动。

- NO CYCLE
  指定一旦达到 MAXVALUE，序列就不能循环，如果超过限制，则在生成后续值时引发异常。

默认值为 NO CYCLE。

- CACHE
  指定预先分配并存储在内存中的后续值的范围，以加快将来的请求。
  默认 CACHE 值为 25。
  最小 CACHE 值为 2。
  CACHE 值必须小于或等于由以下公式确定的值：
  (MAXVALUE - START WITH + 1) / INCREMENT BY

  当缓存耗尽时，将被重新填充序列缓存。

注意：如果发生意外关闭或手动重新启动序列对象的情况，则未使用的预分配值将丢失，并留下意外的间隔，EsgynDB 将继续从其停止的位置缓存新值，因此除非指定了 CYCLE，否则相同的值永远不会出现两次。

- NO CACHE
  指定不预先分配序列的值，即使可能导致性能下降，但是这样降低了丢失值的风险。在这种情况下，每个对新值的请求都将在系统表中同步更新。
  如果未指定 CACHE 和 NO CACHE，默认情况下它将缓存 25 个值。
- ORDER
  指定 ORDER 以保证按请求顺序生成序列号。
  效果与 NO CACHE 相同。
- NO ORDER
  此选项跟 ORDER 相对应，并不按照请求的顺序进行生成。
- DATA TYPE
  序列可以指定为以下数据类型：

类型	最小值	最大值
SMALLINT UNSIGNED	1	2^{16}-1
INTEGER UNSIGHED	1	2^{32}-1
LARGEINT	1	2^{63}-2

- GLOBAL
  指定生成的序列值对于集群是全局唯一的。它是一个 64 位值，可以保证高于任何先前生成的全局序列 ID 值。 2 个连续值之间的差异可能会有所不同。
  如果指定了选项 GLOBAL，则不能同时指定其他选项。例如 CACHE，MAXVALUE 等。

  - [TIMEOUT]
    可选，timeout_val，以微秒为单位。默认值为-1，表示没有指定的

timeout。如果提供了 timeout 值，则表示发送到全局 seq id 服务器的请求将等待的时间段。

2.17.2 CREATE SEQUENCE 注意事项

执行 CREATE SEQUENCE 语句，必须满足以下条件之一:

- 您是 DB_ROOT(数据库根用户)。
- 您正在一个共享的 schema 中创建序列。
- 您是私有 schema 的所有者。
- 您拥有 SQL_OPERATIONS 组件的 CREATE 或 CREATE_SEQUENCE 组件权限。

注意: 在这种情况下，如果您在私有 schema 中创建序列，您必须是 schema 的所有者拥有。

在 GLOBAL sequence 的情况下，两个连续值之间的差异可能会有所不同。不能保证主叫方客户端进程返回的seq号与客户端调用的顺序相同，因为每个主叫方都将进入中央进程以获取下一个唯一值。因此，根据请求排队的方式，将生成 ID。

2.17.3 CREATE SEQUENCE 例子

以下语句创建一个序列 employee_seq，以 1 开头，增量为 2，最大值为 10000，不循环，每次缓存 20 个值，属于 SMALLINT UNSIGNED 数据类型。

```
CREATE SEQUENCE employee_seq
START WITH 1
INCREMENT BY 2
MAXVALUE 10000
NO CYCLE
CACHE 20
SMALLINT UNSIGNED
```

对 SEQNUM(*empolyee_seq*, NEXT) 的第一次引用返回值 1.

```
SQL>SELECT SEQNUM(employee_seq, NEXT) FROM DUAL;
(EXPR)
--------------------
                   1
--- 1 row(s) selected.
```

对 SEQNUM(*empolyee_seq*, NEXT) 的第二次引用返回值 2。每个后续引用

返回的值都比前一个值大 2。

```
SQL>SELECT SEQNUM(employee_seq, NEXT) FROM DUAL;
(EXPR)
--------------------
                  3
--- 1 row(s) selected.
```

2.17.3.1 GLOBAL SEQUENCE 例子

```
SQL>create sequence sg1 global;

--- SQL operation complete.


SQL>create sequence sg0 global timeout 10000;

--- SQL operation complete.


>>select 'seqid',seqnum(sg1) from (values(1)) x(a);

(EXPR)  (EXPR)
------  --------------------

seqid       1652850963536394

--- 1 row(s) selected.
```

2.18 CREATE TABLE 语句

该语句创建一个 EsgynDB SQL 表，从而将关系型 SQL 表映射到 HBase 表。

CREATE VOLATILE TABLE 语句创建一个临时性 EsgynDB SQL 表，该表只有在 SQL 会话期间存在。

CREATE TABLE AS 语句基于 SELECT 查询的数据属性创建一个表，并使用 SELECT 查询返回的数据填充该表。更多信息，请参阅表。

注意： 目前的事务不支持 DDL 语句。这意味着您既无法在用户定义的事务（BEGIN WORK...COMMIT WORK）中运行该语句、也无法在 AUTOCOMMIT 选项关闭时运行该语句。为了运行该语句，必须开启该会话的 AUTOCOMMIT（自动提交）选项（默认）。

```
CREATE [VOLATILE| TEMPORARY] TABLE IF NOT EXISTS table
     { table-spec | like-spec }
     [STORE BY {PRIMARY KEY | (key-column-list)}]
     [SALT USING num PARTITIONS [IN num REGIONS]
     [RANGE SPLIT BY (column[ASC|DESC]
     [,column [ASC|DESC] ...]) split-list]
     [ON (column[, column]...)]]
     [SPLIT BY (column[, column]...]) split-list]
     [HBASE_OPTIONS (hbase-options-list)]
     [LOAD IF EXISTS | NO LOAD]
     [AS select-query]
     [ATTRIBUTES (synchronous | no) replication]
     [ATTRIBUTE NAMESPACE namespace-string]
     [attribute stored desc]
     [attribute incremental backup]
```

注意： EsgynDB 2.1.0 添加了对 CREATE TABLE LIKE 中 SALT 的支持。

```
table-spec is:
     (table-element [,table-element]...)

table-element is:
     column-definition
     | [CONSTRAINT constraint-name] table-constraint

column-definition is:
     column data-type
     [DEFAULT default | NO DEFAULT]
     [[CONSTRAINT constraint-name] column-constraint]...
```

```
data-type is:
     CHAR[ACTER] [(length [CHARACTERS])]
          [CHARACTER SET char-set-name]
          [UPSHIFT] [[NOT]CASESPECIFIC]
     | CHAR[ACTER] VARYING (length [CHARACTERS])
          [CHARACTER SET char-set-name]
          [UPSHIFT] [[NOT]CASESPECIFIC]
     | VARCHAR (length [characters])
           [CHARACTER SET char-set-name]
           [UPSHIFT] [[NOT]CASESPECIFIC]
      | varchar2 (length [characters])
           [character set char-set-name]
           [upshift] [[not]casespecific]

     | NCHAR (length) [CHARACTERS]
            [UPSHIFT] [[NOT]CASESPECIFIC]
     | NCHAR VARYING(length [CHARACTERS])
            [UPSHIFT] [NOT] CASESPECIFIC]

     | NUMERIC [(precision [,scale])] [SIGNED|UNSIGNED]
     | tinyint [signed|unsigned]
     | SMALLINT [SIGNED|UNSIGNED]
     | INT[EGER] [SIGNED|UNSIGNED]
     | LARGEINT
     | DEC[IMAL] [(precision [,scale])] [SIGNED|UNSIGNED]
     | FLOAT [(precision)]
     | REAL
     | DOUBLE PRECISION
     | DATE
     | TIME [(time-precision)]
     | TIMESTAMP [(timestamp-precision)]
     | INTERVAL { start-field TO end-field | single-field }


default is:
       literal
     | NULL
     | CURRENT_DATE
     | CURRENT_TIME
     | CURRENT_TIMESTAMP

column-constraint is:
       NOT NULL
     | UNIQUE
     | PRIMARY KEY [ASC[ENDING] | DESC[ENDING]]
     | CHECK (condition)
     | REFERENCES ref-spec

table-constraint is:
       UNIQUE (column-list)
     | PRIMARY KEY (key-column-list)
```

```
      | CHECK (condition)
      | FOREIGN KEY (column-list) REFERENCES ref-spec

ref-spec is:
      referenced-table [(column-list)]

column-list is:
      column-name [,column-name]...

key-column-list is:
      column-name [ASC[ENDING] | DESC[ENDING]]
           [,column-name [ASC[ENDING] | DESC[ENDING]]]...

like-spec is:
      LIKE source-table [include-option]

include-option is:
        WITH CONSTRAINTS
      | WITHOUT DIVISION
      | WITHOUT LOB COLUMNS
      | WITHOUT SALT
      | WITH [NO] DATA


split-list is:
      ( split-list-element [, split-list-element] ... )

split-list-element is:
ADD FIRST KEY  ( literal [, literal] ... )

hbase-options-list is:
      hbase-option = 'value'[, hbase-option = 'value']...
```

2.18.1 CREATE TABLE 语法

- VOLATILE| TEMPORARY

  指定一个临时表，该表仅存在于创建表会话期间。会话结束时，该表将被自动删除。更多信息，请参阅 CREATE VOLATILE TABLE 注意事项。

  TEMPORARY 和 VOLATILE 等同。

- IF NOT EXISTS

  如果创建表时 HBase 表不存在，那么创建一个 HBase 表。该选项不适用于临时表。

- table

  指定一个表的 ANSI 逻辑名称。更多信息，请参阅数据库对象名称。

该名称在其所属的 schema 内部的表和视图名称里必须是唯一的。

- STORE BY { PRIMARY KEY | (key-column-list)}
  指定聚集键所基于的一组列。聚集键确定保存该表的物理文件内部的行序。存储顺序可以影响您的对象分区方案。

  - PRIMARY KEY
    聚集键基于主键列而定。

  - key-column-list
    聚集键基于 key-column-list 里的列而定。
    key-column-list 里的键列必须被指定为 NOT NULL（非空），并且必须与表上定义的主键列相同。
    如果没有指定 STORE BY，则聚集键将成为主键。

- SALT USING num PARTITIONS [IN num REGIONS] [ON (column[, column]...)]
  在创建表时，将其预分割为多个区域。“加盐”技术会添加行键的哈希值作为键前缀，以防出现序列键“热点”。
  如果您未指定单独的分区和区域编号，则将为每个分区创建一个 HBase 区域。 您指定的分区数量可以是目前 HBase 集群里的区域服务器数量的函数。如果您使用更少量的区域，则未来还有增长空间。您指定的分区数量从 2 到 1024 不等，区域数量可以从 2 到加盐分区数量不等。
  如果您未指定列，则默认使用所有主键列。
  如果使用 LIKE 指定 SALT，则此规范将覆盖源表的规范。注意：这是一项新功能。在早期版本中，如果使用 LIKE 指定 SALT 子句，则该子句会被忽略。

注意： 如果您指定 SALT 选项，EsgynDB 会创建一个带有 ConstantSizeRegionSplitPolicy 的表。您可以使用以下 HBASE_OPTIONS 覆盖此设置。

- SPLIT BY (column [, column]...]) split-list
  在创建一个表时将其预分隔为多个 HBase 区域。该选项不可与 SALT 同时使用。指定的列必须带有 PRIMARY KEY 或 STORE BY key 前缀。如果您使用 DIVISION BY 子句，则首列必须为“_DIVISION_1_”。
  如果您希望显式地指定 HBase 区域边界、而非由 HBase 自动分割区域，则

请使用该选项。

注意： 如果您指定 SALT 选项，EsgynDB 会创建一个带有 ConstantSizeRegionSplitPolicy 的表。您可以使用以下 HBASE_OPTIONS 覆盖此设置。

- *(split-list-element [, split-list-element]...)*
  这是 HBase 区域的分界列表，它按照逻辑顺序排列（ascending 代表升序键列，descending 代表降序键列）。
  如果指定 n 个 *split-list-elements*（拆分列表元素），则已创建的 HBase 表将具有（n+1）个区域。
  第一个区域开始于最低级别的键，此后的 n 个区域都具有 n 个拆分列表元素指定的起始键。

- *(literal [, literal]...)*
  指定一个分割边界的列值。字面值对应于 SPLIT BY 里指定的列，并必须具有相应的数据类型。

- HBASE_OPTIONS (hbase-option = 'value'[, hbase-option = 'value']...)
  为该表设置的 HBase 选项列表
  hbase-option = 'value' 是此类 HBase 选项之一，其分配数值如下：

HBase 选项	**接受值**[5]	**默认值**
BLOCKCACHE	'true' | 'false'	'true'
BLOCKSIZE	'1024' - '64*1024*1024'	'64*1024'
BLOOMFILTER	'NONE' | 'ROW' | 'ROWCOL'	'ROW'
CACHE_BLOOMS_ON_WRITE	'true' | 'false'	'false'
CACHE_DATA_ON_WRITE	'true' | 'false'	'false'
CACHE_INDEXES_ON_WRITE	'true' | 'false'	'false'
COMPACT	'true' | 'false'	'true'
COMPACT_COMPRESSION	'GZ' | 'LZ4' | 'LZO' | 'NONE' | 'SNAPPY'	'NONE'
COMPRESSION	'GZ' | 'LZ4' | 'LZO' | 'NONE' | 'SNAPPY'	'NONE'
DATA_BLOCK_ENCODING	'DIFF' | 'FAST_DIFF' | 'NONE' | 'PREFIX'	'NONE'

[5]粗体字为默认值

DURABILITY	'USE_DEFAULT' | 'SKIP_WAL' | 'ASYNC_WAL' | 'SYNC_WAL' | 'FSYNC_WAL'	'SYNC_WAL'
EVICT_BLOCKS_ON_CLOSE	'true' | 'false'	'false'
IN_MEMORY	'true' | 'false'	'false'
KEEP_DELETED_CELLS	'true' | 'false'	'false'
MAX_FILESIZE	'2*1024*1024' - '2^{63}-1'	'10*1024*1024*1024'
MAX_VERSIONS	'1' | 'positive-integer'	'1'
MEMSTORE_FLUSH_SIZE	1024*1024' - '2^{63}-1'	'128*1024*1024'
MIN_VERSIONS	'0' | 'positive-integer'	'0'
PREFIX_LENGTH_KEY	'positive-integer'，其长度应小于表键的最大长度。只有当 SPLIT_POLICY 为 KeyPrefixRegionSplitPolicy 时，它才适用。	'2'
REPLICATION_SCOPE	'0' | '1'	'1'
SPLIT_POLICY	'org.apache.hadoop.hbase.regionserver.ConstantSizeRegionSplitPolicy' | 'org.apache.hadoop.hbase.regionserver.IncreasingToUpperBoundRegionSplitPolicy' | 'org.apache.hadoop.hbase.regionserver.KeyPrefixRegionSplitPolicy'	'org.apache.hadoop.hbase.regionserver.IncreasingToUpperBoundRegionSplitPolicy'
TTL	'positive-integer'，其长度应小于或等于 2147483447。	'2147483647' (forever)

- LOAD IF EXISTS
  将数据装载到一个现有的表里。必须与 AS select 查询同时使用。更多信息，请参阅 CREATE TABLE AS 语句的 LOAD IF EXISTS 和 NO LOAD 选项的注意事项。

- NO LOAD
  使用 CREATE TABLE AS 语句创建一个表，但不会将数据装载到该表里。更多信息，请参阅 CREATE TABLE AS 语句的 LOAD IF EXISTS 和 NO LOAD 选项的注意事项。

- AS *select-query*

指定一条 select 查询语句，用于填充所创建的表。一条 select 查询可以是任何 SQL select 语句。

- *division-expr-list*
  是一个只能引用主键或STORE BY列的表达式列表。
  以下是DIVISON BY子句中允许的表达式，它们是单调递增的表达式：
  • cast((exact_numeric_column[+ const1]) / const2 as numeric_data_type)
  • DATE_PART('YEAR', date_part_arg)
  • DATE_PART('YEARQUARTER', date_part_arg)
  • DATE_PART('YEARMONTH', date_part_arg)
  • DATE_PART('YEARWEEK', date_part_arg)
    • date_part_arg is one of the following:
      • datetime_col
      • datetime_col + const
      • datetime_col - const
      • ADD_MONTHS(datetime_col, const [, 0])
      • DATE_ADD(datetime_col, const)
      • DATE_SUB(datetime_col, const)
  • DATE_TRUNC(const, datetime_col)
  • DATEDIFF(YEAR, const, datetime_col)
  • DATEDIFF(QUARTER, const, datetime_col)
  • DATEDIFF(MONTH, const, datetime_col)
  • DATEDIFF(WEEK, const, datetime_col)
  • LEFT(character_col, const)
  • SUBSTR[ING](character_col, 1, const)
  • SUBSTR[ING](character_col FROM 1 FOR const)

- *column data-type*
  指定表中列的名称和数据类型。CREATE TABLE 语句里必须至少包含一个列的定义。

  - *column*
    是一种 SQL 标识符。该列在表的列名里必须是唯一的。如果列名是 EsgynDB SQL 保留字，则您必须将其用双引号括起来，以进行分隔。此类分隔部分须区分大小写。例如："join".

  - *data-type*
    是列里可存储数值的数据类型。默认值必须与列具有相同的类型，其中包含字符列的字符集。更多信息，请参阅数据类型。
    数据类型还包括特定大小写格式的信息，例如 UPSHIFT。

- [NOT] CASESPECIFIC
  指定该列包含非特定大小写格式的字符串。默认值为 CASESPECIFIC。
  只有当两个数值都不区分大小写时，才能以不区分大小写的方式对二者进行比较。
  这种做法适用于二元谓词、LIKE 谓词、以及 POSITION/REPLAC 字符串函数搜索的数值比较。更多信息，请参阅 CREATE TABLE 例子。

- DEFAULT *default* | NO DEFAULT
  指定一个列的默认值或指定该列不具有默认值。
  更多信息，请参阅 SQL 子句。

- CONSTRAINT *constraint-name*
  指定一个列名或表约束名。
  constraint-name 必须与 *table* 具有相同的 schema，且其在 schema 中的约束名必须是唯一的。
  如果省略在 *constraint-name* 里指定名称的 schema 部分，则 EsgynDB SQL 将使用表的 schema 以扩展该约束名。更多信息，请参阅约束名和数据库对象名称。

- NOT NULL
  这是一个列约束，它指定该列不能包含空值。
  如果您省略 NOT NULL，则该列中可以出现空值。如果您同时指定 NOT NULL 和 NO DEFAULT，则插入表里的每行都必须包含一个该列的值。
  更多信息，请参阅 Null。

- UNIQUE 或 UNIQUE (*column-list*)
  分别为一种列约束或表约束，它指定该列或多列里不得多次出现同一个数值或同一组数值。如果省略 UNIQUE，则允许出现重复数值，除非该列是 PRIMARY KEY 的一部分。
  column-list 里不得多次出现相同的列。此外，在 UNIQUE 约束上指定的一组列不得匹配为表的主键约束或任何其他 UNIQUE 约束指定的一组列。所有被定义为 UNIQUE 的列都必须被指定为 NOT NULL。
  唯一索引强制实施 UNIQUE 约束。如果 column-list 里已有一个唯一索引，则 EsgynDB SQL 将使用该索引。如果不存在唯一索引，则系统会创建一个此类索引。

- PRIMARY KEY [ASC[ENDING] | DESC[ENDING]], or, PRIMARY KEY (*key-column-list*)
  分别是一个列约束或表约束，它指定一列或多列作为表的主键。key-column-list 里不得多次出现同一列。
  ASCENDING 和 DESCENDING 指定键内一列的输入方向。默认值为 ASCENDING。
  表内各行的 PRIMARY KEY 值必须是唯一的。为一组列定义的 PRIMARY KEY 表明列值是唯一且不为空的。在任何 CREATE TABLE 语句上只能指定一次 PRIMARY KEY。
  EsgynDB SQL 使用主键作为表的聚集键，以防创建一个独立唯一的索引以实施主键约束。
  PRIMARY KEY 在 EsgynDB SQL 里是必需的。

- CHECK (*condition*)
  这是一种约束，它指定了表里各行都必须满足的条件。更多信息，请参阅搜索条件。
  您既无法在 CHECK 约束里引用 CURRENT_DATE、CURRENT_TIME 或 CURRENT_TIMESTAMP，也无法在 CHECK 约束里使用子查询。

- REFERENCES *ref-spec*
  指定一种 REFERENCES 列约束，采用 REFERENCES 约束的各列的最大组合长度为 2048 字节。
  ref-spec 是:
  referenced-table [(column-list)]
  referenced-table 是引用约束里外键所引用的表。
  referenced-table 既不能是一个视图，也不能与表相同。
  referenced-table 对应表的外键。
  column-list 指定 referenced-table 的一列或多列对应于该表的外键。与 REFERENCES 相关联的列表里的列、以及与 FOREIGN KEY 相关联的列表里的列必须具有相同顺序。
  如果省略 *column-list*，则被引用表的 PRIMARY KEY 列将是被引用的列。一个表可以有无数的引用约束，您也可以在多个引用约束里指定相同的外键，但您必须单独定义每个引用约束。您无法创建自引用的外键约束。

- FOREIGN KEY (*column-list*) REFERENCES *ref-spec*
  这是一种表约束，它指定表的引用约束，并声明表里的一列或多列（外键）

仅包含那些匹配 REFERENCES 子句指定表的一列或多列的数值。

两列或一组列必须具备相同的特性（数据类型、长度、尺度、精度）。如果不存在 FOREIGN KEY 子句，则 table 的外键将是已定义的列；如果存在 FOREIGN KEY 子句，则外键将是 FOREIGN KEY 子句里指定的一列或多列。

- LIKE *source-table [include-option]*...

  指导 EsgynDB SQL 创建一个与现有表和源表类似的表，并省略约束（NOT NULL 和 PRIMARY KEY 约束除外）和分区，除非已指定 include-option 子句。

  source-table

  这是现有表的 ANSI 逻辑名称，必须与其 schema 中的表名和视图名不同。

  include-option

  - *WITH CONSTRAINTS*

    指导 EsgynDB SQL 使用来自 *source-table* 的约束。表的约束名是随机生成的唯一名称。

    执行 CREATE TABLE LIKE 语句时，无论是否包含 WITH CONSTRAINTS 子句，目标表都将具有源表上存在的所有 NOT NULL 列约束，但会采用不同的约束名。

  - *WITHOUT DIVISION*

    指导 EsgynDB SQL 不使用 *source-table* 的分区。如果省略此子句，则该表将具有与 *source-table* 相同的分区。

  - *WITHOUT LOB COLUMNS*

    指导 EsgynDB SQL 不包含源表中的 LOB 列。如果省略此子句，则该表将具有与 *source-table* 相同的 LOB 列。

  - *WITHOUT SALT*

    指导 EsgynDB SQL 不要对表使用“加盐”。如果省略此子句，并且没有指定 SALT 子句，那么该表将具有与 *source-table* 相同的分区。

    如果还指定了 SALT 子句，则无法指定此选项。

  - *WITH DATA*

    指导 EsgynDB SQL 包含源表中的数据。如果省略此子句，则该表将创建为一个空表。WITH DATA 子句在表结构创建完成后，将源表的数据加载到此表。

- WITH PARTITIONS
  指导 EsgynDB SQL 使用来自 *source-table* 的分区。每个新表分区和其原有 *source-table* 副本一样，占据相同的空间。新表分区不会沿用原有表的分区名称，EsgynDB SQL 会根据物理文件位置生成新的名称。
  如果指定 LIKE 从句和 SALT USING *num* PARITIONS 从句，则不能指定 WITH PARTITIONS 从句。

- ATTRIBUTES(synchronous | no) replication]
  启用表的同步复制或禁用表的复制。该属性可以与高级版 EsygnDB 的跨数据中心复制功能结合使用。如果该表包含索引，则这些索引也将被自动复制。默认情况下不会复制表。

- *attribute stored desc*
  创建表时直接带有 stored desc 属性，同时生成 descriptor。

- *attribute incremental backup*
  创建表时直接带有 incremental backup 属性。

2.18.2 CREATE TABLE 注意事项

2.18.2.1 必需的权限

发出一条 CREATE TABLE 语句，必须满足以下条件之一：

- 您是 DB__ROOT（数据库根用户）。
- 您正在一个共享 schema 中创建表。
- 您是私有 schema 的所有者。
- 您拥有 SQL_OPERATIONS 组件的 CREATE 或 CREATE_TABLE 权限。

注意：在这种情况下，如果您在私有 schema 中创建函数，您必须是该 schema 的所有者。

2.18.2.2 创建引用完整性约束所需权限

创建一个引用完整性约束（一个表约束，它引用另一个表的某列），必须满足下列条件之一：

- 您是 DB__ROOT（数据库根用户）。
- 您是引用表和被引用表的所有者。
- 您对于引用表和被引用表拥有以下权限：

- 对于引用表而言，您拥有 SQL_OPERATIONS 组件的 CREATE 或 CREATE_TABLE 权限。
- 对于被引用表而言，您可以通过用户名或授予的角色拥有该表的 REFERENCES（或 ALL）权限。

如果该约束在查询表达式里引用另一个表，那么您必须拥有此表的 SELECT 权限。

2.18.2.3 CREATE VOLATILE TABLE 注意事项

- 临时表与会话紧密相联。它们的命名空间在多个并发会话中是唯一的，因此允许多个会话同时使用相同的临时表名，且不会造成任何冲突。
- 临时表支持创建索引。
- 临时表由系统进行分区。默认情况下分区数量限制为四个，这些分区将跨集群分布。不管系统配置如何，默认值都是四个分区。
- 临时表的统计信息不会自动更新。如果需要统计信息，您必须显式地运行 UPDATE STATISTICS 语句。
- 可以使用一部分、两部分或三部分名称创建和访问可变表。但是，必须为临时表上创建的任何 DDL 或 DML 语句使用相同的（一部分、两部分或三部分）名称。更多信息，请参阅 CREATE TABLE 例子。
- EsgynDB SQL 允许用户在包含空值的列上显式地指定主键和 STORE BY 子句。
- EsgynDB SQL 并不要求临时表的首列必须包含非空值、且必须为主键。相反，EsgynDB SQL 会尝试对表进行分区；尽量使用合适的键列作为主键和分区键。
- EsgynDB2.8 更改了临时表机制，数据库在做初始化（INITIALIZE）时会创建一个专门的 schema（_TEMPORARY_）用来存放临时表，该机制使用以下两个 CQD 控制：

CQD	说明	默认值
VOLATILE_TABLE_VERSION2	在新老临时表机制之间切换，ON 代表使用新机制	OFF
VOLATILE_TABLE_ON_SMALL_TABLE	控制新的临时表是否是基于小表创建的，ON 代表新版	OFF

	建的表是小表，否则就是普通的小表	

2.18.2.4 CREATE VOLATILE TABLE 使用限制

- 临时表不支持以下情况:
  - ALTER 语句
  - User 约束
  - 创建视图
  - 在一个临时表上创建非可变索引或在一个非可变表上创建可变索引
  - CREATE TABLE LIKE 语句操作

- EsgynDB SQL 如何支持临时表的可空键?
  - 允许 PRIMARY KEY、STORE BY 和 UNIQUE 约束里出现可空键
  - 空值被视为该列的最高值
  - 一个空值等同于其他空值，该列仅允许出现一个值
- EsgynDB SQL 如何为临时表选择合适的键?

  EsgynDB SQL 在创建的表的列中搜索首个合适的列。一旦发现合适的列，将基于此列对该表进行分区。表里搜素到的列可以（在 CREATE TABLE 语句里）显式地指定或（在 CREATE TABLE AS SELECT 语句里） 隐式地创建。只有在语句里未指定 PRIMARY KEY 或 STORE BY 子句时，才可以选择合适的键列。如果已经指定任何此类子句，则可以使用它们选择键列。

  EsgynDB SQL 遵循这些准则搜索和选择合适的键:

  - Δ 合适的列可以是一个可空列。
  - Δ EsgynDB SQL 里的某些数据类型不可被用作一个分区键。目前，这包括任何浮点列（REAL、DOUBLE PRECISION 和 FLOAT）。
  - Δ EsgynDB SQL 可以基于预定义的顺序搜索一个合适的列:
    - 应首先选择 Numeric 列、进而选择固定的 CHAR、DATETIME、INTERVAL 和 VARCHAR 数据类型。
    - Numeric 数据类型内部的选择顺序为二进制 NUMERIC（LARGEINT、INTEGER、SMALLINT）和 DECIMAL。
    - 无符号列的优先级高于有符号列。
    - 非空列的优先级高于可空列。

- 如果所有数据类型都相同，则应选择首列。

Δ 如果未发现合适的列，则可变表将变为一个非分区表，系统定义的 SYSKEY 将作为其主键；

Δ 如果已发现一个合适的列，则该列将成为分区键，其中主键为 suitable_column、SYSKEY。 这将导致该表进行分区，同时能防止出现重复键和空到非空的错误。

当 EsgynDB SQL 搜索一个合适的键时，Table 1 显示从低到高的数据类型优先顺序。后来出现的数据类型的优先级高于先前出现的数据类型的优先级。未在 Table 里出现的数据类型不能被选作键列。

在合适的键搜索期间内排列数据类型的优先级
VARCHAR
INTERVAL
DATETIME
CHAR(ACTER)
DECIMAL (signed, unsigned)
SMALLINT (signed, unsigned)
INTEGER (signed, unsigned)
LARGEINT (signed only)

- 在一个临时表里创建可空约束

  以下例子为如何在一个临时表里创建可空约束（primary key、STORE BY 和 unique）：

```
create volatile table t (a int, primary key(a));
create volatile table t (a int, store by primary key);
create volatile table t (a int unique);
```

- 创建一个带有可空主键的可变表

  以下例子创建了一个带有可空主键的临时表：

```
>>create volatile table t (a int, primary key(a));
--- SQL operation complete.
```

  仅允许出现唯一空值：

```
>>insert into t values (null);

--- 1 row(s) inserted.
>>insert into t values (null);

*** ERROR[8102] The operation is prevented by a unique
```

```
constraint.

--- 0 row(s) inserted.
```

- 为临时表搜索合适的键的例子

  以下例子为 EsgynDB SQL 选择合适的键时所遵循的顺序，这些优先级规则在“EsgynDB SQL 如何为临时表搜索合适的键”中进行了描述。

  ✓ 选择一列作为主键和分区键：
  ```
  create volatile table t (a int);
  ```

  ✓ 选择 b 列，因为 int 的优先级高于 char：
  ```
  create volatile table t (a char(10), b int);
  ```

  ✓ 选择 b 列，因为非空列的优先级高于可空列。
  ```
  create volatile table t (a int, b int not null);
  ```

  ✓ 选择 b 列，因为 int 的优先级高于 decimal：
  ```
  create volatile table t (a decimal(10), b int);
  ```

  ✓ 选择首列，因为两个列具有相同的数据类型：
  ```
  create volatile table t (a int not null, b int not
  null);
  ```

  ✓ 选择 b 列，因为 char 的优先级高于 date：
  ```
  create volatile table t (a date, b char(10));
  ```

  ✓ 选择 b 列，因为 real 数据类型并非待检查的列的一部分：
  ```
  create volatile table t (a real, b date);
  ```

  ✓ 不得选择任何列作为主键/分区键。SYSKEY 将被自动使用。
  ```
  create volatile table t (a real, b double precision
  not null);
  ```

类似的例子将用于 CREATE TABLE AS SELECT 查询。

2.18.2.5 CREATE TABLE ...LIKE 注意事项

CREATE TABLE LIKE 语句不能基于源表创建新表的视图、所有者信息或权限。使用 LIKE 规范创建的新表的相关权限都已被定义，仿佛该新表是由当前用户显式地创建。

2.18.2.6 CREATE TABLE ...LIKE 和文件属性

CREATE TABLE ... LIKE 语句创建一个类似于其他表（文件属性除外）的表。文件属性包括 COMPRESSION 等。

如果您未将属性值列入 CREATE TABLE ... LIKE 命令的一部分，则 SQL 创建的表具有属性默认值、而非源对象的数值。例如：为使创建的表类似于其他指定压缩率的表，您必须在 CREATE TABLE ... LIKE 语句里指定压缩率属性值。

在以下例子中，原 CREATE TABLE 语句创建一个未经压缩的表。然而，CREATE TABLE ... LIKE 语句里指定了压缩率。

```
-- Original Table
create table NPTEST
(FIRST_NAME CHAR(12) CHARACTER SET ISO88591 COLLATE DEFAULT
NO DEFAULT
NOT NULL
, LAST_NAME CHAR(24) CHARACTER SET ISO88591 COLLATE DEFAULT
NO DEFAULT
NOT NULL
, ADDRESS CHAR(128) CHARACTER SET ISO88591 COLLATE DEFAULT
DEFAULT NULL
, ZIP INT DEFAULT 0
, PHONE CHAR(10) CHARACTER SET ISO88591 COLLATE DEFAULT
DEFAULT NULL ,
SSN LARGEINT NO DEFAULT NOT NULL
, INFO1 CHAR(128) CHARACTER SET ISO88591 COLLATE
DEFAULT DEFAULT NULL , INFO2 CHAR(128) CHARACTER SET
ISO88591 COLLATE
DEFAULT DEFAULT NULL , primary key
(SSN,first_name,last_name)
)

max table size

-- CREATE TABLE LIKE

create table LSCE002 like NPTEST ATTRIBUTE compression type
hardware;
```

2.18.2.7 CREATE TABLE AS 语句注意事项

以下注意事项适用于 CREATE TABLE AS 语句：

- 访问一个由 CREATE TABLE AS 语句构建的表时，需要进行全表扫描，因为主键和聚集键无法轻易定义。
- 不能为 CREATE TABLE AS 表生成编译时间估值和运行时信息。

- 您无法使用 WMS 编译时和运行时规则以管理 CREATE TABLE AS 表。
- 如果没有显式地定义 CREATE TABLE 语句里的所有列，则无法指定 CREATE TABLE AS 表的主键。
- 您无法为一条 CREATE TABLE AS ... INSERT/SELECT 语句生成解释计划。但如果使用 NO LOAD 选项时，您可以为一条 CREATE TABLE AS ... INSERT/SELECT 语句使用解释计划。
- 您无法在一条 CREATE TABLE AS 语句里使用 ORDER BY 子句。编译器会对所选的行进行透明排序，以此提高插入操作的效率。

2.18.2.8 CREATE TABLE AS 语句的 LOAD IF EXISTS 和 NO LOAD 选项的注意事项

CREATE TABLE AS 语句的 LOAD IF EXISTS 选项使数据装载到一个现有的表里。

如果未指定 LOAD IF EXISTS 选项、并尝试将数据装载到一个现有的表里，则 CREATE TABLE AS 语句将执行失败。

以下情况可使用带有 AS 子句的 LOAD IF EXISTS 选项：

- 在没有重新建表的情况下运行 CREATE TABLE AS 语句。该表必须为空。否则，CREATE TABLE AS 语句将返回一个错误。发出 CREATE TABLE AS 语句之前，通过使用 DELETE 语句删除表里的数据。
- 使用 CREATE TABLE AS 语句将数据逐步添加到一个现有的表里。发出 CREATE TABLE AS 语句之前，您必须启动一个用户定义的事务。如果未启动用户定义的事务而尝试执行 CREATE TABLE AS 语句，则将返回一个错误，指出该数据已经存在于表里。借助于用户定义的事务，出现错误时，新添加的行都将被回滚。

运行带有 NO LOAD 选项的 CREATE TABLE AS 语句可以创建一个表，但不会将数据装载到该表里。

如果必须创建一个表以审查其结构、并使用 EXPLAIN 语句分析 CREATE TABLE AS 语句的 SELECT 部分，则该选项是有用的。您也可以使用 EXPLAIN 语句分析 CREATE TABLE AS ...NO LOAD 语句里所涉及的 INSERT/SELECT 部分。

例如：`CREATE TABLE ttgt NO LOAD AS (SELECT ...);`

2.18.3 CREATE TABLE 的 EsgynDB SQL 扩展语句

该语句为 ANSI SQL:1999 入门级标准所支持。CREATE TABLE 语句的 EsgynDB SQL 扩展语句为 ASCENDING、DESCENDING 和 PARTITION 子句。

CREATE TABLE LIKE 语句也是扩展语句。

2.18.4 CREATE TABLE 例子

- 该例子创建了一个表。该表的聚集键是其主键。

```
CREATE TABLE SALES.ODETAIL
( ordernum NUMERIC (6) UNSIGNED NO DEFAULT NOT NULL,
  partnum NUMERIC (4) UNSIGNED NO DEFAULT NOT NULL,
  unit_price NUMERIC (8,2) NO DEFAULT NOT NULL,
  qty_ordered NUMERIC (5) UNSIGNED NO DEFAULT NOT NULL,
  PRIMARY KEY (ordernum, partnum)  );
```

- 该例子创建了一个“加盐”表，它所拥有的尺寸相同的区域可以从 4 个增长到 8、12 或 24 个。

```
CREATE TABLE SALES.ODETAIL_SALT
( ordernum NUMERIC (6) UNSIGNED NO DEFAULT NOT NULL,
  partnum NUMERIC (4) UNSIGNED NO DEFAULT NOT NULL,
  unit_price NUMERIC (8,2) NO DEFAULT NOT NULL,
  qty_ordered NUMERIC (5) UNSIGNED NO DEFAULT NOT NULL
  PRIMARY KEY (ordernum, partnum) )
  SALT USING 24 PARTITIONS IN 4 REGIONS ON (ordernum);
```

- 该例子创建了一个带有 STORE BY 子句、而非主键的表。该表被预分割为 4 个 HBase 区域:

```
CREATE TABLE SALES.ODETAIL_SB
  ( ordernum NUMERIC (6) UNSIGNED NO DEFAULT NOT NULL,
  partnum NUMERIC (4) UNSIGNED NO DEFAULT NOT NULL,
  unit_price NUMERIC (8,2) NO DEFAULT NOT NULL,
  qty_ordered NUMERIC (5) UNSIGNED NO DEFAULT NOT NULL)
STORE BY (ordernum, partnum DESC)
SPLIT BY (ordernum, partnum)
  ( ADD FIRST KEY (1000, 800),
  ADD FIRST KEY (1000, 400),
  ADD FIRST KEY (2000, 800) );
```

- 该例子创建了一个表，它类似于 JOB 表、并具有相同的约束:

```
CREATE TABLE PERSNL.JOB_CORPORATE LIKE PERSNL.JOB WITH
CONSTRAINTS;
```

- 这是一个 NOT CASESPECIFIC 的用例:

```
CREATE TABLE T (a char(10) NOT CASESPECIFIC, b char(10));
INSERT INTO T values ('a', 'A');
```

- 该例子未返回这一行。常量‘A’区分大小写，而‘a’列不区分大小写。
  ```
  SELECT * FROM T WHERE a = 'A';
  ```

- 该例子返回了这一行。双方都区分大小写。
  ```
  SELECT * FROM T WHERE a = 'A' (not casespecific);
  ```

- 该例子里返回了这一行。因为‘b’列区分大小写，所以执行大小写敏感性比较。
  ```
  SELECT * FROM T WHERE b = 'A';
  ```

- 该例子里返回了这一行。因为‘b’列区分大小写，所以执行大小写敏感性比较。
  ```
  SELECT * FROM T WHERE b = 'A' (not casespecific);
  ```

- 这是 DIVISION BY 用法的第一个例子。

  ```
  CREATE TABLE call_home_data
  (id LARGEINT NOT NULL,
  ts TIMESTAMP(6) NOT NULL,
  device_status VARCHAR(200),
  PRIMARY KEY (id, ts))
  SALT USING 16 PARTITIONS ON (id)
  DIVISION BY (date_trunc('day', ts));
  ```

- 这是 DIVISION BY 用法的第二个例子。

  ```
  CREATE TABLE sales1
  (store_id INT NOT NULL,
  item_id INT NOT NULL,
  sale_date DATE DEFAULT DATE '2000-01-01' NOT NULL,
  sale_amt NUMERIC(10,2),
  PRIMARY KEY (store_id, item_id, sale_date))
  DIVISION BY (DATEDIFF(YEAR, '2017-11-02', sale_date));
  ```

- 这是 DIVISION BY 用法的第三个例子。

  ```
  CREATE TABLE sales2
  (store_id INT NOT NULL,
  item_id INT NOT NULL,
  sale_date DATE DEFAULT DATE '2000-01-01' NOT NULL,
  sale_amt NUMERIC(10,2),
  chcol CHAR(20) NOT NULL,
  PRIMARY KEY (store_id, item_id, sale_date, chcol))
  DIVISION BY (SUBSTR(chcol, 1, 5));
  ```

- 该例子使用与 varchar 相同的 varchar2 数据类型来创建表。

```
CREATE TABLE testvarchar2(c1 varchar(32), c2
varchar2(32) upshift not casespecific);
--- SQL operation complete.
SHOWDDL testvarchar2;
CREATE TABLE TRAFODION.SEABASE.TESTVARCHAR2
(
C1 VARCHAR(32) CHARACTER SET ISO88591 COLLATE
DEFAULT DEFAULT NULL NOT SERIALIZED
, C2 VARCHAR(32) CHARACTER SET ISO88591 COLLATE
DEFAULT UPSHIFT DEFAULT NULL NOT SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT
;
--- SQL operation complete.
```

- NATIONAL CHAR (或 NCHAR) 是 CHARACTER SET UCS2 的缩写。因此，在使用 NCHAR 时无法指定 CHARACTER SET 子句，如以下例子所示。

注意： 无论字符集在安装时被指定为 ISO88591 还是 UTF8，NCHAR 或 NATIONAL CHAR 都隐式地假设字符集是 UCS2，并且 NCHAR 或 NATIONAL CHAR 的字符集（UCS2）不能被指定或更改。 对于 ALTER 和 CREATE 语句都是如此。

```
SQL>CREATE TABLE test1 (c1 NCHAR(10) CHARACTER SET USC2);
*** ERROR[15001] A syntax error occurred at or before:
create table test1 (c1 nchar(10) character set usc2);
^ (40 characters from start of SQL statement)
[2018-07-13 04:02:39]
```

- 该例子创建了一个表，该表带有 stored desc 属性

```
SQL>create table table1(a int) attribute stored desc;

--- SQL operation complete.

SQL>showddl table1;


CREATE TABLE TRAFODION.SEABASE.TABLE1
  (
    A                                INT DEFAULT NULL
NOT SERIALIZED
  )
 ATTRIBUTES ALIGNED FORMAT NAMESPACE 'TRAF_RSRVD_3'
STORED DESC
;
```

```
-- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES, ALTER, DROP ON TRAFODION.SEABASE.TABLE1 TO DB__ROOT WITH GRANT OPTION;

--- SQL operation complete.
```

- 该例子创建了一个表，该表带有 incremental backup 属性

```
SQL>create table table1(a int) attribute incremental backup;

--- SQL operation complete.
```

- 该例子使用 ATTRIBUTE NAMESAPCE 语法

```
SQL>create table t1(c1 INT PRIMARY KEY NOT NULL,C2 INT) ATTRIBUTE NAMESPACE 'TRAF_1500000';

--- SQL operation complete.
```

- 该例子使用 RANGE SPLIT BY 语法

```
SQL>create table t1(c1 INT PRIMARY KEY NOT NULL,C2 INT) RANGE SPLIT BY (c1) (ADD FIRST KEY (100));

--- SQL operation complete.
```

2.18.4.1 CREATE TABLE AS 语句例子

该显示了列属性规则，用于生成和指定创建的表的列的名称和数据类型。

- 如果没有指定列属性，则 select 查询里的选择列表项将被用来生成已创建表的列名和数据属性。

  如果选择列表项是一列，则它将被用作已创建的列名。例如：

  ```
  create table t as select a,b from t1
  ```

  t 表带有两个名为（a,b）的列、以及与 t1 表里的列相同的数据属性。

- 如果选择列表项是一个表达式，则必须使用 AS 子句将其重命名。如果表达式没有命名，则将返回一个错误。例如：

  ```
  create table t as select a+1 as c from t1
  ```

  t 表带有一个名为 (c) 的列和(a+1)数据属性。

  ```
  create table t as select a+1 from t1
  ```

  返回一个错误后，表达式必须重命名。

- 如果已指定的列属性包含数据类型信息，则它们将覆盖选择查询里的选择项的属性。这些数据属性必须与 select 查询里的选择列表项相应的数据属性相兼容。

  ```
  create table t(a int) as select b from t1
  ```

  t 表带有一个名为“a”的列，其数据类型为“int”。

  ```
  create table t(a char(10)) as select a+1 b from t1;
  ```

  这将会返回一个错误，因为列“a”的数据属性-char（字符型）并不匹配选择列表项“b”的数据属性-numeric（数字型）。

- 如果已指定的列属性仅包含列名，指定的列名将覆盖 select 查询里派生的任何名称。

  ```
  create table t(c,d) as select a,b from t1
  ```

  表 t 带有两个名为 c 和 d 的列，并具有表 t1 的列 a 和 b 的数据属性。

- 已指定的列属性必须包含 select 查询里的所有选择列表项相应的属性。

  如果存在不匹配的情况，则将返回一个错误。

  ```
  create table t(a int) as select b,c from t1
  ```

  返回了一个错误。这两项需要被指定作为部分表属性。

  列属性必须为所有列指定列名和数据类型信息对、或仅指定列名。您不能仅使用名称和数据类型指定一些列。

  ```
  create table t(a int, b) as select c,d from t1
  ```

  返回了一个错误。

- 以下例子创建了 t1 表。使用 CREATE TABLE AS 语法创建的 t2 表不含表属性。

  ```
  CREATE TABLE t1 (c1 int not null primary key,
  c2 char(50));

  CREATE TABLE t2 (c1 int, c2 char (50) UPSHIFT NOT NULL)
  AS SELECT * FROM t1;
  ```

2.18.4.2 CREATE TABLE LIKE 例子

- 该例子演示了如何在不使用 LOB 列 c3 和 c4 的情况下使用 CREATE TABLE LIKE 语句。

  ```
  CREATE TABLE t1 (c1 INT, c2 CHAR(99), c3 BLOB, c4
  ```

```
CLOB(9999));
--- SQL operation complete.

CREATE TABLE t2 LIKE t1 WITHOUT LOB COLUMNS;
--- SQL operation complete.

SHOWDDL t2;

CREATE TABLE TRAFODION.SEABASE.T2
(
C1 INT DEFAULT NULL NOT SERIALIZED
, C2 CHAR(99) CHARACTER SET ISO88591 COLLATE
DEFAULT DEFAULT NULL NOT SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT
;
-- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON
TRAFODION.SEABASE.T2 TO DB__ROOT WITH GRANT OPTION;

--- SQL operation complete.
```

- 该例子演示如何使用 WITH NO DATA。

```
SELECT * FROM source_table;
C1   C2
---------
 1    1

 --- 1 row(s) selected.
 CREATE TABLE t1 LIKE source_table WITH DATA;
 --- SQL operation complete.
 SELECT * FROM t1;
 C1   C2
 ---------
  1    1

 --- 1 row(s) selected.

 CREATE TABLE t2 LIKE source_table WITH NO DATA;
 --- SQL operation complete.

 SELECT * FROM t2;
 --- 0 row(s) selected.
```

2.18.4.3 CREATE EXTERNAL TABLE 例子

该例子比较了在读取 hive 表时使用外部表和不使用外部表的执行时间。

前者花费的时间较少，因为 EsgynDB 外部表为 varchar 长度设置了上限，这可能产生更好的计划和/或运行时间行为。

注意：运行 UPDATE STATISTICS 或使用 EsgynDB 外部表可以提高性能。要获得完整的性能优势，必须运行 UPDATE STATISTICS 并使用 EsgynDB 外部表。

这是 **hive 表** *test_mix* 的定义，它有一个 EsgynDB 外部表，大小为 137.6G。

```
SQL>SHOWDDL test_mix;
CREATE EXTERNAL TABLE test_mix(
   mix_id int not null not droppable primary key,
   mix_age int,
   mix_name string,
   mix_timestamp01 string,
   mix_timestamp02 string,
   mix_other01 string,
   mix_other02 string,
   mix_other03 string,
   mix_other04 string,
   mix_other05 string,
   mix_other06 string,
   mix_other07 string,
   mix_other08 string,
   mix_other09 string,
   mix_other10 string,
   mix_other11 string,
   mix_other12 string,
   mix_other13 string,
   mix_other14 string,
   mix_other15 string
  )
row format delimited fields terminated by '|'
location '/user/trafodion/data/ExternalTable_data';
-- 01-06 short 06-11medium 12-15 long
```

这是 **EsgynDB 外部表** *test_mix* 的定义，它具有与 hive 表 *test_mix* 相同的结构和大小。

```
SQL>SHOWDDL text_mix;
CREATE EXTERNAL TABLE test_mix(
   mix_id int,
   mix_age int,
   mix_name varchar(20),
   mix_timestamp01 timestamp,
   mix_timestamp02 varchar(20),
   mix_other01 varchar(12),
   mix_other02 varchar(12),
   mix_other03 varchar(12),
   mix_other04 varchar(12),
   mix_other05 varchar(12),
   mix_other06 varchar(12),
```

```
    mix_other07 varchar(64),
    mix_other08 varchar(64),
    mix_other09 varchar(64),
    mix_other10 varchar(64),
    mix_other11 varchar(128),
    mix_other12 varchar(128),
    mix_other13 varchar(128),
    mix_other14 varchar(1024),
    mix_other15 varchar(1024)
  )for hive.hive.test_mix;
-- 01-06 short 07-11medium 12-15 long
```

- 执行以下查询时:

```
SELECT [LAST 1] * FROM hive.hive.test_mix WHERE mix_other02
= 'Ot';
```

使用 EsgynDB 外部表获取结果大约需要 **6 分钟**（平均值）。

	第一次结果	第二次结果	第三次结果	平均值
开始时间	2018/03/07 18:40:31.655159	2018/03/07 09:37:50.801345	2018/03/07 09:45:05.921706	
结束时间	2018/03/07 18:49:08.879780	2018/03/07 09:43:16.695492	2018/03/07 09:48:58.251764	
经过时间	2018/03/07 00:08:37.224621	2018/03/07 00:05:25.894147	2018/03/07 00:03:52.330058	00:06:12.23
编译时间	2018/03/07 00:00:03.497624	2018/03/07 00:00:11.595054	2018/03/07 00:00:00.551781	00:00:04.8
执行时间	2018/03/07 00:08:33.715742	2018/03/07 00:05:14.295840	2018/03/07 00:03:51.708673	**00:06:12**

而在不使用 EsgynDB 外部表的情况下，获取结果大约需要 **14 分钟**（平均值）。

	第一次结果	第二次结果	第三次结果	平均值
开始时间	2018/03/07 13:33:46.722646	2018/03/07 14:39:30.323730	2018/03/07 14:54:58.177258	
结束时间	2018/03/07 13:48:35.028916	2018/03/07 14:53:53.887911	2018/03/07 15:09:11.517646	
经过时间	2018/03/07 00:14:48.306270	2018/03/07 00:14:23.564181	2018/03/07 00:14:13.340388	00:14:28.40
编译时间	2018/03/07 00:00:00.773770	2018/03/07 00:00:00.388777	2018/03/07 00:00:14.856643	00:00:04
执行时间	2018/03/07 00:14:47.530017	2018/03/07 00:14:23.146420	2018/03/07 00:13:58.463850	**00:13:58**

- 执行以下查询时:

```
SELECT [LAST 1] mix_other02, substring(mix_other12 from 1
for 10) FROM
hive.hive.test_mix WHERE substring(mix_other02 from 1 for
1) = 'O';
```

使用EsgynDB外部表获取结果大约需要**6分钟**（平均值）。

	第一次结果	第二次结果	第三次结果	平均值

开始时间	2018/03/09 14:07:59.353015	2018/03/09 14:16:27.725035	2018/03/09 14:41:01.454408	
结束时间	2018/03/09 14:15:05.979546	2018/03/09 14:20:44.939776	2018/03/09 14:46:58.238246	
经过时间	2018/03/09 00:07:06.626531	2018/03/09 00:04:17.214741	2018/03/09 00:05:56.783838	00:05:59
编译时间	2018/03/09 00:00:00.197789	2018/03/09 00:00:00.296705	2018/03/09 00:00:00.227511	00:00:00.23
执行时间	2018/03/09 00:07:06.411065	2018/03/09 00:04:16.873090	2018/03/09 00:05:56.554411	**00:05:59**

而在不使用 EsgynDB 外部表的情况下，获取结果大约需要 35 分钟（平均值）。

	第一次结果	第二次结果	第三次结果	平均值
开始时间	2018/03/09 11:01:12.676307	2018/03/09 11:35:54.514479	2018/03/09 13:15:07.006658	
结束时间	2018/03/09 11:35:16.264756	2018/03/09 12:11:09.587147	2018/03/09 13:49:23.740406	
经过时间	2018/03/09 00:34:03.588449	2018/03/09 00:35:15.072668	2018/03/09 00:34:16.733748	34:44:00
编译时间	2018/03/09 00:00:00.703053	2018/03/09 00:00:00.280146	2018/03/09 00:00:00.536929	00:00:00.5
执行时间	2018/03/09 00:34:02.828529	2018/03/09 00:35:14.743914	2018/03/09 00:34:16.155336	**34:44:00**

2.19 CREATE TRIGGER 语句

该语句在数据库里创建一个触发器。更多关于触发器的信息请参阅 EsgynDB SPSQL 语言参考手册。

```
CREATE [OR REPLACE] TRIGGER
[[catalog.]schema.]trigger_name
before|after {insert|update|delete [or
{insert|update|delete} [or {insert|update|delete}]]}
on [[catalog.]schema.]table_name
for each row |statememt
as
begin
  DML SQL;
end;
```

2.19.1 CREATE TRIGGER 语法

- OR REPLACE
  指定 OR REPLACE 以重新创建触发器，如果触发器已经存在。使用此子句更改现有触发器的定义而不需要首先删除它。
- `trigger_name`

指定一个触发器的 ANSI 逻辑名称。该名称在其所属的 schema 内部的触发器名称里必须是唯一的。

- DML SQAL
  请参考 SP SQL 中的 body 部分

2.19.2 CREATE TRIGGER 注意事项

2.19.2.1 必需的权限

发出一条 CREATE TRIGGER 语句，必须满足以下条件之一：

- 您是 DB_ROOT（数据库根用户）。
- 您正在一个共享的 schema 中创建触发器，并且您对于创建触发器期间使用的库拥有 USAGE（或 ALL）权限。USAGE 权限使您能读取该库的底层 JAR 文件，其包含 SPJ Java 方法。
- 您是私有 schema 的所有者，并且您对于创建触发器期间使用的库拥有 USAGE（或 ALL）权限。USAGE 权限使您能读取该库的底层 JAR 文件，其包含 SPJ Java 方法。
- 您不但对于 SQL_OPERATIONS 组件拥有 CREATE 或 CREATE_ROUTINE 组件级权限，而且对于创建触发器期间使用的库拥有 USAGE（或 ALL）权限。USAGE 权限使您能够读取该库的底层 JAR 文件，其包含 SPJ Java 方法。

目前触发器的名字不能使用双引号。

2.19.3 CREATE TRIGGER 例子

- 该例子显示在对表 test_pk 进行 insert 和 delete 操作时，通过触发器在 test_log 表中记录时间戳和操作类型，在对 test_pk 进行 update 操作时，触发器在 test_log 表中记录时间戳和操作类型的同时，还会将 update 前后的数据记录在 test_py_log 表中。

```
create table test_pk (c1 int primary key,c2 int);
create table test_log(c1 timestamp,c2 varchar(30));
create table test_py_log (c1 int, c2 int);

create or replace trigger test_trig
before insert or update or delete
on test_pk
for each row
as
begin
if inserting then
  insert into test_log values (systimestamp,'insert');
elsif updating then
  insert into test_log values (systimestamp,'update');
```

```
    insert into test_py_log values (new.c1, new.c2);
    insert into test_py_log values (old.c1, old.c2);
  elsif deleting then
    insert into test_log values (systimestamp,'delete');
  else
    null;
  end if;
  end;
```

2.20 CREATE VIEW 语句

该语句创建一个 EsgynDB SQL 视图。更多信息，请参阅视图。

注意： 目前的事务不支持 DDL 语句。这意味着您既无法在用户定义的事务（BEGIN WORK...COMMIT WORK）中运行该语句，也无法在 AUTOCOMMIT 选项关闭时运行该语句。为了运行该语句，必须开启该会话的 AUTOCOMMIT（自动提交）选项（默认）。

```
CREATE [OR REPLACE] VIEW view
      [(column-name ] [, column-name ...)]
      AS query-expr [order-by-clause]
      [WITH CHECK OPTION]
```

2.20.1 CREATE VIEW 语法

- OR REPLACE

  创建一个尚不存在的视图或替换一个同名视图。被替换的视图可能具有相同的视图定义、也可能具有不同的视图定义。

- *view*

  指定一个视图的 ANSI 逻辑名称。更多信息，请参阅数据库对象名称。该名称在其所属的 schema 内部的表和视图名称里必须是唯一的。

- *(column-name [,column-name]...)*

  指定视图的列名。列表里的列名必须一对一地匹配 *query-expr* 子句指定的表列。

  如果省略该子句，则视图里的列将与 *query-expr* 子句里的相应列具有相同名称。

  如果 *query-expr* 子句指定的表里的任意两列都具有相同的名称或该表的任何一列都没有名称，则您必须指定该子句。例如：在该查询表达式 `SELECT MAX(salary), AVG(salary) AS average_salary FROM employee`，首列没有名称。

  一个视图的列名必须各不相同。如果一个视图引用多个表、且选择列表引用了名称相同但不同表的列，则必须指定新的列名，否则将造成重名。

- AS *query-expr*

  指定视图里的列，并设置选择标准，用于确定构成该视图的行。

  更多关于字符串字面值的信息，请参阅字符串字面值。

  更多关于 query-expr 的语法，请参阅 SELECT 语句。

CREATE VIEW 语句提供了 *query-expr* 语法的相关使用限制：视图里不得出现[ANY N]、[FIRST N]选择列表项。

- *order-by-clause*
  指定最终结果表行的排序。更多关于 order-by-clause 的语法和语法描述，请参阅 SELECT 语句。
  CREATE VIEW 语句限制与 access-clause 和 mode-clause 相关的 *order-by-clause* 的使用。
  access-mode 和 *mode-clause* 不能尾随 *order-by-clause*。

- WITH CHECK OPTION
  指定不可通过视图将行插入或更新到数据库里，除非该行满足视图定义，即查询表达式里的 WHERE 子句必须经评估适用于任何已插入或更新的行。该选项仅适用于可更新视图。
  如果您省略该选项，则新插入的行或更新后的行无需满足视图定义，即一行可以被插入进表里或在表里进行更新、但不会出现在视图里。
  每次插入或更新一行的时候都要执行检查。WITH CHECK OPTION 不会影响查询表达式。行必须始终满足视图定义。

2.20.2 CREATE VIEW 注意事项

- 如果您指定 CREATE OR REPLACE VIEW：
  - 如果不存在同名视图，则将创建一个新视图。
  - 如果存在同名视图，则将删除旧的视图定义、并使用新定义创建视图。如果新视图和待替换的视图相同，则无需进行检查。CREATE OR REPLACE VIEW 命令可以单方面删除旧的视图定义，并使用新视图定义替换它。
  - 已授予的旧视图权限将被重新授予给新视图。如果重新授权失败，则 CREATE OR REPLACE VIEW 操作也将执行失败。
  - 当 CREATE OR REPLACE VIEW 替换一个现有的视图时，任何相关的视图都将被删除。
- 您可以指定 GROUP BY，并使用序号引用 SELECT 列表内的相关位置。例如：GROUP BY 3, 2, 1。
- 禁止使用动态参数。

2.20.2.1 在视图定义中添加一列的效果

在表里添加一列对于现有的视图定义或约束定义里的条件没有任何影响。
最初评估定义子句时，视图或约束定义里由 SELECT *指定的任何隐式列引用都将被替换为显式列引用。

2.20.2.2 授权和可用性要求

发出一条 CREATE VIEW 语句，您必须对视图底层对象拥有 SELECT 权限或为视图底层对象的所有者，并且满足以下一种条件：

- 您是 DB_ROOT（数据库根用户）。
- 您正在一个共享 schema 里创建视图。
- 您是私有 schema 的所有者。
- 您拥有 SQL_OPERATIONS 组件的 CREATE 或 CREATE_VIEW 权限。

注意：在这种情况下，如果您在私有 schema 中创建函数，您必须是该 schema 的所有者。

当您在单个表上创建视图时，该视图的所有者将被自动赋予该视图含 WITH GRANT OPTION 的所有权限；然而，当您在多个表上创建视图时，该视图的所有者将仅被赋予该视图含 WITH GRANT OPTION 的 SELECT 权限。如果您尝试将该视图上的非 SELECT 权限授予其他用户，则会收到一个警告，表明您没有该权限的授予选项。

2.20.2.3 可更新和不可更新视图

- 单表视图可以更新。多表视图不可更新。
- 为了定义一个可更新视图，查询表达式还必须满足以下条件：
  - 它不能包含 JOIN、UNION 或 EXCEPT 子句。
  - 它不能包含 GROUP BY 或 HAVING 子句。
  - 它不能直接包含关键字 DISTINCT。
  - FROM 子句必须只能引用一个表或者一个可更新视图。
  - 它不能包含一个含有子查询的 WHERE 子句。
  - 选择列表不能包含表达式、函数或重复列名。

2.20.2.4 ORDER BY 子句准则

CREATE VIEW 语句的 SELECT 部分可以指定 ORDER BY 子句。
单独指定 SELECT 部分时，任何有效的 SELECT 语法在视图定义期间仍然有效。一个 ORDER BY 从句既可以包含 SELECT 列表里的列名、也可以包含 *select-list-index* 里的列名。

如果根据视图发出一条 DML 语句，则以下所记录的规则将被用来应用 ORDER BY 子句。

- **何时使用 ORDER BY 子句**

  一条 ORDER BY 子句位于使用该视图的 Select 查询的根节点时，才可在视图定义里使用该子句。如果一条 ORDER BY 子句出现在其他的中间位置或子查询中，它将被忽略。

  考虑这条 CREATE VIEW 语句：

  ```
  create view v as select a from t order by a;
  select * from v x, v y;
  ```

  或该 INSERT 语句：

  ```
  insert into t1 select * from v;
  ```

  在这两个例子中，DML 语句处理期间忽略了 ORDER BY 子句，因为第一个例子作为派生表的一部分出现，第二个例子则作为 select 子查询出现，二者都在视图扩展之后创建。如果使用显式派生表（不使用视图）发出同样的查询，则将返回语法错误：

  ```
  select * from (select a from t order by a) x, (select a from t order by a) y;
  ```

  该例子将返回语法错误，因为子查询里并不支持 ORDER BY 子句。如果 ORDER BY 子句是视图的一部分或 ORDER BY 子句被用在未获支持的场合里，则它将被忽略。这不同于使用显式 ORDER BY 子句编写相同的查询时返回的错误（如前例所示）。

- **含 No Override 选项的视图定义里的 ORDER BY 子句**

  如果一条 SELECT 查询从不含显式 ORDER BY override 的视图里读取数据，则视图定义里的 ORDER BY 语义将被使用。

  在以下例子中，CREATE VIEW 语句里指定了排序列。

  ```
  create view v as select * from t order by a Select * from v
  ```

  该 SELECT 查询等效于：

  ```
  select * from t order by a;
  ```

- **含 User Override 选项的视图定义里的 ORDER BY 子句**

如果一条 SELECT 查询包含显式 ORDER BY 子句，它将覆盖视图定义里指定的 ORDER BY 子句。

例如：

```
create view v as select a,b from t order by a;
select * from v order by b;
```

在此例子中，order by b 子句将覆盖视图定义里指定的 order by a。该 SELECT 查询等效于：

```
select a,b from t order by b;
```

2.20.2.5 嵌套视图的定义

在嵌套视图定义的情况下，最顶层视图定义里的 ORDER BY 子句将覆盖任何嵌套视图定义的 ORDER BY 子句。

例如：

```
create view v1 as select a,b from t1 order by a;
create view v2 as select a,b from v1 order by b;
select * from v2;
```

在该例子中，视图 v2 定义里指定的 ORDER BY 子句将覆盖视图 v1 定义里指定的 ORDER BY 子句。

该 SELECT 查询等效于：

```
select a,b from (select a, b from t) x order by b;
```

2.20.3 CREATE VIEW 例子

- 该例子在不含“视图列列表”的单个表上创建了一个视图：

  ```
  CREATE VIEW SALES.MYVIEW1 AS
  SELECT ordernum, qty_ordered FROM SALES.ODETAIL;
  ```

- 该例子使用不同的视图定义替换了 MYVIEW1 视图：

  ```
  CREATE OR REPLACE VIEW SALES.MYVIEW1 AS
     SELECT ordernum, qty_ordered FROM SALES.ODETAIL
     WHERE unit_price > 100;
  ```

- 该例子使用一个““列”列表”创建了一个视图：

  ```
  CREATE VIEW SALES.MYVIEW2
  (v_ordernum, t_partnum) AS
  SELECT v.ordernum, t.partnum
  FROM SALES.MYVIEW1 v, SALES.ODETAIL t;
  ```

- 该例子使用 INNER JOIN 从两个表中创建了一个视图：

  ```
  CREATE VIEW MYVIEW4
  ```

```
(v_ordernum, v_partnum) AS
SELECT od.ordernum, p.partnum
     FROM SALES.ODETAIL OD INNER JOIN SALES.PARTS P
     ON od.partnum = p.partnum;
```

2.20.3.1 垂直分区例子

该例子为一个表创建了三个逻辑垂直分: vp0、vp1 和 vp2，然后创建了一个 vp 视图，用于访问它们。

一个视图可被用来获取一组密切相关的表的复合表达。以下例子中的各表——vp0、vp1 和 vp2 都有一个键列 a。该键列包含这三个表相同的行，且这三个表分别包含 b 列、c 列和 d 列。可以创建一个 vp 视图，将这三个表组合在一起，并为归属一个对象的 a 列、b 列、c 列和 d 列提供了接口。

EsgynDB SQL 能消除一个查询里的冗余联接。冗余联接在以下情况下出现:

- 一个联接的输出仅包含其两个子联接之一的表达式。
- 该子联接的各行都仅匹配另一个子联接的一行。

假设 A 表和 B 表都表示通用表。为了核对规则“该子联接的各行都仅匹配另一个子联接的一行”是否属实，EsgynDB SQL 使用下列事实作为依据——A 表或 B 子查询的联接将保留 A 表的所有行，如果联接谓词包含一个引用 B 键的等值联接谓词、并且以下情况之一属实: 该联接是左外联接，其中 B 是内部表。

在该例子中，联接 vp0 和 vp1 时，vp0 会填补 A 表的角色，vp1 会填补 B 表的角色。联接 vp1 和 vp2 时，vp1 会填补 A 表的角色，vp2 会填补 B 表的角色。

该例子中所示的 vp 视图使用左外联接将三个底层表组合在一起。因此，如果访问 vp 的查询里的选择列表不含 vp2 里的 d 列，则不会在 vp 视图里联接表 vp2。

```
create table vp0(a integer not null, b integer, primary
key(a));
create table vp1(a integer not null, c integer, primary
key(a));
create table vp2(a integer not null, d integer, primary
key(a));

create view vp(a,b,c,d) as
select vp0.a, b, c, d
from vp0 left outer join vp1 on vp0.a=vp1.a
left outer join vp2 on vp0.a=vp2.a;
```

```
select a, b from vp; -- reads only vp0
select a, c from vp; -- reads vp0 and vp1
select d from vp; -- reads vp0 and vp2
```

2.21 DELETE 语句

该语句是一种 DML 语句，它从一个表或一个可更新视图中删除一行或多行。

从一个视图里删除行后，该视图的基表里也将删除相应的行。DELETE 语句不能删除一个表或视图，即便您已经删除表或视图里的最后一行也是如此。

EsgynDB SQL 提供搜索到的 DELETE 语句——删除那些满足搜索条件的行。

对于搜索到的 DELETE 表单，如果不存在 WHERE 子句，则将从表或视图里删除所有行。

搜索到的 DELETE 语句为:

```
DELETE
     [ WITH NO ROLLBACK ] [“[“ FIRST N “]”]
     FROM table
     [ t_alias ]
     [ WHERE search-condition ]
     [[FOR] access-option ACCESS ]

access-option is:
     READ COMMITTED
```

2.21.1 DELETE 语法

- *Table-name*

  对删除行所在的用户表或视图进行命名。该表必须是一个基表或一个可更新视图。为了引用一个表或视图，请使用 ANSI 逻辑名称。

- WITH NO ROLLBACK

  非事务性删除。只发生在事务外部。这意味着如果语句被中断，任何已完成的删除仍然是永久性的。

  注意：只有在表里不含索引的情况下,该选项才会得到支持。

- FIRST *N*

  指定待删除的 N 行(假设该表至少有 N 行,并且 WHERE 子句里指定了限定标准。如果存在的话，将至少选择 N 行。)

  您必须用方括号（[]）将 FIRST N 括起来。语法图里每个方括号周围的引号（""）表示该括号是您必须键入的字符，如图所示（例如：[FIRST 5]）。

  不得在 FIRST 子句里加入引号。

注意：在没有 WITH NO ROLLBACK 选项时使用改选项时，如果您有一个较大的 N（数以万计），请将 N 拆分成更小的数字、重复执行 DELETE 语句，以获得更好的性能。

- `t_alias`
  为要在语句中的其他位置引用的表，视图，实例化视图，子查询或集合值提供相关别名。

- WHERE `search-condition`
  指定一个搜索条件，它选择需要删除的行。在搜索条件内部，任何被比较的列都是需要从表或视图中删除的列。更多信息，请参阅搜索条件。
  如果不指定一个搜索条件，则表或视图里的所有行都将被删除。

- [FOR] `access-option` ACCESS
  指定数据访问选项，这些数据用于评估搜索条件。更多信息，请参阅数据一致性和访问选项。

- READ COMMITTED
  指定用于评估搜索条件的任何数据都必须来自已经提交的行。
  默认访问选项是所包含事务的隔离级别。

2.21.2 DELETE 注意事项

2.21.2.1 授权要求

执行 DELETE 语句需要有权对需要进行删除操作的表或视图进行读写操作，并有权读取搜索条件里所用的子查询指定的表或视图。

2.21.2.2 发起和终止事务

除非制定了 WITH NO ROLLBACK，如果没有处于活动状态的事务，则 DELETE 语句将会自动发起一个事务。否则，可以使用 BEGIN WORK 语句显式地发起一个事务。
一个事务启动后，SQL 语句将在该事务内部执行，直到遇到 COMMIT 或 ROLLBACK、或出现错误为止。

2.21.2.3 语句的事务和访问选项的隔离级别

SQL 事务隔离级别定义事务内部数据操作受并发事务影响的程度。如果您在一个事务内部指定了 DML 语句的访问选项，则其中所包含事务的隔离级别将被覆盖。每条语句都按照其各自的访问选项执行。

2.21.3 DELETE 例子

- 从 JOB 表里删除所有行:

```
DELETE FROM persnl.job;

--- 10 row(s) deleted.
```

- 从 ORDERS 表里删除任何客户通过销售代表 220 提交的订单。编号为 1234 的客户除外;

```
DELETE FROM sales.orders
WHERE salesrep = 220 AND custnum <> 1234;

--- 2 row(s) deleted.
```

- 从 PARTSUPP 表里删除所有不在德克萨斯州的供应商;

```
DELETE FROM invent.partsupp
WHERE suppnum IN
   (SELECT suppnum FROM samdbcat.invent.supplier
   WHERE state <> 'TEXAS');

--- 41 row(s) deleted.
```

- 该语句可以达到同样的结果:

```
DELETE FROM invent.partsupp
WHERE suppnum NOT IN
(SELECT suppnum FROM samdbcat.invent.supplier
WHERE state = 'TEXAS');

--- 41 row(s) deleted.
```

- 这是一个自引用的 DELETE 语句的例子，删除行的表将在子查询里被扫描:

```
DELETE FROM table1 WHERE a IN
   (SELECT a FROM table1 WHERE b > 200);
```

- 该例子执行满足 WHERE 条件的前 1000 行的非事务性删除。

```
DELETE WITH NO ROLLBACK [FIRST 1000]
FROM test
WHERE id<30;
```

- 该例子显示只有在表里不含索引时，WITH NO ROLLBACK 选项才会得到支持。

```
SQL>DELETE WITH NO ROLLBACK FROM test;
*** ERROR[3232] INSERT, UPDATE, or DELETE of table
TRAFODION.SEABASE.TEST with
dependent Index : TRAFODION.SEABASE.INDEX_TEST is not
allowed when NO ROLLBACK
transaction setting is in effect. [2018-05-30 05:42:47]
SQL>SHOWDDL test;
CREATE TABLE TRAFODION.SEABASE.TEST
(
C1 INT DEFAULT NULL NOT SERIALIZED
, C2 INT DEFAULT NULL NOT SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT
;
CREATE INDEX INDEX_TEST ON TRAFODION.SEABASE.TEST
(
C1 ASC
)
ATTRIBUTES ALIGNED FORMAT
;
-- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON
TRAFODION.SEABASE.TEST TO
DB__ROOT WITH GRANT OPTION;
--- SQL operation complete.
```

- 对 N 行执行非事务型删除，满足 WHERE 条件

```
DELETE WITH NO ROLLBACK [FIRST 160000]
   FROM mytable
   WHERE unique2<160000;
```

- 删除 N 行，满足 WHERE 条件

```
DELETE [FIRST 160000]
   FROM mytable
   WHERE unique2<160000;
```

- 支持别名

```
SQL>delete from staff a where a.salary = 10000;

--- 1 row(s) deleted.
```

2.22 DROP FUNCTION 语句

DROP FUNCTION 语句从 EsgynDB 数据库里删除一个用户定义函数（UDF）。

注意： 目前的事务不支持 DDL 语句。这意味着您既无法在用户定义的事务（BEGIN WORK...COMMIT WORK）中运行该语句，也无法在 AUTOCOMMIT 选项关闭时运行该语句。为了运行该语句，必须开启该会话的 AUTOCOMMIT（自动提交）选项（默认）。

```
DROP FUNCTION [[catalog-name.]schema-name.]function-name
```

2.22.1 DROP FUNCTION 语法

[[catalog-name.]schema-name.]function-name

指定函数的 ANSI 逻辑名称，其中每部分都是一个有效的 SQL 标识符，最长 128 个字符。

指定 schema 里已注册的函数名称。如果您没有完全限定一个函数名称，EsgynDB SQL 将根据当前会话的 schema 限定它。更多信息，请参阅标识符和数据库对象名称。

2.22.2 DROP FUNCTION 注意事项

必需的权限

发出一条 DROP FUNCTION 语句，必须满足以下条件之一：

- 您是 DB__ROOT（数据库根用户）。
- 您是该函数的所有者。
- 您拥有 SQL_OPERATIONS 组件的 DROP 或 DROP_ROUTINE 权限。

2.22.3 DROP FUNCTION 例子

- 该语句从默认 schema 中删除一个名为 ADD2 的函数。
  ```
  DROP FUNCTION add2;
  ```
- 该语句从默认 schema 中删除一个名为 MMA5 的函数。
  ```
  DROP PROCEDURE mma5;
  ```
- 该语句从默认 schema 中删除一个名为 REVERSE 的函数。
  ```
  DROP PROCEDURE reverse;
  ```

2.23 DROP INDEX 语句

该语句删除一个 EsgynDB SQL 索引。更多信息，请参阅索引。

该语句是 EsgynDB SQL 扩展语句。

注意： 目前的事务不支持 DDL 语句。这意味着您既无法在用户定义的事务（BEGIN WORK...COMMIT WORK）中运行该语句，也无法在 AUTOCOMMIT 选项关闭时运行该语句。为了运行该语句，必须开启该会话的 AUTOCOMMIT（自动提交）选项（默认）。

```
DROP [VOLATILE] INDEX index
```

2.23.1 DROP INDEX 语法

`index`

待删除的索引。更多信息，请参阅数据库对象名称。

2.23.2 DROP INDEX 注意事项

2.23.2.1 必需的权限

发出一条 DROP INDEX 语句，必须满足以下条件之一：

- 您是 DB__ROOT（数据库根用户）。
- 您是索引的所有者或与索引关联的表的所有者。
- 您拥有 SQL_OPERATIONS 组件的 DROP 或 DROP_INDEX 权限。

2.23.2.2 DROP INDEX 例子

- 该例子可以删除一个索引：
  ```
  DROP INDEX myindex;
  ```
- 该例子可以删除一个临时索引：
  ```
  DROP VOLATILE INDEX vindex;
  ```

2.24 DROP LIBRARY 语句

该语句从 EsgynDB 数据库里删除一个库对象和此库对象引用的库文件。

DROP LIBRARY 是 EsgynDB SQL 扩展语句。

注意： 目前的事务不支持 DDL 语句。这意味着您既无法在用户定义的事务（BEGIN WORK...COMMIT WORK）中运行该语句、也无法在 AUTOCOMMIT 选项关闭时运行该语句。为了运行该语句，必须开启该会话的 AUTOCOMMIT（自动提交）选项（默认）。

```
DROP LIBRARY [[catalog-name.]schema-name.]library-name
     [RESTRICT | CASCADE]
```

2.24.1 DROP LIBRARY 语法

[[catalog-name.]schema-name.]library-name

指定库对象的 ANSI 逻辑名称，其中每部分都是一个有效的 SQL 标识符，最长 128 个字符。

指定 schema 里已注册的库对象的名称。

如果您没有完全限定一个库名，EsgynDB SQL 将根据当前会话的 schema 限定它。

更多信息，请参阅标识符和数据库对象名称。

[RESTRICT | CASCADE]

如果您指定 RESTRICT 选项，并且所有 Java 存储过程（SPJ）或用户定义函数（UDF）基于指定的库而创建，则 DROP LIBRARY 语句操作将执行失败。

如果您指定 CASCADE 选项，则任何此类相关的存储过程或函数都将作为 DROP LIBRARY 语句操作的一部分而被删除。

默认值为 RESTRICT。

2.24.2 DROP LIBRARY 注意事项

- RESTRICT 选项要求您在删除库对象之前删除引用该对象的所有存储过程和函数。CASCADE 选项可以自动删除任何使用该库的存储过程或函数。
- 如果该库对象所引用的库文件名并不存在，则 EsgynDB SQL 会发出一个警告。

2.24.2.1 必需的权限

发出一条 DROP LIBRARY 语句，必须满足以下条件之一：

- 您是 DB__ROOT（数据库根用户）。
- 您是库的所有者。

- 您拥有 SQL_OPERATIONS 组件的 DROP 或 DROP_LIBRARY 权限。

2.24.3 DROP LIBRARY 例子

- 该 DROP LIBRARY 语句可以从 SALES schema 里删除一个名为 SALESLIB 的库、该库引用的 Sales2.jar 文件和基于该库创建的任何 Java 存储过程（SPJ）。

  ```
  DROP LIBRARY sales.saleslib CASCADE;
  ```

- 该 DROP LIBRARY 语句可以从默认 schema 里删除一个名为 MYUDFS 的库和该库引用的$TMUDFS 库文件。

  ```
  DROP LIBRARY myudfs RESTRICT;
  ```

  RESTRICT 选项可以防止 DROP LIBRARY 语句删除任何基于该库创建的用户定义函数（UDF）。如果任何 UDF 都基于该库创建，则 DROP LIBRARY 语句操作将执行失败。

2.25 DROP PROCEDURE 语句

该语句从 EsgynDB 数据库里删除一个 Java 存储过程（SPJ）。

注意： 目前的事务不支持 DDL 语句。这意味着您既无法在用户定义的事务（BEGIN WORK...COMMIT WORK）中运行该语句、也无法在 AUTOCOMMIT 选项关闭时运行该语句。为了运行该语句，必须开启该会话的 AUTOCOMMIT（自动提交）选项（默认）。

```
DROP PROCEDURE [[catalog-name.]schema-name.]procedure-name
```

2.25.1 DROP PROCEDURE 语法

`[[catalog-name.]schema-name.]procedure-name`

指定一个存储过程的 ANSI 逻辑名称，其中每部分都是一个有效的 SQL 标识符，最长 128 个字符。指定 schema 里已注册的存储过程的名称。

如果您没有完全限定一个存储过程名称，EsgynDB SQL 将根据当前会话的 schema 限定它。更多信息，请参阅标识符和数据库对象名称。

2.25.2 DROP PROCEDURE 注意事项

2.25.2.1 必需的权限

发出一条 DROP PROCEDURE 语句，必须满足以下条件之一：

- 您是 DB__ROOT（数据库根用户）。
- 您是该存储过程的所有者。
- 您拥有 SQL_OPERATIONS 组件的 DROP 或 DROP_ROUTINE 权限。

2.25.3 DROP PROCEDURE 例子

- 该语句从 SALES schema 中删除一个名为 LOWERPRICE 的存储过程：

  ```
  DROP PROCEDURE sales.lowerprice;
  ```

- DROP PROCEDURE 语句能从默认的会话 schema（即 SALES schema）中删除 TOTALPRICE 存储过程：

  ```
  SET SCHEMA sales;
  DROP PROCEDURE totalprice;
  ```

2.26 DROP ROLE 语句

该语句删除一个 SQL 角色。更多信息，请参阅角色。

```
DROP ROLE role-name
```

2.26.1 DROP ROLE 语法

role-name

它是一个现有的角色名。

如果以下条件之一成立，则不可删除该角色：

- 该角色已被授予任一权限。
- 该角色被授予给任一用户。
- 该角色拥有任一 schema。

2.26.2 DROP ROLE 注意事项

- 为了删除一个角色，您必须拥有该角色或具有该角色的用户管理权限。如果您已被授予 MANAGE_ROLES 组件权限，则将具有该角色的用户管理权限。最初只有 DB__ROOT（数据库根用户）被授予 MANAGE_ROLES 组件权限。

- 以 DB__开头的角色名将被保留，并仅可被 DB__ROOT（数据库根用户）删除。

- 您可 s 以通过使用 SHOWDDL ROLE 语句确定所有被授予该角色的用户。更多信息，请参阅 SHOWDDL 语句。

2.26.2.1 删除一个角色之前

删除一个角色之前，您必须遵循这些准则：

- 您必须撤销所有授予该角色的权限。
- 所有拥有该角色的用户都必须先撤去该角色。
- 您必须删除角色为管理者或所有者的所有 schema。

您可以通过使用 SHOWDDL 语句确定所有被授予该角色的用户。更多信息，请参阅 SHOWDDL 语句。

2.26.2.2 用户的活动会话

当您撤销一个用户的角色时，权限变更情况将被自动传播、并被活动的会话检

测到。用户无需断开再重新连接会话，即可看到更新后的权限集。

2.26.3 DROP ROLE 例子

- 删除一个角色:
  ```
  DROP ROLE clerks;
  ```

- 删除一个带有依赖权限的角色:
  - 用户管理员创建一个角色:
    ```
    CREATE ROLE clerks;
    ```
  - 用户管理员将一个表上的权限都授予给该角色:
    ```
    GRANT ALL PRIVILEGES ON TABLE invent.partloc TO clerks;
    ```
  - 用户管理员将该角色授予给一个用户:
    ```
    GRANT ROLE clerks TO JSmith;
    ```
  - JSmith 基于被授予的权限创建一个视图:
    ```
    CREATE VIEW invent.partlocView (partnum, loc_code)
    AS SELECT partnum, loc_code FROM invent.partloc;
    ```
  - 如果用户管理员尝试删除该角色，操作将会失败，因为该视图基于已授予的权限而创建。为了成功删除该角色，必须首先删除依赖性视图和授权。对于该例子而言:
    1) JSmith 删除该视图
       ```
       DROP VIEW invent.partlocView;
       ```
    2) 用户管理员撤销该用户的角色
       ```
       REVOKE ROLE clerks FROM JSmith;
       ```
    3) 用户管理员撤销所授予角色的一切权限
       ```
       REVOKE ALL ON invent.partloc FROM clerks;
       ```
    4) 用户管理员删除该角色:
       ```
       DROP ROLE clerks;
       ```
    5) 删除完毕。

2.27 DROP SCHEMA 语句

该语句从数据库里删除一个 schema。更多信息，请参阅 Schemas。

注意： 目前的事务不支持 DDL 语句。这意味着您既无法在用户定义的事务（BEGIN WORK...COMMIT WORK）中运行该语句、也无法在 AUTOCOMMIT 选项关闭时运行该语句。为了运行该语句，必须开启该会话的 AUTOCOMMIT（自动提交）选项（默认）。

```
DROP SCHEMA schema-name [RESTRICT | CASCADE]
```

2.27.1 DROP SCHEMA 语法

- schema-name
  这是将要删除的 schema 名称。

- IF EXISTS
  如果存在，则删除 schema。如果省略，则在 schema 不存在时引发错误。

- RESTRICT
  如果指定 RESTRICT 选项，且指定的 schema 不为空，则将报告一个错误。默认值为 RESTRICT。

- CASCADE
  如果指定 CASCADE，则指定的 schema 自身和其中的对象都将被删除。依赖于该 schema 里的对象的其他 schema 里的所有对象也将被删除。

2.27.2 DROP SCHEMA 注意事项

2.27.2.1 授权要求

删除一个 schema，以下条件之一必须成立：

- 您是该 schema 的所有者。
- 您已被授予该 schema 的所有者角色。
- 您已被授予 DROP_SCHEMA 权限。

2.27.3 DROP SCHEMA 例子

- 删除一个空的 schema：
  ```
  DROP SCHEMA sales;
  ```

2.28 DROP SEQUENCE 语句

该语句从 EsgynDB 数据库中删除一个序列。

注意： 目前的事务不支持 DDL 语句。这意味着您既无法在用户定义的事务（BEGIN WORK...COMMIT WORK）中运行该语句，也无法在 AUTOCOMMIT 选项关闭时运行该语句。为了运行该语句，必须开启该会话的 AUTOCOMMIT（自动提交）选项（默认）。

如果要更改序列的初始值，可以删除序列，然后使用相同的名称和不同的 START WITH 值重新创建序列。

DROP SEQUENCE 是 EsgynDB SQL 扩展语句。
更多信息，请参阅 CREATE SEQUENCE 语句或 ALTER SEQUENCE 语句。

```
DROP SEQUENCE [[catalog-name.]schema-name.]sequence-name
```

2.28.1 DROP SEQUENCE 语法

```
[[catalog-name.]schema-name.]sequence-name
```

指定要删除的序列名称，以及可选的目录和 schema 名称，其中每部分都是一个有效的 SQL 标识符，最长 128 个字符。

2.28.2 DROP SEQUENCE 注意事项

发出一条 DROP SEQUENCE 语句，必须满足以下条件之一：

- 您是 DB__ROOT（数据库根用户）。
- 您是该序列的所有者。
- 您拥有 SQL_OPERATIONS 组件的 DROP 或 DROP_ SEQUENCE 权限。

2.28.3 DROP SEQUENCE 例子

以下语句删除一个名为 employee_seq 的序列，该序列在 CRAETE SEQUENCE 的例子中创建。

```
DROP SEQUENCE employee_seq
```

2.29 DROP TABLE 语句

该语句删除一个 EsgynDB SQL 表和其依赖性对象诸如索引和约束。 更多信息，请参阅表。

注意： 目前的事务不支持 DDL 语句。这意味着您既无法在用户定义的事务（BEGIN WORK...COMMIT WORK）中运行该语句，也无法在 AUTOCOMMIT 选项关闭时运行该语句。为了运行该语句，必须开启该会话的 AUTOCOMMIT（自动提交）选项（默认）。

```
DROP [VOLATILE] TABLE [IF EXISTS] table [RESTRICT | CASCADE]
```

2.29.1 DROP TABLE 语法

- VOLATILE
  指定将要删除的表是一个临时表。

- IF EXISTS
  删除已存在的 HBase 表。该选项不适用于可变表。

- `table`
  这是将要删除的表名。

- RESTRICT
  如果您指定 RESTRICT、且该表为另一个对象所引用，则无法删除指定的表。默认值为 RESTRICT。

- CASCADE
  如果您指定 CASCADE、且该表和所有引用该表的对象（诸如视图）都将被删除。

2.29.2 DROP TABLE 注意事项

2.29.2.1 授权要求

发出一条 DROP TABLE 语句，以下条件之一必须成立：

- 您是 DB__ROOT（数据库根用户）。
- 您是该表的所有者。
- 您拥有 SQL_OPERATIONS 组件的 DROP 或 DROP_TABLE 权限。

2.29.3 DROP TABLE 例子

- 删除一个表:

  ```
  DROP TABLE mysch.mytable;
  ```

- 删除一个可变表:

  ```
  DROP VOLATILE TABLE vtable;
  ```

2.30 DROP TRIGGER 语句

该语句删除一个触发器。

```
DROP TRIGGER [[catalog.]schema.]trigger_name
```

2.30.1 DROP TRIGGER 语法

- `[[catalog.]schema.]trigger_name`
  指定一个触发器的 ANSI 逻辑名称。该名称在其所属的 schema 内部的触发器名称里必须是唯一的。

2.30.2 DROP TRIGGER 注意事项

发出一条 DROP TRIGGER 语句，必须满足以下条件之一:

- 您是 DB__ROOT（数据库根用户）。
- 您是该触发器的所有者。
- 您拥有 SQL_OPERATIONS 组件的 DROP 或 DROP_ROUTINE 权限。

2.30.3 DROP TRIGGER 例子

- 该例子删除一个触发器。
  drop trigger test_trig;

2.31 DROP VIEW 语句

该语句删除一个 EsgynDB SQL 视图。更多信息，请参阅视图。

注意： 目前的事务不支持 DDL 语句。这意味着您既无法在用户定义的事务（BEGIN WORK...COMMIT WORK）中运行该语句，也无法在 AUTOCOMMIT 选项关闭时运行该语句。为了运行该语句，必须开启该会话的 AUTOCOMMIT（自动提交）选项（默认）。

```
DROP VIEW view [RESTRICT | CASCADE]
```

2.31.1 DROP VIEW 语法

- `view`
  这是将要删除的视图名。

- RESTRICT
  指定 RESTRICT 选项时，如果其被任何其他视图的查询表达式或另一个对象约束里引用，则将无法删除指定的视图。默认值为 RESTRICT。

- CASCADE
  如果指定 CASCADE，则任何依赖性对象都将被删除。

2.31.2 DROP VIEW 注意事项

2.31.2.1 授权要求

发出一条 DROP VIEWE 语句，以下条件之一必须成立：

- 您是 DB__ROOT（数据库根用户）。
- 您是该视图的所有者。
- 您拥有 SQL_OPERATIONS 组件的 DROP 或 DROP_VIEW 权限。

2.31.3 DROP VIEW 例子

- 删除一个视图：
  ```
  DROP VIEW invent.partlocView;
  ```

2.32 EXECUTE 语句

EXECUTE 语句在 EsgynDB 命令界面（TrafCI）会话里执行一条通过 PREPARE 语句预编译的 SQL 语句。

```
EXECUTE statement-name
     [ USING param [,param]…] ]

param is:
     ?param-name | literal-value
```

2.32.1 EXECUTE 语法

- *statement-name*
  这是一条预处理的 SQL 语句名称，即用在 PREPARE 语句里的语句名称。
  statement-name 是一种 SQL 标识符。更多信息，请参阅标识符。

- USING *param [,param]... param* is: ? *param-name | literal-value*
  以参数名（?param-name）或字面值（literal-value）的形式指定一条预处理语句中未命名参数（表示为?）的值。一个参数值的数据类型必须和预处理语句里的相关参数的数据类型相兼容。参数值（param）按照位置替换预处理语句中的未命名参数—USING 子句里的第 i 个值为预处理语句里的第 i 个参数。
  如果 USING 子句里的参数值少于 PREPARE 语句里的未命名参数，则 EsgynDB SQL 将返回一个错误；如果 USING 子句里的参数值多于 PREPARE 语句里的未命名参数，则 EsgynDB SQL 将发出警告 15019。
  USING 子句不会在一条预处理语句里设置命名参数（表示为?param-name）的参数值。为了设置命名参数的参数值，请使用 SET PARAM 命令。更多信息，请参阅《EsgynDB 命令界面指南》。

- *?param-name*
  必须使用 SET PARAM 命令预先指定*?param-name* must 的数值。*param-name* 区分大小写。更多有关 SET PARAM 命令的信息，请参阅《EsgynDB 命令界面指南》。

- *literal-value*
  这是一种数字或字符字面值，指定未命名参数的数值。
  如果 *literal-value* 是一种字符字面值、并且目标列为字符型，则您不

必将它括在单引号里。

它的数据类型取决于该字面值赋给列的数据类型。如果 *literal-value* 包含前导或尾随空格、逗号，或者如果它匹配已设置的任何参数名，则请将其括在单引号里。更多信息，请参阅 PREPARE 语句。

更多关于 SET PARAM 命令的信息，请参阅《EsgynDB 命令界面指南》。

2.32.2 EXECUTE 注意事项

2.32.2.1 EXECUTE 的作用域

EXECUTE（执行）一条语句之前必须使用 PREPARE 将其编译；但一次编译后，您就可以多次执行该语句而无需重新编译。必须在同一个 TrafCI 会话期间执行已编译的语句。

2.32.3 EXECUTE 例子

- 使用 PREPARE 编译语句一次，然后即可多次执行具有不同参数值的该语句。该例子使用 SET PARAM 命令在预处理语句里设置命名参数（表示为 *?param-name*）的参数值。

```
SQL>prepare findemp from
+>select * from persnl.employee
+>where salary > ?sal and jobcode = ?job;

--- SQL command prepared.

SQL>set param ?sal 40000.00;
SQL>set param ?sal 450;
SQL>execute findemp;

EMPNUM FIRST_NAME  LAST_NAME DEPTNUM JOBCODE SALARY
------ ---------- ---------- ------- ------- -------
232     THOMAS     SPINNER    4000    450    45000.00

--- 1 row(s) selected.

SQL>set param ?sal 20000.00;
SQL>set param ?sal 300;
SQL>execute findemp;

EMPNUM FIRST_NAME LAST_NAME DEPTNUM JOBCODE SALARY
------ ---------- --------- ------- ------- --------
```

```
75      TIM       WALKER     3000   300     32000.00
89      PETER     SMITH      3300   300     37000.40
...
--- 13 row(s) selected.
```

- 在 EXECUTE 语句的 USING 子句里指定字面值，用于预处理语句的未命名参数：

```
SQL>prepare findemp from
+>select * from persnl.employee
+>where salary > ? and jobcode = ?;

--- SQL command prepared.

SQL>execute findemp using 40000.00,450;

EMPNUM FIRST_NAME  LAST_NAME DEPTNUM JOBCODE SALARY
------ ---------- ---------- ------- ------- -------
232     THOMAS     SPINNER    4000    450    45000.00

--- 1 row(s) selected.

SQL>execute findemp using 20000.00, 300;

EMPNUM FIRST_NAME LAST_NAME DEPTNUM JOBCODE SALARY
------ ---------- --------- ------- ------- --------
75      TIM       WALKER     3000   300     32000.00
89      PETER     SMITH      3300   300     37000.40
...
--- 13 row(s) selected.
```

- 使用 SET PARAM 命令赋值给一个参数名，并指定 EXECUTE USING 子句里的参数名和字面值：

```
SQL>prepare findemp from
+>select * from persnl.employee
+>where salary > ? and jobcode = ?;

--- SQL command prepared.

SQL>set param ?Salary 40000.00;
SQL>execute findemp using 20000.00, 450;
```

```
EMPNUM FIRST_NAME  LAST_NAME DEPTNUM JOBCODE SALARY
------ ---------- ---------- ------- ------- -------
232     THOMAS     SPINNER    4000    450    45000.00
```

2.33 EXPLAIN 语句

EXPLAIN 语句帮助您审查查询的执行计划。在任何可以执行其他 SQL 语句（例如：SELECT）的地方，您都可以使用 EXPLAIN 语句。

更多有关 EXPLAIN 函数的信息，请参阅 EXPLAIN 函数。

EXPLAIN 是 EsgynDB SQL 扩展语句。

```
EXPLAIN [OPTIONS {'f'}] {FOR QID query-text | prepared-stmt-name}
```

EXPLAIN 语句选项		
语法	**选项类型**	**目的**
选项'f'	格式化	提供查询执行计划里所包含的简单、基本的信息。这种格式化信息适合读取，具体限制为每个运算符（每行）79 个字符。

EXPLAIN 语句里显示的计划将自上（根节点操作符）而下（叶节点操作符）排序。

2.33.1 EXPLAIN 语法

- f
  格式化。更多信息，请参阅 Formatted [OPTIONS 'f']注意事项。

- *CURRENT*
  提供最新编译查询的信息。

- *query-id*
  指定一条预处理或正在执行的查询中的查询 ID，该 ID 是由 SQL 编译器生成的唯一标识符。

- *query-text*
  一条诸如 SELECT * FROM T3 的 DML 语句。

- *prepared-stmt-name*
  这是一种 SQL 标识符，它包含已在本会话中预处理的语句名称。SQL 标识符不区分大小写（将采用大写字母），除非加上双引号。

如果包含空格、小写字母或特殊字符，则必须加上双引号。它必须以字母开头。当您在一条 SELECT 语句里引用一条预处理查询时，必须使用大写字母。

注意：一些可用于预处理语句名的非保留字不能直接用于 EXPLAIN。例如，使用'EXPLAIN access;'会导致语法错误。但是，您仍然可以通过使用分隔标识符来使用该名称。即，用大写字母指定名称，并用双引号括起来。例如，使用“"EXPLAIN "ACCESS";'就会起作用。

2.33.2 EXPLAIN 注意事项

2.33.2.1 必需的权限

为了发出一条 EXPLAIN 语句，以下条件之一必须成立：

- 您是 DB__ROOT（数据库根用户）。
- 您拥有（即已发出）EXPLAIN 语句里指定的查询。
- 您拥有 SQL_OPERATIONS 组件的 SHOW 权限。SHOW 组件权限默认授予给 PUBLIC。

2.33.2.2 查询运行期间获取 EXPLAIN 计划

EsgynDB SQL 能使您随时捕获一条使用 FOR QID 选项运行的查询的 EXPLAIN 计划。默认情况下，该行为将被 EsgynDB 数据库会话所禁用。

注意：您在预处理和执行查询之前应启用该功能。

启用该功能后，在 EXPLAIN 语句里使用 FOR QID 选项即可获得一条正在运行的查询的执行计划。

EXPLAIN 函数或语句返回预处理一条查询时生成的计划。带有 FOR QID 选项的 EXPLAIN 可以从一条正在执行的查询的初始计划里检索所有信息。该计划直到查询执行结束时都可以获得。此后它将被删除或释放。

2.33.2.3 大小写字母注意事项

在大多数情况下，这些命令里的单词既可以是大写字母、也可以是小写字母。选项字母在必须是加上单引号的小写字母。

2.33.2.4 数字注意事项

这些成本费用以通用劳动力单元的形式给出。它们可以显示一个操作的相对成本费用。

如果尾随的十进制数字都是零，则它们将被删除。例如：6.4200 可以显示为 6.42, 5.0 则可以显示为 5，且不含小数点。

2.33.2.5 Formatted [OPTIONS 'f']注意事项

格式化选项是最简单的选项。它提供了该计划基本而又简短的信息，其中显示了操作符和它们在查询执行计划里的顺序。

OPTIONS 'f'可以将 EXPLAIN 输出格式化为这些字段:

LC	**左孩子的序列号**
RC	右孩子序列号
OP	查询计划里操作符的序列号
OPERATOR	操作符类型
OPT	所应用的查询优化
DESCRIPTION	关于操作符的其他信息
CARD	该计划所返回的估算行数 CARDINALITY 和 ROWS_OUT 是相同的。

该例子使用 OPTIONS 'f':

```
>>explain options 'f' select * from region;

LC      RC     OP      OPERATOR     OPT    DESCRIPTION    CARD
---     ---   ----     ---------   ----    -----------    ------
1        .     2        root                          1.00E+002
1        .     2        trafodion_scan     REGION     1.00E+002

--- SQL operation complete.
```

为了同时使用 EXPLAIN 语句和一条预处理语句，请首先对该查询进行预处理。然后使用 EXPLAIN 语句:

```
PREPARE q FROM SELECT * FROM REGION;

EXPLAIN options 'f' q;
```

EXPLAIN 还可直接与查询文本一起使用:

```
EXPLAIN options 'f' SELECT * FROM REGION;
```

2.33.2.6 hint（提示）

1. hint (提示)是用来影响执行计划的，目前有三种:
   cardinality
   selectivity
   index

   a. 该例子显示如何使用 cardinality
      select * from t1<<+ cardinality 1e10>>;

   b. 该例子显示如何使用 selectivity
      select * from tt<<+ selectivity 1.0>>;

   c. 该例子通过使用 index 来显示表的 index 被使用到:

```
>>create table tt(id int, name varchar(30), schname varchar(30));

--- SQL operation complete.
>>insert into tt values(1,'sdf','rwer'),(2,'abc','cdd'),(3,'aaa','bbb'),(4,'sssf','wwwe');
--- 4 row(s) inserted.

>>create index tt_name on tt(name);
--- SQL operation complete.

>>create index tt_name_schname on tt(name, schname);
--- SQL operation complete.

>>explain options 'f' select name from tt;

LC RC OP OPERATOR OPT         DESCRIPTION    CARD
---- ---- ---- -------------------- -------- ----------
---

1  .  2  root                                1.00E+002
.  .  1  trafodion_index_scan TT_NAME        1.00E+002
--- SQL operation complete.

>>fc
>>explain options 'f' select name from tt;
.. i<<+ index tt_name_schname>>
>>explain options 'f' select name from tt<<+ index tt_name_schname>>;
..

LC RC OP OPERATOR OPT DESCRIPTION CARD
---- ---- ---- -------------------- -------- ----------
---------- ---------
```

```
1  .   2 root                      1.00E+002
. . 1 trafodion_index_scan TT_NAME_SCHNAME 1.00E+002

--- SQL operation complete.
>>fc
>>explain  options  'f'  select  name  from  tt<<+  index
tt_name_schname>>;
.. *
>>explain  options  'f'  select  *  from  tt<<+  index
tt_name_schname>>;
..

LC RC OP OPERATOR OPT   DESCRIPTION CARD
---- ---- ---- -------------------- -------- ----------
---------- ---------

1  . 2   root                       1.00E+002
.  . 1   trafodion_scan  TT         1.00E+002

--- SQL operation complete.
>>control query shape join(cut,cut);

--- SQL operation complete.
>>fc exp;
>>explain  options  'f'  select  *  from  tt<<+  index
tt_name_schname>>;
..

LC RC OP OPERATOR OPT           DESCRIPTION      CARD
---- ---- ---- -------------------- -------- ----------
---------- ---------

4  .  5  root                                 1.00E+002
1  3  4  nested_join                          1.00E+002
2  .  3  probe_cache                          1.00E+000
.  .  2  trafodion_vsbb_scan    TT            1.00E+000
.    .    1    trafodion_index_scan       TT_NAME_SCHNAME
1.00E+002

--- SQL operation complete.
>>fc
>>explain  options  'f'  select  *  from  tt<<+  index
tt_name_schname>>;
.. ddddddddd
>>explain options 'f' select * from tt<<+ index tt_name>>;
..

LC RC OP OPERATOR OPT                DESCRIPTION CARD
---- ---- ---- -------------------- -------- ----------
---------- ---------
```

```
4   .   5 root                                  1.00E+002
1   3   4 nested_join                           1.00E+002
2   .   3 probe_cache                           1.00E+000
.   .   2 trafodion_vsbb_scan        TT         1.00E+000
.   .   1 trafodion_index_scan       TT_NAME    1.00E+002

--- SQL operation complete.
```

2. 兼容 Oracle hint

为兼容 Oracle，支持以下语法：

```
{DELETE|INSERT|SELECT|UPDATE} /*+ hint [text]
[hint[text]]... */
```

举例：

```
>>select * from staff /* +index(t1,idx_t1_c1)*/;

LAST_NAME   SALARY
-------------------------------------------------------
--
Tom         10000
Peter       10000
Mike        10000
Jake        2000

>>select /* +index(t1,idx_t1_c1)*/ * from staff;

LAST_NAME   SALARY
-------------------------------------------------------
--
Tom         10000
Peter       10000
Mike        10000
Jake        2000

--- 4 row(s) selected.
```

EsgynDB 对此仅做语法支持，实际并不会做任何事情。

2.34 GET 语句

该语句显示 EsgynDB 数据库实例里存在的数据库对象名称、组件、组件权限、角色或用户。

GET 是 EsgynDB SQL 扩展语句。

```
GET option
option is:
     CATALOGS
   | COMPONENT PRIVILEGES ON component-name [FOR auth-name]
   | COMPONENTS
   |   FUNCTIONS   FOR   LIBRARY   [[catalog-name.]schema-
   name.]library-name
   | FUNCTIONS [IN SCHEMA [catalog-name.]schema-name]
   | HBASE REGISTERED TABLES IN CATALOG TRAFODION
   | HIVE EXTERNAL TABLES IN CATALOG TRAFODION
   | HIVE REGISTERED {OBJECTS | SCHEMAS | TABLES | VIEWS}
   IN CATALOG TRAFODION
   | INDEXES [IN SCHEMA schema-name]
   | INDEXES [ON TABLE table-name]
   | LIBRARIES [IN SCHEMA [catalog-name.]schema-name]
   |   PROCEDURES   FOR   LIBRARY   [[catalog-name.]schema-
   name.]library-name
   | PROCEDURES [IN SCHEMA [catalog-name.]schema-name]
   | ROLES [FOR USER database-username]
   | SEQUENCES [IN CATALOG catalog-name]
   | SEQUENCES [IN SCHEMA schema-name}]
   | SCHEMAS [IN CATALOG catalog-name]
   | SCHEMAS FOR [USER | ROLE] authorization-id
   | TABLES [IN CATALOG catalog-name]
   | TABLES [IN SCHEMA [catalog-name.]schema-name]
   | TABLES [IN VIEW view-name]
   | USERS [FOR ROLE role-name]
   | VIEWS [IN CATALOG catalog-name]
   | VIEWS [IN SCHEMA [catalog-name.]schema-name]
   | VIEWS [IN VIEW view-name]
   | VIEWS [ON VIEW view-name]
   |  VIEWS  ON  TABLE  [[catalog-name.]schema-name.]table-
   name
   | PRIVILEGES FOR {USER database-username | ROLE role-
   name}
   | PRIVILEGES ON SEQUENCE sequence-name
   | PRIVILEGES ON TABLE table-name
   | PRIVILEGES ON VIEW view-name
   | NAMESPACES
```

2.34.1 GET 语法

- COMPONENT PRIVILEGES ON *component-name*

显示指定组件可用的组件权限名称。

- COMPONENT PRIVILEGES ON *component-name* FOR *auth-name*
  显示了指定组件的权限，该组件已被赋予了指定授权名。
  auth-name 是一个已注册的数据库用户名或现有的角色名。它可以是一个常规或分隔的不区分大小写字母的标识符。更多信息，请参阅不区分大小写字母的分隔标识符。

- COMPONENTS
  显示所有现有组件的列表。

- FUNCTIONS
  显示当前会话的 catalog 和 schema 里的 UDF 的名称。默认 catalog 为 TRAFODION，默认 schema 为 SEABASE。

- FUNCTIONS FOR LIBRARY *[[catalog-name.]schema-name.]library-name*
  显示引用指定库的 UDF。

- FUNCTIONS IN SCHEMA *[catalog-name.]schema-name*
  显示指定 schema 里的所有 UDF。

- LIBRARIES
  显示当前会话的 catalog 和 schema 里的所有库的名称。默认 catalog 为 TRAFODION，默认 schema 为 SEABASE。

- LIBRARIES IN SCHEMA *[catalog-name.]schema-name*
  显示指定 schema 里的库。

- PROCEDURES
  显示当前会话的 catalog 和 schema 里的所有存储过程的名称。默认 catalog 为 TRAFODION，默认 schema 为 SEABASE。

- PROCEDURES FOR LIBRARY *[[catalog-name.]schema-name.]library-name*
  显示引用指定库的存储过程。

- PROCEDURES IN SCHEMA *[catalog-name.]schema-name*

显示指定 schema 里的所有存储过程。

- ROLES
  显示所有现有角色的列表。

- ROLES FOR USER *database-username*
  显示指定数据库用户被授予的所有角色。database-username（数据库用户名）是一种不区分大小写字母的常规或分隔标识符。
  更多信息，请参阅不区分大小写字母的分隔标识符。

- SCHEMAS
  显示当前会话的 catalog 里的所有 schema 的名称。默认 catalog 为 TRAFODION。

- SCHEMAS IN CATALOG *catalog-name*
  显示指定 catalog 里的所有 schema 的名称。对于 *catalog-name*，您仅可指定 TRAFODION。

- SCHEMAS FOR [USER | ROLE] *authorization-id*
  显示一个指定用户或角色所管理（或拥有）的所有 schema。*authorization-id* 是用户名或角色名。
  您可以为用户或角色分别指定 USER 或 ROLE。

- TABLES
  显示当前会话的 catalog 和 schema 里的所有表。默认 catalog 为 TRAFODION，默认 schema 为 SEABASE。

- TABLES IN SCHEMA *[catalog-name.]schema-name*
  显示指定 schema 里的所有表。

- TABLES [IN VIEW view-name]
  返回由视图 *view-name* 引用的表。

- USERS
  显示所有已注册数据库用户的列表。

- USERS FOR ROLE *role-name*
  显示已被授予指定角色的所有数据库用户。*role-name*（角色名）可以是

常规或分隔标识符。更多信息，请参阅不区分大小写字母的分隔标识符。

- VIEWS
  显示当前会话的 catalog 和 schema 里的所有视图的名称。默认 catalog 为 TRAFODION，默认 schema 为 SEABASE。

- VIEWS IN SCHEMA *[catalog-name.]schema-name*
  显示指定 schema 里的所有视图的名称。对于 *catalog-name*，您仅可指定 TRAFODION。

- `VIEWS [IN VIEW view-name]`
  返回由视图 *view-name* 引用的视图。

- VIEWS ON TABLE *[[catalog-name.]schema-name.]table-name*
  显示针对指定表创建的所有视图的名称。如果您没有使用 catalog 和 schema 名称限定表名，则 GET 将使用当前会话的 catalog 和 schema。
  对于 *catalog-name*，您仅可指定 TRAFODION。

- `PRIVILEGES FOR {USER | ROLE} authorization-id`
  显示一个指定用户或角色的所有权限。

2.34.2 GET 注意事项

重要提示： 只有当 EsgynDB 里启用认证和授权功能时，才可运行 GET COMPONENT PRIVILEGES、GET COMPONENTS、GET ROLES FOR USER 和 GET USERS FOR ROLE 语句。更多信息，请参阅《EsgynDB 安装部署指南南》中的 Enabling Security Features（启用安全功能）。

GET 语句以其内部格式显示分隔对象名称。例如： GET 语句将分隔名称"my ""table""" 以 my "table"的形式返回。

2.34.2.1 必需的权限

为了发出一条 GET 语句，以下条件之一必须成立：

- 您是 DB__ROOT（数据库根用户）。
- 您拥有 SQL_OPERATIONS 组件的 SHOW 权限。SHOW 组件权限默认授予给 PUBLIC。

2.34.3 GET 例子

- 该 GET 语句显示当前会话的 catalog（这里是 TRAFODIONcatalog）里的所有 schema 的名称:
  ```
  GET SCHEMAS;
  ```

- 该 GET 语句显示指定的 catalog（TRAFODION）里的所有 schema 的名称。
  ```
  GET SCHEMAS IN CATALOG TRAFODION;
  ```

- 该 GET 语句显示 DB__ROOT（数据库根用户）所拥有的 schema 的名称:
  ```
  GET SCHEMAS FOR USER DB__ROOT;
  ```

- 该 GET 语句显示当前会话的 catalog 和 schema（这里是 TRAFODION.SEABASE）里的所有表名:
  ```
  GET TABLES;
  ```

- 该 GET 语句显示 TRAFODIONcatalog 的指定 schema（SEABASE2）里的所有表名:
  ```
  GET TABLES IN SCHEMA SEABASE2;
  ```

- 该 GET 语句显示当前会话的 catalog 和 schema（这里是 TRAFODION.SEABASE）里的所有视图名:
  ```
  GET VIEWS;
  ```

- 该 GET 语句显示 TRAFODIONcatalog 的指定 schema（SEABASE2）里的所有视图名:
  ```
  GET VIEWS IN SCHEMA SEABASE2;
  ```

- 该 GET 语句显示针对 TRAFODION.SEABASEschema 里的指定 T 表创建的所有视图的名称:
  ```
  GET VIEWS ON TABLE T;
  ```

- 该例子在 GET VIEWS [IN VIEW *view-name*]、GET VIEWS [ON VIEW *viewname*]、GET VIEWS [ON TABLE *table-name*]和 GET TABLES [IN VIEW *view-name*]之间进行比较。

  创建表 *t1*.

  ```
  SQL>CREATE TABLE t1 (c1 int);
  --- SQL operation complete.
  ```

基于表*t1*创建视图*v1*。

```
SQL>CREATE VIEW v1 AS SELECT * FROM t1;
--- SQL operation complete.
```

基于视图 *v1* 创建视图 *v2*。

```
SQL>CREATE VIEW v2 AS SELECT * FROM v1;
--- SQL operation complete.
```

该语句不返回任何内容，因为*v1*未引用任何视图。

```
SQL>GET VIEWS IN VIEW v1;
--- SQL operation complete.
```

该语句返回*v1*，因为*v1*被*v2*引用。

```
SQL>GET VIEWS IN VIEW v2;
TRAFODION.SEABASE.V1
--- SQL operation complete.
```

该语句返回 *v2*，因为 *v2* 引用了 *v1*。

```
SQL>GET VIEWS ON VIEW v1;
TRAFODION.SEABASE.V2
--- SQL operation complete.
```

该语句不返回任何内容，因为没有视图引用*v2*。

```
SQL>GET VIEWS ON VIEW v2;
--- SQL operation complete.
```

该语句返回 *v1*，因为 *v1* 引用了 *t1*。

```
SQL>GET VIEWS ON TABLE t1;
TRAFODION.SEABASE.V1
--- SQL operation complete.
```

该语句返回*t1*，因为*t1*被*v1*引用。

```
SQL>GET TABLES IN VIEW v1;
TRAFODION.SEABASE.T1
--- SQL operation complete.
```

该语句不返回任何内容，因为*v2*未引用任何视图。

```
SQL>GET TABLES IN VIEW v2;
--- SQL operation complete.
```

- 该 GET 语句显示当前会话的 catalog 和 schema（这里是 TRAFODION.SEABASE）里的库名：
  ```
  GET LIBRARIES;
  ```
- 该 GET 语句显示 TRAFODION._MD_schema 里的库名：
  ```
  GET LIBRARIES IN SCHEMA "_MD_";
  ```
- 该 GET 语句显示 TRAFODION._MD_.UDR_LIBRARY 库里注册的存储过程名称：
  ```
  GET PROCEDURES FOR LIBRARY "_MD_".UDR_LIBRARY;
  ```
- 该 GET 语句显示 TRAFODION._MD_ schema 里的存储过程名称：
  ```
  GET PROCEDURES IN SCHEMA "_MD_";
  ```
- 该 GET 语句显示当前会话的 catalog 和 schema（这里是 TRAFODION.SEABASE）里的存储过程名称：
  ```
  GET PROCEDURES;
  ```
- 该 GET 语句显示当前会话的 catalog 和 schema（这里是 TRAFODION.SEABASE）里的 UDF 名称：
  ```
  GET FUNCTIONS;
  ```
- 该 GET 语句显示 MYSCHEMA 里的 UDF 名称：
  ```
  GET FUNCTIONS IN SCHEMA MYSCHEMA;
  ```
- 该 GET 语句显示 TRAFODION.MYSCHEMA.MYUDFS 库里创建的 UDF 名称：
  ```
  GET FUNCTIONS FOR LIBRARY MYSCHEMA.MYUDFS;
  ```
- 该 GET 语句显示所有现有组件的列表：
  ```
  get components;
  ```
- 该 GET 语句显示指定 SQL_OPERATIONS 组件可用的组件权限名称：
  ```
  get component privileges on sql_operations;
  ```
- 该 GET 语句显示 DB_ROOT 用户被授予的 SQL_OPERATIONS 组件权限：

```
get component privileges on sql_operations for db__root;
```

- 该 GET 语句显示所有现有角色的列表:

```
get roles;
```

- 该 GET 语句显示指定 DB__ROOT 用户被授予的所有角色:

```
get roles for user db__root;
```

- 该 GET 语句显示所有已注册数据库用户的列表:

```
get users;
```

- 该 GET 语句显示已被授予 DB__ROOTROLE 角色的所有数据库用户。

```
get users for role db__rootrole;
```

- 该例子演示了如何向指定的数据库用户 *testuser1* 授予权限并显示权限。

  testuser1 授予 *project* 表上的 *testuser2* 一些权限。

```
SQL>GRANT DELETE, INSERT, REFERENCES, SELECT
(PROJCODE,PROJDESC), UPDATE ON TABLE
project TO testuser2;
--- SQL operation complete.
```

  该 GET 语句显示已授予 *testuser2* 的所有权限。

```
SQL>GET PRIVILEGES FOR USER testuser2;
-IDU-R- TRAFODION.SEABASE.PROJECT
S------ TRAFODION.SEABASE.PROJECT <Column> PROJCODE
S------ TRAFODION.SEABASE.PROJECT <Column> PROJDESC
--- SQL operation complete.
```

- 该例子演示如何向指定的角色*"TECHNICAL_WRITER"* 授予权限并显示权限。

  testuser1 授予 *project* 表上的*"TECHNICAL_WRITER"*一些权限。

```
SQL>GRANT DELETE, INSERT, REFERENCES, SELECT
(start_date,ship_timestamp,est_complete), UPDATE ON
TABLE project TO
"TECHNICAL_WRITER";
--- SQL operation complete.
```

  该 GET 语句显示已授予*"TECHNICAL_WRITER"*的所有权限。

```
SQL>GET PRIVILEGES FOR ROLE "TECHNICAL_WRITER";
-IDU-R- TRAFODION.SEABASE.PROJECT
S------ TRAFODION.SEABASE.PROJECT <Column>
EST_COMPLETE
S------ TRAFODION.SEABASE.PROJECT <Column>
SHIP_TIMESTAMP
S------ TRAFODION.SEABASE.PROJECT <Column> START_DATE
--- SQL operation complete.
```

- 该 GET 语句显示所有 NAMESPACE 的列表：

```
GET NAMESPACES;
TRAF_150000
TRAF_RSRVD_1
TRAF_RSRVD_2
TRAF_RSRVD_3
TRAF_RSRVD_4
Default
hbase
--- SQL operation complete.
```

2.35 GET HBASE OBJECTS 语句

该语句显示一个直接来自 HBase、而非 EsgynDB 元数据的 HBase 对象列表；它可运行在任何 SQL 界面中，例如 EsgynDB 命令界面（TrafCI）。该命令等效于从 HBase shell 里运行 list 命令，但无需启动和连接至 HBase shell。

GET HBASE OBJECTS 是 EsgynDB SQL 扩展语句。

```
GET [ USER | SYSTEM | EXTERNAL | ALL ] HBASE OBJECTS
```

2.35.1 GET HBASE OBJECTS 语法

- USER

  显示一个 EsgynDB 用户对象列表。

- SYSTEM

  显示一个 EsgynDB 系统对象（诸如元数据、存储库、权限和分布式事务管理器 [DTM] 表）的列表。

- EXTERNAL

  显示一个非 EsgynDB 对象列表。

- ALL

  显示一个所有对象（包含用户、系统和外部对象）的列表。

2.35.2 GET HBASE OBJECTS 例子

- GET HBASE OBJECTS 语句显示 HBase 里的 EsgynDB 用户对象:

```
>>get user hbase objects;

TRAFODION.SCH.SB_HISTOGRAMS
TRAFODION.SCH.SB_HISTOGRAM_INTERVALS
TRAFODION.SCH.T006T1
TRAFODION.SCH.T006T2
TRAFODION.SCH.T006T3
TRAFODION.SCH.T006T4
TRAFODION.SCH.T006T5
TRAFODION.SCH.T006T6
TRAFODION.SCH.T006T7
TRAFODION.SCH.T006T8
TRAFODION.SCH.X1
TRAFODION.SCH.X2
TRAFODION.SCH.X3

--- SQL operation complete.
```

- GET HBASE OBJECTS 语句显示 HBase 里的 EsgynDB 系统对象:

```
>>get system hbase objects;

TRAFODION._DTM_.TLOG0_CONTROL_POINT
…
TRAFODION._DTM_.TLOG1_LOG_f
TRAFODION._MD_.AUTHS
TRAFODION._MD_.COLUMNS
TRAFODION._MD_.DEFAULTS
TRAFODION._MD_.INDEXES
TRAFODION._MD_.KEYS
TRAFODION._MD_.LIBRARIES
TRAFODION._MD_.LIBRARIES_USAGE
TRAFODION._MD_.OBJECTS
TRAFODION._MD_.OBJECTS_UNIQ_IDX
TRAFODION._MD_.REF_CONSTRAINTS
TRAFODION._MD_.ROUTINES
TRAFODION._MD_.TABLES
TRAFODION._MD_.SEQ_GEN
TRAFODION._MD_.TABLES
TRAFODION._MD_.TABLE_CONSTRAINTS
TRAFODION._MD_.TEXT
TRAFODION._MD_.UNIQUE_REF_CONSTR_USAGE
```

```
TRAFODION._MD_.VERSIONS
TRAFODION._MD_.VIEWS
TRAFODION._MD_.VIEWS_USAGE
TRAFODION._REPOS_.METRIC_QUERY_AGGR_TABLE
TRAFODION._REPOS_.METRIC_QUERY_TABLE
TRAFODION._REPOS_.METRIC_SESSION_TABLE
TRAFODION._REPOS_.METRIC_TEXT_TABLE

--- SQL operation complete.
```

- GET HBASE OBJECTS 语句显示 HBase 里的外部非 EsgynDB 对象:

```
>>get external hbase objects;

obj1 obj2

--- SQL operation complete.
```

2.36 GET VERSION OF METADATA 语句

该语句显示 EsgynDB 数据库实例里的元数据版本，并标注元数据是否为当前版本。

GET VERSION OF METADATA 是 EsgynDB SQL 扩展语句。

```
GET VERSION OF METADATA
```

2.36.1 GET VERSION OF METADATA 注意事项

- 如果元数据与安装的 EsgynDB 软件版本相兼容，则 GET VERSION OF METADATA 语句将标注元数据是否为当前版本:

```
Current Version 1.0.Expected Version 1.0.
Metadata is current.
```

- 如果元数据与安装的 EsgynDB 软件版本不兼容，则 GET VERSION OF METADATA 语句将提示您需要升级或重新初始化元数据:

```
Current Version 1.0.Expected Version 1.1.
Metadata need to be upgraded or reinitialized.
```

2.36.2 GET VERSION OF METADATA 例子

- GET VERSION OF METADATA 语句显示元数据版本:

```
get version of metadata;

Current Version 3.0.Expected Version 3.0.Metadata is
current.

--- SQL operation complete.
```

- GET VERSION OF METADATA语句显示EsgynDB Release 0.9.0实例里的元数据版本:

```
>> get version of metadata;
Current Version 2.3. Expected Version 2.3.
Metadata is current.
--- SQL operation complete.
```

- 如果元数据与安装的EsgynDB软件版本不兼容，输出里将提示“您需要升级或重新初始化元数据”:

```
get version of metadata;

Current Version 2.3.Expected Version 3.0.Metadata need
to be upgraded or reinitialized.

--- SQL operation complete.
```

2.37 GET VERSION OF SOFTWARE 语句

该语句显示系统上安装的 EsgynDB 软件版本，并标注其是否当前版本。

GET VERSION OF SOFTWARE 是 EsgynDB SQL 扩展语句。

```
GET VERSION OF SOFTWARE
```

2.37.1 GET VERSION OF SOFTWARE 注意事项

- 如果系统上的软件为当前版本，则 GET VERSION OF SOFTWARE 语句将显示以下输出：

  ```
  System Version 1.0.0.Expected Version 1.0.0.
  Software is current.
  ```

- 在少数情况下，当 EsgynDB 软件安装出现问题时或安装了不匹配的对象时，GET VERSION OF SOFTWARE 语句将显示以下输出：

  ```
  System Version 0.9.1.Expected Version 1.0.0.
  Version of software being used is not compatible with
  version of
  software on the system.
  ```

2.37.2 GET VERSION OF SOFTWARE 例子

- GET VERSION OF SOFTWARE 语句显示 EsgynDB Release 1.0.0 的软件版本：

  ```
  get version of software;
  System Version 1.0.0. Expected Version 1.0.0.Software
  is current.
  --- SQL operation complete.
  ```

- GET VERSION OF SOFTWARE语句显示EsgynDB Release 0.9.0的软件版本：

  ```
  get version of software;
  System Version 0.9.0. Expected Version 0.9.0.
  Software is current.
  --- SQL operation complete.
  ```

- 当 EsgynDB 软件安装出现问题时或安装了不匹配的对象时，您将看到以下输出，提示您正在使用的软件与系统上的软件不兼容：

  ```
  get version of software;
  ```

```
System Version 0.9.1. Expected Version 1.0.0.
Version of software being used is not compatible with
version of software on the
system.
--- SQL operation complete.
```

2.38 GRANT 语句

该语句向指定的用户或角色授予 SQL 对象访问权限。可以给对象授予权限，或者单独或同时给一列或多列授予权限。

提示：创建对象的用户或角色成为对象所有者，并拥有该对象的所有权限。无需向所有者授予权限，出于安全考虑，所有者可以向其他用户或角色授予/撤销权限。

重要提示：只有在 EsgynDB 中启用了身份验证和授权时，此语句才有效。更多信息，请参阅《EsgynDB 安装部署指南》中的 Enabling Security Features（启用安全功能）。

```
GRANT { privilege [, privilege]... | ALL [PRIVILEGES]}
  ON [object-type] [schema.] object
  TO { grantee [, grantee ]...}
     [WITH GRANT OPTION]
     [[GRANTED] BY grantor]

Privilege-name is:
  DELETE
| EXECUTE
| INSERT     [column-list]
| REFERENCES [column-list]
| SELECT     [column-list]
| UPDATE     [column-list]
| USAGE

object-type is:
   TABLE
  | PROCEDURE
  | LIBRARY
  | FUNCTION
  | SEQUENCE
  | SCHEMA

grantee is:
     auth-name

grantor is:
     role-name
column-list is:
  (column [,colummn] ...)
```

2.38.1 GRANT 语法

`privilege [,privilege ] ...| ALL [PRIVILEGES]`

指定将要授予的权限。可以为一个对象指定这些权限。

DELETE	可以使用 DELETE 语句。
EXECUTE	可以使用一条 CALL 语句执行一个存储过程或可以执行一个 UDF。
INSERT [column-list]	可以使用 INSERT 语句。
REFERENCES [column-list]	可以创建引用该对象的约束。
SELECT [column-list]	可以使用 SELECT 语句。
UPDATE [column-list]	可以在表对象上使用 UPDATE 语句。
USAGE	可以使用 CREATE PROCEDURE 或 CREATE FUNCTION 语句访问一个库。该权限使您能够对该库底层的库文件进行读访问。
ALL	所有适用的权限。 为表或视图指定 ALL 时，这包括 SELECT、DELETE、INSERT、REFERENCES 和 UPDATE 权限。如果对象是一个存储过程或 UDF，则只有 EXECUTE 权限得到应用。 如果对象是一个库，则只有 UPDATE 和 USAGE 权限得到应用。

- `ON [object-type] [schema-name.]object-name`
  指定一个将被授权的对象。更多信息，请参阅数据库对象名称。

  如果未指定，则默认为 TABLE。

  `[object-type][schema-name.]object-name` 可以是：
  - `FUNCTION [schema-name.]function-name`，这里的`function-name`是数据库里一个用户定义函数（UDF）的名称。
  - `LIBRARY [schema-name.]library-name`，这里的`library-name`是数据库里一个库对象的名称。
  - `PROCEDURE [schema-name.]procedure-name`，这里的`procedure-name`是数据库里已注册的一个Java存储过程（SPJ）的名称。
  - `SEQUENCE` *`[schema-name.]sequence-name`*，这里的`sequence-`

name是数据库里一个序列对象的名称。

- [TABLE] *[schema-name.]object*，这里的object是一个表或视图。

- TO {grantee}
  指定一个将被授予权限的 *auth-name*。

- auth-name
  指定一个将被授予权限的授权 ID。更多信息，请参阅授权 ID。auth-name 是一种不区分大小写字母的常规或分隔标识符。更多信息，请参阅不区分大小写字母的分隔标识符。

  授权ID必须是以下之一：

  - 一个已注册的数据库用户名
  - 一个现有的角色名
  - PUBLIC（如果授予一种权限给PUBLIC，则该权限仍可用于所有用户，除非后来将它从PUBLIC撤销。）

- WITH GRANT OPTION
  指定已被授予一种权限的 *auth-name* 同时也能授予相同的权限给其他用户或角色。

注意： WITH GRANT OPTION 不能授权给 PUBLIC。

例子：

```
SQL>GRANT SELECT ON t1 TO PUBLIC WITH GRANT OPTION;
*** ERROR[1007] The WITH GRANT OPTION is not
supported. [2018-06-11 10:56:21]

SQL>GRANT ALL PRIVILEGES ON t1 TO PUBLIC WITH GRANT
OPTION;
*** ERROR[1007] The WITH GRANT OPTION is not
supported. [2018-06-11 10:52:30]
```

- [GRANTED] BY *grantor*
  允许您以一个角色的名义授予权限。如果没有指定，则当前用户/授予者的权限将以您的名义被授予。

- *role-name*
  指定一个代理执行 GRANT 操作的角色。为了以一个角色的名义授予权限，您必须是该角色的成员之一，且该角色必须有权授予权限，即该角色必须被授予包含 WITH GRANT OPTION 子句的权限。

- *column-list*
  指定一个要授予请求权限的列的列表。

2.38.2 GRANT 注意事项

2.38.2.1 授权和可用性要求

- 为了授予某种权限给某个对象，您必须同时拥有该权限和授予该权限的权利。
- 这些权限可以直接授予给您或您被授予的角色之一。
- 您可以授予某种权限给某个对象。
  - 如果您是该对象的所有者（这样您可以被隐式地授予一个对象的所有权限）
  - 包含该对象的 schema 的所有者
  - 如果您已被同时授予该权限和该权限的 WITH GRANT OPTION 子句权限
- 如果以一个角色的名义授予权限，则您必须在[GRANTED] BY 子句里指定该角色。 为了以一个角色的名义授予权限，您必须是该角色的成员之一、且该角色必须有权授予权限，即该角色必须被授予 WITH GRANT OPTION 的权限。
- 如果您无权授予一个或多个指定的权限，则 SQL 将返回一个警告（但会授予您有权授予的指定权限）。如果您无权授予任何指定的权限，则 SQL 将返回一个错误。

例子:

customer 表的所有者是 testuser1，该用户使用 WITH GRANT OPTION 将 customer 表上的 DELETE、INSERT 和 REFERENCES 权限授予用户 testuser2。

testuser1:

```
SQL>GRANT DELETE, INSERT, REFERENCES ON customer TO
testuser2 WITH GRANT OPTION;
--- SQL operation complete.
```

然后，testuser2 尝试将 customer 表的所有权限授予 testuser3。

grant 命令返回一个警告，指出只授予了部分权限。

DELETE、INSERT 和 REFERENCES 权限被授予，因为 testuser2 已通过 with grant option 被授予了这些权限。

未授予 SELECT 和 UPDATE 权限。

testuser2:

```
SQL>GRANT ALL PRIVILEGES ON customer TO testuser3;
*** WARNING[1013] Not all privileges were granted. You lack
grant option for the
SELECT privilege. [2018-06-11 16:07:34]
*** WARNING[1013] Not all privileges were granted. You lack
grant option for the
UPDATE privilege. [2018-06-11 16:07:34]
--- SQL operation complete.
```

现在检查 testuser1、testuser2 和 testuser3 的权限。

```
SQL>SHOWDDL customer;
CREATE TABLE TRAFODION.SEABASE.CUSTOMER
(
CUSTNUM NUMERIC(4, 0) UNSIGNED NO DEFAULT NOT NULL
NOT DROPPABLE NOT SERIALIZED
, CUSTNAME CHAR(18) CHARACTER SET ISO88591 COLLATE
DEFAULT NO DEFAULT NOT NULL NOT DROPPABLE NOT SERIALIZED
, STREET CHAR(22) CHARACTER SET ISO88591 COLLATE
DEFAULT NO DEFAULT NOT NULL NOT DROPPABLE NOT SERIALIZED
, CITY CHAR(14) CHARACTER SET ISO88591 COLLATE
DEFAULT NO DEFAULT NOT NULL NOT DROPPABLE NOT SERIALIZED
, STATE CHAR(12) CHARACTER SET ISO88591 COLLATE
DEFAULT DEFAULT _ISO88591' ' NOT NULL NOT DROPPABLE NOT
SERIALIZED
, PRIMARY KEY (CUSTNUM ASC)
)
ATTRIBUTES ALIGNED FORMAT NAMESPACE 'TRAF_RSRVD_3'
;
-- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON
TRAFODION.SEABASE.CUSTOMER
TO TESTUSER1 WITH GRANT OPTION;
GRANT INSERT, DELETE, REFERENCES ON
TRAFODION.SEABASE.CUSTOMER TO TESTUSER2
WITH GRANT OPTION;
```

```
GRANT INSERT, DELETE, REFERENCES ON
TRAFODION.SEABASE.CUSTOMER TO TESTUSER3
GRANTED BY TESTUSER2;
--- SQL operation complete.
```

如果您没有授予任何指定特权的权限，SQL 会返回一个错误。

例子:

(接以上例子)

用户 testuser3 尝试向 testuser4 授予 DELETE 权限，但由于 testuser3 没有权限而失败。

```
SQL>GRANT DELETE ON CUSTOMER TO testuser4;
*** ERROR[1012] No privileges were granted. TESTUSER3 lacks
grant option on the
specified privileges. [2018-06-11 16:21:31]
```

2.38.3 GRANT 例子

- 授予一个表的列级 SELECT 和对象级 DELETE 权限:

  ```
  GRANT SELECT (part_no, part_name), DELETE ON TABLE
  invent.partloc TO "MO.Neill@company.com";]
  ```

- 将一个表的 SELECT 权限授予给一个用户:

  ```
  GRANT SELECT ON TABLE invent.partloc TO ajones;
  ```

- 该例子解释了如何将 SELECT 权限授予给 PUBLIC。

  testuser1 创建了表 t1。*testuser2* 和 *testuser3* 没有表 t1 的 SELECT 权限。

  testuser2:

  ```
  SQL>SELECT * FROM t1;
  *** ERROR[4481] The testuser2 does not have SELECT
  privilege on table or view
  TRAFODION.SEABASE.T1. [2018-06-11 11:39:16]
  ```

  testuser1 将 SELECT 权限授予给所有当前用户（*testuser2* 和 *testuser3*）和未来用户。

  testuser1:

  ```
  SQL>GRANT SELECT ON t1 TO PUBLIC¬
  --- SQL operation complete.
  ```

  然后，testuser2 和 testuser3 拥有了表 t1 的 SELECT 权限。

testuser3:

```
SQL>SELECT * FROM t1;
C1 C2
----------- -----------
1 2
3 4
--- 2 row(s) selected.
```

- 该例子演示了如何向指定的角色*"TECHNICAL_WRITER"*授予权限并显示权限。

```
SQL>GRANT DELETE, INSERT, REFERENCES, SELECT
(start_date,ship_timestamp,est_complete),   UPDATE   ON
TABLE project TO
"TECHNICAL_WRITER";
--- SQL operation complete.
SQL>GET PRIVILEGES FOR ROLE "TECHNICAL_WRITER";
-IDU-R- TRAFODION.SEABASE.PROJECT
S------ TRAFODION.SEABASE.PROJECT <Column> EST_COMPLETE
S------       TRAFODION.SEABASE.PROJECT       <Column>
SHIP_TIMESTAMP
S------ TRAFODION.SEABASE.PROJECT <Column> START_DATE
--- SQL operation complete.
```

- 该例子演示了如何将指定角色"TECHNICAL_WRITER"授予给 testuser1。

```
SQL>GRANT ROLE "TECHNICAL_WRITER" TO testuser1;
--- SQL operation complete.
```

- 该例子说明了如何授予 USAGE 权限。

testuser1 创建序列 sequence_testuser1，此时，testuser2 不能使用此序列生成器。

testuser2:

```
SQL>SELECT SEQNUM(sequence_testuser1, NEXT) FROM DUAL;
*** ERROR[4491] The user does not have the USAGE
privilege on sequence
TRAFODION.SEABASE.SEQUENCE_TESTUSER1. [2018-06-11
16:49:20]
```

然后，testuser1 将 USAGE 权限授予给 testuser2。

```
SQL>GRANT USAGE ON SEQUENCE sequence_testuser1 TO
testuser2;
```

```
--- SQL operation complete.
```

现在 *testuser2* 可以使用此序列生成器。

testuser2:

```
SQL>SELECT SEQNUM(sequence_testuser1, next) FROM DUAL;
(EXPR)
--------------------
112
--- 1 row(s) selected.
```

- 该例子演示了如何向指定的角色"technical_writer"授予 schema 级别权限。
  GRANT SELECT ON SCHEMA sch1 TO technical_writer WITH GRANT OPTION;

```
--- SQL operation complete.
```

2.39 GRANT COMPONENT PRIVILEGE 语句

该语句将一个或多个组件权限授予给一个用户或角色。更多信息，请参阅权限和角色。

GRANT COMPONENT PRIVILEGE 是 EsgynDB SQL 扩展语句。

重要提示：只有在 EsgynDB 中启用了身份验证和授权时，此语句才有效。更多信息，请参阅《EsgynDB 安装部署指南》中的 Enabling Security Features（启用安全功能）。

```
GRANT COMPONENT PRIVILEGE { privilege-name [, privilege-
name]...}
ON component-name
TO grantee
[WITH GRANT OPTION] [[GRANTED] BY grantor]

grantee is:
auth-name

grantor is:
role-name
```

2.39.1 GRANT COMPONENT PRIVILEGE 语法

- *privilege-name*
  指定一个或多个将授予的组件权限。逗号分隔的列表只能包含同一个组件

内的权限。

组件	组件权限	描述
SQL_OPERATIONS	ALTER	用于更改数据库对象的权限
	ALTER_LIBRARY	用于更改库的权限
	ALTER_ROUTINE	用于更改 routine 的权限
	ALTER_SCHEMA	用于更改 schema 的权限
	ALTER_SEQUENCE	用于更改序列生成器的权限
	ALTER_TABLE	用于更改表的权限
	ALTER_VIEW	用于更改视图的权限
	COMMENT	用于添加/更改注释的权限
	CREATE	用于创建数据库对象的权限
	CREATE_INDEX	用于创建索引的权限
	CREATE_LIBRARY	用于创建数据库里的库的权限
	CREATE_PROCEDURE	用于创建 Java 存储程序（SPJs）的权限
	CREATE_ROUTINE	用于创建数据库里 Java 存储过程（SPJ）、用户定义函数

		（UDF）、表映射函数和其他例程的权限
	CREATE_SCHEMA	用于创建数据库里 schema 的权限
	CREATE_SEQUENCE	用于在数据库中创建序列生成器的权限
	CREATE_TABLE	用于创建数据库里表的权限
	CREATE_VIEW	用于创建数据库里视图的权限
	DML_SELECT_METADATA	用于选取元数据的权限
	DROP	用于删除数据库对象的权限
	DROP_INDEX	用于删除 index 的权限
	DROP_LIBRARY	用于删除 libraries 的权限
	DROP_PROCEDURE	用于删除 Java 存储程序（SPJs）的权限
	DROP_ROUTINE	用于删除数据库里 Java 存储过程（SPJ）、用户定义函数（UDF）、表映射函数和其他例程的权限
	DROP_SCHEMA	用于删除 schema 的权限

	DROP_SEQUENCE	用于删除序列生成器的权限
	DROP_TABLE	用于删除表的权限
	DROP_VIEW	用于删除视图的权限
	MANAGE	用于管理所有组件特权
	MANAGE_COMPONENTS	用于创建或删除组件以及向/从用户/角色授予/撤消组件权限的权限
	MANAGE_LIBRARY	用于执行库相关命令的权限，例如：用于创建和删除库的权限
	MANAGE_LOAD	用于执行 LOAD 和 UNLOAD 命令的权限
	MANAGE_PRIVILEGES	用于向用户/角色授予/撤消对象权限的权限
	MANAGE_RESOURCE_GROUPS	用于为多租户创建、更改或删除资源组的权限
	MANAGE_ROLES	用于创建、更改、删除、授予和撤销角色的权限
	MANAGE_STATISTIS	用于更新和显示统计信息的

		权限
	MANAGE_TENANTS	用于注册、更改或注销多租户的特权
	MANAGE_USERS	用于注册或注销用户、更改用户、授权或撤销组件权限
	QUERY_ACTIVATE	在查询被挂起后，用于激活或执行查询的权限
	QUERY_CANCEL	取消正在执行的查询的权限
	QUERY_SUSPEND	挂起正在执行的查询的权限
	SHOW	运行 EXPLAIN、GET、INVOKE 和 SHOW 命令。SHOW 权限已默认授权给 PUBLIC

- ON *component-name*
  指定一个将被授予组件权限的组件名称。目前唯一有效的组件名称是 SQL_OPERATIONS。

- TO *grantee*
  指定一个将被授予组件权限的 *auth-name*。

- *auth-name*
  指定一个将被授予权限的授权 ID。更多信息，请参阅授权 ID。
  授权 ID 必须是一个已注册的数据库用户名、一个现有的角色名或 PUBLIC。该名称是一种不区分大小写字母的常规或分隔标识符。更多信息，请参阅

不区分大小写字母的分隔标识符。

如果您授予一种权限给 PUBLIC，则该权限仍可用于所有用户，除非以后将其从 PUBLIC 撤销。

- WITH GRANT OPTION
  指定已被授予一个组件权限的 auth-name 同时也能授予相同的组件权限给其他用户或角色。

- [GRANTED] BY *grantor*
  允许您以一个角色的名义授予组件权限。如果没有指定，则当前用户/授予者的权限将以您的名义授予。

- *role-name*
  指定一个代理执行 GRANT COMPONENT PRIVILEGE 操作的角色。
  为了以一个角色的名义授予权限，您必须是该角色的成员之一、且该角色必须有权授予权限，即该角色必须被授予 WITH GRANT OPTION 的权限。

2.39.2 GRANT COMPONENT PRIVILEGE 注意事项

- 一个被授予含 WITH GRANT OPTION 组件权限的用户或角色可以授予相同的组件权限给其他用户角色。
- 如果所有的组件权限都已被授予，则 SQL 将返回一个错误。
- 如果其中的一个或多个组件权限已被授予，则 SQL 将静默忽略那些已授予的权限、并将继续执行授予操作。

2.39.2.1 授权和可用性要求

为了授予一个组件权限，您必须具有以下这些权限之一：

- 用户管理权限（即一个已被授予 MANAGE_USERS 组件权限的用户）。最初，数据库用户里只有 DB__ROOT（数据库根用户）被授予 MANAGE_USERS 组件权限。
- 一个拥有 WITH GRANT OPTION 组件权限的用户（用户管理员除外）。
- 用户被授予的一个角色具有 WITH GRANT OPTION 组件权限。

2.39.3 GRANT COMPONENT PRIVILEGE 例子

- 将 SQL_OPERATIONS 组件的 CREATE_TABLE 组件权限授予给 SQLUSER1：
  ```
  GRANT COMPONENT PRIVILEGE CREATE_TABLE ON SQL_OPERATIONS
  TO sqluser1;
  ```

2.40 GRANT ROLE 语句

该语句授予用户一个或多个角色。更多信息，请参阅角色。

重要提示：只有在 EsgynDB 中启用了身份验证和授权时，此语句才有效。更多信息，请参阅《EsgynDB 安装部署指南》中的启用安全功能。

```
GRANT ROLE { role-name [, role-name ]...}
TO grantee

grantee is:
     database-username
```

2.40.1 GRANT ROLE 语法

- *role-name [,role-name] ...*
  指定已经存在的角色。

- TO *grantee*
  指定您希望授予角色的已注册数据库的用户名。

2.40.2 GRANT ROLE 注意事项

- 为了授予一个角色给其他被授权者，您必须拥有这些角色或具有这些角色的用户管理权限。如果您已被授予 MANAGE_ROLES 组件权限，则将具有这些角色的用户管理权限。最初只有 DB__ROOT（数据库根用户）被授予 MANAGE_ROLES 组件权限。

- 您授予一个角色给用户时，权限变更情况将被自动传播，并被活动的会话检测到。用户无需断开再重新连接会话，即可看到更新后的权限集。
- 如果在处理一条可以命名多个角色的 GRANT ROLE 语句时出现错误，则将无法完成任何授予操作。

- 尝试授予一个角色时，如果已存在相同的授予角色和被授权人，则 SQL 将忽略该请求，并将返回一个成功操作。

2.40.3 GRANT ROLE 例子

- 授予多个角色给一个被授权人：
  `GRANT ROLE clerks, sales TO jsmith;`

2.41 INSERT 语句

该语句是一种 DML 语句，它将行插入一个表或视图里。

```
INSERT INTO table [(target-col-list)] insert-source

target-col-list is:
     colname [, colname]...

insert-source is:
     query-expr   [order-by-clause]   [access-clause]   |
     DEFAULT VALUES
```

2.41.1 INSERT 语法

- *table*

  对一个需要插入行的用户表或视图进行命名。该表必须是一个基表或一个可更新视图。

- *(target-col-list)*

  对需要插入数值的表或视图里的列进行命名。每个目标列的数据类型都必须与其相应源值的数据类型相兼容。

  列表中的每个目标列都必须与关联的源值具有相同的位置；该位置取决于查询表达式（query-expr）求值所派生的表的列。

  如果您没有指定 target-col-list（目标列列表）里的表的所有列，则列的默认值将被插入该列表中未出现的列里。更多信息，请参阅列的默认设置。

  如果您没有指定 target-col-list，则源表的行值将被插入该表的所有列里。

  源表里的列值顺序必须与该表 CREATE TABLE 语句里指定的列序相同（该顺序与 SHOWDDL 表的结果表里所列的列序相同）。

- *insert-source*

  指定将要插入该表所有列或指定列（可选）里的行值。

- *query-expr*
  更多关于 query-expr、order-by-clause 和 access-clause 的信息，请参阅 SELECT 语句。

- DEFAULT VALUES
  它等效于 VALUES (DEFAULT, ...)的 *query-expr*。
  每个 DEFAULT 的数值都是 *colname* 列描述符里定义的默认值，其包含在 *table* 的表描述符里。每个默认值都将被插入该表的列中，以便构成一个新的行。如果您已指定 DEFAULT VALUES，则不得指定“列列表”。只有当 *table* 的所有列都具有默认值时，您才可以使用 DEFAULT VALUES。

2.41.2 INSERT 注意事项

2.41.2.1 授权要求

执行 INSERT 语句必需有权对接收数据的表或视图进行读写操作，并有权读取 INSERT 语句的查询表达式（或任何子查询）里指定的表或视图。

2.41.2.2 发起和终止事务

如果目前没有处于活动状态的事务，则 INSERT 语句将会自动发起一个事务。
此外，您还可以使用 BEGIN WORK 语句显式地发起一个事务。
启动一个事务后，SQL 语句将在该事务内部执行，直到遇到 COMMIT 或 ROLLBACK、或出现一个错误为止。
如果开启 AUTOCOMMIT 选项，则该事务将在 INSERT 语句执行结束时终止。

2.41.2.3 自引用 INSERT 和 BEGIN WORK 或 AUTOCOMMIT OFF

一条自引用 INSERT 语句应在其插入源里引用需要插入行的同一个表或视图。
更多信息，请参阅自引用插入操作例子。
如果使用 BEGIN WORK 或 AUTOCOMMIT OFF，则自引用 INSERT 语句将无法正确执行或抛出一个错误，除非编译器在插入这些行之前已对它们进行排序。
如果您希望使用一条自引用 INSERT 语句，则应避免使用 BEGIN WORK 或 AUTOCOMMIT OFF。更多关于 AUTOCOMMIT 的信息，请参阅 SET TRANSACTION 语句。

2.41.2.4 事务隔离级别和语句访问选项

SQL 事务隔离级别定义事务内部数据操作受并发事务影响的程度。
如果您在一个事务内部指定了 DML 语句的访问选项，则其中所包含事务的隔离

级别将被覆盖。每条语句都将按照其各自的访问选项执行。

2.41.2.5 为源查询表达式使用 VALUES 子句

如果查询表达式含有 VALUES 关键字和行值，则每行都将含有一个数值表达式列表或一条行子查询（即：可以返回一行列值的子查询）。

行值也可以是一条标量子查询（一条子查询，可以返回一个含有单列数值的行）。

在一个 VALUES 子句里，一个数值表达式的操作数可以是数字、字符串、日期时间或间隔值。 然而，操作数无法引用一个列（除非在一个标量或行子查询的结果表里返回一个或多个值）。

2.41.2.6 对于插入行的要求

所插入的每行都必须满足表或视图底层基表的约束。

如果 Check 条件不是假的（它是真的或有一个未知的数值），则可以满足表约束。

2.41.2.7 使用兼容性数据类型

为了插入一行，您必须为表里不含默认值的每列都提供一个数值。每行插入数值的数据类型都必须与相应的目标列的数据类型相兼容。

- 插入 Character（字符）值

  任何字符串数据类型都必须与具有相同字符集的所有其他字符串数据类型相兼容。任何字符串数据类型都必须与其他所有字符串数据类型使用相同的字符集。

  对于固定长度而言，小于列长度的插入值将被填充在右侧，并留有适当字符集的空格字符，例如：ISO88591 空格（HEX20）。

  如果该值大于列长度，则非空尾部字符串被截断时将返回一个错误，被截断的字符串也将不会被插入。一个更短的可变长度插入值将不会被填充。

  固定长度的插入值也是同样的情况。如果该值大于列长度，则非空尾部字符串被截断时将返回一个错误，被截断的字符串也将不会被插入。

- 插入 Numeric（数字）值

  任何 Numeric（数字）数据类型都必须与所有其他 Numeric（数字）数据类型相兼容。如果您将一个数值插入到一个不够大的 Numeric 列，则会出现溢出错误。 如果一个数值小数点右边的位数超过了列定义指定的标度，则该数值将被截断。

- 插入 Interval（间隔）值

只有当一种 Interval 数据类型与另一种 Interval 数据类型都为 year-month（年-月）或 day-time interval（日-时间间隔）时，二者的数值才会兼容。

- 插入 Date（日期）和 Time（时间）值

  Date、time 和 timestamp 是三种 EsgynDB SQL datetime（日期时间）数据类型。 只有当一种 datetime（日期时间）数据类型与另一种 datetime（日期时间）数据类型具有相同的的 datetime（日期时间）字段时，二者的数值才会兼容。

- 插入空值

  当插入特定数据类型的数值时，您可能要插入空值。插入空值时，请使用关键字 NULL。NULL 只能和 VALUES 子句同时使用。

  为 *select-list* 使用 cast (null as type)。

2.41.3 INSERT 例子

- 在不使用 target-col-list 的情况下将一行插入 CUSTOMER 表里:

```
INSERT INTO sales.customer
   VALUES (4777, 'ZYROTECHNIKS', '11211 40TH ST.',
        'BURLINGTON', 'MASS.', '01803', 'A2');

--- 1 row(s) inserted.
```

  该 INSERT 语句没有指定“列的列名”。该操作能运行的原因在于 VALUES 子句里所列的数值的数量等于 CUSTOMER 表里的列数，并且所列的数值与 CUSTOMER 表的 CREATE TABLE 语句里指定的列以相同的顺序出现。

  以下 SELECT 语句显示这种特定的顺序:

```
SELECT * FROM sales.customer WHERE custnum = 4777;

CUSTNUM CUSTNAME     STREET          ... POSTCODE CREDIT
------- ------------ -------------       -------- ------
4777    ZYROTECHNIKS 11211 4OTH ST.  ... 01803     A2

--- 1 row(s) selected.
```

- 使用 target-col-list 将一行插入 CUSTOMER 表里:

```
INSERT INTO sales.customer
   (custnum, custname, street, city, state, postcode)
   VALUES (1120, 'EXPERT MAILERS', '5769 N. 25TH PL',
```

```
        'PHOENIX', 'ARIZONA', '85016');

--- 1 row(s) inserted.
```

与前面的例子不同，该语句的插入源不包含具有默认值的 CREDIT 列的一个数值。因此，该 INSERT 语句必须包含“列名列表”。

该 SELECT 语句为 CREDIT 显示默认值'C1':

```
SELECT * FROM sales.customer WHERE custnum = 1120;

CUSTNUM CUSTNAME      STREET            POSTCODE CREDIT
------- ------------- ---------------- -------- ------
1120    EXPERT MAILERS 5769 N. 25TH PL   85016   C1

--- 1 row(s) selected.
```

- 仅通过使用一条 INSERT 语句将多行插入到 JOB 表里:

```
INSERT INTO persnl.job
  VALUES (100,'MANAGER'),
  (200,'PRODUCTION SUPV'),
  (250,'ASSEMBLER'),
  (300,'SALESREP'),
  (400,'SYSTEM ANALYST'),
  (420,'ENGINEER'),
  (450,'PROGRAMMER'),
  (500,'ACCOUNTANT'),
  (600,'ADMINISTRATOR'),
  (900,'SECRETARY');

--- 10 row(s) inserted.
```

- PROJECT 表包含 5 列，它们分别具有 numeric、varchar、date、timestamp 和 interval 数据类型。通过使用这些数据类型插入数值:

```
INSERT INTO persnl.project
  VALUES (1000, 'SALT LAKE CITY', DATE '2007-10-02',
  TIMESTAMP  '2007-12-21  08:15:00.00',  INTERVAL  '30'
  DAY);

--- 1 row(s) inserted.
```

- 假设 CUSTLIST 是 CUSTOMER 表的所有列（不含 credit rating 列）的一个视图。通过 CUSTLIST 视图将信息从 SUPPLIER 表插入 CUSTOMER 表里，然

后更新 credit rating（信用评级）列：

```
INSERT INTO sales.custlist
   (SELECT * FROM invent.supplier
   WHERE suppnum = 10);

UPDATE sales.customer
   SET credit = 'A4'
   WHERE custnum = 10;
```

以下情况可使用该序列。假设您的一个供应商日后将成为客户，如果使用相同的客户编号和供应商编号，则可为这个新客户从 SUPPLIER 表里选择信息，并通过 CUSTLIST 视图将其插入 CUSTOMER 表里（如例所示）。

该操作可以运行，因为 SUPPLIER 表列包含与 CUSTLIST 视图列相对应的数值。此外，CUSTOMER 表里的 credit rating 列被指定具有默认值。如果您希望得到一个不同于该默认值的 credit rating，则必须更新这个新客户数据行里的该列。

2.41.3.1 自引用插入操作例子

- 这是一个自引用插入操作的例子：

```
insert into table1 select pk+?, b, c from table1
```

- 这是一个自引用插入操作的例子，其中的插入目标 table1 也被用在插入源的子查询里。

```
insert into table1
   select a+16, b, c from table2 where table2.b not in
        (select b from table1 where a > 16)
```

源表不受插入操作的影响。

2.42 INVOKE 语句

该语句生成一条记录描述，它对应于指定的表、视图或索引里的一行。记录描述包含表、视图或索引里每行的数据项，其中包含主键、但不包含 SYSKEY 列。只有当一个视图显式地列出定义里的列时，该描述才会包含该视图的 SYSKEY 列。

INVOKE 是 EsgynDB SQL 扩展语句。

```
INVOKE table-name
```

2.42.1 INVOKE 语法

- *table-name*
  指定用于生成一条记录描述的表、视图或索引的名称。更多信息，请参阅数据库对象名称。

2.42.2 INVOKE 注意事项

2.42.2.1 必需的权限

为了发出一条 INVOKE 语句，以下条件之一必须成立：

- 您是 DB__ROOT（数据库根用户）。
- 您是该表的所有者。
- 您拥有 SQL_OPERATIONS 组件的 SHOW 权限。SHOW 组件权限默认授予给 PUBLIC。
- 您拥有目标表的 SELECT 权限。

2.42.3 INVOKE 例子

- 该命令可以生成表 T 的一条记录描述：

```
SQL>invoke trafodion.seabase.t;

-- Definition of Trafodion table TRAFODION.SEABASE.T
-- Definition current Wed Mar 5 10:36:06 2014

(
A INT NO DEFAULT NOT NULL NOT DROPPABLE
)
PRIMARY KEY (A ASC)

--- SQL operation complete.
```

2.43 MERGE 语句

- 如果该行存在，应更新表；否则应将该行插入表里。
- 将匹配行从一个表更新（合并）到另一个表。

```
MERGE INTO table [using-clause]
     on-clause
     {[when-matched-clause]|[when-not-matched-clause]}  ...

using-clause is:
     USING (select-query) AS derived-table-name [derived-
     column-names]

on-clause is:
     ON predicate

when-matched-clause is:
     WHEN MATCHED THEN UPDATE SET set-clause [WHERE
     predicate]
     WHEN MATCHED THEN DELETE

when-not-matched-clause is:
     WHEN NOT MATCHED THEN INSERT insert-values-list

insert-values-list is:
     [(column1, ..., columnN )] VALUES (value1, ..., valueN)
```

2.43.1 MERGE 语法

- *table*
  指定一个表的 ANSI 逻辑名称。

- ON *predicate*
  用于确定某一行是否存在表里。
  如果 MERGE 有一个 *when-not-matched-clause*，则 ON 谓词必须是表的聚集键上的一个谓词。聚集键既可以是单列键、也可以是多列键。
  如果 MERGE 语句带有 *when-not-matched-clause* 子句，则 ON 谓词必须选择唯一的一行。

2.43.2 MERGE 注意事项

2.43.2.1 使用单行的 Upsert 语句

一条 MERGE 语句允许您指定一组列值；如果找到该行，则应更新该组列值；否则应插入另一行。如果 MERGE 语句有一条 INSERT 子句，ON 谓词必须选择

将要被更新的一行。

在一条 MERGE 语句里，必须指定至少一条子句-when-matched 或 when-not-matched。

请注意下列规则：

- 如果存在 when-matched 子句，且 UPDATE 语句里的 WHERE 谓词得到满足，则 SET 子句里的列将被更新。
- 如果存在 when-matched 子句，且 UPDATE 语句里的 WHERE 谓词未得到满足，则 SET 子句里的列将不被更新。
- 如果存在一条 when-matched 子句，并且 UPDATE 语句里没有 WHERE 谓词，则 SET 子句里的列将被更新。
- 如果存在一条 when-not-matched 子句，并且 INSERT 子句里显式地指定了列，则这些列的指定值都将被插入。 缺失的列将使用这些列的默认值进行更新。

2.43.2.2 例子

- 如果已找到数值为 10 的 a 键列所在的行，则该例子里可将 b 列更新为 20。如果 t 表里未找到该行，则插入一个新行（10, 30）。

```
MERGE INTO t ON a = 10
WHEN MATCHED THEN UPDATE SET b = 20
WHEN NOT MATCHED THEN INSERT VALUES (10, 30)
```

- 如果已找到数值为 10 的 a 列，则该例子里可将 b 列更新为 20。如果未找到数值为 10 的 a 列，则无需进行任何操作。

```
MERGE INTO t ON a = 10
WHEN MATCHED THEN UPDATE SET b = 20
```

- 如果未找到数值为 10 的 a 列，则该例子里将插入数值(10, 30)。如果已找到数值为 10 的 a 列，则无需进行任何操作。

```
MERGE INTO t ON a = 10
WHEN NOT MATCHED THEN INSERT VALUES (10, 30)
```

2.43.2.3 使用单行的条件式 Upsert 语句

在该例子中，如果 d 表里不含带有 keycol（参数指定）数值的一行，则 MERGE 语句将使用一种单行的条件式 Upsert 语句插入一行数值（keycol, col, seqnum）。否则，如果该行的 seqnum （序号）高于当前（参数指定）的序号，MERGE 语句将更新该行的 col 和 seqnum 列。如果匹配行的 seqnum 列值不高于当前的序号，则匹配行不会被更新。

```
MERGE INTO d ON keycol = ?
     WHEN MATCHED THEN UPDATE SET (col, seqnum) =
     (?, ?)WHERE seqnum <
     ?
     WHEN NOT MATCHED THEN INSERT (keycol, col, seqnum)
     VALUES (?, ?,?)
```

只有满足给定条件、希望进行更新时，*when-matched-then-update* 子句里的可选 WHERE 谓词才会得到使用。考虑这个用例。假设对象 X 被表示为 T 表里的一行，另外还存在对象 X 的更新流。

这些更新通过源序列号进行标记。然而，贯穿网络的更新流并不能保证“先入先出的交付方式”。

事实上，这些更新可能无序地到达数据库中。这种情况下，应以数据库的最后一次更新（当前序号最高的更新）为准。

上述 MERGE 语句可被用来满足这个用例：

- d 表现有的更新流将按其来源的序列号 seqnum 进行排序。
- 贯穿网络的更新可以按照任何顺序到达数据库。
- 您希望保证以数据库的最后一次更新（序号最高的更新）为准。

2.43.2.4 使用限制

- MERGE 语句不使用 ESP 并行度。
- 合并后的表不能是视图。
- 如果表里有约束，则不得进行合并操作。
- on-clause 不能包含一条子查询。不得使用该语句：

  ```
  MERGE INTO t ON a = (SELECT a FROM t1) WHEN ...
  ```

- when-matched 子句里的可选 WHERE 谓词不得包含一条子查询或一个聚合函数。不得使用这些语句：

  ```
  MERGE INTO t ON a = 10
  WHEN MATCHED THEN UPDATE SET b=4 WHERE b=(SELECT b FROM
  t1) WHEN NOT
  MATCHED THEN INSERT VALUES (10,30);

  MERGE INTO t ON a=10
  WHEN MATCHED THEN UPDATE SET b=4 WHERE b=MAX(b) WHEN NOT
  MATCHED THEN
  INSERT VALUES (10,30);
  ```

- MERGE 语句里的 UPDATE SET 子句不得包含一条子查询。不得使用该语句：
  ```
  MERGE INTO t ON a = 1
     WHEN MATCHED THEN UPDATE SET b = (SELECT a FROM t1)
  ```

- MERGE 语句里的 insert-values-list 子句不得包含一条子查询。不得使用该语句：
  ```
  MERGE INTO t ON a = 1
     WHEN NOT MATCHED THEN INSERT VALUES ((SELECT a FROM
     t1))
  ```

- 只有不存在 INSERT 子句时，才可使用 MERGE update 的非唯一性 ON 子句。
  ```
  MERGE INTO t USING (SELECT a,b FROM t1) x ON t.a=x.a
     WHEN MATCHED THEN UPDATE SET b=x.b;
  ```
  在该例子里，t.a=x.a 并非一种完全限定的唯一性主键谓词。

- 只有不存在 INSERT 子句时，才可以使用 MERGE delete 的非唯一性 ON 子句。
  ```
  MERGE INTO t USING (SELECT a,b FROM t1) x ON t.a=x.a
   WHEN MATCHED THEN DELETE;
  ```

2.43.2.5 将一个表的数据合并到另一个表中

MERGE 语句可被用来将源表的所有匹配行更新插入到目标表里。源表里的每行都将作为单条更新插入语句的来源。

using-clause 包含一条 *select-query*，其输出将被用作 MERGE 语句的来源。

必须使用 AS 子句重命名 *select-query* 来源。

```
MERGE INTO t ON
     USING (select-query) AS Z(X) ON col = Z.X
     WHEN MATCHED THEN ...
```

从该 select-query（选择查询）里选出的每行都将进行 MERGE 语句求值。所选的数值将被用于 ON 子句里，以便联接合并后表的列。如果已找到该数值，则应对其进行更新。如果没有找到该数值，则需将其插入表里。

这里的使用限制与“使用单行进行更新插入操作”的使用限制相同。

2.43.3 MERGE 例子

该查询 USING 查询里抽取派生 a 列和 b 列作为派生 z 表，并使用其中的每行基于 *on-clause*（ON 子句）联接合并后的 t 表。

对于每个匹配行，都应使用派生 z 表里的 b 列更新 t 表里的 b 列：对于不匹配的行，则将插入数值 z.a 和 z.b。

```
MERGE INTO t USING
     (SELECT * FROM t1) z(a,b) on a = z.a
     WHEN MATCHED THEN UPDATE SET b = z.b
     WHEN NOT MATCHED THEN INSERT VALUES (z.a, z.b);
```

2.44 PREPARE 语句

该语句编译一条 SQL 语句，以便之后在同一个 EsgynDB 命令界面（TrafCI）会话里和 EXECUTE 语句同时使用。

您也可以使用 PREPARE 语句检查语句的语法，而无需在同一个 TrafCI 会话里执行该语句。

```
PREPARE statement-name FROM statement
```

2.44.1 PREPARE 语法

- *statement-name*

  这是一种 SQL 标识符，它指定预处理语句所用的名称。更多信息，请参阅标识符。

  该语句名应为一个字符串、而非 Numeric（数字）数值。

  如果您指定一个现有的预处理语句的名称，则新语句将覆盖之前的语句。

- *statement*

  指定将被预处理的 SQL 语句。

2.44.2 PREPARE 注意事项

2.44.2.1 PREPARE 语句的可用性

如果一条 PREPARE 语句执行失败，则随后在已命名语句上运行 EXECUTE 语句也将失败。只有执行 PREPARE 语句的 TrafCI 会话才能在预处理语句上运行 EXECUTE 语句。

该预处理语句可用于运行 EXECUTE 语句，直到 TrafCI 会话终止为止。在语句上运行 EXECUTE 语句之前，必须通过 PREPARE 对其编译。 编译该语句后，可以在该语句上多次运行 EXECUTE 语句而无需重新编译。

2.44.3 PREPARE 例子

- 对一条 SELECT 语句进行预处理，检查是否存在语法错误：

```
SQL>prepare empsal from
+>select salary from employee
+>where jobcode = 100;

*** ERROR[4082] Table, view or stored procedureTRAFODION.SEABASE.EMPLOYEE does not exist or is inaccessible.
*** ERROR[8822] The statement was not prepared.
```

- 使用一个未命名参数（? ）对一条 SELECT 语句进行预处理，再在该语句上运行 EXECUTE 语句：

```
SQL>prepare findsal from
+>select salary from persnl.employee
+>where jobcode = ?;

--- SQL command prepared.
SQL>execute findsal using 450;
SALARY
----------
32000.00
33000.50
40000.00
32000.00
45000.00

--- 5 row(s) selected.SQL>
```

- 使用一个命名参数（? ）对一条 SELECT 语句进行预处理，再在该语句上运行 EXECUTE 语句：

```
SQL>prepare findsal from
+>select salary from persnl.employee
+>where jobcode = ?job;
--- SQL command prepared.

SQL>set param ?job 450
SQL>execute findsal;
SALARY
----------
32000.00
33000.50
40000.00
32000.00
45000.00
--- 5 row(s) selected.
```

更多信息，请参阅 EXECUTE 语句。

2.45 REGISTER USER 语句

该语句在 SQL 数据库里注册一个用户，同时将用户登录名和数据库用户名关联起来。

REGISTER USER 是 EsgynDB SQL 扩展语句。

注意：用户登录名也是目录服务里定义一个用户时所用的名称，因此，以下语法将其称作 directory-service 用户名。

```
REGISTER USER directory-service-username [ AS database-
username ]
```

2.45.1 REGISTER USER 语法

- directory-service-username

  该名称识别目录服务里的用户。它也是用户登录到 EsgynDB 数据库时指定的名称。

  directory-service-username 是一种不区分大小写字母的常规或分隔标识符。

  更多信息，请参阅不区分大小写字母的分隔标识符。

- database-username

  这是一种不区分大小写字母的常规或分隔标识符,代表数据库里定义的用户名。

  该数据库用户名既不能与已注册的数据库用户名相同、也不能与现有的角色名相同。 然而，它可以与 directory-service 用户名相同。

  如果省略 AS *database-username* 子句，该数据库用户名将与 directory-service 用户名相同。

2.45.2 REGISTER USER 注意事项

2.45.2.1 谁可以注册一个用户？

注册用户必须拥有用户管理权限。

如果您已被授予 MANAGE_ROLES 组件权限，则您将拥有用户管理权限。

最初，数据库用户只有 DB__ROOT（数据库根用户）被授予 MANAGE_USERS 组件权限。

2.45.2.2 注册用户前将其添加到目录里

注册用户前，应将其添加到适当的目录服务里。否则，REGISTER USER 语句

将执行失败。

2.45.2.3 AS database-username 子句

使用 AS database-username 子句分配一个数据库用户名，不能与目录服务里定义的用户名相同。

它特别适合于分配一个比目录服务用户名更短、且更易于输入的数据库用户名。例如：如果用户作为 John.Allen.Doe.the.Second@mycompany.com 登录，则您可以分配的数据库用户名是 JDoe。

数据库用户名是授权 ID。如果您指定一个已分配给其他用户或现有角色的名称，则该命令将运行失败。更多信息，请参阅授权 ID。

2.45.2.4 保留名称

PUBLIC、_SYSTEM、NONE 和以 DB_开头的数据库用户名都被保留。

您不能用任何此类名称注册用户。

2.45.2.5 用户名的长度

数据库用户名不能超过 128 个字符。

2.45.3 REGISTER USER 例子

- 为了注册一个用户，并分配一个不同于其他登录名的数据库用户名：
  ```
  REGISTER USER "jsmith@company.com" AS jsmith;
  ```

- 如果在不指定数据库用户名的情况下注册一个用户，则数据库用户名将与该用户的登录名相同。
  ```
  REGISTER USER "jsmith@company.com";
  ```

2.46 REVOKE 语句

该语句撤销指定用户或角色被授予的 SQL 对象访问权限。

重要提示：只有在 EsgynDB 中启用了身份验证和授权时，此语句才有效。更多信息，请参阅《EsgynDB 安装部署指南》中的启用安全功能。

```
REVOKE [GRANT OPTION FOR]
     { privilege [, privilege]...| ALL [PRIVILEGES]}
     ON [object-type] [schema.] object
     FROM { grantee [, grantee]...}
     [[GRANTED] BY grantor]
     [RESTRICT | CASCADE]

privilege 是:
     SELECT
     | DELETE
     | INSERT
     | REFERENCES
     | UPDATE
     | EXECUTE
     | USAGE

object-type 是:
     TABLE
     | PROCEDURE
     | LIBRARY
     | FUNCTION

grantee 是:
     auth-name

grantor 是:
     role-name
```

2.46.1 REVOKE 语法

- GRANT OPTION FOR
  指定被授权者将指定的权限授予其他用户或角色（即 WITH GRANT OPTION）的权力可以被撤销。这是一条可选子句。如果指定了该子句，则仅仅将取消授予权限给其他用户的能力。

- privilege [, privilege] ...| ALL [PRIVILEGES]
  指定将要撤销的权限。您可以为一个对象指定撤销以下这些权限：

SELECT	**撤销使用 SELECT 语句的能力。**
DELETE	撤销使用 DELETE 语句的能力。
INSERT	撤销使用 INSERT 语句的能力。
REFERENCES	撤销创建可以引用对象的约束的能力。
UPDATE	撤销使用 UPDATE 语句的能力。
EXECUTE	撤销使用 CALL 语句执行存储过程的能力或取消执行用户定义函数（UDF）的能力。
USAGE	撤销使用 CREATE PROCEDURE 或 CREATE FUNCTION 语句访问一个库的能力。取消该库底层库文件的读访问权限。
ALL	撤销使用该对象类型的所有适用权限的能力。 当您为一个表或视图指定 ALL 权限时，它将包含 SELECT、DELETE、INSERT、REFERENCES 和 UPDATE 权限。 如果该对象是一个存储过程或用户定义函数（UDF），则它将包含 EXECUTE 权限。 如果该对象是一个库，则它将包含 UPDATE 和 USAGE 权限。

- ON *[object-type] [schema.]object*
  指定一个将被授予权限的对象。object-type（对象类型）可以是：
  - [TABLE] [*schema.]object*，这里的 *object* 是一个表或视图。更多信息，请参阅数据库对象名称。
  - [PROCEDURE] *[schema.]procedure-name*，这里的 *procedure-name* 是数据库里注册的一个 Java 存储过程（SPJ）的名称。 更多信息，请参阅数据库对象名称。
  - [LIBRARY] *[schema.]library-name*，这里的 *library-name* 是

数据库里的一个库对象的名称。更多信息，请参阅数据库对象名称。

- [FUNCTION] *[schema.]function-name*，这里的 *function-name* 是数据库里的一个用户定义函数（UDF）的名称。更多信息，请参阅数据库对象名称。

- FROM {*grantee [,grantee]* ...}
  指定一个将被撤销权限的 *auth-name*。

- *auth-name*
  指定一个将被撤销权限的授权 ID。更多信息，请参阅授权 ID。
  授权 ID 必须是一个已注册的数据库用户名、一个现有的角色名或 PUBLIC。
  该名称是一种不区分大小写的常规或分隔标识符。更多信息，请参阅不区分大小写字母的分隔标识符。

- [GRANTED] BY *grantor*
  允许您以一个角色的名义撤销权限。如果没有指定，则将以您作为当前用户或授予者的名义撤销权限。

- *role-name*
  指定一个代理执行 GRANT 操作的角色。
  为了以一个角色的名义撤销权限，您必须是该角色的成员之一、并且该角色必须有权撤销权限，即该角色必须被授予 WITH GRANT OPTION 的权限。

- column-list
  指定一个要撤销指定权限的列的列表。

- [RESTRICT | CASCADE]
  如果您指定 RESTRICT 选项，如果已经授予了权限或基于给定的权限创建了对象，REVOKE 操作将失败。
  如果您指定 CASCADE 选项，则任何依赖的权限和对象都将作为 REVOKE 操作的一部分而被删除。
  默认值为 RESTRICT。

2.46.2 REVOKE 注意事项

2.46.2.1 授权和可用性要求

您可以撤销以您为授予者（直接或代理授予）出去的权限。

如果要撤销以一个角色的名义授予的权限，则您必须是该角色的成员之一、并

且必须在[GRANTED] BY 子句里指定该角色。

如果一个或多个权限未被授予，SQL 将返回一个警告。

如果您指定 CASCADE 选项，则所有基于该撤销权限而创建的对象都将被删除。

2.46.3 REVOKE 例子

- 撤销一个用户对于一个表的 SELECT 和 DELETE 权限的授予能力：
  `REVOKE GRANT OPTION FOR SELECT, DELETE ON TABLE invent.partloc FROM jsmith;`

- 撤销一个用户/角色对于一个表的 SELECT 和 DELETE 权限的授予能力：
  `REVOKE GRANT OPTION FOR SELECT, DELETE ON TABLE invent.partloc FROM jsmith, clerks;`

- 撤销一个用户对一个表的 SELECT 权限：
  - 用户管理员将 SELECT 权限授予给 JSMITH：
    ```
    GRANT SELECT ON TABLE invent.partloc TO jsmith
    WITH GRANT OPTION;
    ```
  - JSMITH 将 SELECT 权限授予给 AJONES：`GRANT SELECT ON TABLE invent.partloc TO ajones;`
  - 如果用户管理员尝试删除 JSMITH 的 SELECT 权限，则该操作将会失败，因为授予 AJONES 的权限是基于授予 JSMITH 的权限而定的。
    为了成功地撤销 JSMITH 的 SELECT 权限，必须首先撤销 AJONES 被授予的 SELECT 权限。对于该例子而言：
    1) JSMITH 撤销 AJONES 被授予的 SELECT 权限：
       `REVOKE SELECT ON TABLE invent.partloc FROM ajones;`
    2) 用户管理员撤销 JSMITH 对于该表的 SELECT 权限：
       ```
       REVOKE SELECT ON TABLE invent.partloc FROM jsmith
       RESTRICT;
       ```
       REVOKE 操作成功。

- 使 REVOKE 操作成功的一种更简单的方法是使用 CASCADE 选项：
  `REVOKE SELECT ON TABLE invent.partloc FROM jsmith CASCADE;`

  REVOKE 操作将成功，因为 CASCADE 选项可以撤销所有指定的权限和所有基于指定权限而授予的权限。

- 运输部门的管理层将决定CLERKS角色不再能授予invent.partloc表的权限。最近 Fred 已经调到了另一个部门，因此 JSMITH 将撤销 Fred 对于 invent.partloc 表的 SELECT 权限，该权限由 CLERKS 角色授予。然后，JSMITH将撤销CLERKS角色的GRANT OPTION（授予选项）：

```
REVOKE SELECT on table invent.partloc FROM fred GRANTED
BY clerks;

REVOKE GRANT OPTION FOR SELECT ON TABLE invent.partloc
FROM
clerks;
```

2.47 REVOKE COMPONENT PRIVILEGE 语句

该语句删除一个用户或角色所拥有的一个或多个组件权限。更多信息，请参阅权限和角色。

REVOKE COMPONENT PRIVILEGE 是 EsgynDB SQL 扩展语句。

重要提示：只有在 EsgynDB 中启用了身份验证和授权时，此语句才有效。更多信息，请参阅《EsgynDB 安装部署手册》中的 Enabling Security Features（启用安全功能）。

```
REVOKE [GRANT OPTION FOR]
     COMPONENT PRIVILEGE { privilege-name[, privilege-
     name]...}
     ON component-name
     FROM grantee
     [[GRANTED] BY grantor]

grantee is:
     auth-name

grantor is:
     role-name
```

2.47.1 REVOKE COMPONENT PRIVILEGE 语法

- GRANT OPTION FOR
- 撤销被授权者将指定权限授予其他用户或角色（即 WITH GRANT OPTION）的权力。这是一条可选子句。
  如果指定了该子句，那么只有向其他用户授予组件权限的能力将被撤销。

- *privilege-name*
  指定一个或多个将撤销的组件权限。逗号分隔的列表仅可以包含同一个组件内的权限。

- ON *component-name*
  指定一个将被撤销组件权限的有效的组件名称。目前唯一有效的组件名称是 SQL_OPERATIONS。

- FROM *grantee*
  指定一个将被撤销组件权限的 auth-name。

- *auth-name*
  指定一个将被撤销权限的授权 ID。更多信息，请参阅授权 ID。
  授权 ID 必须是一个已注册的数据库用户名、一个现有角色名或 PUBLIC。
  该名称是一种不区分大小写的常规或分隔标识符。更多信息，请参阅不区分大小写字母的分隔标识符。

- [GRANTED] BY *grantor*
  允许您以一个角色的名义撤销组件权限。如果没有指定，组件权限将以您作为当前用户或授予者的名义撤销。

- *role-name*
  指定一个代理执行 GRANT COMPONENT PRIVILEGE 操作的角色。
  为了以一个角色的名义撤销权限，您必须是该角色的成员之一，并且该角色必须有权撤销权限，即该角色必须被授予带有 WITH GRANT OPTION 的权限。

2.47.2 REVOKE COMPONENT PRIVILEGE 注意事项

- 撤销权限时，被授予 WITH GRANT OPTION 的所有权限都将被删除。即该撤销行为是 CASCADE（级联）。
- 如果没有授予任何组件权限，则 SQL 将返回一个错误。
- 如果没有授予一个或多个组件权限，则 SQL 将会静默忽略这些权限，并继续进行撤销操作。
- 必须在删除角色或注销用户之前撤销组件权限。如果一个角色或用户已被授予任何权限，则删除该角色或注销该用户时将返回一个错误。更多信息，请参阅 DROP ROLE 语句和 UNREGISTER USER 语句。

2.47.2.1 授权和可用性要求

您可以撤销作为授权者（直接或代理授予）授予的组件权限。

如果要撤销以一个角色的名义授予的权限，则您必须是该角色的成员之一，并且必须在[GRANTED] BY 子句里指定该角色。

2.47.3 REVOKE COMPONENT PRIVILEGE 例子

- 撤销 SQLUSER1 的组件权限:

```
REVOKE COMPONENT PRIVILEGE CREATE_TABLE ON
SQL_OPERATIONS FROM sqluser1;
```

2.48 REVOKE ROLE 语句

该语句删除一个用户所拥有的一个或多个角色。更多信息，请参阅角色。

重要提示：只有在 EsgynDB 中启用了身份验证和授权时，此语句才有效。更多信息，请参阅《EsgynDB 安装部署手册》中的 Enabling Security Features（启用安全功能）。

```
REVOKE ROLE { role-name [, role-name]...}
     FROM grantee
     [RESTRICT | CASCADE]

grantee is:
     database-username
```

2.48.1 REVOKE ROLE 语法

- *role-name* [, role-name] ...

  指定要删除的有效角色。

- FROM *grantee*

  指定您希望撤销其角色的已注册数据库用户名。

- [RESTRICT | CASCADE]

  如果您指定 RESTRICT 选项，如果已经授予了权限或基于给定的权限创建了对象，REVOKE 操作将失败。

  如果您指定 CASCADE 选项，则任何依赖性权限和对象都将作为 REVOKE ROLE 操作的一部分而被删除。

  默认值为 RESTRICT。

2.48.2 REVOKE ROLE 注意事项

- 为了撤销用户的角色，您必须拥有这些角色或具有这些角色的用户管理权限。如果您已被授予 MANAGE_ROLES 组件权限，则将具有这些角色的用户管理权限。 最初，数据库用户里只有 DB__ROOT（数据库根用户）被授予 MANAGE_USERS 组件权限。
- 如果已经指定 RESTRICT 选项（或未指定任何选项），并且如果您希望撤销一个用户的角色（该用户仅根据角色权限创建了对象）则您必须在撤销该角

色之前删除该用户所创建的对象。然而，如果您指定了 CASCADE 选项，依赖性对象将被自动删除、角色将被撤销。

- 所有指定的角色必须都已授予指定的用户。 如果还未授予该用户任何角色，则操作时将返回一个错误，并且不会撤销任何角色。
- 当您撤销一个用户的角色时，权限减少情况将被自动检测以及自动传播给活动会话。 用户无需断开再重新连接会话，即可看到更新后的权限集。
- 如果 REVOKE ROLE 语句里列出了多个角色，并且处理期间出现了任何错误，则不会执行任何撤销操作。

2.48.3 REVOKE ROLE 例子

- 为了撤销一个用户的多个角色:

```
REVOKE ROLE clerks, sales FROM jsmith;
```

- 为了撤销一个用户的一个带有从属对象的角色:
  - CMILLER 向 AJONES 授予一个角色:

    ```
    GRANT ROLE sales TO ajones;
    ```

  - CMILLER 向该角色授予一种权限:

    ```
    GRANT SELECT ON TABLE invent.partloc TO sales;
    ```

  - AJONES 基于其被授予角色的授予权限而创建一个视图:

    ```
    CREATE VIEW invent.partlocview (partnum, loc_code)
    AS SELECT partnum, loc_code FROM invent.partloc;
    ```

  - 如果 CMILLER 尝试撤销 AJONES 的角色，则该操作将失败，因为该视图是基于 AJONES 被授予角色的授予权限而创建。

    CMILLER 使用带有 CASCADE 选项的语句撤销 AJONES 的角色:

    ```
    REVOKE ROLE sales from AJONES CASCADE;
    ```

    REVOKE ROLE 操作成功，所有从属对象权限都被撤销。

2.49 ROLLBACK WORK 语句

该语句在当前的事务执行期间撤销所有数据库对象更改，并结束该事务。更多信息，请参阅事务管理。

2.49.1 ROLLBACK WORK 语法

```
ROLLBACK [WORK]
```

WORK 是一个没有效果的可选关键字。

活动状态事务之外发布 ROLLBACK WORK 语句时将返回成功。

2.49.2 ROLLBACK WORK 注意事项

2.49.2.1 开始和结束一个事务

BEGIN WORK 语句启动一个事务，COMMIT WORK 或 ROLLBACK WORK 结束一个事务。

2.49.3 ROLLBACK WORK 例子

假设您添加了一个订单到 ORDERS 和 ODETAIL 表里，旨在订购两个编号为 4130 的部件。 当您更新 PARTLOC 表、以减少可用数量时，您会发现给定的位置已经不存在这个部件编号。

使用 ROLLBACK WORK 语句终止该事务，同时不提交数据库更改:

```
BEGIN WORK;
INSERT INTO sales.orders
     VALUES (124, DATE '2007-04-10',
          DATE '2007-06-10', 75, 7654);
INSERT INTO sales.odetail
     VALUES (124, 4130, 25000, 2);
UPDATE invent.partloc
     SET qty_on_hand = qty_on_hand - 2
     WHERE partnum = 4130 AND loc_code = 'K43';
ROLLBACK WORK;
```

ROLLBACK WORK 语句取消事务执行期间的插入和更新操作。

2.50 SAVEPOINT 语句

使用 SAVEPOINT 语句标识事务中可以稍后回滚的点。

2.50.1SAVEPONT 语法

BEGIN SAVEPOINT

创建保存点。

ROLLBACK SAVEPOINT

指定回滚到保存点。

COMMIT SAVEPOINT

提交保存点。

2.50.2SAVEPOINT 注意事项

SAVEPOINT 受 CQD TRAF_SAVEPOINTS 开关控制。只有在开关值为 ON 的时候才起效，开关的默认值是 OFF。
CQD TRAF_SAVEPOINTS ‘ON’ 用来打开该开关。

2.50.3SAVEPOINT 例子

- 以下例子显示如何使用 COMMIT SAVEPOINT

```
begin work;
update staff set salary=5000;
begin Savepoint;
update staff set salary=1000 where last_name='Tom';
commit savepoint;
commit;
```

- 以下例子显示如何使用 ROLLBACK SAVEPOINT

```
begin work;
update staff set salary=5000;
begin Savepoint;
update staff set salary=1000 where last_name='Tom';
rollback savepoint;
commit;
```

2.51 SELECT 语句

SELECT 语句是一种 DML 语句，它能在表、视图、通过查询表达式求值的派生表或联接表里检索数据。

```
sql-query is:
     query-specification
     | query-expr-and-order
     [for_update_clause]

query-specification is:
[with-clause]
SELECT [ "[" ANY N "]" | "[" FIRST N "]" ] | "[" LAST N
"]" ] [ALL | DISTINCT] select-list
     FROM table-ref [,table-ref]...
     [WHERE search-condition]
     [SAMPLE sampling-method]
     [TRANSPOSE transpose-set [transpose-set]...
          [KEY BY key-colname]]...
     [START WITH condition ]
     [CONNECT BY [ NOCYCLE ] condition]
     [GROUP BY {colname | colnum} [,{colname | colnum}]...]
     [HAVING search-condition]
     [access-clause ]
     [mode-clause]

With-clause is:
WITH with_clause_elements

with_clause_elements is:
with_clause_element [,with_clause_element] …

with_clause_element is:
cte-table-name AS (sql-query)

query-expr-and-order is:
     query-expr [order-by-clause] [access-clause] [mode-
     clause]

query-expr is:
     query-primary
     | query-expr UNION [ALL] query-primary
     | query-expr INTERSECT query-primary
     | query-expr EXCEPT query-primary

query-primary is:
     simple-table | (query-expr)

simple-table is:
     VALUES (row-value-const) [,(row-value-const)]...
```

```
     | TABLE table
     | query-specification

row-value-const is:
     row-subquery
     | {expression | NULL} [,{expression | NULL}]...

order-by-clause is:
     [ORDER BY {colname | colnum} [ASC[ENDING] |
     DESC[ENDING]] [NULLS FIRST|NULLS LAST]
          [,{colname | colnum} [ASC[ENDING] |
          DESC[ENDING]]]...]
     [access-clause]

access-clause is:
     [FOR] access-option ACCESS


access-option is:
     READ COMMITTED

[LIMIT num]

select-list is:
     *    | select-sublist [,select-sublist]...

select-sublist is:
     corr.* | [corr.]single-col [[AS]name] | col-expr [[AS]
     name]

table-ref is:
     table [[AS] corr [(col-expr-list)]]
     | view    [[AS] corr [(col-expr-list)]]
     | (query-expr) [AS] corr [(col-expr-list)]
     | (delete-statement [RETURN select-list])
          [AS] corr [(col-expr-list)]
     | (update-statement [RETURN select-list])
          [AS] corr [(col-expr-list)]
     | (insert-statement) [AS] corr [(col-expr-list)]
     | joined-table

joined-table is:
     table-ref [join-type] JOIN table-ref join-spec
     | table-ref NATURAL [join-type] JOIN table-ref
     | table-ref CROSS JOIN table-ref
     | (joined-table)

join-type is:
     INNER | LEFT [OUTER] | RIGHT [OUTER] | FULL [OUTER]

join-spec is:
```

```
     ON search-condition

sampling-method is:
     RANDOM percent-size
     | FIRST rows-size
          [SORT BY colname [ASC[ENDING] | DESC[ENDING]]
               [,colname [ASC[ENDING] | DESC[ENDING]]...]
     | PERIODIC rows-size EVERY number-rows ROWS
          [SORT BY colname [ASC[ENDING] | DESC[ENDING]]
               [,colname [ASC[ENDING] | DESC[ENDING]]...]

percent-size is:
     percent-result PERCENT [ROWS]
     | BALANCE      WHEN condition
               THEN percent-result PERCENT [ROWS]
               [WHEN condition THEN percent-result PERCENT
               [ROWS]]...  [ELSE  percent-result  PERCENT
               [ROWS]] END

rows-size is:
     number-rows ROWS
     | BALANCE WHEN condition THEN number-rows ROWS
          [WHEN condition THEN number-rows ROWS]...
          [ELSE number-rows ROWS] END

transpose-set is:
     transpose-item-list AS transpose-col-list

transpose-item-list is:
     expression-list  |  (expression-list)  [,(expression-
     list)]...

expression-list is:
     expression [, expression]...

transpose-col-list is:
     colname | (colname-list)

colname-list is:
     colname [, colname]...

for_update_clause is:
     FOR UPDATE
```

2.51.1 SELECT 语法

- with-clause

  WITH子句，即公共表表达式（CTEs）或子查询因子子句，在SQL-99标准

中引入，并已实现。

WITH子句为复杂表达式指定别名。别名被视为临时表或内联视图，仅在查询期间有效，并且可以在同一查询中多次引用。

通过将查询的复杂部分抽象为更简单、独立、具有逻辑的块，并可能具体化这些部分的结果以避免多次重新计算，WITH子句具有以下优点:

- 简化复杂的查询，提高可读性，减少重复引用。
- 构建可重复使用的单元并降低维护成本。
- 缩短响应时间并提高查询性能。
- 提高与其他支持WITH子句的系统的兼容性。

- cte-table-name

  指定要创建的CTE的唯一名称，该名称是一个有效SQL标识符，最长128个字符。单个WITH子句中不允许有重复的名称。

- "[" ANY N "]" | "[" FIRST N "]"[" LAST N "]" `

  指定返回 N 行（假设该表至少有 N 行，并且 WHERE 子句里指定了限定条件。如果存在，将至少选择 N 行）。您并不关注将从限定行里选出、并实际返回哪些（N）行。

  ANY N 或 FIRST N 必须用方括号（[]）括起来。上述语法描述中每个方括号外面的双引号（""）表示该括号是语法的一部分（例如：[ANY 10]或[FIRST 5]）。无需在 ANY 或 FIRST 子句里包含引号。

  只有当您在选择列表的任何一列上使用 ORDER BY 对 SELECT 语句的结果表进行排序时，[FIRST N]才不同于[ANY N]。N 是一个无标度、无符号的 Numeric（数字型）字面值。如果 N 大于一个表的行数，则所有的行都将被返回。[ANY N]和[FIRST N]不得出现在嵌套的 SELECT 语句里和 UNION 操作的两侧。

  [LAST N] 执行整个查询并计算运行时间。N 必须是 0 或 1。[LAST 0] 不返回任何行。[LAST 1] 只返回最后一个限定行。

- ALL | DISTINCT

  指定是否检索所有的行（其列由选择列表 [ALL] 指定），或仅检索没有重复（DISTINCT）的行。在删除重复行时空值时当作彼此相等的。默认值为 ALL。

- *select-list*

  在 FROM 子句中引用的表中选定的列或列表达式。

  - *

    FROM 子句中的表、视图、联接表或通过查询表达式求值的派生表的所有列。

  - *corr.**

    指特定表引用的所有列。在 FROM 子句中，该特定表引用通过使用关联名 corr 的表引用。

  - *corr.single-col* [[AS]name]

    在 FROM 子句中，使用关联名的一列或特定表引用。

    AS 子句可以将列和关联名相关联。name 是一种 SQL 标识符。更多信息，请参阅标识符。

  - *single-col* [[AS]name]

    表示一列。AS 子句可以将列和关联名相关联。name 是 SQL 标识符。更多信息，请参阅标识符。

  - *col-expr* [[AS]name]

    表示列表中由 SQL 数值表达式计算出的派生列。AS 子句可以将列和关联名相关联。name 是一种 SQL 标识符。

- FROM *table-ref [,table-ref]...*

  指定一些表、视图、派生表或联接表的列表，这些表、视图、派生表或联接表决定中间结果表的内容。中间结果表来自您在 select-list 中已指定、并由 EsgynDB SQL 返回的列。

如果您仅指定了一个 table-ref，则中间结果表将包含表引用的派生行。 如果您指定了多个 table-ref，则中间结果表将包含单个表引用所派生的结果表的交叉积。

Δ *table [[AS]corr [(col-expr-list)]] |view [[AS]corr [(col-expr-list)]] | (query-expr) [AS]corr [(col-expr-list)] | (delete-statement [RETURN select-list]) [AS]corr [(col-expr-list)] | (update-statement [RETURN select-list]) [AS]corr [(col-expr-list)] | (insert-statement) [AS]corr*

[(col-expr-list)] |joined-table

指定 *table-ref* 作为单表、视图、通过查询表达式求值确定的派生表或一个联接表。

您可以为一个表或视图指定该可选子句。对于派生表，该子句是必需的：

[AS]*corr [(col-expr-list)]*

为在 FROM 子句中前面的表引用 table-ref 指定一个关联名 *corr*。

Δ *col-expr [[AS]name] [,col-expr [[AS]name]]...*

指定派生列列表 *col-expr-list* 中的各项。

AS 子句可以将一个派生列 *col-expr* 和一个关联名（*name*）相关联。*name* 是一种 SQL 标识符。

Δ (*delete-statement [RETURN select-list]) [AS]corr [(col-expr-list)]*

使一个应用程序能通过单一操作读取并删除行。

更多关于 delete-statement（DELETE 语句）语法的信息，请参阅 DELETE 语句。

- RETURN *select-list*

  从已删除的行返回多列或单列表达式。Select-list 的各项可以具有下列形式：

  ◊ [OLD.]*

  嵌入式删除操作所暴露的旧表里的行。该旧表引用了删除

之前的列值。不得使用 NEW。没有指定列表的删除操作都有一个隐式 OLD.*返回列表。

◊ *col-expr* [[AS]*name*]

指定通过列表里的 SQL 数值表达式求值确定的一个派生列，一个数值表达式里引用的任何列都来自删除操作所暴露的旧表。旧表引用了删除之前的列值。

AS 子句可以将一个派生列 *col-expr* 和一个关联名（*name*）相关联。

- [AS]*corr [(col-expr-list)]*

  指定关联名 *corr* 和 *select-list* 中前述项目的可选列列表。

Δ *(update-statement* [RETURN*select-list*]) [AS]*corr [(col-expr-list)*]

使一个应用程序能通过单一的操作读取和更新行。更多关于 update-statement（UPDATE 语句）语法的信息，请参阅 UPDATE 语句。

- RETURN *select-list*

  指定从已更新的行返回多列或单列表达式。select-list 的各项可以具有以下形式:

  ◊ [OLD.|NEW.]*

  指定更新操作所暴露的旧表或新表里的行。旧表引用了更新操作前的列值，而新表则引用了更新操作后的列值。如果一列尚未更新，则新值与旧值相等。

  如果更新操作未指定返回列表，这它有隐式 NEW.*返回列表。

  ◊ *col-expr* [[AS]*name*]

  指定通过列表里的 SQL 数值表达式求值确定的一个派生列。数值表达式里引用的任何列都可被指定来自更新操作所暴露的旧/新表里的一行。

例如：RETURN
old.empno,old.salary,new.salary,
(new.salary - old.salary).
AS 子句可以将一个派生列 *col-expr* 和一个名称（name）相关联。*name* 是一种 SQL 标识符。更多信息，请参阅标识符。

- [AS]*corr*[(*col-expr-list*)]
  指定关联名 corr 和 select-list 中前述项目的可选列列表。
  例如：
  RETURN old.empno,old.salary,new.salary,
  (new.salary-old.salary)
  AS emp (empno, oldsalary, newsalary, increase).

Δ (*insert-statement*)[AS]*corr [(col-expr-list*)]
更多关于 insert-statement 语法的信息，请参阅 INSERT 语句。
[AS]*corr [(col-expr-list)]*
指定关联名称 corr 和可选的列列表。

Δ *joined-table*
joined-table 可以被指定为：
table-ref [join-type] JOIN table-ref join-spec
| table-ref NATURAL [join-type] JOIN table-ref
| table-ref CROSS JOIN table-ref
| (joined-table)
join-type is:INNER | LEFT [OUTER] | RIGHT [OUTER]
| FULL [OUTER]
这是一个联接表。通过 CROSS、INNER、OUTER、LEFT、RIGHT 和 FULL 关键字指定 join-type。
如果省略可选的 OUTER 关键字、并在联接查询里使用 LEFT、RIGHT 或 FULL 关键字，则 EsgynDB SQL 将默认该联接是一个外联接。
如果指定 join-type 是 CROSS join，则不可指定 NATURAL join 或 join-spec。
如果指定 *join-type* 是 INNER、LEFT、RIGHT 或 FULL join，并且没有指定 NATURAL join，则必须使用 ON 子句作为 *join-spec*，FULL OUTER JOIN 的联接谓词里不得出现子查询。

- ON *search-condition*

  指定联接查询的 *search-condition*。

  search-condition 里的每个列引用都必须是 JOIN 关键字的左/右侧的表引用所派生的两个结果表里的一列。如果这些行都满足该条件，则将出现结果表里的两行联接。

  所使用的联接类型和联接规范能确定通过两个表引用联接的行，详情如下：

  ◊ *table-ref* CROSS JOIN *table-ref*

  将左侧的 *table-ref* 表引用）的各行与右侧 *table-ref* 的各行联接起来。

  ◊ *table-ref* NATURAL JOIN *table-ref*

  仅联接两个表里所有具备匹配数值的同名列所涉及的行。该选项等效于 NATURAL INNER。

  ◊ *table-ref* NATURAL LEFT JOIN *table-ref*

  联接两个表里所有具备匹配数值的同名列所涉及的行、以及不满足该条件的左侧 *table-ref* 里的行。

  ◊ *table-ref* NATURAL RIGHT JOIN *table-ref*

  联接两个表里所有具备匹配数值的同名列所涉及的行、以及不满足该条件的右侧 *table-ref* 里的行。

  ◊ *table-ref* NATURAL FULL JOIN *table-ref*

  联接两个表里所有具备匹配数值的同名列所涉及的行、以及不满足该条件的任何一侧的 *table-ref* 里的行；对于缺失的数值应填 NULL。

  ◊ *table-ref* JOIN *table-ref join-spec*

  仅联接满足 *join-spec* 子句里条件的行。该选项等效于 INNER JOIN ... ON。

  ◊ *table-ref* LEFT JOIN *table-ref join-spec*

联接满足 *join-spec* 子句里条件的行、以及不满足该条件的左侧 *table-ref* 里的行。

◊ *table-ref* RIGHT JOIN *table-ref join-spec*
联接满足 *join-spec* 子句里条件的行、以及不满足该条件的右侧 *table-ref* 里的行。

◊ *table-ref* FULL OUTER JOIN *table-ref join-spec*
结合左外联接和右外联接的查询结果。这些联接显示两个表里的记录，并在任何一侧的匹配缺失处填写 NULL。

Δ *simple-table*
一个 simple-table 可以被指定为：

```
VALUES (row-value-const) [,(row-value-const)]...
| TABLE table
| query-specification
```

一个 *simple-table* 可以是一个表值构造函数。它以 VALUES 关键字开头，后面跟有一个行值构造函数序列，每个构造函数都用括号括起来。
row-value-const 是一个表达式列表（或 NULL）或一条行子查询（返回一行列值的子查询）。
一个表达式的操作数无法引用一个列（除非该操作数是一个可以在结果表里返回单列值的标量子查询）。
EsgynDB SQL 扩展语句里使用 NULL 作为 *row-value-const* 元素。
使用表明后的 TABLE 关键字表名指定一个 *simple-table*，这等效于查询规范 SELECT * FROM table。

simple-table 可以成为一种查询规范，即一条含有 SELECT ...FROM ...的 SELECT 语句带有可选的 WHERE、SAMPLE、TRANSPOSE、GROUP BY 和 HAVING 子句。

Δ WHERE *search-condition*
指定一条联接查询的 *search-condition*。更多信息，请参阅搜索条件。

WHERE 子句里不得包含一个聚合（集合）函数。

search-condition 适用于 FROM 子句的表引用所派生的结果表的各行或多表引用里的单个表引用所派生的结果表的交叉积。

您在 *search-condition* 里指定的每列通常是中间结果表里的一列。如果存在用于提供比较值的嵌套子查询，该列也可以是一个外部引用。更多信息，请参阅子查询。

为了遵循 ANSI 标准，EsgynDB SQL 既不会将聚合谓词从 WHERE 子句移至 HAVING 子句，也不会将非聚合谓词从 HAVING 子句移至 WHERE 子句。

Δ SAMPLE *sampling-method*

指定一个抽样方法，用于选择 SELECT 语句中间结果表的一个子集。每种方法都使用相同的抽样大小。三种抽样方法为随机抽样、首次抽样和定期抽样，它们被指定为：

- RANDOM *percent-size*

  指导 EsgynDB SQL 随机地选择行（每行被选概率相同），不会从结果表中重复取样。通过使用结果表的百分比确定抽样大小。

- FIRST *rows-size* [SORT BY *colname [,colname]...]*

  指导 EsgynDB SQL 从排序后的结果表里选择第一个 rows-size 行。通过使用指定的行数确定抽样大小。

- PERIODIC *rows-size* EVERY *number-rows* ROWS [SORT BY *colname [,colname]*...]

  指导 EsgynDB SQL 从每个块（周期）的连续排序行里选择第一行。通过使用从每个块里选取的指定行数确定抽样大小。

- SAMPLE

  它是 EsgynDB SQL 扩展语句。更多信息，请参阅 SAMPLE 子句。

Δ TRANSPOSE *transpose-set[transpose-set]*...[KEY BY *key-colname]*

指定 TRANSPOSE 子句里的 *transpose-sets* 和可选的 KEY 子句。可以在一条 SELECT 语句里使用多条 TRANSPOSE 子句。

- *transpose-item-list* AS *transpose-col-list*
  指定一个 *transpose-set*。可以在一条 TRANSPOSE 子句里使用多个转置数据集。
  TRANSPOSE 子句可以为 FROM 子句里的表引用所派生的源表的每行都生成一行数据，用于所有转置数据集的每个 *transpose-item-list* 里的各项。
  TRANSPOSE 子句的结果表里包含源表的所有列、加上所有转置数据集的每个 *transpose-item-list* 里指定的一个或多个值列、以及一个可选的主键字段 *key-colname*。

- KEY BY *key-colname*
  选择指定可选的主键字段 *key-colname*。它根据表达式在 *transpose-item-list* 里的位置标识其在“转置列列表”里的相应数值。*key-colname* 是一种 SQL 标识符。 精确的数据类型为 Numeric，数值为 NOT NULL。

- TRANSPOSE
  它是 EsgynDB SQL 扩展语句。更多信息，请参阅 TRANSPOSE 子句。

Δ [START WITH condition]
[CONNECT BY [NOCYCLE] condition]

如果表包含分层数据，则可以使用分层查询子句以分层顺序选择行。

START WITH 指定层次结构的根所在的行。

CONNECT BY 指定层次结构的父行和子行之间的关系。

即使数据中存在 CONNECT BY LOOP，NOCYCLE 参数也会指示数据库从查询中返回行。

在分层查询中，必须使用 PRIOR 运算符限定条件中的一个表达式以引用父行。 例如，

```
... PRIOR expr = expr
or
... expr = PRIOR expr
```

如果 CONNECT BY 条件是复合的，那么只有一个条件需要 PRIOR 运

算符，尽管您可以有多个 PRIOR 条件。 例如：

```
CONNECT BY last_name != 'King' AND PRIOR
employee_id = manager_id ...
CONNECT BY PRIOR employee_id = manager_id and
           PRIOR account_mgr_id = customer_id ...
```

PRIOR 是一元运算符，与一元+和-算术运算符具有相同的优先级。它为分层查询中当前行的父行计算紧接着的表达式。

EsgynDB 暂不支持 CONNECT_BY_ROOT。

Δ GROUP BY *[col-expr] {colname |colnum} [,{colname |colnum}]...]*

指定分组列，以为 SELECT 语句的结果表定义一个组集。GROUP BY 子句里的表达式必须与选择列表里的表达式完全相同。这些列必须出现在 SELECT 语句的 FROM 子句里的表引用的“字段列表”里。

如果语句包含一个 GROUP BY 子句，则在 *select-list* 里引用的列都必须是分组列或一个聚合（或集合）函数的参数。

这些分组列定义一个组集，其中的每个组都包含指定列具有相同数值的行。列名可以通过表名、视图名或关联名进行限定。例如：CUSTOMER.CITY。

例如：如果您指定 AGE，则结果表里将包含一组 AGE 等于 40 的行和一组 AGE 等于 50 的行。如果指定了 AGE 和 JOB，则结果表将包含各年龄所属的组、以及各年龄组内每个工作代码的子组。

可以指定 GROUP BY 子句使用序号引用 SELECT 列表内的相关位置。例如：GROUP BY 3, 2, 1。

为了进行分组，所有空值都将被视为彼此相等。一个 GROUP BY 子句的结果表只能有一个空组。更多信息，请参阅 GROUP BY 子句注意事项。

Δ HAVING *search-condition*

指定 *search-condition* 用于 SELECT 语句里的前述 GROUP BY 子句所产生的分组表的各组。

为了遵循 ANSI 标准，EsgynDB SQL 既不会将聚合谓词从 WHERE 子句移至 HAVING 子句，也不会将非聚合谓词从 HAVING 子句移至

WHERE 子句。

如果不存在 GROUP BY 子句，则 *search-condition* 将适用于整个表（包含 WHERE 子句所产生的一组）或（FROM 子句所产生的一组，如果不存在 WHERE 子句）。

在 *search-condition* 里，可以指定任何列作为一个聚合（或集合）函数的参数；例如：AVG (SALARY)。一个聚合函数适用于分组表里的各组。

非聚合函数参数的列必须是分组列。当引用分组列时，您正在引用一个单值，因为该组里的每行的分组列都包含相同的数值。更多信息，请参阅搜索条件。

Δ [FOR] *access-option* ACCESS

当访问通过 SELECT 语句或表引用指定的数据时（该表引用在由查询表达式［即 SELECT 语句］求值所派生的 FROM 子句中），可以指定 *access-option*。更多信息，请参阅数据一致性和访问选项。

Δ READ COMMITTED

指定任何访问数据都必须来自已提交的行。

Δ UNION [ALL]*select-stmt*

指定一条 SELECT 语句的结果表和另一条 SELECT 语句的结果表之间的并集操作。

合并操作可以得到一个表，该表的行只属于两个有效表其中的一个。

如果指定 UNION ALL，则该表将包含所有通过 SELECT 语句检索到的行，否则将删除重复行。

一个合并操作的两条 SELECT 语句里的选择列表必须具有相同的列数，并且列表中相应位置的列必须具备兼容的数据类型。Select list 前不得有[ANY N]或[FIRST N]。

合并操作的结果表的列数与每个选择列表的列数相同。合并操作的结果表的列名与左表 SELECT 语句的选择列表里的相应名称相同。

表达式或常量的合并操作所产生的列名[EXPR]。更多信息，请参阅 UNION 注意事项。

Δ `select-stmt INTERSECT select-stmt`

指定一条SELECT语句的结果表与另一条SELECT语句的结果表之间的set intersect操作。INTERSECT操作的结果是一个由同时出现在两个结果集中的行组成的表。INTERSECT的结果不包含任何重复行。一个INTERSECT操作的两条SELECT语句里的选择列表必须具有相同的列数，并且列表中相应位置的列必须具备兼容的数据类型。选择列表前面不得有[ANY N]或[FIRST N]。

Δ `select-stmt EXCEPT select-stmt`

指定一条 SELECT 语句的结果表和另一条 SELECT 语句的结果表之间的 set except 操作。EXCEPT 操作的结果是一个表，该表的行在 left SELECT 语句的结果里，不在 right SELECT 语句的结果里。

EXCEPT 的结果不包含任何重复行。

EXCEPT 操作的两条 SELECT 语句里的 select 列表必须具有相同的列数，并且列表中相应位置的列必须具备兼容的数据类型。select 列表前面不得有[ANY N] or [FIRST N]。

Δ `ORDER BY {colname | colnum} [ASC[ENDING] | DESC[ENDING]] [NULLS FIRST|NULLS LAST] [,{colname | colnum} [ASC[ENDING] | DESC[ENDING]]]...`

指定最终结果表的行的排序。

- *colname*

  命名 *select-list* 里的一列或 SELECT 语句的 FROM 子句的表引用里的一列。*colname* 可以通过表名、视图名或关联名进行限

定。例如：CUSTOMER.CITY。 如果一列已有别名，则必须使用该别名。

- *colnum*

  根据 *select-list* 里位置指定一列。使用 *colnum* 引用那些未命名列，例如派生列。

- ASC|DESC

  指定排序。默认值为 ASC。如果按照一个可能包含空值的列对结果表进行排序，这些空值可被视为彼此相等、但均大于其他所有非空值。更多信息，请参阅 ORDER BY 注意事项。

- NULLS FIRST|NULLS LAST

  为实现与 Oracle 兼容，支持该语法，指定返回的包含空值的行在排序中是出现在最前面还是最后面。NULLS LAST 是升序的默认值，NULLS FIRST 是降序的默认值。

Δ LIMIT num

限制一条查询所返回的行数；如果 *num* 为空或小于零，则不存在限制。LIMIT 子句在 ORDER BY 子句之后执行，以便支持 TopN 查询。

- for_update_clause

  FOR UPDATE 子句允许您锁定所选行，以便在您结束事务之前，其他用户无法锁定或更新行。只能在顶级 SELECT 语句中指定此子句，不能在子查询中指定。

2.51.2 SELECT 注意事项

2.51.2.1 授权要求

执行 SELECT 语句必需有权读取该语句里引用的所有视图和表，包括该语句里被引用视图的底层表。

2.51.2.2 在 SELECT 语句里使用视图

如果在 SELECT 语句里引用一个视图，则该视图的定义规范将与该语句结合起

来。这种结合可能导致 SELECT 语句无效。

如果您收到一条错误消息指示存在问题，但 SELECT 语句似乎是有效的，请检查视图定义。

例如，假设名为 AVESAL 的视图包含一个被定义为 AVG (X)的 A 列，选择列表里包含 MAX (A)的 SELECT 语句是无效的，因为选择列表里实际包含的是 MAX (AVG (X))，并且一个聚合函数不得带有另一个聚合函数所包含的参数。

2.51.2.3 联接限制

注意：建议将一条联接查询里的表数量设置为最多为64个，包括联接操作里被引用视图的基表。涉及大量表的联接查询将无法保证编译。

2.51.2.4 SELECT 语句里的对象名

只能在 SELECT 语句的 FROM 子句里使用完全限定的名称。

2.51.2.5 AS 和 ORDER BY 的冲突

如果使用 AS 动词对 SELECT 语句里的一列重命名，并且 ORDER BY 子句使用初始列名，则查询将失败。

如果一列已重命名为另一个名字，则必须使用该别名。ANSI 标准不支持这类查询。

2.51.2.6 嵌入式插入的使用限制

- 一条嵌入式 INSERT 语句不得用在联接查询里。
- 一条嵌入式 INSERT 语句不得出现在子查询里。
- 一条嵌入式 INSERT 语句的 WHERE 子句里不能有子查询。
- 一条 INSERT 语句不能包含嵌入式 INSERT 语句。
- 不支持嵌入式 INSERT 表达式之间的联接操作。
- 不支持在一条嵌入式 INSERT 语句里声明游标。

2.51.2.7 DISTINCT 聚合函数

聚合函数能接受一个被指定为 DISTINCT 的参数，它能在应用该聚合函数之前消除重复值。一个给定的分组可以出现多个 DISTINCT 聚合函数，它们可以和非 DISTINCT 聚合函数同时使用。此处存在一个使用限制：DISTINCT STDDEV 和 VARIANCE 不能与多个 DISTINCT 聚合函数同时使用。

2.51.2.8 DISTINCT 聚合函数的使用限制

- DISTINCT 聚合函数的数量没有限制。
- 不支持 DISTINCT STDDEV 和 DISTINCT VARIANCE 与多个 DISTINCT 聚合函数同时使用。例如，以下语句将导致错误。

  ```
  SELECT sum(distinct a), stddev(distinct b) from T group by d;
  ```

2.51.2.9 多个 Distinct 聚合函数的例子

- 该语句包含多个 distinct 聚合函数：

  ```
  SELECT sum(distinct a), count(distinct b), avg(distinct c)
  from T group by d;
  ```

- 该语句不含多个 distinct 聚合函数，因为每个 distinct 聚合函数位于都在同一列(a)上，这被视为一个不同的数值。

  ```
  SELECT sum(distinct a), count(distinct a), avg(distinct a) from T
  group by d;
  ```

- 该语句显示多个 distinct 聚合函数可以与非 Distinct 聚合函数一起使用：

  ```
  SELECT sum(distinct a), avg(distinct b), sum(c) from T group by d;
  ```

2.51.2.10with-clause 注意事项

- 默认情况下，不启用使用临时表物化CTE。
- EsgynDB只支持非递归公共表表达式，这意味着WITH子句不能自引用，但它可以在同一WITH子句中引用之前定义的CTE。
- EsgynDB只允许在SELECT语句的开头指定一次WITH子句。尚不支持

INSERT、UPDATE、DELETE和CREATE VIEW语句。

2.51.2.11Select List 注意事项

- *and corr.* 形式的 select-list 规范非常方便。 然而，此类规范使得 SELECT 结果表里的列序依赖于当前所引用的表或视图定义里的列序。
- *col-expr* 是单个列的名称或派生列。派生列是一种 SQL 数值表达式。其操作数可以是数字、字符串、日期时间、或间隔字面值、列、定义在列上的函数（包括聚合函数）、标量子查询、CASE 表达式或 CAST 表达式。col-expr 里使用的任何列都必须来自 FROM 子句里指定的表或视图。更多关于聚合函数列表的信息，请参阅聚合（集合）函数。
- 如果 *col-expr* 是一个列名，则 SELECT 结果表里的该列将是一个命名列。结果表里的所有其他列都是未命名列（带有 EXPR 标题），除非您使用 AS 子句为派生列指定名称。

2.51.2.12GROUP BY 子句注意事项

- 如果您的语句包含一个 GROUP BY 子句，则您在 select-list 里引用的列都必须是分组列或一个聚合（或集合）函数的参数。例如：如果 AGE 不是一个分组列，您可以引用 AGE、并仅将其作为一个诸如 AVG (AGE)函数的参数。
- GROUP BY 子句里的表达式必须与选择列表里的表达式完全相同。否则将返回一个错误。它不得包含聚合函数或子查询。
- 如果 col-expr 的值是一个数字常数，它将引用选择列表项的位置、并被视为使用序数功能的当前 GROUP BY 子句。
- 可以指定使用序数的 GROUP BY 子句，以便引用 SELECT 列表中的相对位置。例如：GROUP BY 3, 2, 1。
- 如果语句不包含 GROUP BY 子句，但您在选择列表里指定了一个聚合函数，则结果表的所有行将构成唯一的组。例如：AVG 的执行结果是单个值，它适用于整个表。

2.51.2.13ORDER BY 注意事项

指定一个 ORDER BY 子句和它的排序列时，请注意：

- ORDER BY 仅可用在一条查询语句的外层或一条 INSERT/SELECT 语句的 SELECT 部分。不得将它用在嵌套 SELECT 语句（诸如子查询）的内部。
- 如果您指定了 DISTINCT，则排序列必须在 select-list 中。
- 如果您指定了一个 GROUP BY 子句，则排序列必须也是分组列。
- 如果一个 ORDER BY 子句应用于 SELECT 语句的联合操作，则必须在最左侧 SELECT 语句的 selec-list 中显式地引用排序列（不在聚合函数和表达式内）。
- SQL 不保证特定或一致的行序，除非您指定一个 ORDER BY 子句。ORDER BY 子句可能会降低性能，所以，建议您在需要特定的顺序时使用它。
- 如果语句中出现了 group by, distinct 或者聚合函数，order by 的子句不用一定要显式出现在 select list 中。

  比如下例中 order by 的列没有意义，EsgynDB 会忽略这个没有意义的 order By 语句，不会报错：

```
select nvl(sum(qianfjee),0) from T where farendma='9999' and kehuzhao='6224980010135762' and zhanghao='99011560102017800005 70' and (shouqibz in ('0','2') and jiluztai='0') order by shjndjxh;

(EXPR)
--------------------

                  246

--- 1 row(s) selected.
```

- 为兼容 Oracle，支持在 ORDER-BY 语句中使用函数来进行排序。

  举例：

```
>>select * from staff order by last_name;
LAST_NAME     SALARY
-------------------------------------
Mike           1000
```

```
Peter          1000
Tom            1000

--- 3 row(s) selected.
>>select  substr(LAST_NAME,2,1)  as  A,  last_name  as
Original_lastname     from     staff     order     by
substr(LAST_NAME,2,1);
A  ORIGINAL_LASTNAME
-----------------------------------------
e  Peter
i  Mike
o  Tom

--- 3 row(s) selected.
```

2.51.2.14UNION 注意事项

假设有效的 SELECT 语句分别被命名为 SELECT1 和 SELECT2，通过 SELECT 语句得到的有效表分别被命名为 TABLE1 和 TABLE2，通过 UNION 操作得到的表则被命名为 RESULT。

2.51.2.15UNION 列的特性

构成 RESULT 表的 TABLE1 和 TABLE2 里的列:

- 如果两列都包含字符串，则 RESULT 表的相应的列也将包含一个字符串，其长度等于两个有效列长度的较大者。
- 如果两列都包含可变长度字符串，则 RESULT 表也将包含一个可变长度字符串，其长度等于两个有效列长度的较大者。
- 如果两列都具有精确的 numeric 数据类型，则 RESULT 表也将包含一个精确的 numeric 数值，其精度和标度均等于两个有效列精度和标度的较大者。
- 如果两列都具有近似的 numeric 数据类型，则 RESULT 表也将包含一个近似的 numeric 数值，其精度等于两个有效字段精度的较大者。

- 如果两列都具有 datetime 数据类型（DATE、TIME 或 TIMESTAMP），则 RESULT 表里的相应列也具有相同的数据类型。
- 如果两列都具有INTERVAL数据类型，且均为year-month（年-月型）或day-time（日-时型），则RESULT表里也将包含一个INTERVAL数值，则字段范围涵盖有效列 INTERVAL 字段的最高有效起始字段至最低有效终止字段。（year-month 字段为 YEAR 和 MONTH。day-time 字段为 DAY、HOUR、MINUTE 和 SECOND。）

  例如：假设 TABLE1 里的列具有数据类型 INTERVAL HOUR TO MINUTE；TABLE2 里的字段具有数据类型 INTERVAL DAY TO HOUR。则通过联合查询操作产生的字段的数据类型将是 INTERVAL DAY TO MINUTE。如果两列都被描述为 NOT NULL（非空），则 RESULT 表里的相应字段也不得为空。否则，该字段可以为空。

2.51.2.16ORDER BY 子句和 UNION 操作符

在一条包含 UNION 操作符的查询里，ORDER BY 子句可以定义联合查询结果里的排序。在这种情况下，SELECT 语句不能包含一个单独的 ORDER BY 子句。只可以在最后一条 SELECT 语句后面的最后一个子句（本例为 SELECT2）之后添加 ORDER BY 子句。

RESULT 表里的 ORDER BY 子句通过使用一个整数或 SELECT1 选择列表里的列名指定了排序列的序数位置。

- 这条 SELECT 语句显示了 ORDER BY 子句的正确使用方式：

  ```
  SELECT A FROM T1 UNION SELECT B FROM T2 ORDER BY A
  ```

- 这条 SELECT 语句不正确，因为 ORDER BY 子句并没有跟在最终 SELECT 语句后面：

  ```
  SELECT A FROM T1 ORDER BY A UNION SELECT B FROM T2
  ```

- 这条 SELECT 语句也不正确：

  ```
  SELECT A FROM T1 UNION (SELECT B FROM T2 ORDER BY A)
  ```

  因为子查询（SELECT B FROM T2...）被首先处理，所以 ORDER BY 子句不是跟在最后一条 SELECT 语句后面。

2.51.2.17GROUP BY 子句、HAVING 子句和 UNION 操作符

在一条包含 UNION 操作符的查询里，GROUP BY 子句或 HAVING 子句将关联所属的 SELECT 语句（不同于 ORDER BY 子句；ORDER BY 子句可以关联合并操作的结果），这些组在特定 SELECT 语句的结果表里都可见。GROUP BY 子句和 HAVING 子句不可用来形成合并操作的结果里的组。

2.51.2.18UNION ALL 和关联方式

UNION ALL 是一种左结合运算，即这两条查询返回相同的结果。

```
(SELECT * FROM TABLE1 UNION ALL
     SELECT * FROM TABLE2) UNION ALL SELECT * FROM TABLE3;
SELECT * FROM TABLE1 UNION ALL
     (SELECT * FROM TABLE2 UNION ALL SELECT * FROM TABLE3);
```

如果 UNION ALL 和 UNION 操作符同时出现在一条查询里，则求值顺序始终为从左到右。首先对 SELECT 语句括号内的 UNION 操作从左到右求值，然后对 SELECT 语句剩余的 UNION 操作进行求值。

2.51.2.19For Update

For Update 受以下限制：

SELECT 查询必须有一个 WHERE 子句，该子句使用所有主键作为谓词，并且所有谓词都必须使用相等的比较器。

不能使用以下其他构造指定此子句：DISTINCT 运算符、游标表达式、set 运算符、group U by U 子句或聚合函数。

2.51.3 SELECT 例子

- 以下例子定义了两个CTE，w1和w2。w2引用w1，w1在w2之前定义。

  ```
  >>select * from t1
  C1 C2
  ----------
  ```

```
1 1
2 2
3 3
4 4
5 5
--- SQL operation complete.
>>with w1 as (select * from t1),
>>w2 as (select * from w1)
>>select * from w2;
C1 C2
----------
1 1
2 2
3 3
4 4
5 5
--- SQL operation complete.
```

- 以下例子定义了两个CTE，w1和w2，然后在它们之间执行联接(JOIN)。

```
>>select * from t1
C1 C2
----------
1 1
2 2
3 3
4 4
5 5
--- SQL operation complete.
>>select * from t2
C1 C2
----------
3 3
4 4
5 5
6 6
7 7
--- SQL operation complete.
>>with w1 as (select c1, c2 from t1),
>>w2 as (select c1, c2 from t2)
>>select * from w1, w2 where w1.c1 = w2.c1;
C1 C2 C1 C2
----------
3 3 3 3
4 4 4 4
5 5 5 5
--- SQL operation complete.
```

- 从 EMPLOYEE 表里检索 job code（工作代码）大于 500、所在部门编号小于或等于 3000 的雇员信息，并将结果按 job code 升序显示：

```
SELECT jobcode, deptnum, first_name, last_name, salary
FROM persnl.employee
WHERE jobcode > 500 AND deptnum <= 3000
ORDER BY jobcode;

JOBCODE DEPTNUM FIRST_NAME  LAST_NAME   SALARY
------- ------- ----------- ----------- ----------
600     1500    JONATHAN    MITCHELL    32000.00
600     1500    JIMMY       SCHNEIDER   26000.00
900     2500    MIRIAM      KING        18000.00
900     1000    SUE         CRAMER      19000.00
…
```

- 按 job code 分组升序显示所选的行：

```
SELECT jobcode, AVG(salary)
FROM persnl.employee
WHERE jobcode > 500 AND deptnum <= 3000
GROUP BY jobcode
ORDER BY jobcode;

JOBCODE EXPR
------- ----------------------
600     29000.00
900     25100.00
--- 2 row(s) selected.
…
```

该 select list 仅包含分组列和聚合函数。每个输出行都可以汇总一个组内的所选数据。

- 通过在 FROM 子句里指定表名、并在 WHERE 子句里指定结果行选择条件，从多个表中选择数据：

```
SELECT jobdesc, first_name, last_name, salary
   FROM persnl.employee E, persnl.job J
   WHERE E.jobcode = J.jobcode AND
         E.jobcode IN (900, 300, 420);

   JOBDESC   FIRST_NAME LAST_NAME  SALARY
```

```
--------- ---------- ---------- -----------
SALESREP  TIM        WALKER     32000.00
SALESREP  HERBERT    KARAJAN    29000.00
...
ENGINEER  MARK       FOLEY      33000.00
ENGINEER  MARIA      JOSEF      18000.10
...
SECRETARY BILL       WINN       32000.00
SECRETARY DINAH      CLARK      37000.00

...

--- 27 row(s) selected.
```

该条件类型有时称为联接谓词。该查询可以将 EMPLOYEE 表的每行和 JOB 表的每行组合在一起，首先对 EMPLOYEE 和 JOB 表进行联接查询。中间结果是这两个表的笛卡儿乘积。

这个联接谓词规定（中间结果里）相同 job code 的任何行都应包含在结果表里。WHERE 条件还规定 job code 必须为 900、300 或 420。所有其他行都将被剔除。

四个逻辑步骤可以确定前一条查询的中间结果和最后结果，具体如下：

1) 联接各表。

EMPLOYEE 表			**JOB 表**	
EMPNUM …	JOBCODE …	SALARY	JOBCODE	JOBDESC

2) 删除 job code 不同的行。

EMPLOYEE 表			**JOB 表**	
EMPNUM …	JOBCODE …	SALARY	JOBCODE	JOBDESC
1	100	175500	100	MANAGER
…	…	…	…	…
75	300	32000	300	SALESREP
…	…	…	…	…
178	900	28000	900	SECRETARY
…	…	…	…	…
207	420	33000	420	ENGINEER
…	…	…	…	…
568	300	39500	300	SALESREP

3) 删除 job code 不是 900、300 或 420 的行。

EMPLOYEE 表			**JOB 表**	
EMPNUM …	JOBCODE …	SALARY	JOBCODE	JOBDESC
…	…	…	…	…
75	300	32000	300	SALESREP
…	…	…	…	…
178	900	28000	900	SECRETARY
…	…	…	…	…
207	420	33000	420	ENGINEER
…	…	…	…	…
568	300	39500	300	SALESREP

4) 处理选择列表，只留下四列。

JOBDESC	**FIRST_NAME**	**LAST_NAME**	**SALARY**
SALESREP	TIM	WALKER	32000
…	…	…	…
SECRETARY	JOHN	CHOU	28000
…	…	…	…
ENGINEER	MARK	FOLEY	33000
…	…	…	…
SALESREP	JESSICA	CRINER	39500

最终结果如下所示：

```
JOBDESC       FIRST_NAME    LAST_NAME        SALARY
------------- ------------- ---------------- ----------
SALESREP      TIM           WALKER           32000.00
...
SECRETARY     JOHN          CHOU             28000.00
...
```

- 从这三个表里进行选择操作，并将所选的行按 job code 和部门编号（job code 内部）进行分组，并按各组的最高工资对这些组进行排序：

```
SELECT E.jobcode, E.deptnum, MIN (salary), MAX (salary)
FROM persnl.employee E,
   persnl.dept D, persnl.job J
WHERE E.deptnum = D.deptnum AND E.jobcode = J.jobcode
   AND E.jobcode IN (900, 300, 420)
```

```
GROUP BY E.jobcode, E.deptnum
ORDER BY 4;

JOBCODE DEPTNUM  (EXPR)   (EXPR)
------- ------- ----------- -----------
 900    1500   17000.00    17000.00
 900    2500   18000.00    18000.00
 ...
 300    3000   19000.00    32000.00
 900    2000   32000.00    32000.00
 ...
 300    3200   22000.00    33000.10
 420    4000   18000.10    36000.00
 ...

 --- 16 row(s) selected.
```

仅选择 job code 为 300、420 和 900。

计算每个部门里相同工作的最低和最高工资，并按最高工资对这些行进行排序。

- 从两个联接表中选择数据，这两个表在匹配的部件编号上使用 INNER JOIN:

```
SELECT OD.*, P.*
FROM sales.odetail OD INNER JOIN sales.parts P
ON OD.partnum = P.partnum;

Order/Num  Part/Num  Unit/Price  Qty/Ord  Part/Num  Part
Description PRICE Qty/Avail

---------- -------- ------------ ---------- --------
------------------ ------------ -----------
400410 212 2450.00 12 212
PCSILVER, 20 MB 2500.00 3525

500450 212 2500.00 8 212
PCSILVER, 20 MB 2500.00 3525

100210 244 3500.00 3 244
PCGOLD, 30 MB 3000.00 4426

800660 244 3000.00 6 244
```

```
PCGOLD, 30 MB 3000.00 4426

... ... ... ... ...
... ... ...

...

--- 72 row(s) selected.
```

- 从这三个表中选择数据，并按雇员编号排序显示。在匹配的部门编号上使用 LEFT JOIN 联接两个表，然后在匹配的工作代码上联接一个附加表：

```
SELECT empnum, first_name, last_name, deptname, location,
jobdesc
  FROM employee e LEFT JOIN dept d ON e.deptnum =
  d.deptnum
       LEFT JOIN job j ON e.jobcode = j.jobcode
  ORDER BY empnum;
```

- 假设已经使用 CREATE LIKE 语句从 JOB 表中创建了 JOB_CORPORATE 表。以下语句可以形成这两个表的联合查询：

```
SELECT * FROM job UNION SELECT * FROM job_corporate;

JOBCODE JOBDESC
--- ------------------
100 MANAGER
200 PRODUCTION SUPV
250 ASSEMBLER
300 SALESREP
400 SYSTEM ANALYST
420 ENGINEER
450 PROGRAMMER
500 ACCOUNTANT
600 ADMINISTRATOR
900 SECRETARY
100 CORP MANAGER
300 CORP SALESREP
400 CORP SYSTEM ANALYS
500 CORP ACCOUNTANT
600 CORP ADMINISTRATOR
```

```
900 CORP SECRETARY


--- 16 row(s) selected.
```

- FULL OUTER JOIN 结合左外联接和右外联接的查询结果。这些联接显示两个表里的记录，并在任何一侧的匹配缺失处填写 NULL（空值）。

```
SELECT *
FROM employee
   FULL OUTER JOIN
   department
        ON employee.DepartmentID =
        department.DepartmentID;


LastName DepartmentID DepartmentName DepartmentID
------- ------------ -------------- -------------
Smith             34 Clerical                 34
Jones             33 Engineering              33
Robinson          34 Clerical                 34
Jasper            36 NULL                   NULL
Steinberg         33 Engineering              33
Rafferty          31 Sales                    31
NULL            NULL Marketing                35
```

- 介绍两种从加州客户提交的表中选择相同数据的方式。
  - 第一种方式

```
SELECT OD.ordernum, SUM (qty_ordered * price)
FROM sales.parts P, sales.odetail OD
WHERE OD.partnum = P.partnum AND OD.ordernum IN
     (SELECT O.ordernum
     FROM sales.orders O, sales.customer C
     WHERE  O.custnum  =  C.custnum  AND  state  =
     'CALIFORNIA')
GROUP BY OD.ordernum;


ORDERNUM (EXPR)
---------- ---------------------
200490                   1030.00
300350                  71025.00
300380                  28560.00
--- 3 row(s) selected.
```

- 第二种方式

```
SELECT OD.ordernum, SUM (qty_ordered * price)
FROM sales.parts P, sales.odetail OD
WHERE OD.partnum = P.partnum AND OD.ordernum IN
     (SELECT O.ordernum
     FROM sales.orders O
     WHERE custnum IN
          (SELECT custnum
          FROM sales.customer
          WHERE state = 'CALIFORNIA'))
GROUP BY OD.ordernum;
```

```
ORDERNUM (EXPR)
---------- ---------------------
200490                   1030.00
300350                  71025.00
300380                  28560.00
--- 3 row(s) selected.
```

针对每个订单号计算总量价格。

- 显示每位雇员、工资和工资占工资总额的百分比。该子查询是选择列表里表达式的一部分。

```
SELECT empnum, first_name, last_name, salary,
CAST(salary * 100 / (SELECT SUM(salary) FROM
persnl.employee) AS
NUMERIC(4,2))
FROM persnl.employee ORDER BY salary, empnum;
```

```
Employee/Number First Name Last Name salary (EXPR)
--------------- --------------- -------------------- --
--------- -------
209 SUSAN CHAPMAN 17000.00 .61
235 MIRIAM KING 18000.00 .65
224 MARIA JOSEF 18000.10 .65
...
23
JERRY
HOWARD
137000.10
4.94
```

```
32 THOMAS RUDLOFF 138000.40 4.98
1 ROGER GREEN 175500.00 6.33
...
--- 62 row(s) selected.
```

- 在 GROUP BY 子句中使用表达式例子

```
SELECT a+1 FROM t GROUP BY a+1;
SELECT cast(a AS int) FROM t GROUP BY cast(a AS int);
SELECT a+1 FROM
t GROUP BY 1;
```

- GROUP BY 子句中不支持的表达式:

```
SELECT sum(a) FROM t GROUP BY sum(a);
SELECT (SELECT a FROM t1) FROM t GROUP BY (SELECT a FROM
t1); SELECT a+1 FROM t GROUP BY 1+a;
```

- 使用[LAST N]选项的例子:

```
SQL>SELECT [LAST 0] * FROM employee;
--- 0 row(s) selected.
SQL>SELECT [LAST 1] * FROM employee WHERE jobcode <>
100 AND salary = 30000;
EMPNUM FIRST_NAME LAST_NAME DEPTNUM JOBCODE SALARY
------ --------------- -------------------- ------- ---
---- ----------
227 XAVIER SEDLEMEYER 3300 300 30000.00
--- 1 row(s) selected.
SQL>SELECT [LAST 2] * FROM employee;
*** ERROR[15002] Internal parser error: Number of rows
must be 0 or 1 with LAST
option.
. [2018-02-28 18:05:12]
```

- 使用[NULLS FIRST|NULLS LAST]选项的例子:

```
create table Table4(a varchar(1));
insert into Table4 values(NULL),('A'),('B'),(NULL);
SQL>select * from Table4 order by a nulls first;

A
----
NULL
NULL
A
B
```

--- 4 row(s) selected.

```
SQL>select * from Table4 order by a nulls last;

A
----
A
B
NULL
NULL
```

2.52 SET SCHEMA 语句

该语句为当前会话的非限定对象名称设置默认逻辑 schema。

```
SET SCHEMA default-schema-name
```

2.52.1 SET SCHEMA 语法

- *default-schema-name*

  指定一个 schema 的名称。更多信息，请参阅 Schemas。

- *default-schema-name* 是一种 SQL 标识符。例如：您可以使用 MYSCHEMA、myschema 或分隔标识符“My_Schema”。更多信息，请参阅标识符。

2.52.2 SET SCHEMA 注意事项

使用 SET SCHEMA 语句指定的默认 schema 的效用一直延续到会话结束，或直到执行另一条 SET SCHEMA 语句。

如果没有使用 SET SCHEMA 语句指定 schema 名称，则默认的 schema 为 SEABASE，它已存在于 TRAFODION catalog 中。更多关于如何创建 schema 的信息，请参阅创建和删除 schema。

2.52.3 SET SCHEMA 例子

- 设置默认 schema 名称：

  ```
  SET SCHEMA myschema;
  ```

2.53 SET TRANSACTION 语句

该语句设置该事务的 autocommit（自动提交）属性。该语句的效用一直延续到该会话结束时或直到发出下一条 SET TRANSACTION 语句时（以先到时间为准）。

因此，SET TRANSACTION 语句可以为该会话里的所有后续事务设置 autocommit 属性。

```
SET TRANSACTION autocommit-option

autocommit-option is:
     AUTOCOMMIT [ON] | AUTOCOMMIT OFF
```

2.53.1 SET TRANSACTION 语法

autocommit-option

指定 EsgynDB SQL 在语句执行结束时是否自动提交或回滚该语句。

该选项适用于任何由系统发起一个事务的语句。

如果该选项设置为 ON（开启），则 EsgynDB SQL 将在语句执行结束时自动提交或回滚针对数据库所做的任何更改。会话开始时，AUTOCOMMIT 选项默认开启；如果该选项设置为 OFF（关闭），则当前事务将保持活动状态，直到会话结束，除非您显式地提交或回滚该事务。

AUTOCOMMIT 是 EsgynDB SQL 扩展选项。不得将它与任何其他选项同时使用。在一条 SET TRANSACTION 语句里使用 AUTOCOMMIT 选项不会重置前一条 SET TRANSACTION 语句里可能已经指定的其他事务属性。

同样，一条未指定 AUTOCOMMIT 属性的 SET TRANSACTION 语句也不会重置该属性。

2.53.2 SET TRANSACTION 注意事项

2.53.2.1 隐式事务

大多数 DML 语句都可以发起事务，即语句开始执行时，系统会自动发起一个事务。

例外情况（不能发起事务的语句）包括：

- COMMIT、FETCH、ROLLBACK 和 SET TRANSACTION
- 只有在关联语句发起事务的情况下，EXECUTE 才能发起事务。

2.53.2.2 显式事务

即使开启 AUTOCOMMIT 选项，您也可以发出一条显式 BEGIN WORK 语句。AUTOCOMMIT 选项将被暂时禁用，直到您显式地发出 COMMIT 或 ROLLBACK 为止。

2.53.3 SET TRANSACTION 例子

以下 SET TRANSACTION 语句关闭 AUTOCOMMIT 选项，以使当前事务保持活动状态，直到会话结束（除非您显式地提交或回滚该事务）。语句执行结束时，EsgynDB SQL 不会自动提交或回滚针对数据库所做的任何更改。

相反，当您发出一条 COMMIT WORK 语句时，EsgynDB SQL 会提交所有更改。

```
SET TRANSACTION AUTOCOMMIT OFF;
--- SQL operation complete.

BEGIN WORK;
 --- SQL operation complete.

DELETE FROM persnl.employee WHERE empnum = 23;
--- 1 row(s) deleted.

INSERT INTO persnl.employee
     (empnum, first_name, last_name, deptnum, salary)
     VALUES (50, 'JERRY','HOWARD', 1000, 137000.00);
--- 1 row(s) inserted.

UPDATE persnl.dept SET manager = 50
WHERE deptnum = 1000;
--- 1 row(s) updated.

COMMIT WORK;
--- SQL operation complete.
```

2.54 SHOWCONTROL 语句

该语句显示有效的默认属性。SHOWCONTROL 是 EsgynDB SQL 扩展语句。

```
SHOWCONTROL {ALL | [QUERY] DEFAULT [attribute-name[, MATCH
{FULL | PARTIAL }]]}
```

2.54.1 SHOWCONTROL 语法

- `ALL`

  显示 EsgynDB instance 中设置的所有硬编码默认属性。

- `[QUERY] DEFAULT`

  显示对该会话有效的 CONTROL QUERY DEFAULT 语句。更多信息，请参阅 CONTROL QUERY DEFAULT 语句。

- `attribute-name[, MATCH {FULL | PARTIAL }]`

  只显示全部或部分匹配 CONTROL QUERY DEFAULT 语句所使用的属性默认值。更多关于这些属性的信息，请参阅附录 B 控制查询默认（CQD）属性。

  MATCH FULL 指定该属性名称必须与 CONTROL QUERY DEFAULT 语句里所用的属性名称相同。

  MATCH PARTIAL 指定该属性名称必须包含在 CONTROL QUERY DEFAULT 语句所用的属性名称中。

  默认值为 MATCH PARTIAL。

  如果该属性名称是一个诸如 MIN、MAX 或 TIME 的保留字，这必须将其转换为大写字母、并分隔在双引号（“）内。 该规则唯一的例外情况是保留字 CATALOG 和 SCHEMA，您既可以将它们转换为大写字母、并用双引号分隔，也可以指定它们不带引号。

2.54.2 SHOWCONTROL 例子

输入一条 SHOWCONTROL DEFAULT 命令后，发出多条 SHOWCONTROL DEFAULT 语句：

```
CONTROL QUERY DEFAULT CACHE_HISTOGRAMS_REFRESH_INTERVAL
'7200';
--- SQL operation complete.

CONTROL QUERY DEFAULT HIST_NO_STATS_REFRESH_INTERVAL
'7200';
--- SQL operation complete.

SHOWCONTROL DEFAULT;

CONTROL QUERY DEFAULT
     CACHE_HISTOGRAMS_REFRESH_INTERVAL 7200
     HIST_NO_STATS_REFRESH_INTERVAL 7200

--- SQL operation complete.
```

2.55 SHOWDDL 语句

该语句描述创建一个元数据现有对象时所用的 DDL 语法、也能够以 GRANT 语句的形式返回用户、角色或组件的描述。

SHOWDDL 语句是 EsgynDB SQL 扩展语句。

```
SHOWDDL showddl-spec

showddl-spec
     [TABLE | LIBRARY | PROCEDURE] [schema-name.]object-
     name[, PRIVILEGES ]
     | COMPONENT component-name
     | USER database-username
     | ROLE role-name[, GRANTEES ]
```

2.55.1 SHOWDDL 语法

- `object-name`
  指定现有的表、视图、库或存储过程的 ANSI 名称。更多信息，请参阅数据库对象名称。如果对象名没有完全限定，则 SHOWDDL 语句将使用该会话的默认 schema。

- `PRIVILEGES`
  描述该对象的关联 PRIVILEGES（权限）。一个对象指定的权限将以 GRANT 语句的形式显示。

- *`component-name`*
  指定一个现有的组件。目前唯一有效的组件名称是 SQL_OPERATIONS。

- *`database-username`*
  指定一个已注册的数据库用户名。

- *`role-name`*
  指定一个现有的角色。

- `GRANTEES`
  以 GRANT ROLE 语句的形式显示所有已被授予该角色的用户。这是一条可选子句。

2.55.2 SHOWDDL 注意事项

- SHOWDDL 语句与创建对象所用的初始 DDL 不同。
- SHOWDDL 语句可用于 TrafCI 里。
- SHOWDDL 语句 [TABLE | LIBRARY | PROCEDURE]显示以下信息:
  - 可能禁用一个约束。
  - 一个表可能处于脱机状态。
  - 一个对象上可能存在一个处于活动状态的 DDL 锁。
- SHOWDDL 语句 USER 语句将用户信息显示为 REGISTER USER 语句。
- SHOWDDL 语句 ROLE 语句将角色信息显示为 CREATE ROLE 语句。

2.55.2.1 必需的权限

发出一条 SHOWDDL 语句，以下条件之一必须成立:

- 您是 DB_ROOT（数据库根用户)。
- 您是该数据库对象的所有者。
- 您拥有 SQL_OPERATIONS 组件的 SHOW 权限。SHOW 组件权限被默认授予给 PUBLIC。
- 您拥有目标对象的 SELECT 权限。

2.55.2.2 SHOWDDL 语句输出和初始 DDL 语句之间的区别

- 所有的列约束（NOT NULL、PRIMARY KEY 和 CHECK）都将转变为表约束。
- 所有 NOT NULL 约束将被合并成单一的 Check 约束。
- Check 约束将被移出 CREATE TABLE 语句、并封装在一条独立的 ALTER TABLE ADD CONSTRAINT 语句里。
- SHOWDDL 语句可以为添加到表中的每个字段生成 ALTER TABLE ADD COLUMN 语句。

- 输出里的所有 ANSI 名称都有限定的 schema 名称。
- SHOWDDL 语句显示约束名，即便创建约束期间未指定名称它也将显示。
- SHOWDDL 语句总会为 SPJ 生成一个 Java 签名。

2.55.2.3 PRIVILEGES 选项

PRIVILEGES 选项包含适用的 GRANT 语句。每一个权限都使用独立的 GRANT 语句指定，即便它们是通过一条语句授予的。

2.55.3 SHOWDDL 例子

- 该 SHOWDDL 语句显示数据库里指定表的创建语句和该表的授予权限：

```
SQL>showddl tab41;

CREATE TABLE TRAFODION.SCH41.TAB41
(
A INT DEFAULT NULL
, B INT DEFAULT NULL
)
;

-- GRANT DELETE, INSERT, SELECT, UPDATE, REFERENCES ON
TRAFODION."SCH41"."TAB41" TO PAULLOW41 WITH GRANT OPTION;

--- SQL operation complete.
```

- 该 SHOWDDL 语句显示将指定用户注册在数据库里所用的语句。

```
SQL>showddl user sqluser_admin;

REGISTER USER "SQLUSER_ADMIN";
--- SQL operation complete.
```

- 该 SHOWDDL 语句显示数据库里创建指定角色的语句和被授予该角色的用户：

```
SQL>showddl role db__rootrole;

CREATE ROLE "DB__ROOTROLE";

-- GRANT ROLE "DB__ROOTROLE" TO "DB__ROOT" WITH ADMIN
```

```
OPTION;

--- SQL operation complete.
```

2.56 SHOWDDL SCHEMA 语句

该语句将依照存在于元数据中的信息显示创建 schema 的 DDL 语法。并显示该 schema 所有者的授权 ID。

SHOWDDL SCHEMA 是 EsgynDB SQL 扩展语句。

```
SHOWDDL SCHEMA [catalog-name.] schema-name
```

2.56.1 SHOWDDL SCHEMA 语法

`schema-name`

指定一个现有的 cataolog 和 schema 的 ANSI 名称。

如果没有完全限定 schema 名称没，则 SHOWDDL 语句将使用该会话的默认 catalog。 更多信息，请参阅数据库对象名称。

2.56.2 SHOWDDL SCHEMA 注意事项

如果没有说明，则 catalog 为当前的默认 catalog。

2.56.2.1 必需的权限

发出一条 SHOWDDL SCHEMA 语句，以下条件之一必须成立：

- 您是 DB__ROOT（数据库根用户）。
- 您是该 schema 的所有者。
- 您拥有 SQL_OPERATIONS 组件的 SHOW 组件权限。SHOW 组件权限默认授予给 PUBLIC。

2.56.3 SHOWDDL SCHEMA 例子

依照 MYSCHEMA 存在于元数据中的信息，SHOWDDL SCHEMA 语句将显示创建 schema 所用的 DDL 语法。同时还显示该 schema 所有者的授权 ID。

```
SHOWDDL SCHEMA MYSCHEMA;

CREATE PRIVATE SCHEMA "TRAFODION"."MYSCHEMA" AUTHORIZATION
"DB__ROOT";

--- SQL operation complete.
```

2.57 SHOWSTATS 语句

该语句显示表内的一组或多组列的直方图统计信息。这些统计信息用于设计优化的访问计划。

SHOWSTATS 是 EsgynDB SQL 扩展语句。

```
SHOWSTATS FOR TABLE table-name ON group-list [DETAIL]

group-list is:
     column-list[, column-list]...
     | EVERY COLUMN[, column-list]...
     | EVERY KEY[, column-list]...
     | EXISTING COLUMN[S][, column-list]...

column-list for a single-column group is:
     column-name
     | (column-name)
     | column-name TO column-name
     | (column-name) TO (column-name)
     | column-name TO (column-name)
     | (column-name) TO column-name

column-list for a multicolumn group is:
     (column-name, column-name[, column-name]...)
```

2.57.1 SHOWSTATS 语法

- `table-name`（表名）

  显示统计信息的表的 ANSI 名称。

- `ON group-list`

  指定显示直方图统计信息的一组或多组列(group-list)。

  `group-list` 是：`column-list [, column-list]...| EVERY COLUMN [, column-list]...| EVERY KEY [, column-list]...| EXISTING COLUMN[S] [, column-list]...`

  指定 group-list 的定义方式。列列表代表一个单列组或一个多列组。

  - `EVERY COLUMN`

    为表的每个独立的列，以及构成主键和索引的多个列显示直方图统计信息。对于不含直方图的列，该选项将返回 No histogram data for column(s)-

-->。

- EVERY KEY
  为构成主键和索引的列显示直方图统计信息。

- EXISTING COLUMN[S]
  仅为确实具有直方图的表列显示直方图统计信息。
  由于不含直方图数据的列将被省略，所以该选项能生成一个更简洁的报告。
  该选项包含所有现有的多列直方图。

- DETAIL
  显示相应直方图区间的统计信息和其他细节。
  如果未选择 DETAIL 关键字，则默认显示界面将列出基本的直方图信息，包含直方图 ID、区间数量、总行数、总 UEC 和字段名。详细的显示界面还包括高/低值和定距数据。

- 单列组的 column-list 是：column-name | (column-name) | column-name TO column-name | (column-name) TO (column-name) | column-name TO (column-name) | (column- name) TO column-name
  指定单列组定义列名的方式。使用 TO 关键字指定的列范围将导致该范围内的所有列都被包含在内，并按照表里的声明进行定义。

- 多列组的 column-list 是：(column-name, column-name [, column-name]...)
  指定多列组定义字段名的方式。
  例如，(abc, def) 表示包含字段 abc 和 def 的多字段直方图、而非两个单字段直方图。

  更多关于 column-list 语法和指定列的信息，请参阅 UPDATE STATISTICS 语句。

2.57.2 SHOWSTATS 注意事项

2.57.2.1 必需的权限

- 发出一条 SHOWSTATS 语句，以下条件之一必须成立：
- 您是 DB__ROOT（数据库根用户）。
- 您是该数据库对象的所有者。
- 您具有 SQL_OPERATIONS 组件的 SHOW 权限。SHOW 组件权限默认授予给 PUBLIC。
- 您具有目标对象的 SELECT 权限。
- 您具有 SQL_OPERATIONS 组件的 MANAGE_STATISTICS 权限。

2.57.3 SHOWSTATS 例子

- 该例子显示使用 EVERY KEY 关键字的表 A 的直方图的统计信息。此外，选择 DETAIL 关键字：

```
SHOWSTATS FOR TABLE A ON EVERY KEY DETAIL;
```

- 该例子显示表 CAT.SCH.A 的统计信息，并选择了从 abc 到 def 的所有列：

```
SHOWSTATS FOR TABLE CAT.SCH.A ON ABC TO DEF;
```

- 该例子显示表 A 的统计信息。括号内的列表引用了一个多字段组：

```
SHOWSTATS FOR TABLE A ON (ABC,DEF);
```

- 该例子显示使用 EXISTING COLUMNS 关键字的表 A 的直方图的统计信息。此外，选择 DETAIL 关键字：

```
SHOWSTATS FOR TABLE A ON EXISTING COLUMNS DETAIL;
Default output example:
>>SHOWSTATS FOR TABLE A ON EXISTING COLUMNS;

Histogram data for Table CAT.SCH.A Table
ID:341261536378386


Hist ID # Ints Rowcount UEC Colname(s)
========== ====== =========== ================
623327638    1       11 10     ABC, DEF, GHI
623327633    10      11 10     ABC
623327628    9       11 9      DEF
623327623    10      11 10     GHI
--- SQL operation complete.
```

```
>>SHOWSTATS FOR TABLE A ON ABC;

Histogram data for Table CAT.SCH.A Table ID:
341261536378386

Hist ID # Ints Rowcount UEC Colname(s)
========= ====== =========== ==============
623327633    10     11 10     ABC
--- SQL operation complete.

>>SHOWSTATS FOR TABLE A ON DEF DETAIL;

Detailed Histogram data for Table CAT.SCH.A Table ID:
341261536378386

Hist ID: 623327628
Column(s): DEF Total Rows: 11
Total UEC: 9
Low Value: (1)
High Value: (199)
Intervals: 9
Number Rowcount UEC Boundary
====== =========== ==============================
0 0        0 (1)
1 1        1 (1)
2 3        1 (2)
3 1        1 (4)
4 1        1 (11)
5 1        1 (12)
6 1        1 (14)
7 1        1 (99)
8 1        1 (123)
9 1        1 (199)

--- SQL operation complete.
```

2.58 TABLE 语句

该语句等效于查询规范 SELECT * FROM table。

TABLE *table*

2.58.1 TABLE 语法

table
命名用户表或视图。

2.58.2 TABLE 注意事项

与 SELECT 语句的关系

TABLE 语句的执行结果是一个简易表，指的是 SELECT 语句中表引用的定义。更多信息，请参阅 SELECT 语句。

2.58.3 TABLE 例子

TABLE 语句与 SELECT * FROM job 返回相同的结果:

```
TABLE job;

Job/Code Job Description
-------- --------------------
     100 MANAGER
     200 PRODUCTION SUPV
     250 ASSEMBLER
     300 SALESREP
     400 SYSTEM ANALYST
     420 ENGINEER
     450 PROGRAMMER
     500 ACCOUNTANT
     600 ADMINISTRATOR
     900 SECRETARY
--- 10 row(s) selected.
```

2.59 UNREGISTER USER 语句

该语句从 SQL 数据库里删除一个数据库用户名。用户将不再能登录到数据库。UNREGISTER USER 是 EsgynDB SQL 扩展语句。

```
UNREGISTER USER database-username [RESTRICT | CASCADE]
```

2.59.1 UNREGISTER USER 语法

- *database-username*
  这是当前已注册的数据库用户名，它不区分大小写字母的常规或分隔标识符。更多信息，请参阅不区分大小写字母的分隔标识符。

- [RESTRICT|CASCADE]
  如果指定 RESTRICT 选项，且该用户拥有的数据库里存在任何对象或 schema、或该用户已被授予任何权限或角色，则 UNREGISTER USER 操作将失败。
  如果指定 CASCADE，则该用户拥有的所有对象和 schema 都将被删除；作为 UNREGISTER USER 操作的一部分，该用户被授予的所有权限或角色也将被撤销。默认值为 RESTRICT。

2.59.2 UNREGISTER USER 注意事项

- 为了注销一个用户，您必须拥有该用户的管理权限。如果您已被授予 MANAGE_ROLES 组件权限，则将具有用户管理权限。最初，数据库用户里只有 DB__ROOT（数据库根用户）才可以被授予 MANAGE_USERS 组件权限。

- 您不能注销任何以 DB__开头的用户名。以 DB__开头的角色名都被 EsgynDB 保留。

- 如果指定 RESTRICT（或未指定），且该用户拥有任何对象或 schemas、或该用户已被授予任何权限或角色，则 UNREGISTER USER 语句操作将失败，

2.59.3 UNREGISTER USER 例子

- 注销用户:

```
UNREGISTER USER "jsmith@hp.com";
```

2.60 UPDATE 语句

该语句是一种 DML 语句，它更新一个表或一个可更新视图里的数据。更新一个视图里的行将会同时更新其基表里的行。

```
Searched UPDATE is:
UPDATE table
     { set-clause-type1 | set-clause-type2 }

set-clause-type1 is:
     SET set-clause[, set-clause ]..
     set-clause is:
          column-name = { expression | NULL}

set-clause-type2 is:
     SET (column1, ..., columnN) = {(value1, ..., valueN)
| (query-expr)}
[WHERE search-condition]
[[FOR] access-option ACCESS]

access-option is:
     READ COMMITTED
```

2.60.1 UPDATE 语法

- table
  指明更新的用户表或视图。table 必须是一个基表或一个可更新的视图。为了引用一个表或视图，请使用 ANSI 逻辑名称。更多信息，请参阅数据库对象名称。

- set-clause-type1
  该类 SET 子句可以将一个数值与正在更新的表中的特定字段关联起来。对于每条 set-clause，指定目标列名的数值都将被置换为更新源表达式的数值或 NULL（空值）。每个目标字段的数据类型都必须与其源值的数据类型相兼容。
  - column-name
    指明更新的表中的列。不能限定或重复一个列名。您不能更新作为主键一部分的字段的数值。
  - expression
    这是一种 SQL 数值表达式，指定列的数值。该表达式不能包含针对一个列定义的聚合函数。 表达式的数据类型都必须与列名的数据类型相兼容。

如果表达式引用了正在更新的字段，EsgynDB SQL 将使用初始值执行表达式求值，并确定新值。更多信息，请参阅表达式。

- NULL
  指定更新源的数值。

- set-clause-type2
  这类 SET 子句允许在赋值操作符的左侧指定多个字段。这些字段都使用赋值操作符右侧指定的多个数值进行更新。赋值操作符的右侧既可以是一个简单数值，也可以是一条子查询。
  - column1, ..., columnN
    命名多个将更新的表中的列。您不能限定或重复一个列名。您不能更新作为主键一部分的字段的数值。

  - value1, ..., valueN
    此为赋值操作符右侧指定的数值，它们适用于赋值操作符左侧指定的字段。每种数据类型必须和赋值操作符左侧相应字段的数据类型相兼容。

  - query-expr
    这是一条 SELECT 子查询。SET 子句的右侧仅可指定一条子查询。该子查询不能引用一个正在更新的表。更多关于“query-expr 的语法描述”，请参阅 SELECT 语句。

- WHERE search-condition
  指定一个搜索条件，以便选择将更新的行。正在搜索条件中比较的列也将同时在表或视图里进行更新。 更多信息，请参阅搜索条件。

  如果您没有指定一个搜索条件，则表或视图里的所有行都将被删除。不得在包含 SELECT 的 WHERE 子句里对同一个表使用 UPDATE 语句。
  对同一个表进行读取、插入、删除或更新操作会产生一个错误。此时应改用一种定位型(WHERE CURRENT OF) UPDATE 语句。更多信息，请参阅 MERGE 语句。

- [FOR] access-option ACCESS
  指定搜索条件求值所用数据必备的访问选项。更多信息，请参阅数据一致性和访问选项。

- READ COMMITTED
  指定搜索条件求值所用的任何数据都必须来自已提交的行。

2.60.2 UPDATE 注意事项

2.60.2.1 性能

与非主键字段的 UPDATE 操作相比，主键字段的 UPDATE 操作可能表现不佳。因为 UPDATE 操作会删除前映像里的所有记录，并将后映像里的记录插入表里，因此将会移动磁盘里的记录。

2.60.2.2 授权要求

UPDATE 操作必需有权读、写正在更新的表或视图，并有权读取搜索条件的子查询中指定的任何表或视图。如果基表中底层列可被更新，则视图里的列也可被更新。

2.60.2.3 发起和终止事务

如果目前不存在处于活动状态的事务，则 UPDATE 语句将会自动发起一个事务。否则，您可以使用 BEGIN WORK 语句显式地发起一个事务。

启动一个事务后，SQL 语句将在该事务内部执行，直到遇到 COMMIT 或 ROLLBACK、或出现一个错误为止。

2.60.2.4 事务隔离级别和语句访问选项

EsgynDB SQL 事务隔离级别定义事务内部数据操作受并发事务影响的程度。如果您在一个事务内部指定了 DML 语句的访问选项，则包含该语句的事务的隔离级别将被覆盖。 每条语句都将按照其各自的访问选项执行。

2.60.2.5 并发应用程序里的更新冲突

如果您正在一个事务内部使用 READ COMMITTED 隔离级别，则应用程序可以在不同时间读取相同数据的不同提交值。此外，两个并发应用程序可以（可能在出现错误时）更新同一行的相同列。

2.60.2.6 行数据的要求

所更新的每行都必须满足表或视图底层基表的约束。 除非所有这些约束都得到满足，否则不会出现列更新。如果 Check 条件不是 false（它是真的或者是未知的数值），则可以满足表约束。

此外，使用 WITH CHECK OPTION 创建的视图里的一个候选行必须满足视图选择标准。该选择标准被指定在 CREATE VIEW 语句的 AS query-expr 子句的 WHERE 子句里。

2.60.2.7 更新情况报告

如果一个 UPDATE 操作成功完成，EsgynDB SQL 将报告操作期间行的更新次数。在某些条件下，更新一个带有索引的表会导致 EsgynDB SQL 多次更新同一行，同时导致所报告的更新数量高于更改行的实际数量。然而，表里的数据和所报告的更新数量都是正确的。

如果以下所有条件为真，则将出现上述行为：

- 优化器选择一个替代索引作为访问路径。
- 该更新不会改变 WHERE 搜索条件里指定的索引列。
- 同一个索引内部的另一列被更新为一个更高的值（如果该列按升序存储）或一个更低的值（如果该列按降序存储）。

当这些条件出现时，索引条目的顺序可以保证 EsgynDB SQL 将在表处理后期遇到（满足相同的搜索条件的)同一行。然后将再次使用相同的数值更新该行。

例如，假设 MYTABLE 的索引包括 A 列和 B 列，指定的 UPDATE 语句如下：

```
UPDATE MYTABLE
SET B = 20
WHERE A > 10;
```

如果 UPDATE 操作前的 A 列和 B 列的内容分别为 1 和 12，UPDATE 操作后，EsgynDB SQL 将遇到由数值 1 和 20 索引的同一行。

2.60.2.8 更新 Character（字符）值

对于固定长度的字符列而言，短于该列长度的更新值将被填充单字节 ASCII 空格（HEX20），再填入该列中。 如果更新值大于列长度，则非空尾部字符串被

截断时将返回一个错误，该列也不会被更新。

对于变长字符列而言，更新值将不会被填充。其长度为指定数值的长度。 固定长度的更新也是同样如此。如果更新值大于字段长度，则非空尾部字符串被截断时将返回一个错误，该字段也不会被更新。

2.60.2.9 SET 子句的使用限制和错误情况

SET 子句有以下使用限制:

- 每个赋值操作符左侧的列数应匹配右侧数值的数量或 SELECT 列表元素的数量。 以下为错误示例:

  ```
  UPDATE t SET (a,b)=(10,20,30)
  UPDATE t set (b,c)=(SELECT r,t,s FROM x)
  ```

- 如果指定了多列更新语法，并且右侧包含一条子查询，则不得使用只有一个元素的子查询:

  ```
  UPDATE t SET (a,b)=(10, (SELECT a FROM t1))
  ```

- 如果使用了多列语法，则不得使用多条子查询。

  ```
  UPDATE t SET (a,b)=(SELECT x,y FROM z), (c,d)=(SELECT x,y FROM a))
  ```

- 如果使用了一条子查询，则它必须至多返回一行。

2.60.3 UPDATE 例子

- 更新 ORDERS 表里包含订单号 200300 相关信息的一行数据，并更改交付日期:

```
UPDATE sales.orders
SET deliv_date = DATE '2008-05-02'
WHERE ordernum = 200300;
```

- 更新 CUSTOMER 表里的多行:

```
UPDATE sales.customer SET credit = 'A1'
WHERE custnum IN (21, 3333, 324);
```

- 将 CUSTOMER 表里所有行都更新为默认 credit 'C1':

```
UPDATE sales.customer SET credit = 'C1';
```

- 更新位于芝加哥的所有部门的雇员工资:

```
UPDATE persnl.employee
SET salary = salary * 1.1
WHERE deptnum IN
   (SELECT deptnum FROM persnl.dept
   WHERE location = 'CHICAGO');
```

  针对 DEPT 表的每行进行子查询求值，返回位于芝加哥的部门编号。

- 以下为自引用UPDATE语句的例子，在这里更新的表将在子查询里进行扫描:

```
UPDATE table3
SET b = b + 2000
   WHERE a, b = (SELECT a, b FROM table3 WHERE b > 200);
```

2.61 UPDATE USING UPSERT 语句

UPDATE USING UPSERT 语法适用于大批量更新表数据,例在 5 百万记录里面去更新 8 万条记录，使用 update table set …直接去更新时，2 节点的虚拟机集群环境大概需要 35 分钟左右，但是使用 update using upsert 去更新时，大约需要 10 秒左右，性能提升很多，但是以前不支持对带 index 的 table 操作，从 EsgynDB 2.7.0 版本开始，才支持了 index 功能。

```
Update using upsert tableName set columnName = <expression>
[where columnName <condition>]
```

2.61.1 UPDATE USING UPSERT 注意事项

- 正在更新的列不能是主键的一部分。
- 表上没有显式或隐式索引。

2.61.2 UPDATE USING UPSERT 例子

- 下列用来更新没有索引的表

```
SQL>create table t_update2upsert (a int, b varchar(10));

--- SQL operation complete.

SQL>upsert using load into t_update2upsert select
element, cast('abc' || element as varchar(10)) from
udf(series(1,10000));

--- 10000 row(s) inserted.

SQL>select * from t_update2upsert limit 10;

A           B
----------- ----------
          1 abc1
          2 abc2
          3 abc3
          4 abc4
          5 abc5
          6 abc6
          7 abc7
          8 abc8
          9 abc9
         10 abc10

--- 10 row(s) selected.
```

```
SQL>explain   options   'f'   update   using   upsert
t_update2upsert set a = 1 where a <=1000;


LC   RC   OP   OPERATOR              OPT       DESCRIPT
ION           CARD
---- ---- ---- --------------------  --------  --------
------------  ---------

4    .    5    root                            x
              3.29E+001
2    3    4    tuple_flow
              3.29E+001
.    .    3    trafodion_vsbb_upser            T_UPDATE
2UPSERT       1.00E+000
1    .    2    sort
              3.29E+001
.    .    1    trafodion_scan                  T_UPDATE
2UPSERT       3.29E+001

--- SQL operation complete.

SQL>explain options 'f' update t_update2upsert set a =
1 where a <=1000;


LC   RC   OP   OPERATOR              OPT       DESCRIPT
ION           CARD
---- ---- ---- --------------------  --------  --------
------------  ---------

3    .    4    root                            x
              3.29E+001
1    2    3    tuple_flow
              3.29E+001
.    .    2    trafodion_update                T_UPDATE
2UPSERT       1.00E+000
.    .    1    trafodion_scan                  T_UPDATE
2UPSERT       3.29E+001

--- SQL operation complete.

SQL>set statistics on;

SQL>update using upsert t_update2upsert set a = 1 where
a <=1000;

--- 1000 row(s) updated.


Start Time             2020/02/27 10:46:05.703188
```

```
End Time                2020/02/27 10:46:05.984037
Elapsed Time                       00:00:00.280849
Compile Time                       00:00:00.023074
Execution Time                     00:00:00.257287


Table
Name              Records        Records      Hdfs
Hdfs I/O    Hdfs Access
                       Accessed           Used      I/Os
          Bytes     Time(usec)
TRAF_RSRVD_3:TRAFODION.SEABASE.T_UPDATE2UPSERT
                         10000           1000       102
         519964         211687
TRAF_RSRVD_3:TRAFODION.SEABASE.T_UPDATE2UPSERT
                             0           1000         7
          60978          17453


SQL>select count(*) from t_update2upsert where  a =1;

(EXPR)
--------------------
                1000

--- 1 row(s) selected.
```

- 下列用来更新有索引的表

```
SQL>create table t_update2upsert (a int, b varchar(10));

--- SQL operation complete.

SQL>create index idx_1 on t_update2upsert(a);

--- SQL operation complete.

SQL>upsert  using  load  into  t_update2upsert  select
element,  cast('abc'  ||  element  as  varchar(10))  from
udf(series(1,10000));

--- 10000 row(s) inserted.

SQL>explain   options   'f'   update   using   upsert
t_update2upsert set a = 1 where a <=1;


LC   RC   OP   OPERATOR              OPT       DESCRIPT
ION           CARD
---- ---- ---- --------------------  --------  --------
------------  ---------
```

```
10    .     11    root                              x
                  6.69E+001
6     9     10    nested_join
                  6.59E+001
7     8     9     ordered_union
                  2.00E+000
.     .     8     trafodion_vsbb_upser              IDX_1
                  1.00E+000
.     .     7     trafodion_vsbb_delet              IDX_1
                  1.00E+000
4     5     6     nested_join
                  3.29E+001
.     .     5     trafodion_vsbb_upser              T_UPDATE
2UPSERT           1.00E+000
3     .     4     sort
                  3.29E+001
1     2     3     nested_join
                  3.29E+001
.     .     2     trafodion_vsbb_scan               T_UPDATE
2UPSERT           1.00E+000
.     .     1     trafodion_index_scan              IDX_1
                  3.29E+001

--- SQL operation complete.

SQL>explain options 'f' update t_update2upsert set a =
1 where a <=1000;


LC    RC    OP    OPERATOR                OPT       DESCRIPT
ION             CARD
---- ---- ---- --------------------  --------  --------
------------  ---------

8     .     9     root                              x
                  6.69E+001
4     7     8     nested_join
                  6.59E+001
5     6     7     ordered_union
                  2.00E+000
.     .     6     trafodion_vsbb_upser              IDX_1
                  1.00E+000
.     .     5     trafodion_vsbb_delet              IDX_1
                  1.00E+000
2     3     4     nested_join
                  3.29E+001
.     .     3     trafodion_update                  T_UPDATE
2UPSERT           1.00E+000
1     .     2     sort
                  3.29E+001
```

```
.    .    1    trafodion_index_scan            IDX_1
             3.29E+001

--- SQL operation complete.

SQL>set statistics on;

SQL>update using upsert t_update2upsert set a = 1 where
a <=1000;

--- 1000 row(s) updated.


Start Time             2020/02/27 10:55:17.892307
End Time               2020/02/27 10:55:18.181715
Elapsed Time                      00:00:00.289408
Compile Time                      00:00:00.043505
Execution Time                    00:00:00.245393


Table
Name             Records        Records      Hdfs
Hdfs I/O    Hdfs Access
                     Accessed           Used      I/Os
          Bytes     Time(usec)
TRAF_RSRVD_3:TRAFODION.SEABASE.IDX_1
                         1000           1000        12
          40000          11081
TRAF_RSRVD_3:TRAFODION.SEABASE.T_UPDATE2UPSERT
                            0           1000         1
          51964         183829
TRAF_RSRVD_3:TRAFODION.SEABASE.T_UPDATE2UPSERT
                            0           1000         1
          60966          16572
TRAF_RSRVD_3:TRAFODION.SEABASE.IDX_1
                            0           1000         1
          15002          18348
TRAF_RSRVD_3:TRAFODION.SEABASE.IDX_1
                            0           1000         2
          55004          16817


SQL>select count(*) from t_update2upsert where  a =1;

(EXPR)
--------------------
                1000

--- 1 row(s) selected.

SQL>set statistics off;
```

```
SQL>set parserflags 1;

--- SQL operation complete.

SQL>select count(*) from table(index_table idx_1) where
"A@" = 1;

(EXPR)
--------------------
                1000

--- 1 row(s) selected.
```

2.62 UPSERT 语句

该语句可以更新表里的一个已存在的行或将一个不存在的行插入表里。

UPSERT 是 EsgynDB SQL 扩展语句。

```
UPSERT [USING LOAD] INTO table [(target-col-list)] { query-
expr | values-
clause}

target-col-list is:
     column-name[, column-name]...

values-clause is:
     VALUES ( expression[, expression]...)
```

2.62.1 UPSERT 语法

- `USING LOAD`

  允许在没有事务的情况下进行 UPSERT 操作。

  将数据插入一个空表时，可以使用该子句。如果您没有指定该子句，则 UPSERT 操作将在一个事务内部出现。

  注意:

  `UPSERT USING LOAD` 语句不启动事务，这是因为为了提高装载数据的性能，一般都会使用不记录日志或者不启动事务的方式，即将日志模式设置成 nologging，由于这样只会记录一部分日志，所以不能保证用户数据在出错情况下能正常回滚。

- *`table`*

  命名插入行或更新行所在的用户表。该表必须为基表。

- *`(target-col-list)`*

  指定将要插入或更新数值的表里的列。 每个目标列的数据类型都必须与其相应源表列的数据类型相兼容。

  在列表内，每个目标列都必须与关联的源表字段具有相同的位置；该位置取决于 query-expr 求值所派生的表的列。

  如果未指定 target-col-list 里的目标表的所有列，则列的默认值将在列表（该列未出现在列表中）中被插入或更新。

  如果未指定 target-col-list，则源表的行值将在表的所有列中被插入或更新。

源表里的列值顺序必须与该表 CREATE TABLE 语句里指定的列顺序相同（该顺序与 SHOWDDL 表的结果表中列的顺序相同。）

- *column-name*

  指定将要插入或更新数值的目标表里的字段。您不能限定或重复一个字段名。

- *query-expr*

  这是一条 SELECT 子查询，它返回在目标表中将被插入或更新的数据。该子查询不能引用一个正在操作的表。 更多关于“query-expr 的语法描述”，请参阅 SELECT 语句。

- VALUES *(expression[, expression]...)*

  指定一个 SQL 数值表达式或一组表达式，以便指定将被插入到目标表里的数值或在该表里进行更新的数值。 表达式的数据类型都必须与目标表的相应列的数据类型相兼容。更多信息，请参阅表达式。

2.62.2 UPSERT 例子

- 该 UPSERT 语句将 source_table 中的 1000000 行插入 target_table 中。

  假设我们有以下表:
  source_table:

```
SQL>SELECT COUNT(*) FROM source_table;
(EXPR)
--------------------
1000000
--- 1 row(s) selected.
```

  target_table:

```
SQL>CREATE TABLE target_table
(
ID INT NO DEFAULT NOT NULL NOT DROPPABLE NOT
SERIALIZED
```

```
, NUM INT DEFAULT NULL NOT SERIALIZED
, CARD_ID LARGEINT DEFAULT NULL NOT SERIALIZED
, PRICE DECIMAL(11, 3) DEFAULT NULL NOT SERIALIZED
, START_DATE DATE DEFAULT NULL NOT SERIALIZED
, START_TIME TIME(0) DEFAULT NULL NOT SERIALIZED
, END_TIME TIMESTAMP(6) DEFAULT NULL NOT SERIALIZED
, B_YEAR INTERVAL YEAR(10) DEFAULT NULL NOT
SERIALIZED
, B_YM INTERVAL YEAR(5) TO MONTH DEFAULT NULL NOT
SERIALIZED
, B_DS INTERVAL DAY(10) TO SECOND(3) DEFAULT NULL
NOT SERIALIZED
, PRIMARY KEY (ID ASC)
)
SALT USING 9 PARTITIONS
ATTRIBUTES ALIGNED FORMAT NAMESPACE 'TRAF_1500000'
HBASE_OPTIONS
(
MEMSTORE_FLUSH_SIZE = '1073741824'
);
```

完成加载1000000行大约需要10秒。

```
SQL>SET STATISTICS ON;
SQL>UPSERT USING LOAD INTO target_table SELECT * FROM
source_table;
--- 1000000 row(s) inserted.
Start Time 2018/01/18 11:38:02.739433
End Time 2018/01/18 11:48:25.822903
Elapsed Time 00:10:23.083470
Compile Time 00:00:00.381337
Execution Time 00:10:22.700870
SQL>SELECT COUNT(*) FROM target_table;
(EXPR)
--------------------
1000000
--- 1 row(s) selected.
```

- 该语句将 ODETAIL 表里的 part number 为 244 和 unit price 字段插入 PARTS 表的 part number 和 price 字段中，或使用这些数据更新 PARTS 表的 part number 和 price 字段:

```
UPSERT INTO sales.parts (partnum, price) SELECT partnum,
unit_price
FROM sales.odetail WHERE partnum = 244;
```

- 该 UPSERT 语句将从 EMPLOYEE_EUROPE 表查询的行插入到 EMPLOYEE 表

中、或更新 EMPLOYEE 表中现有的行。

```
UPSERT INTO persnl.employee
   SELECT * FROM persnl.employee_europe;
```

- 该 UPSERT 语句可以将指定值插入 DEPT 表中，或使用指定值更新 DEPT 表中的行:

```
UPSERT INTO persnl.dept
  VALUES (3500,'CHINA SALES',111,3000,'HONG KONG');
```

- 该 UPSERT 语句可以将指定值插入 DEPT 表中，或使用指定值更新 DEPT 表中的行:

```
UPSERT INTO persnl.dept (deptnum, deptname, manager)
   VALUES (3600,‘JAPAN SALES’, 996);
```

2.63 VALUES 语句

VALUES 语句以 VALUES 关键字开头，后跟有行值构造序列，其中的每个行都用括号括起来。 该语句显示行值构造函数内的表达式求值结果和行子查询的结果。

```
VALUES (row-value-const) [, (row-value-const)]...

row-value-const is:
     row-subquery
     | { expression | NULL} [,{ expression | NULL}...
```

2.63.1 VALUES 语法

row-value-const

指定表达式列表（或 NULL）或行的子查询（即可返回一行列值的子查询）。表达式的操作数无法引用一列（除非该操作数能在结果表里返回单列值的标量子查询）。

行值构造函数内的表达式求值结果和行子查询的结果必须具有兼容的数据类型。

2.63.2 VALUES 注意事项

2.63.2.1 与 SELECT 语句的关系

VALUES 语句的执行结果是一个简易表，它是 SELECT 语句里的表引用定义的一部分。 更多信息，请参阅 SELECT 语句。

2.63.2.2 与 INSERT 语句的关系

对于 INSERT 语句的直接 VALUES 子句，EsgynDB SQL 允许在该子句里使用关键字 DEFAULT（类似 NULL）。 更多信息，请参阅 INSERT 语句。

2.63.3 VALUES 例子

- 该 VALUES 语句显示两个带有简单常数的行:

  ```
  VALUES (1,2,3), (4,5,6);
  ```

```
(EXPR) (EXPR) (EXPR)
------ ------ -----
1            2     3
4            5     6

--- 2 row(s) selected.
```

- VALUES 语句将表达式和行子查询的结果显示在列表中:

```
VALUES (1+2, 3+4), (5, (select count (*) from t));

(EXPR) (EXPR)
------ ----------------- ------
     3                       7
     5                       2

--- 2 row(s) selected.
```

3. SQL 实用工具

实用工具是一种在 EsgynDB SQL 内部运行的工具，它可以执行任务，本描述了 EsgynDB SQL 实用工具。

本章目录:

3.1 CLEANUP 语句	该语句清除元数据中信息不一致的对象。
3.2 DUMP 语句	该语句导出指定对象的数据结构到指定文件。
3.3 LOAD 语句	使用 EsgynDB Bulk Loader 从一个源表（EsgynDB 表或 Hive 表）装载数据到目标 EsgynDB 表。
3.4 POPULATE INDEX 语句	装载索引。
3.5 PURGEDATA 语句	从表和索引里清除数据。
3.6 TRUNCATE 语句	从表和索引里清除数据。
3.7 UNLOAD 语句	将 EsgynDB 表里的数据卸载到您指定的 HDFS 位置。
3.8 UPDATE STATISTICS 语句	更新表中一组或多组列的直方图统计信息。这些统计信息可被用来设计优化的访问计划。

注意： EsgynDB SQL 实用工具的命令既可以交互输入，也可以使用诸如 Trafodion 命令界面（TrafCI）的基于客户端的工具从脚本文件输入。更多关于安装和配置能连接和发出SQL实用工具命令的客户端应用程序的信息，请参阅《EsgynDB安装部署指南》。

3.1 CLEANUP 语句

该语句清除元数据中信息不一致的对象。此命令可用于清除schema中的单个对象或所有对象。此命令还可用来定位EsgynDB元数据中所有不一致的对象，然后将其清除。

通过使用以下选项之一，可清除不一致的对象并将其从元数据和HBase中删除。

```
CLEANUP object-type object-name [ ,UID {object-uid} ]
CLEANUP [ PRIVATE | SHARED ] SCHEMA {schema-name}
CLEANUP UID {object-uid}
CLEANUP METADATA [ , CHECK ] [ , RETURN DETAILS ]

object-type is: TABLE | INDEX | SEQUENCE | OBJECT
object-name is: Name of the object that needs to be cleaned
up
object-uid is: UID (Unique ID) of the object that is stored
in metadata
schema-name is: Name of the schema whose contents need to
be cleaned up
```

3.1.1 CLEANUP 语法

- `CLEANUP object-type object-name [ , UID {object-uid} ]`

  此命令将清除所有EsgynDB元数据表和HBase中指定的对象。如果在元数据中找不到对象，但该对象存在于HBase中，则它将从HBase中删除。所有依赖对象（如索引、视图、LOB、内部序列和约束）也将从元数据和HBase中删除。

- `object-type`
  - 如果将对象类型指定为TABLE、INDEX或SEQUENCE，并且该对象存在于元数据中，则将执行检查以确保指定的object-name具有指定的类型。如果它们不相同，则返回一个错误。

  - 如果将对象类型指定为OBJECT，则任何与指定名称匹配的对象都将从元数据和HBase中删除。

- `object-name`
  指定要删除的对象名称。

- `object-uid`
  如果OBJECTS元数据表中存在对象，则将存储对象UID与指定对象UID进行比较。如果它们不匹配，则不进行清理。如果OBJECTS元数据表中不存在对象，则从元数据表中删除与指定对象UID匹配的所有条目。

- `CLEANUP [ PRIVATE | SHARED ] SCHEMA {schema-name}`

  该命令将清除在指定schema中创建的所有对象。

- `CLEANUP UID {object-uid}`
  该命令将从元数据表中删除所有与指定对象UID匹配的条目。如果指定对象UID在OBJECTS元数据表中，并且可以检索到与该UID关联的对象名，则该对象也将从HBase中删除。

- `CLEANUP METADATA [ , CHECK ] [ , RETURN DETAILS ]`
  此命令将查找元数据中不一致的条目，返回相关信息并进行清理。
  进行多次检查以验证元数据的一致性。

- `CHECK`

  可以检查不一致的元数据条目，而不实际清理它们。这可通过指定CHECK选项来完成。如果指定了CHECK，会同时执行以下检查，并将返回属于下列类别的不一致对象的摘要。

  - 孤立元数据对象
    此检查用于验证元数据中存在的对象是否也存在于HBase中。如果该对象在HBase中不存在，则将其标记为孤立元数据对象。
  - 孤立HBase对象
    此检查用于验证HBase中存在的对象是否也存在于元数据中。如果该对象在元数据中不存在，则将其标记为孤立HBase对象。
  - 不一致的对象条目

此检查用于确保 OBJECTS 表中所有条目同样存在于 OBJECTS 唯一索引(OBJECTS_UNIQ_IDX)中，并且 OBJECTS_UNIQ_IDX 中的所有条目也存在于 OBJECTS 表中。它还要确保 COLUMNS 表中的所有条目在 OBJECTS 表中都有相应的条目。

如果之前的任何检查失败，则该条目被标记为不一致。

- 不一致的视图条目

  此检查用于确保视图和 OBJECTS 表中的所有视图条目都一致。

- RETURN DETAILS

  如果指定了 RETURN DETAILS，则还会显示有关对象不一致的详细信息。

3.1.2 CLEANUP 注意事项

3.1.2.1 安全和权限

要清理对象，需要具有 DROP 权限。如果无法从元数据中检索对象权限，或者正在进行元数据清理，则需要 DB_ROOT 权限来执行清理。

3.1.2.2 访问期间的一致性检查

如果一个对象存在于元数据中，但不存在于 HBase 中，则在任何 DML 或 DDL 操作期间访问该对象，将返回错误 4254。

在任何复杂的 DDL 操作期间，对象也被标记为不一致。例如，如果 CREATE TABLE 也创建隐式索引或约束，则在完成复杂的 DDL 操作之前，该对象将被标记为不一致。如果发生异常错误（例如进程崩溃或节点故障），则该对象仍然不一致。任何访问该对象的尝试都将返回错误 4254。

在访问不一致的对象之前，需要清理它。

3.1.3 CLEANUP 例子

- 访问不一致的对象

  表 T026T1 已被标记为与先前会话不一致。对它的任何操作都将返回错误

4254。对于此测试用例，已完成元数据更新，并且该表在 OBJECTS 表中被标记为无效。

```
>>invoke t026t1;
*** ERROR[4254] Object TRAFODION.SCH026.T026T1 has
invalid state and cannot be
accessed. Use cleanup command to drop it.
*** ERROR[4082] Object TRAFODION.SCH026.T026T1 does not
exist or is inaccessible.
--- SQL operation failed with errors.
>>drop table t026t1;
*** ERROR[4254] Object TRAFODION.SCH026.T026T1 has
invalid state and cannot be
accessed. Use cleanup command to drop it.
--- SQL operation failed with errors.
>>select * from t026t1;
*** ERROR[4254] Object TRAFODION.SCH026.T026T1 has
invalid state and cannot be
accessed. Use cleanup command to drop it.
*** ERROR[4082] Object TRAFODION.SCH026.T026T1 does not
exist or is inaccessible.
*** ERROR[8822] The statement was not prepared.
>>
```

- 访问不正确的对象

  如果指定了对象类型，则存储在元数据中的对象类型必须与其匹配。在本例中，对象“TI”是索引，因此无法作为表进行清理。

```
>>create index ti on t(b);
--- SQL operation complete.
>>cleanup table ti;
*** ERROR[1389] Object TI does not exist in Trafodion.
*** ERROR[4256] Object type specified for this object
does not match the object type
stored in metadata.
--- SQL operation failed with errors.
>>
```

- 清理不一致的对象

  使用 CLEANUP 命令清理处于无效状态的对象。

```
>>drop table t026t1;
*** ERROR[4254] Object TRAFODION.SCH026.T026T1 has
invalid state and cannot be
```

```
accessed. Use cleanup command to drop it.
--- SQL operation failed with errors.
>>cleanup table t026t1;
--- SQL operation complete.
>>invoke t026t1;
*** ERROR[4082] Object TRAFODION.SCH026.T026T1 does not
exist or is inaccessible.
--- SQL operation failed with errors.
>>
```

DROP TABLE、CREATE TABLE、DROP INDEX 和 CREATE INDEX 失败将调用 cleanup。

- 通过指定对象的UID来清理对象

  通过对象 UID 来指定对象 T 并清理它。

```
>>select object_uid from "_MD_".objects where
schema_name = 'SCH026' and object_name
= 'T';
OBJECT_UID
--------------------
127287511636506969
--- 1 row(s) selected.
>>cleanup uid 127287511636506969;
--- SQL operation complete.
>>invoke t;
*** ERROR[4082] Object TRAFODION.SCH026.T does not
exist or is inaccessible.
--- SQL operation failed with errors.
>>
```

- 元数据清理

  该例子显示元数据是一致的，没有什么问题。

```
>>cleanup metadata, check, return details;
Metadata Cleanup: started, check only
Start: Cleanup Orphan Objects Entries
End: Cleanup Orphan Objects Entries (0 entries found)
Start: Cleanup Orphan Hbase Entries
End: Cleanup Orphan Hbase Entries (0 entries found)

Start: Cleanup Inconsistent Objects Entries
End: Cleanup Inconsistent Objects Entries (0 entries
found)
Start: Cleanup Inconsistent Views Entries
End: Cleanup Inconsistent Views Entries (0 entries
```

```
found)
Metadata Cleanup: done
--- SQL operation complete.
>>
```

该例子显示元数据中的不一致性:

```
>>cleanup metadata, check, return details;
Metadata Cleanup: started, check only
Start: Cleanup Orphan Objects Entries
Entry #1(OBJECT): TRAFODION.SCH026.T026T10
End: Cleanup Orphan Objects Entries (1 entry found)
Start: Cleanup Orphan Hbase Entries
Entry #1(OBJECT): TRAFODION.SCH026.T026T11
End: Cleanup Orphan Hbase Entries (1 entry found)
Start: Cleanup Inconsistent Objects Entries
Entry #1(OBJECT): TRAFODION.SCH026.T026T12
Entry #2(UID): 29899934458095403
End: Cleanup Inconsistent Objects Entries (2 entries
found)
Start: Cleanup Inconsistent Views Entries
End: Cleanup Inconsistent Views Entries (0 entries
found)
Metadata Cleanup: done
--- SQL operation complete.
>>
```

运行 CLEANUP METADATA 命令清除不一致的元数据:

```
>>cleanup metadata, return details;
Metadata Cleanup: started
Start: Cleanup Orphan Objects Entries
Entry #1(OBJECT): TRAFODION.SCH026.T026T10
End: Cleanup Orphan Objects Entries (1 entry cleaned
up)
Start: Cleanup Orphan Hbase Entries
Entry #1(OBJECT): TRAFODION.SCH026.T026T11
End: Cleanup Orphan Hbase Entries (1 entry cleaned up)
Start: Cleanup Inconsistent Objects Entries
Entry #1(OBJECT): TRAFODION.SCH026.T026T12
Entry #2(UID): 29899934458095403
End: Cleanup Inconsistent Objects Entries (2 entries
cleaned up)
Start: Cleanup Inconsistent Views Entries
End: Cleanup Inconsistent Views Entries (0 entries
cleaned up)
Metadata Cleanup: done
--- SQL operation complete.
>>
>>cleanup metadata, check, return details;
```

```
Metadata Cleanup: started, check only
Start: Cleanup Orphan Objects Entries
End: Cleanup Orphan Objects Entries (0 entries found)
Start: Cleanup Orphan Hbase Entries
End: Cleanup Orphan Hbase Entries (0 entries found)
Start: Cleanup Inconsistent Objects Entries
End: Cleanup Inconsistent Objects Entries (0 entries
found)
Start: Cleanup Inconsistent Views Entries
End: Cleanup Inconsistent Views Entries (0 entries
found)
Metadata Cleanup: done
--- SQL operation complete.
```

3.2 DUMP 语句

该语句的作用是导出指定对象的数据结构到指定文件。

```
DUMP object object_name TO FILE file_path

object is:
     SCHEMA
     | TABLE
     | VIEW
     | LIBRARY
     | FUNCTION
     | SEQUENCE
     | PROCEDURE
     | TRIGGER
     | PACKAGE
```

3.2.1 DUMP 语法

- object

  object 代表的是要 DUMP 出的对象类型。当导出对象为 SCHEMA 时，会导出该 SCHEMA 下除系统对象外的所有对象。

- file_path

  file_path 是你指定的导出的文件路径。可以使用绝对路径，也可以使用相对路径。使用相对路径时。如果使用的命令行工具是 sqlci，则相对路径以此时使用 sqlci 的路径为基准。如果使用的命令行工具是 trafci 时，则以 TRAF_HOME 目录为基准。当路径的最后一级目录不存在且有写权限创建时会自动创建，否则会报错找不到对应目录。

3.2.2 DUMP 注意事项

3.2.2.1 关于导出对象的权限

执行一个 DUMP 语句，以下条件之一必须成立：

- 您是 DB__ROOT（数据库根用户）。
- 您拥有 SQL_OPERATIONS 组件的 MANAGE_LOAD 组件权限。

3.2.2.2 关于导出路径的注意事项

执行一个 DUMP 语句，必须注意：

- 您对指定的导出路径有写权限。

主要为写入权限。如果导出路径不具备相应的权限，将会导出失败。

- 导出路径必须使用单引号。
- 导出路径区分大小写。
- 导出路径可存在，可不存在。当导出路径存在时直接导出即可。当导出路径不存在时，只限于最后一级目录。如果只有最后一级目录不存在时，会直接创建目录且导出文件，如果连续多个目录不存在时，会提示失败。

3.2.2.3 关于导出文件的注意事项

执行一个 DUMP 语句

如果执行的导出对象是 SCHEMA 时，导出文件的命名为：

CATALOGNAME_SCHEMANAME

如果执行的导出对象不是 SCHEMA 时，导出文件的命名为：

CATALOGNAME_SCHEMANAME_OBJECTNAME

3.2.3 DUMP 例子

- 该例子导出指定的表到指定路径，此处使用绝对路径：

  `DUMP TABLE trafodion.myschema.mytable TO FILE '/home/esgyn/dump';`

  会在“/home/esgyn/dump”下生成表的导出文件“TRAFODION_MYSCHEMA_MYTABLE”。

- 该例子导出指定 schema 到指定路径，此处使用相对路径：

  `DUMP SCHEMA myschema TO FILE './';`

  会在当前目录下生成 SCHEMA 的导出文件“TRAFODION_MYSCHEMA”。

3.3 LOAD 语句

该语句使用 EsgynDB Bulk Loader 从一个源表（EsgynDB 表或 Hive 表）装载数据到目标 EsgynDB 表。

EsgynDB Bulk Loader 直接在区域服务器里预处理和装载 HFile，由此绕过了写入路径和相关的成本。

写入路径开始于客户端，再将数据移至区域服务器，并在数据最终写入 HFile 的 HBase 数据文件后结束。

EsgynDB 批量装载进程将分为以下几个阶段进行:

- **禁用索引（如果未使用增量索引构建）**
- **准备（花费大部分时间，批量装载操作的核心）**
  - 读取源文件（EsgynDB 表、Hive 表或 Hive 外部表）
  - 以 EsgynDB 编码数据
  - 重新分区和排序数据以匹配目标表的区域
  - 写入 HFile 数据
  - 数据重新分区并写入索引 HFiles（如果使用增量索引构建）
- **完成**（有无快照恢复）
  - 拍摄表格的快照
  - 将 HFiles 合并到 HBase 表中（非常快 - 移动，不是副本）
  - 如果合并失败，则删除快照或从快照还原
- **填充索引**（如果未使用增量索引构建）
- **清理**
  - HFiles 临时空间清理

LOAD 是 EsgynDB SQL 扩展语句。

```
LOAD [WITH option[[,] option]...] INTO target-table
SELECT ... FROM source-table

option is:
     CONTINUE ON ERROR
    | LOG ERROR ROWS [TO error-location-name]
    | STOP AFTER num ERROR ROWS
```

```
| TRUNCATE TABLE
| NO POPULATE INDEXES
| NO DUPLICATE CHECK
| NO OUTPUT
| NO RECOVERY
| INDEX TABLE ONLY
| REBUILD INDEXES
| UPSERT USING LOAD
```

3.3.1 LOAD 语法

- `target-table`

  这是数据将被载入的 EsgynDB 目标表的名称。更多信息，请参阅数据库对象名称。

- `source-table`

  这是带有源数据的 EsgynDB 表或 Hive 表的名称。

  可以使用 HIVE.HIVEschema（例如 hive.hive.orders）在 EsgynDB 里访问 Hive 表。 在 EsgynDB 访问之前，Hive 表需要存在于 Hive 中。

  如果您希望装载 HDFS 文件夹里现有的数据，则需要创建一个外部 Hive 表，它具有正确的字段、并指向包含数据的 HDFS 文件夹。您也可以在源数据上指定一个 WHERE 子句作为过滤器。

- `[WITH option[[,] option]...]`

  可以为装载操作指定的一组选项。可以指定其中一个或多个选项：

  - `CONTINUE ON ERROR`

    在从源表扫描行或加载到目标表中时，LOAD 语句将在可忽略的错误之后继续。可忽略的错误通常是数据转换错误。

    装载或排序阶段的错误将导致 LOAD 语句中止。

    如果指定了 `LOG ERROR ROWS [TO *error-location-name*]`（日志错误行）或 `STOP AFTER *num* ERROR ROWS`（num 错误行之后停止），则隐含此选项。

  - `LOG ERROR ROWS [TO error-location-name]`

    - 错误行

如果必须将错误行写入指定位置，则要指定 TO error-location-name，否则它们将被写入默认位置。 *error-location-name* 必须是 EsgynDB 具有写入访问权限的 HDFS 目录名。

如果未指定错误日志位置，错误行将记录在**目录** /user/trafodion/bulkload/logs 中的 HDFS 文件中。

子目录的默认名称是ERR_catalog.schema.target_table_date_id，其中 id 是出现错误进程的唯一数字标识符时间戳（YYYYMMDD_HHMMSS）。

错误文件的默认名称是 *loggingFileNamePrefix_catalog.schema.target_table_instanceID*，其中 *loggingFileNamePrefix* 是 hive_scan_err 或 traf_upsert_err，具体取决于数据源表，*instanceID* 是从 0 开始的实例 ID，通常只有一个实例。

例如，test_load_log 表的完整路径是 /user/trafodion/bulkload/logs/test/ERR_TRAFODION.SEABASE.TEST_LOAD_LOG_20171218_035918/traf_upsert_err_TRAFODION.SEABASE.TEST_LOAD_LOG_0，

其中：

1. /user/trafodion/bulkload/logs/test 是**目录**的默认名称。

2.ERR_TRAFODION.SEABASE.TEST_LOAD_LOG_20171218_035918 是**子目录**的默认名称。

3.

traf_upsert_err_TRAFODION.SEABASE.TEST_LOAD_LOG_0 是**错误文件**的默认名称。

- 错误日志

  错误日志由表示给定位置中的 load 命令子目录下的 load 命令中涉及的进程写入单独的文件中。

  实际的日志文件位置显示在 load 命令输出中。建议使用相同的位置进行装载，因为这样更容易找到错误日志。

- STOP AFTER num ERROR ROWS

  在执行此语句期间遇到 num 错误后，使用此选项可以停止 LOAD 语句的处理。

  重复值检查错误和使用前面的选项继续处理的错误都包含在计数中。

  - num

    指定错误行数。

- TRUNCATE TABLE

  导致 Bulk Loader 在装载之前截断目标表。
  默认情况下，Bulk Loader 不会在装载数据之前截断目标表。

- NO RECOVERY

  指定 Bulk Loader 不得使用 HBase 快照进行数据恢复。
  默认情况下，Bulk Loader 会使用 HBase 快照机制处理数据恢复。

- NO POPULATE INDEXES

  指定 Bulk Loader 不得处理索引维护或填充索引。
  默认情况下，Bulk Loader 处理索引维护，在装载之前禁用索引，并在装载之后填充它们。

- NO DUPLICATE CHECK

导致 Bulk Loader 忽略源数据里的重复值。

默认情况下，Bulk Loader 检查源数据里是否有重复值，并在检测到重复值时产生一个错误。

- NO OUTPUT

  防止 LOAD 语句显示状态消息。默认情况下，LOAD 语句可以打印状态消息，其中将列出 Bulk Loader 正在执行的步骤。

- INDEX TABLE ONLY

  指定目标表（索引）填充来自父表的数据。

- UPSERT USING LOAD

  指定在没有事务的情况使用行集（rowset）插入数据目标表的数据。

- REBUILD INDEXES

  指定在更新源表时将自动更新目标表的索引。这是 LOAD 语句的默认行为，也就是说，即使未指定此选项，LOAD 语句也将重建索引，除非 CQD TRAF_LOAD_ALLOW_RISKY_INDEX_MAINTENANCE 已打开。

  默认情况下，此 CQD 处于关闭状态。在这种情况下，使用不带选项的 LOAD 语句所用的时间几乎相等。

- UPSERT USING LOAD

  指定使用没有事务的行集插入将数据插入到目标表。

  在这种情况下，EsgynDB 在一次插入多行时使用常规 HBase 接口（Put）而不是 HBase Bulk Loader。

3.3.2 LOAD 注意事项

3.3.2.1 必需的权限

执行 LOAD 语句，必须满足以下条件之一：

- 您是 DB__ROOT（数据库根用户）。
- 您是目标表的所有者。
- 您具有以下这些权限：
  - 目标表的 SELECT 和 INSERT 权限。

- 目标表的 DELETE 权限（如果已指定 TRUNCATE TABLE）。
- 您拥有 SQL_OPERATIONS 组件的 MANAGE_LOAD 组件权限。

3.3.2.2 运行 LOAD 语句之前的配置

运行 LOAD 语句之前必须保证您已经根据这些准则配置了暂存文件夹、源表和 HBase。

3.3.2.3 HFiles 的暂存文件夹

Bulk Loader 使用 HDFS 文件夹作为 HFiles 的暂存区。默认情况下，EsgynDB 使用/bulkload/作为暂存文件夹。该文件夹必须与运行 EsgynDB 的文件夹为同一个用户拥有。EsgynDB 也必须拥有该文件夹的完全权限。HBase 用户（运行 HBase 的用户）必须具有该文件夹的读写访问权限。

例如：`drwxr-xr-x - trafodion trafodion 0 2014-07-07 09:49 /bulkload`.

3.3.2.4 提高吞吐量

以下 CQD（控制查询默认值）设置有助于提高 Bulk Loader 吞吐量：

- TRAF_LOAD_MAX_HFILE_SIZE

  指定 HFile 大小限制，超过该限制将关闭当前文件并为同一分区创建新文件。调整此设置以最小化 HBase 拆分/合并。

  默认值：10GB
- `TRAF_LOAD_PREP_TMP_LOCATION`

  指定在装载期间创建 HFile 的 HDFS 目录。

  默认值：`/bulkload`

另外，如果您的源数据是干净的，请考虑使用 `NO DUPLICATE CHECK` 来提高吞吐量。

3.3.2.5 Hive 源表

为了装载 HDFS 里存储的数据，您需要在启动装载之前创建一个 Hive 表，它具有正确的字段和类型，并且指向包含数据的 HDFS 文件夹。

3.3.2.6 HBase 快照

如果您没有在 LOAD 语句里指定 NO RECOVERY OPTION，则 Bulk Loader 将使用 HBase 快照作为数据恢复机制。

快照是一种轻量级操作，它可以复制一些元数据（数据不会被复制）。装载开始

之前拍摄一个快照，并在装载成功后删除此快照。如果出现问题，可以进行数据恢复；此时，快照可以将表恢复到装载之前的初始状态。

为了使用这种数据恢复机制，需要配置 HBase 为“允许快照”。

3.3.2.7 增量装载

Bulk Loader 默认允许增量装载。在第二阶段开始之前生成快照，在批量装载完成时删除。

如果装载出现问题，则恢复快照到之前的状态。

3.3.2.8 非增量装载

以下批量装载选项可用于执行非增量装载：

- NO RECOVERY：不要生成表的快照。
- TRUNCATE TABLE：在开始装载之前截断表。

3.3.2.9 空间使用

在转换期间，将自动创建 SYSKEY、SALT、identity 和 divisioning 列的目标表值。 HBase 文件的大小取决于编码、压缩、HDFS 复制因子和行格式。通过向表中添加数据，可以使用 salting（一种 Java 程序）将目标表预先分割为多个区域。

3.3.2.10 性能

总体吞吐量受行格式、行长度、列数、数据偏差等 LOAD sas upsert 语义学（未使用现有数据检查重复约束）因素的影响。LOAD 具有比类似的持续装载（INSERT、UPSERT 或 UPSERT USING LOAD）更低的 CPU 和磁盘活动。而且，LOAD 在完成后的压缩活动也比持续加载低。

3.3.2.11 Hive 扫描

Hive 表数据的直接访问支持：

- 仅文本输入格式和序列文件。
- 仅结构化数据类型。

必须通过 Hive 本身创建/删除/更改表。

EsgynDB:

- 读取 Hive 元数据以确定有关表的信息。
- 可以在 Hive 表上执行 UPDATE STATISTICS——提高性能!
- 可以以文本和序列格式（由 UNLOAD 使用）写入 Hive 表。

3.3.3 LOAD 例子

3.3.3.1 CONTINUE ON ERROR 例子

假设我们有以下表格:

source_table:

```
SQL>SELECT * FROM source_table;
A B
------------ ----
           1 aaaa
           2 bbbb
           3 cccc
           4 dd
           5 ee
           6 fff
--- 6 row(s) selected.

SQL>SHOWDDL source_table;
CREATE TABLE TRAFODION.SEABASE.SOURCE_TABLE
    (
      A INT DEFAULT NULL NOT SERIALIZED
    , B CHAR(4) CHARACTER SET ISO88591 COLLATE
        DEFAULT DEFAULT NULL NOT SERIALIZED
    )
ATTRIBUTES ALIGNED FORMAT
;
--- SQL operation complete.
```

target_table1:

```
SQL>SELECT * FROM target_table1;
--- 0 row(s) selected.
SQL>SHOWDDL target_table1;
CREATE TABLE TRAFODION.SEABASE.TARGET_TABLE1
(
```

```
A INT DEFAULT NULL NOT SERIALIZED
, B CHAR(3) CHARACTER SET ISO88591 COLLATE
DEFAULT DEFAULT NULL NOT SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT
;
--- SQL operation complete.
```

- 下面的例子演示了当发生可忽略的数据转换错误时，LOAD 语句在有或没有 CONTINUE ON ERROR 情况下的行为。
- 如果未指定 CONTINUE ON ERROR，则将数据从 *source_table* 加载到 *target_table1* 时，操作因出现数据转换错误而失败。

```
SQL>LOAD INTO target_table1 SELECT * FROM source_table;
UTIL_OUTPUT
-------------------------------------------------------
---------------------------
---------
Task: LOAD Status: Started Object:
TRAFODION.SEABASE.TARGET_TABLE1
Task: CLEANUP Status: Started Time: 2018-01-03
16:15:53.222441
Task: CLEANUP Status: Ended Time: 2018-01-03
16:15:53.250826
Task: CLEANUP Status: Ended Elapsed Time: 00:00:00.028
Task: LOADING DATA Status: Started Time: 2018-01-03
16:15:53.250909
*** ERROR[8402] A string overflow occurred during the
evaluation of a character
expression. Conversion of Source
Type:CHAR(REC_BYTE_F_ASCII,4 BYTES,ISO88591)
Source Value:aaaa to Target
Type:CHAR(REC_BYTE_F_ASCII,3 BYTES,ISO88591). [2018-01-
03 16:15:54]
```

- 如果指定了 CONTINUE ON ERROR，则将数据从 source_table 加载到 target_table1 时，操作在出现可忽略的数据转换错误之后成功。

```
SQL>LOAD WITH CONTINUE ON ERROR INTO target_table1
SELECT * FROM source_table;
Task: LOAD Status: Started Object:
TRAFODION.SEABASE.TARGET_TABLE1
Task: CLEANUP Status: Started Object:
TRAFODION.SEABASE.TARGET_TABLE1
Task: CLEANUP Status: Ended Object:
```

```
TRAFODION.SEABASE.TARGET_TABLE1
Task:    PREPARATION    Status:    Started    Object:
TRAFODION.SEABASE.TARGET_TABLE1
Rows Processed: 3
Task: PREPARATION Status: Ended ET: 00:00:03.151
Task:    COMPLETION    Status:    Started    Object:
TRAFODION.SEABASE.TARGET_TABLE1
Task: COMPLETION Status: Ended ET: 00:00:01.137
--- 3 row(s) loaded.
UTIL_OUTPUT
-------------------------------------------------------
---------------------------
---------
Task:      LOAD      Status:      Started      Object:
TRAFODION.SEABASE.TARGET_TABLE1
Task:   CLEANUP   Status:   Started   Time:   2018-01-03
16:19:43.543405
Task:   CLEANUP   Status:   Ended   Time:   2018-01-03
16:19:43.568828
Task: CLEANUP Status: Ended Elapsed Time: 00:00:00.025
Task:  LOADING  DATA  Status:  Started  Time:  2018-01-03
16:19:43.568899
Rows Processed: 6
Error Rows: 3
Task:  LOADING  DATA  Status:  Ended  Time:  2018-01-03
16:19:44.211150
Task:   LOADING   DATA   Status:   Ended   Elapsed   Time:
00:00:00.642
Task:  COMPLETION  Status:  Started  Time:  2018-01-03
16:19:44.211192
Rows Loaded: 3
Task:   COMPLETION   Status:   Ended   Time:   2018-01-03
16:19:45.171458
Task:   COMPLETION   Status:   Ended   Elapsed   Time:
00:00:00.960
--- SQL operation complete.
SQL>SELECT * FROM target_table1;
A B
----------- ----
4 dd
5 ee
6 fff
--- 3 row(s) selected.
```

3.3.3.2 LOG ERROR ROWS [TO error-location-name]例子

假设我们有两个相同的表（source_table 和 target_table1），如 CONTINUE ON ERROR 例中所示。

- 该例子说明了 LOAD 语句如何加载数据并将错误行记录到默认目录 user/trafodion/bulkload/logs。

```
SQL>LOAD WITH LOG ERROR ROWS INTO target_table1 SELECT *
FROM source_table;
UTIL_OUTPUT
------------------------------------------------------------
---------------------------
-------
Task: LOAD Status: Started Object:
TRAFODION.SEABASE.TARGET_TABLE1
Task: CLEANUP Status: Started Time: 2018-01-03
16:23:03.142862
Task: CLEANUP Status: Ended Time: 2018-01-03
16:23:03.151725
Task: CLEANUP Status: Ended Elapsed Time: 00:00:00.009
Logging Location:
/user/trafodion/bulkload/logs/ERR_TRAFODION.SEABASE.TARGE
T_TABLE_20180103_082303
Task: LOADING DATA Status: Started Time: 2018-01-03
16:23:03.151815
Rows Processed: 6
Error Rows: 3
Task: LOADING DATA Status: Ended Time: 2018-01-03
16:23:03.920270
Task: LOADING DATA Status: Ended Elapsed Time:
00:00:00.768
Task: COMPLETION Status: Started Time: 2018-01-03
16:23:03.920313
Rows Loaded: 3
Task: COMPLETION Status: Ended Time: 2018-01-03
16:23:04.301579
Task: COMPLETION Status: Ended Elapsed Time: 00:00:00.381
--- SQL operation complete.
SQL>SELECT * FROM target_table1;
A B
----------- ----
4 dd
5 ee
6 fff
--- 3 row(s) selected.
[root@cent-1 bin]$ hdfs dfs -ls
/user/trafodion/bulkload/logs/
Found 1 items
drwxr-xr-x - trafodion trafodion 0 2018-01-13 16:23
/user/trafodion/bulkload/logs/ERR_TRAFODION.SEABASE.TARGE
T_TABLE_20180103_082303
```

- 该例子显示了LOAD语句如何将错误行加载并记录到指定目录 user/trafodion/bulkload/error_log。

```
SQL>LOAD WITH LOG ERROR ROWS TO '/BULKLOAD/ERROR_LOG'
INTO target_table1 SELECT *
FROM source_table;
UTIL_OUTPUT
------------------------------------------------------------
---------------------------
-------
Task: LOAD Status: Started Object:
TRAFODION.SEABASE.TARGET_TABLE1
Task: CLEANUP Status: Started Time: 2018-01-03
17:19:43.436689
Task: CLEANUP Status: Ended Time: 2018-01-03
17:19:43.456761
Task: CLEANUP Status: Ended Elapsed Time: 00:00:00.020
Logging Location:
/bulkload/error_log/ERR_TRAFODION.SEABASE.TARGET_TABLE_20
180103_091943
Task: LOADING DATA Status: Started Time: 2018-01-03
17:19:43.456804
Rows Processed: 6
Error Rows: 3
Task: LOADING DATA Status: Ended Time: 2018-01-03
17:19:43.722825
Task: LOADING DATA Status: Ended Elapsed Time:
00:00:00.266
Task: COMPLETION Status: Started Time: 2018-01-03
17:19:43.722868
Rows Loaded: 3
Task: COMPLETION Status: Ended Time: 2018-01-03
17:19:44.591544
Task: COMPLETION Status: Ended Elapsed Time: 00:00:00.869
--- SQL operation complete.
[root@cent-1 bin]$ hdfs dfs -ls /bulkload/error_log
Found 1 items
drwxr-xr-x - trafodion trafodion 0 2018-01-03 17:19
/bulkload/error_log/ERR_TRAFODION.SEABASE.TARGET_TABLE_20
180103_091943
```

3.3.3.3 STOP AFTER num ERROR ROWS 例子

假设我们有两个相同的表（source_table 和 target_table1），如 CONTINUE ON ERROR 例中所示。

下面的例子说明了 LOAD 语句如何根据不同的 num 执行操作。

```
SQL>LOAD WITH STOP AFTER 2 ERROR ROWS INTO target_table1
SELECT * FROM source_table;
```

```
UTIL_OUTPUT
------------------------------------------------------------
------------------------------
------
Task: LOAD Status: Started Object:
TRAFODION.SEABASE.TARGET_TABLE1
Task: CLEANUP Status: Started Time: 2018-01-05
10:53:52.20569
Task: CLEANUP Status: Ended Time: 2018-01-05
10:53:52.45689
Task: CLEANUP Status: Ended Elapsed Time: 00:00:00.025
Task: LOADING DATA Status: Started Time: 2018-01-05
10:53:52.45757
*** WARNING[8114] The number of error rows is 3 [2018-01-
05 10:53:53]
*** ERROR[8113] The maximum number of error rows is
exceeded. [2018-01-05 10:53:53]
*** WARNING[8114] The number of error rows is 3 [2018-01-
05 10:53:53]
SQL>SELECT * FROM target_table1;
--- 0 row(s) selected.
SQL>LOAD WITH STOP AFTER 3 ERROR ROWS INTO target_table1
SELECT * FROM source_table;
UTIL_OUTPUT
------------------------------------------------------------
------------------------------
------
Task: LOAD Status: Started Object:
TRAFODION.SEABASE.TARGET_TABLE1
Task: CLEANUP Status: Started Time: 2018-01-05
15:55:58.975459
Task: CLEANUP Status: Ended Time: 2018-01-05
15:55:59.20219
Task: CLEANUP Status: Ended Elapsed Time: 00:00:00.045
Task: LOADING DATA Status: Started Time: 2018-01-05
15:55:59.20322
Rows Processed: 6
Error Rows: 3
Task: LOADING DATA Status: Ended Time: 2018-01-05
15:55:59.960109
Task: LOADING DATA Status: Ended Elapsed Time:
00:00:00.940
Task: COMPLETION Status: Started Time: 2018-01-05
15:55:59.960180
Rows Loaded: 3
Task: COMPLETION Status: Ended Time: 2018-01-05
15:56:00.448496
Task: COMPLETION Status: Ended Elapsed Time: 00:00:00.488
--- SQL operation complete.
SQL>SELECT * FROM target_table1;
A B
```

```
----------- ----
4 dd
5 ee
6 fff
--- 3 row(s) selected.
```

3.3.3.4 INDEX TABLE ONLY 例子

假设我们有以下表格:

source_table:

```
SQL>SELECT * FROM source_table;
A B
----------- ----
1 aaaa
2 bbbb
3 cccc
4 dd
5 ee
6 fff
--- 6 row(s) selected.
SQL>SHOWDDL source_table;
CREATE TABLE TRAFODION.SEABASE.SOURCE_TABLE
(
A INT DEFAULT NULL NOT SERIALIZED
, B CHAR(4) CHARACTER SET ISO88591 COLLATE
DEFAULT DEFAULT NULL NOT SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT
;
--- SQL operation complete.
```

target_table1:

```
SQL>SELECT * FROM target_table1;
--- 0 row(s) selected.
SQL>SHOWDDL target_table1;
CREATE TABLE TRAFODION.SEABASE.TARGET_TABLE1
(
A INT DEFAULT NULL NOT SERIALIZED
, B CHAR(3) CHARACTER SET ISO88591 COLLATE
DEFAULT DEFAULT NULL NOT SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT
;
--- SQL operation complete.
```

target_table2:

```
SQL>SELECT * FROM target_table2;
--- 0 row(s) selected.
SQL>SHOWDDL target_table2;
CREATE TABLE TRAFODION.SEABASE.TARGET_TABLE2
(
A INT NO DEFAULT NOT NULL NOT DROPPABLE NOT
SERIALIZED
, B CHAR(4) CHARACTER SET ISO88591 COLLATE
DEFAULT DEFAULT NULL NOT SERIALIZED
, PRIMARY KEY (A ASC)
)
ATTRIBUTES ALIGNED FORMAT
;
--- SQL operation complete.
```

target_table3:

```
SELECT * FROM target_table3;
--- 0 row(s) selected.
SHOWDDL target_table3;
CREATE TABLE TRAFODION.SEABASE.TARGET_TABLE3
(
A INT NO DEFAULT NOT NULL NOT DROPPABLE NOT
SERIALIZED
, B CHAR(4) CHARACTER SET ISO88591 COLLATE
DEFAULT DEFAULT NULL NOT SERIALIZED
, PRIMARY KEY (A ASC)
)
ATTRIBUTES ALIGNED FORMAT
;
--- SQL operation complete.
```

target_table4:

```
SELECT * FROM target_table4;
--- 0 row(s) selected.
CREATE TABLE TRAFODION.SEABASE.TARGET_TABLE4
(
A INT DEFAULT NULL NOT SERIALIZED
, B CHAR(4) CHARACTER SET ISO88591 COLLATE
DEFAULT DEFAULT NULL NOT SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT
;
--- SQL operation complete.
```

- 下面的例子演示了如何填充索引表和目标表。
    - 如果指定NO POPULATE，填充索引表，但不填充目标表。

```
SQL>CREATE INDEX index_target_table1 ON target_table1(b)
NO POPULATE;
--- SQL operation complete.
SQL>SET PARSERFLAGS 1;
--- SQL operation complete.
SQL>LOAD WITH INDEX TABLE ONLY INTO TABLE(INDEX_TABLE
index_target_table1) SELECT
b,a FROM source_table;
--- SQL operation complete.
SQL>SELECT * FROM target_table1;
--- 0 row(s) selected.
SELECT * FROM table(index_table index_target_table1);
B@ A
---- --------------------
aaaa 1
bbbb 2
cccc 3
dd 4
ee 5
fff 6
--- 6 row(s) selected.
```

- 如果未指定NO POPULATE，填充索引表，并填充目标表。

```
SQL>CREATE INDEX index_target_table1 ON target_table1(b);
--- SQL operation complete.
SQL>SET PARSERFLAGS 1;
--- SQL operation complete.
SQL>LOAD WITH INDEX TABLE ONLY INTO TABLE(INDEX_TABLE
index_target_table1) SELECT
b,a FROM source_table;
--- SQL operation complete.
SQL>SELECT * FROM target_table1;
A B
----------- ----
1 aaaa
2 bbbb
3 cccc
4 dd
5 ee
6 fff
--- 6 row(s) selected.
SQL>SELECT * FROM table(index_table index_target_table1);
B@ A
---- --------------------
aaaa 1
bbbb 2
cccc 3
dd 4
ee 5
fff 6
```

```
--- 6 row(s) selected.
```

- 下面的例子说明了如何根据不同的目标表填充索引表。
  - *target_table2* 定义了 A 列（主键列）和 B 列（索引列），在本例中，使用 *source_table* 中的列 B 和 A 列填充索引表。

```
SQL>CREATE INDEX index_target_table2 ON target_table2(b)
NO POPULATE;
--- SQL operation complete.
SQL>SET PARSERFLAGS 1;
--- SQL operation complete.
SQL>SHOWDDL TABLE(INDEX_TABLE index_target_table2);
CREATE TABLE TRAFODION.SEABASE.INDEX_TARGET_TABLE2
(
"B@" CHAR(4) CHARACTER SET ISO88591 COLLATE
DEFAULT NO DEFAULT NOT SERIALIZED
, A INT NO DEFAULT NOT NULL NOT DROPPABLE NOT
SERIALIZED
, PRIMARY KEY ("B@" ASC, A ASC)
)
ATTRIBUTES ALIGNED FORMAT
;
SQL>LOAD WITH INDEX TABLE ONLY INTO TABLE(INDEX_TABLE
index_target_table2) SELECT
b,a FROM source_table;
--- SQL operation complete.
SQL>SELECT * FROM target_table2;
--- 0 row(s) selected.
SQL>SELECT * FROM TABLE(INDEX_TABLE index_target_table2);
B@ A
---- --------------------
aaaa 1
bbbb 2
cccc 3
dd 4
ee 5
fff 6
--- 6 row(s) selected.
```

  - *target_table3*定义了A列（主键列和索引列）和B列，在本例中，使用source_table中的A列填充索引表。

```
SQL>CREATE INDEX index_target_table3 ON target_table3(a)
NO POPULATE;
--- SQL operation complete.
SQL>SET PARSERFLAGS 1;
--- SQL operation complete.
SQL>SHOWDDL TABLE(INDEX_TABLE index_target_table3);
CREATE TABLE TRAFODION.SEABASE.INDEX_TARGET_TABLE3
```

```
(
"A@" INT NO DEFAULT NOT NULL NOT DROPPABLE NOT
SERIALIZED
, PRIMARY KEY ("A@" ASC)
)
ATTRIBUTES ALIGNED FORMAT
;
SQL>LOAD WITH INDEX TABLE ONLY INTO TABLE(INDEX_TABLE
index_target_table3) SELECT
a FROM source_table;
--- SQL operation complete.
SQL>SELECT * FROM target_table3;
--- 0 row(s) selected.
SQL> SELECT * FROM TABLE(INDEX_TABLE
index_target_table3);
A@
-----------
1
2
3
4
5
6
--- 6 row(s) selected.
```

- *target_table4*定义了A列（索引列）和B列，在本例中，使用source_table中的A列和syskey填充索引表。

```
SQL> CREATE INDEX index_target_table4 ON target_table4(a)
NO POPULATE;
--- SQL operation complete.
SQL>SET PARSERFLAGS 1;
--- SQL operation complete.
SQL>SHOWDDL TABLE(INDEX_TABLE index_target_table4);
CREATE TABLE TRAFODION.SEABASE.INDEX_TARGET_TABLE4
(
"A@" INT NO DEFAULT NOT SERIALIZED
, SYSKEY LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
NOT SERIALIZED
, PRIMARY KEY ("A@" ASC, SYSKEY ASC)
)
ATTRIBUTES ALIGNED FORMAT
;
--- SQL operation complete.
SQL>LOAD WITH INDEX TABLE ONLY INTO TABLE(INDEX_TABLE
index_target_table4) SELECT
a,syskey FROM source_table;
--- SQL operation complete.
SQL>SELECT * FROM target_table4;
--- 0 row(s) selected.
```

```
SQL>SELECT * FROM TABLE(INDEX_TABLE index_target_table4);
A@ SYSKEY
----------- --------------------
1 4239726128363214004
2 4239726128363256924
3 4239726128363258834
4 4239726128363260240
5 4239726128363261628
6 4239726128363263088
--- 6 row(s) selected.
```

注意： 此时，如果要删除索引，例如上面创建的 index_target_table2、index_target_table3 或 index_target_table4，请在删除之前从其父表中填充索引，参见下面的例子。更多信息，请参阅 POPULATE INDEX 语句。

```
SQL> DROP INDEX index_target_table4;
*** ERROR[4254] Object
TRAFODION.SEABASE.INDEX_TARGET_TABLE4 has invalid state
and
cannot be accessed. Use cleanup command to drop it.
SQL> POPULATE INDEX index_target_table4 ON target_table4;
--- SQL operation complete.
SQL> DROP INDEX index_target_table4;
--- SQL operation complete.
```

3.3.3.5 REBUILD INDEXES 例子

假设我们有以下表格:

source_table:

```
SQL>select count(*) from source_table;
(EXPR)
--------------------
1000000
--- 1 row(s) selected.
```

target_table1 和 target_table2 都具有以下结构:

```
SQL>CREATE TABLE target_table1
(
ID INT NO DEFAULT NOT NULL NOT DROPPABLE NOT
SERIALIZED
, NUM INT DEFAULT NULL NOT SERIALIZED
, CARD_ID LARGEINT DEFAULT NULL NOT SERIALIZED
, PRICE DECIMAL(11, 3) DEFAULT NULL NOT SERIALIZED
, START_DATE DATE DEFAULT NULL NOT SERIALIZED
, START_TIME TIME(0) DEFAULT NULL NOT SERIALIZED
```

```
, END_TIME TIMESTAMP(6) DEFAULT NULL NOT SERIALIZED
, B_YEAR INTERVAL YEAR(10) DEFAULT NULL NOT
SERIALIZED
, B_YM INTERVAL YEAR(5) TO MONTH DEFAULT NULL NOT
SERIALIZED
, B_DS INTERVAL DAY(10) TO SECOND(3) DEFAULT NULL
NOT SERIALIZED
, PRIMARY KEY (ID ASC)
)
SALT USING 9 PARTITIONS
ATTRIBUTES ALIGNED FORMAT NAMESPACE 'TRAF_1500000'
HBASE_OPTIONS
(
MEMSTORE_FLUSH_SIZE = '1073741824'
)
;
```

- 该例子比较了在默认情况下关闭CQD TRAF_LOAD_ALLOW_RISKY_INDEX_MAINTENANCE时，使用带选项的LOAD语句和使用LOAD WITH REBUILD INDEXES的执行时间。

这两条语句花费了几乎相同的时间。

```
SQL>CREATE INDEX index_target_table1 ON target_table1(id);
--- SQL operation complete.
SQL>SET STATISTICS ON;
SQL>LOAD INTO target_table1 SELECT * FROM source_table
WHERE id < 301;
UTIL_OUTPUT
------------------------------------------------------------
------------------------
Task: LOAD Status: Started Object:
TRAFODION.SEABASE.TARGET_TABLE1
Task: CLEANUP Status: Started Time: 2018-01-18 13:33:52.310
Task: CLEANUP Status: Ended Time: 2018-01-18 13:33:52.328
Task: CLEANUP Status: Ended Elapsed Time: 00:00:00.019
Task: DISABLE INDEXE Status: Started Time: 2018-01-18
13:33:52.328
Task: DISABLE INDEXE Status: Ended Time: 2018-01-18
13:34:04.709
Task: DISABLE INDEXE Status: Ended Elapsed Time:
00:00:12.381
Task: LOADING DATA Status: Started Time: 2018-01-18
13:34:04.709
Rows Processed: 300
Error Rows: 0
Task: LOADING DATA Status: Ended Time: 2018-01-18
13:34:21.629
Task: LOADING DATA Status: Ended Elapsed Time: 00:00:16.919
```

```
Task: COMPLETION Status: Started Time: 2018-01-18
13:34:21.629
Rows Loaded: 300
Task: COMPLETION Status: Ended Time: 2018-01-18
13:34:22.436
Task: COMPLETION Status: Ended Elapsed Time: 00:00:00.808
Task: POPULATE INDEX Status: Started Time: 2018-01-18
13:34:22.436
Task: POPULATE INDEX Status: Ended Time: 2018-01-18
13:34:31.116
Task: POPULATE INDEX Status: Ended Elapsed Time:
00:00:08.680
--- SQL operation complete.
Start Time 2018/01/18 13:33:51.478782
End Time 2018/01/18 13:34:31.549491
Elapsed Time 00:00:40.070709
Compile Time 00:00:00.510024
Execution Time 00:00:39.559433
SQL>LOAD INTO target_table1 SELECT * FROM source_table
WHERE id > 300;
UTIL_OUTPUT
------------------------------------------------------------
------------------------
Task: LOAD Status: Started Object:
TRAFODION.SEABASE.TARGET_TABLE1
Task: CLEANUP Status: Started Time: 2018-01-18
13:35:01.804
Task: CLEANUP Status: Ended Time: 2018-01-18 13:35:01.823
Task: CLEANUP Status: Ended Elapsed Time: 00:00:00.018
Task: DISABLE INDEXE Status: Started Time: 2018-01-18
13:35:01.823
Task: DISABLE INDEXE Status: Ended Time: 2018-01-18
13:35:13.840
Task: DISABLE INDEXE Status: Ended Elapsed Time:
00:00:12.017
Task: LOADING DATA Status: Started Time: 2018-01-18
13:35:13.840
Rows Processed: 999700
Error Rows: 0
Task: LOADING DATA Status: Ended Time: 2018-01-18
13:35:19.720
Task: LOADING DATA Status: Ended Elapsed Time:
00:00:05.879
Task: COMPLETION Status: Started Time: 2018-01-18
13:35:19.720
Rows Loaded: 999700
Task: COMPLETION Status: Ended Time: 2018-01-18
13:35:22.436
Task: COMPLETION Status: Ended Elapsed Time: 00:00:02.717
Task: POPULATE INDEX Status: Started Time: 2018-01-18
```

```
13:35:22.436
Task: POPULATE INDEX Status: Ended Time: 2018-01-18
13:35:33.346
Task: POPULATE INDEX Status: Ended Elapsed Time:
00:00:10.910
--- SQL operation complete.
Start Time 2018/01/18 13:35:00.624490
End Time 2018/01/18 13:35:33.779394
Elapsed Time 00:00:33.154904
Compile Time 00:00:00.825703
Execution Time 00:00:32.321890
SQL>SET PARSERFLAGS 1;
--- SQL operation complete.
SQL>SELECT COUNT(*) FROM TABLE(INDEX_TABLE
index_target_table1);
(EXPR)
--------------------
1000000
--- 1 row(s) selected.


SQL>CREATE INDEX index_target_table2 ON
target_table2(id);
--- SQL operation complete.
SQL>SET STATISTICS ON;
SQL>LOAD WITH REBUILD INDEXES INTO target_table2 SELECT *
FROM source_table WHERE id
< 301;
UTIL_OUTPUT
------------------------------------------------------------
------------------------
Task: LOAD Status: Started Object:
TRAFODION.SEABASE.TARGET_TABLE2
Task: CLEANUP Status: Started Time: 2018-01-18
13:34:37.836
Task: CLEANUP Status: Ended Time: 2018-01-18 13:34:37.847
Task: CLEANUP Status: Ended Elapsed Time: 00:00:00.011
Task: DISABLE INDEXE Status: Started Time: 2018-01-18
13:34:37.847
Task: DISABLE INDEXE Status: Ended Time: 2018-01-18
13:34:45.445
Task: DISABLE INDEXE Status: Ended Elapsed Time:
00:00:07.598
Task: LOADING DATA Status: Started Time: 2018-01-18
13:34:45.445
Rows Processed: 300
Error Rows: 0
Task: LOADING DATA Status: Ended Time: 2018-01-18
13:35:03.576
Task: LOADING DATA Status: Ended Elapsed Time:
00:00:18.131
```

```
Task: COMPLETION Status: Started Time: 2018-01-18
13:35:03.577
Rows Loaded: 300
Task: COMPLETION Status: Ended Time: 2018-01-18
13:35:04.873
Task: COMPLETION Status: Ended Elapsed Time: 00:00:01.296
Task: POPULATE INDEX Status: Started Time: 2018-01-18
13:35:04.873
Task: POPULATE INDEX Status: Ended Time: 2018-01-18
13:35:12.461
Task: POPULATE INDEX Status: Ended Elapsed Time:
00:00:07.589
--- SQL operation complete.
Start Time 2018/01/18 13:34:37.053647
End Time 2018/01/18 13:35:12.893891
Elapsed Time 00:00:35.840244
Compile Time 00:00:00.435855
Execution Time 00:00:35.402620
SQL>LOAD WITH REBUILD INDEXES INTO target_table2 SELECT *
FROM source_table WHERE id
> 300;
UTIL_OUTPUT
------------------------------------------------------------
------------------------
Task: LOAD Status: Started Object:
TRAFODION.SEABASE.TARGET_TABLE2
Task: CLEANUP Status: Started Time: 2018-01-18
13:35:25.480
Task: CLEANUP Status: Ended Time: 2018-01-18 13:35:25.493
Task: CLEANUP Status: Ended Elapsed Time: 00:00:00.013
Task: DISABLE INDEXE Status: Started Time: 2018-01-18
13:35:25.493
Task: DISABLE INDEXE Status: Ended Time: 2018-01-18
13:35:38.844
Task: DISABLE INDEXE Status: Ended Elapsed Time:
00:00:12.591
Task: LOADING DATA Status: Started Time: 2018-01-18
13:35:38.845
Rows Processed: 999700
Error Rows: 0
Task: LOADING DATA Status: Ended Time: 2018-01-18
13:35:43.491
Task: LOADING DATA Status: Ended Elapsed Time:
00:00:05.407
Task: COMPLETION Status: Started Time: 2018-01-18
13:35:43.491
Rows Loaded: 999700

Task: COMPLETION Status: Ended Time: 2018-01-18
13:35:45.920
Task: COMPLETION Status: Ended Elapsed Time: 00:00:01.601
```

```
Task: POPULATE INDEX Status: Started Time: 2018-01-18
13:35:45.920
Task: POPULATE INDEX Status: Ended Time: 2018-01-18
13:35:56.322
Task: POPULATE INDEX Status: Ended Elapsed Time:
00:00:11.230
--- SQL operation complete.
Start Time 2018/01/18 13:35:24.693410
End Time 2018/01/18 13:35:56.754441
Elapsed Time 00:00:32.061031
Compile Time 00:00:00.449236
Execution Time 00:00:31.611112
SQL>SET PARSERFLAGS 1;
--- SQL operation complete.
SQL>SELECT COUNT(*) FROM TABLE(INDEX_TABLE
index_target_table2);
(EXPR)
--------------------
1000000
--- 1 row(s) selected.
```

- 该例子比较了在打开CQD TRAF_LOAD_ALLOW_RISKY_INDEX_MAINTENANCE时，使用不带选项的LOAD语句和使用LOAD WITH REBUILD INDEXES的执行时间。前者比后者花费的时间少。

```
SQL>CQD TRAF_LOAD_ALLOW_RISKY_INDEX_MAINTENANCE 'ON';
--- SQL operation complete.
SQL>CREATE INDEX index_target_table1 ON target_table1(id);
--- SQL operation complete.
SQL>SET STATISTICS ON;
SQL>LOAD INTO target_table1 SELECT * FROM source_table
WHERE id < 301;
UTIL_OUTPUT
------------------------------------------------------------
-------------------------
Task: LOAD Status: Started Object:
TRAFODION.SEABASE.TARGET_TABLE1
Task: CLEANUP Status: Started Time: 2018-01-18 13:46:01.730
Task: CLEANUP Status: Ended Time: 2018-01-18 13:46:01.756
Task: CLEANUP Status: Ended Elapsed Time: 00:00:00.027
Task: LOADING DATA Status: Started Time: 2018-01-18
13:46:01.756
Rows Processed: 300
Error Rows: 0
Task: LOADING DATA Status: Ended Time: 2018-01-18
```

```
13:46:22.415
Task: LOADING DATA Status: Ended Elapsed Time: 00:00:20.659
Task:   COMPLETION   Status:   Started   Time:   2018-01-18
13:46:22.415
Rows Loaded: 300
Task:   COMPLETION   Status:   Ended   Time:   2018-01-18
13:46:26.353
Task: COMPLETION Status: Ended Elapsed Time: 00:00:03.938
--- SQL operation complete.

Start Time 2018/01/18 13:46:00.954518
End Time 2018/01/18 13:46:26.795757
Elapsed Time 00:00:25.841239
Compile Time 00:00:00.455681
Execution Time 00:00:25.384158
SQL>LOAD  INTO  target_table1  SELECT  *  FROM  source_table
WHERE id > 300;
UTIL_OUTPUT
------------------------------------------------------------
------------------------
Task:       LOAD       Status:       Started       Object:
TRAFODION.SEABASE.TARGET_TABLE1
Task: CLEANUP Status: Started Time: 2018-01-18 13:46:57.811
Task: CLEANUP Status: Ended Time: 2018-01-18 13:46:57.870
Task: CLEANUP Status: Ended Elapsed Time: 00:00:00.058
Task:  LOADING  DATA  Status:  Started  Time:  2018-01-18
13:46:57.870
Rows Processed: 999700
Error Rows: 0
Task:  LOADING  DATA  Status:  Ended  Time:  2018-01-18
13:47:12.411
Task: LOADING DATA Status: Ended Elapsed Time: 00:00:14.541
Task:   COMPLETION   Status:   Started   Time:   2018-01-18
13:47:12.411
Rows Loaded: 999700
Task:   COMPLETION   Status:   Ended   Time:   2018-01-18
13:47:16.292
Task: COMPLETION Status: Ended Elapsed Time: 00:00:03.881
--- SQL operation complete.
Start Time 2018/01/18 13:46:57.400422
End Time 2018/01/18 13:47:16.738970
Elapsed Time 00:00:19.338548
Compile Time 00:00:00.010545
Execution Time 00:00:19.321781
SQL>SET PARSERFLAGS 1;
--- SQL operation complete.
SQL>SELECT      COUNT(*)      FROM      TABLE(INDEX_TABLE
index_target_table1);
(EXPR)
--------------------
1000000
```

```
--- 1 row(s) selected.


SQL>CREATE INDEX index_target_table2 ON target_table2(id);
--- SQL operation complete.
SQL>SET STATISTICS ON;
SQL>LOAD WITH REBUILD INDEXES INTO target_table2 SELECT *
FROM source_table WHERE id
< 301;
UTIL_OUTPUT
------------------------------------------------------------
------------------------
Task:       LOAD       Status:       Started       Object:
TRAFODION.SEABASE.TARGET_TABLE2
Task: CLEANUP Status: Started Time: 2018-01-18 13:46:28.303
Task: CLEANUP Status: Ended Time: 2018-01-18 13:46:28.796
Task: CLEANUP Status: Ended Elapsed Time: 00:00:00.049

Task: DISABLE INDEXE Status: Started Time: 2018-01-18
13:46:28.796
Task: DISABLE INDEXE Status: Ended Time: 2018-01-18
13:46:38.479
Task: DISABLE INDEXE Status: Ended Elapsed Time:
00:00:10.400
Task: LOADING DATA Status: Started Time: 2018-01-18
13:46:38.479
Rows Processed: 300
Error Rows: 0
Task: LOADING DATA Status: Ended Time: 2018-01-18
13:46:58.143
Task: LOADING DATA Status: Ended Elapsed Time: 00:00:19.665
Task: COMPLETION Status: Started Time: 2018-01-18
13:46:58.144
Rows Loaded: 300
Task: COMPLETION Status: Ended Time: 2018-01-18
13:47:00.186
Task: COMPLETION Status: Ended Elapsed Time: 00:00:02.043
Task: POPULATE INDEX Status: Started Time: 2018-01-18
13:47:00.187
Task: POPULATE INDEX Status: Ended Time: 2018-01-18
13:47:09.966
Task: POPULATE INDEX Status: Ended Elapsed Time:
00:00:08.910
--- SQL operation complete.
Start Time 2018/01/18 13:46:27.063642
End Time 2018/01/18 13:47:09.529257
Elapsed Time 00:00:42.465615
Compile Time 00:00:00.532541
Execution Time 00:00:41.928812
SQL>LOAD WITH REBUILD INDEXES INTO target_table2 SELECT *
FROM source_table WHERE id
```

```
> 300;
UTIL_OUTPUT
------------------------------------------------------------
------------------------
Task:       LOAD       Status:       Started       Object:
TRAFODION.SEABASE.TARGET_TABLE2
Task: CLEANUP Status: Started Time: 2018-01-18 13:47:18.187
Task: CLEANUP Status: Ended Time: 2018-01-18 13:47:18.198
Task: CLEANUP Status: Ended Elapsed Time: 00:00:00.011
Task: DISABLE INDEXE Status: Started Time: 2018-01-18
13:47:18.198
Task: DISABLE INDEXE Status: Ended Time: 2018-01-18
13:47:30.670
Task: DISABLE INDEXE Status: Ended Elapsed Time:
00:00:12.472
Task: LOADING DATA Status: Started Time: 2018-01-18
13:47:30.670
Rows Processed: 999700
Error Rows: 0
Task: LOADING DATA Status: Ended Time: 2018-01-18
13:47:39.311
Task: LOADING DATA Status: Ended Elapsed Time: 00:00:08.641
Task: COMPLETION Status: Started Time: 2018-01-18
13:47:39.311
Rows Loaded: 999700
Task: COMPLETION Status: Ended Time: 2018-01-18
13:47:40.497
Task: COMPLETION Status: Ended Elapsed Time: 00:00:01.186
Task: POPULATE INDEX Status: Started Time: 2018-01-18
13:47:40.497
Task: POPULATE INDEX Status: Ended Time: 2018-01-18
13:47:52.367
Task: POPULATE INDEX Status: Ended Elapsed Time:
00:00:11.539
--- SQL operation complete.
Start Time 2018/01/18 13:47:17.447093
End Time 2018/01/18 13:47:52.469190
Elapsed Time 00:00:35.022097
Compile Time 00:00:00.412718
Execution Time 00:00:34.608571



SQL>SET PARSERFLAGES 1;
--- SQL operation complete.
SQL>SELECT COUNT(*) FROM TABLE(INDEX_TABLE
index_target_table2);
(EXPR)
--------------------
1000000
--- 1 row(s) selected.
```

3.3.3.6 从 Hive 表装载数据例子

- 对于驻留在/hive/tpcds/customer_demographics 里的客户统计数据，应使用以下 Hive SQL 创建一个外部 Hive 表：

```
create external table customer_demographics
(
cd_demo_sk int
, cd_gender string
, cd_marital_status string
, cd_education_status string
, cd_purchase_estimate int
, cd_credit_rating string
, cd_dep_count int
, cd_dep_employed_count int
, cd_dep_college_count int
)
row format delimited fields terminated by '|' location
'/hive/tpcds/customer_demographics';
```

- 使用这条 DDL 定义您希望数据载入进的 EsgynDB 表：

```
create table customer_demographics_salt
(
cd_demo_sk int not null
, cd_gender char(1)
, cd_marital_status char(1)
, cd_education_status char(20)
, cd_purchase_estimate int
, cd_credit_rating char(10)
, cd_dep_count int
, cd_dep_employed_count int
, cd_dep_college_count int
, primary key (cd_demo_sk)
)
salt using 4 partitions on (cd_demo_sk);
```

- 该例子显示了LOAD语句如何从Hive hive.hive.customer_demographics里装载customer_demographics_salt表：

```
>>load into customer_demographics_salt
+>select * from hive.hive.customer_demographics where
cd_demo_sk <= 5000;
Task: LOAD Status: Started Object:
TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Task: DISABLE INDEX Status: Started Object:
TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Task: DISABLE INDEX Status: Ended Object:
```

```
TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Task: PREPARATION Status: Started Object:
TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Rows Processed: 5000
Task: PREPARATION Status: Ended ET: 00:00:03.199
Task: COMPLETION Status: Started Object:
TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Task: COMPLETION Status: Ended ET: 00:00:00.331
Task: POPULATE INDEX Status: Started Object:
TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Task: POPULATE INDEX Status: Ended ET: 00:00:05.262
```

- 对于驻留在/hive/tpcds/customer_demographics里的客户统计数据，应使用以下Hive SQL创建一个外部Hive表：

```
create external table customer_demographics (
   cd_demo_sk int,
   cd_gender string,
   cd_marital_status string,
   cd_education_status string,
   cd_purchase_estimate int,
   cd_credit_rating string,
   cd_dep_count int,
   cd_dep_employed_count int,
   cd_dep_college_count int
)
row format delimited fields terminated by '|'
location '/hive/tpcds/customer_demographics'
```

- 使用这条DDL定义您希望将数据载入进的EsgynDB表：

```
create table customer_demographics_salt (
cd_demo_sk   int   not   null,   cd_gender   char(1),
cd_marital_status char(1), cd_education_status char(20),
cd_purchase_estimate  int,  cd_credit_rating  char(10),
cd_dep_count int, cd_dep_employed_count
int, cd_dep_college_count int, primary key (cd_demo_sk)
)
salt using 4 partitions on (cd_demo_sk);
```

- 该例子显示了 LOAD语句如何从Hive (hive.hive.customer_demographics) 里装载customer_demographics_salt表：

```
>>load into customer_demographics_salt
+>select * from hive.hive.customer_demographics where
cd_demo_sk <= 5000;
Task:LOAD Status:Started
Object:TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Task:DISABLE INDEX Status:Started Object:
TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Task:DISABLE INDEX Status:Ended
Object:TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Task:PREPARATION Status:Started
Object:TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Rows Processed:5000
Task:PREPARATION Status:Ended ET:00:00:03.199
Task:COMPLETION Status:Started
Object:TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Task:COMPLETION Status:Ended ET:00:00:00.331
Task:POPULATE INDEX Status:Started Object:
TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS_SALT
Task:POPULATE INDEX Status:Ended ET:00:00:05.262
```

3.4 POPULATE INDEX 语句

POPULATE INDEX 语句将母表数据快速插入到索引中。您可以在诸如 TrafCI 的基于客户端的工具里使用 POPULATE INDEX 语句。

```
POPULATE INDEX index ON table [index-option]

index-option is:
     ONLINE | OFFLINE
```

3.4.1 POPULATE INDEX 语法

- *index*

  这是一种 SQL 标识符，指定索引的简易名称。不能用该索引所属 schema 的名称来限定它。 一个 schema 内的索引都有各自的命名空间，因此索引名可能与表名或约束名相同。 然而，一个 schema 内的索引名必须各不相同。

- *table*

  这是需要填充索引的表名。更多信息，请参阅数据库对象名称。

- ONLINE

  指定填充操作应该在线完成，即填充时，ONLINE 允许基表执行 DML 读写访问。 此外，索引填充期间，ONLINE 还可以读取审计跟踪数据，以便重现基表的更新情况。

  如果已经生成了大量的审计数据，并且您执行了很多 CREATE INDEX 操作，我们建议您不要使用 ONLINE 操作，因为它将为读取审计跟踪数据引入更多竞争。默认值为 ONLINE。

- OFFLINE

  指定填充操作应该脱机完成。OFFLINE 允许对基表进行 DML 读取访问。

  此时基表不可用于写入操作。 OFFLINE 必须显式地指定。允许 SELECT 操作。

3.4.2 POPULATE INDEX 注意事项

POPULATE INDEX 的执行步骤如下:

- POPULATE INDEX 操作可以运行在很多事务中。
- 实际装载操作运行于一个事务之外。

  如果操作失败，则将更快地进行回滚，因为它无需处理大量审计数据。

  另外，如果操作失败，索引将仍为空、未经审计、并且没有连接到基表（脱机)。
- 脱机执行 POPULATE INDEX 时，可以访问基表进行 DML 读操作。 联机执行 POPULATE INDEX 期间（不含最后提交阶段)，可以访问基表进行 DML 读/写操作。
- 如果 POPULATE INDEX 操作意外失败，您可能需要再次删除索引、重新创建一个索引并填充。
- 通过读/写访问，Online POPULATE INDEX 可以读取审计跟踪数据，以便重现更新情况。如果您计划并行创建很多索引，或如果您有高等级的审计跟踪活动，则应考虑使用 OFFLINE 选项。

  如果无法访问来源基表或目标索引、或如果数据装载操作因为某些资源问题或文件系统问题失败，则可能出现错误。

3.4.2.1 必需的权限

执行一个 POPULATE INDEX 语句，以下条件之一必须成立:

- 您是 DB__ROOT（数据库根用户)。
- 您是该表的所有者。
- 您拥有关联表的 SELECT 和 INSERT（或 ALL）权限。

3.4.3 POPULATE INDEX 例子

- 该例子从指定的表里装载指定的索引:

  ```
  POPULATE INDEX myindex ON myschema.mytable;
  ```

- 该例子从使用默认 schema 的指定表里装载指定的索引:

  ```
  POPULATE INDEX index2 ON table2;
  ```

3.5 PURGEDATA 语句

PURGEDATA 语句快速删除一个表和其相关索引的数据。您可以在诸如 TrafCI 之类的基于客户端的工具里使用 PURGEDATA 语句。

PURGEDATA 语句在行为上跟 TRUNCATE 语句是一样的。

```
PURGEDATA object
```

3.5.1 PURGEDATA 语法

object

这是需要清除数据的表名。更多信息，请参阅数据库对象名称。

3.5.2 PURGEDATA 注意事项

- 该对象可以是一个表名。
- 如果该表无法访问、或如果资源问题或文件系统问题导致删除操作失败，则将返回错误。
- 临时表不支持运行 PURGEDATA。

3.5.2.1 必需的权限

执行一个 PURGEDATA 操作，以下条件之一必须成立：

- 您是 DB__ROOT（数据库根用户）。
- 您是该表的所有者。
- 您拥有关联表的 SELECT 和 DELETE（或 ALL）权限。

3.5.2.2 可用性

PURGEDATA 将一个表标记为 OFFLINE，并在处理期间设置损坏位。

如果 PURGEDATA 操作在完成之前失败，则该表和它的从属索引都将不可用，并且您必须再次运行 PURGEDATA，以便完成操作、并删除数据。

在这种情况下将返回错误 8551、以及附带的文件系统错误 59 或错误 1071。

3.5.3 PURGEDATA 例子

- 该例子清除指定表里的数据。如果一个表拥有多个索引，则它们的数据也将被清除。

  ```
  PURGEDATA myschema.mytable;
  ```

3.6 TRUNCATE 语句

TRUNCATE 语句快速删除一个表和其相关索引的数据。您可以在诸如 TrafCI 之类的基于客户端的工具里使用 TRUNCATE 语句。

TRUNCATE 语句在行为上跟 PURGEDATA 语句是一样的。

TRUNCATE [TABLE] *table-name*;

3.6.1 TRUNCATE 语法

table-name
这是需要清除数据的表名。更多信息，请参阅数据库对象名称。

3.6.2 TRUNCATE 注意事项

- 该对象可以是一个表名。
- 如果该表无法访问、或如果资源问题或文件系统问题导致删除操作失败，则将返回错误。
- 临时表不支持运行 TRUNCATE。
- 支持在事务中使用 TRUNCATE 语句。该功能由 CQD ALLOW_TRUNCATE_IN_USER_XN 控制，值为 ON 时 EsgynDB 支持此功能。

3.6.2.1 必需的权限

执行一个 TRUNCATE 操作，以下条件之一必须成立：

- 您是 DB__ROOT（数据库根用户）。
- 您是该表的所有者。
- 您拥有关联表的 SELECT 和 DELETE（或 ALL）权限。

3.6.2.2 可用性

TRUNCATE 将一个表标记为 OFFLINE，并在处理期间设置损坏位。

如果 TRUNCATE 操作在完成之前失败，则该表和它的从属索引都将不可用，并且您必须再次运行 TRUNCATE，以便完成操作、并删除数据。

在这种情况下将返回错误 8551、以及附带的文件系统错误 59 或错误 1071。

3.6.3 TRUNCATE 例子

- 该例子清除指定表里的数据。如果一个表拥有多个索引，则它们的数据也将被清除。

  TRUNCATE `myschema.mytable;`

- 在事务中使用 TRUNCATE 语句。

```
SQL>cqd ALLOW_TRUNCATE_IN_USER_XN 'on';

---SQL operation complete.

SQL>truncate table KTLB_GYSJIA;

--- SQL operation complete.

SQL>rollback;

--- SQL operation complete.

SQL>select count(*) from KTLB_GYSJIA;

(EXPR)
--------------------
                   0

--- 1 row(s) selected.
```

3.7 UNLOAD 语句

UNLOAD 语句将 EsgynDB 表里的数据卸载到您指定的 HDFS 位置。

抽取的数据既可以是压缩数据，也可以是未压缩数据，您可以自行选择。

UNLOAD 是 EsgynDB SQL 扩展语句。

```
UNLOAD [WITH option[ option]...]INTO ' target-location'
     SELECT ...FROM source-table ...

option is:
     DELIMITER { ' delimiter-string' | delimiter-ascii-
     value }
     | RECORD_SEPARATOR {' separator-literal' | separator-
     ascii-value }
     | NULL_STRING ' string-literal'
     | PURGEDATA FROM TARGET
     | COMPRESSION GZIP
     | MERGE FILE merged_file-path [OVERWRITE]
     | NO OUTPUT
     | { NEW | EXISTING } SNAPSHOT HAVING SUFFIX ' string'
```

3.7.1 UNLOAD 语法

- `'target-location'`

  这是抽取数据将被写入的 HDFS 目标文件夹的完整路径名称。将该文件夹的名称用单引号括起来。

  指定文件夹名称为完整路径名称、而非相对路径。您必须对 HDFS 目标文件夹拥有写入权限。

  如果您并行运行 UNLOAD 语句，则 target-location（目标位置）中将产生多个文件。已创建文件的数量将等于 ESP 的数量。

- `SELECT ...FROM source-table ...`

  这是一条简单查询或是一条包含 GROUP BY、JOIN 或 UNION 子句的复杂查询。source-table 是带有源数据的 EsgynDB 表的名称。更多信息，请参阅数据库对象名称。

- `[WITH option[ option]...]`

  这是为卸载操作指定的一组选项。如果您多次指定一个选项，则 EsgynDB 将返回一个带有 SQLCODE:-4489 的错误。

  您可以指定其中的一个或多个选项：

- `DELIMITER { 'delimiter-string' | delimiter-ascii-value }`
  指定分隔符是一个分隔字符串或一个 ASCII 值。
  如果您没有指定该选项，EsgynDB 将使用字符“|”作为分隔符。
  delimiter-string（分隔字符串）可以是任何 ASCII 或 Unicode 字符串。您也可以指定分隔符是一个 ASCII 值。有效值的范围从 1 到 255。指定一个十进制数值；目前暂不支持十六进制或八进制表示法。
  如果您正在使用一个 ASCII 值，则分隔符宽度仅限一个字符。如果指定一个 ASCII 值作为分隔符，则不得使用引号。

- `RECORD_SEPARATOR { 'separator-literal' | separator-ascii-value }`
  该字符将被用于分隔输出文件里的连续记录或行。您可以指定一个字面值或 ASCII 值作为分隔符。 默认值为换行符。
  separator-literal 可以为任何 ASCII 或 Unicode 字符。您也可以指定分隔符是一个 ASCII 值。有效值的范围从 1 到 255。指定一个十进制数值；目前不支持十六进制或八进制表示法。
  如果您正在使用一个 ASCII 值，则分隔符宽度仅限一个字符。如果指定一个 ASCII 值作为分隔符，则不得使用引号。

- `NULL_STRING 'string-literal'`
  指定一个用于表示空值的字符串。默认值为空字符串。

- `PURGEDATA FROM TARGET`
  卸载前删除目标 HDFS 文件夹里的文件。

- `COMPRESSION GZIP`
  在抽取节点里使用 gzip 压缩，将数据以这种压缩格式写入磁盘。
  GZIP 是目前唯一支持的压缩类型。如果您未指定该选项，则抽取的数据将被解压。

- `MERGE FILE` *`merged_file-path`* `[OVERWRITE]`
  将卸载文件合并到指定的 merged-file-path（合并文件路径）的单一文件里。
  如果您指定了压缩率，则卸载数据和合并文件都将具有压缩格式。

如果您指定了可选的 OVERWRITE 关键字，则现有文件将被覆盖。否则，EsgynDB 会抛出一个错误。

- `NO OUTPUT`
  防止 UNLOAD 语句显示状态消息。默认情况下，UNLOAD 语句可以打印状态消息，将列出 Bulk Loader 正在执行的步骤。

- `{ NEW | EXISTING } SNAPSHOT HAVING SUFFIX 'string'`
  在卸载期间发起 HBase 快照扫描。快照扫描期间，Bulk Unloader 将从查询解释计划中获得一份 EsgynDB 表的清单，然后将为这些表创建和验证快照。指定一个后缀字符串'string'，它将被追加到每个表名后面。

3.7.2 UNLOAD 注意事项

- 您必须对 HDFS 目标文件夹拥有写入权限。
- 如果您多次指定一个 WITH 选项，则 EsgynDB 将返回一个带有 SQLCODE:-4489 的错误。

3.7.2.1 必需的权限

发出一条 UNLOAD 语句，以下条件之一必须成立：

- 您是 DB__ROOT（数据库根用户）。
- 您是目标表的所有者。
- 您拥有目标表的 SELECT 权限。
- 您拥有 SQL_OPERATIONS 组件的 MANAGE_LOAD 组件权限。

3.7.3 UNLOAD 例子

该例子显示 UNLOAD 语句如何将一个 EsgynDB 表 TRAFODION.HBASE.CUSTOMER_DEMOGRAPHICS 里的数据抽取到一个 HDFS 文件夹 `/bulkload/ customer_demographics`:中。

```
>>UNLOAD
+>WITH PURGEDATA FROM TARGET
+>MERGE FILE 'merged_customer_demogs.gz' OVERWRITE
+>COMPRESSION GZIP
+>INTO '/bulkload/customer_demographics'
+>select * from trafodion.hbase.customer_demographics
```

```
+><<+ cardinality 10e10 >>; Task:UNLOAD Status:Started
Task:EMPTY TARGET Status:Started
Task:EMPTY TARGET Status:Ended ET:00:00:00.014

Task:EXTRACT Status:Started Rows Processed:200000
Task:EXTRACT Status:Ended ET:00:00:04.743 Task:MERGE FILES
Status:
Started
Task:MERGE FILES Status:Ended ET:00:00:00.063

--- 200000 row(s) unloaded.
```

3.8 UPDATE STATISTICS 语句

该语句更新表里的一组或多组列的直方图统计信息。这些统计信息可被用来设计优化的访问计划。

UPDATE STATISTICS 是 EsgynDB SQL 扩展语句。

```
UPDATE STATISTICS FOR TABLE table [CLEAR | on-clause]

     [LIKE sourceTable-name]

on-clause is:
     ON column-group-list CLEAR
     | ON column-group-list [histogram-option]...

column-group-list is:
     column-list [,column-list]...
     | EVERY COLUMN [,column-list]...
     | EVERY KEY [,column-list]...
     | EXISTING COLUMN[S] [,column-list]...
     | NECESSARY COLUMN[S] [,column-list]...

column-list for a single-column group is:
     column-name
     | (column-name)
     | column-name TO column-name
     | (column-name) TO (column-name)
     | column-name TO (column-name)
     | (column-name) TO column-name

column-list for a multicolumn group is:
     (column-name, column-name [,column-name]...)

histogram-option is:
     GENERATE n INTERVALS
     | SAMPLE [sample-option]

sample-option is:
     [r ROWS]
     | RANDOM percent PERCENT
     | PERIODIC size ROWS EVERY period ROWS
```

3.8.1 UPDATE STATISTICS 语法

- *table*

  命名一个将要更新统计信息的表。为了引用一个表，请使用 ANSI 逻辑名称. 更多信息，请参阅数据库对象名称。

- CLEAR

  删除该表某些或所有直方图。当新的应用程序不再使用某些直方图统计信息时，可以使用该选项。

  如果您没有指定 column-group-list，则应删除表中所有直方图；如果您指定了 column- group-list, 则仅删除组列表里的列。

- ON column-group-list

  指定为一个或多个列组生成直方图统计信息，并可以清除该信息。您必须使用 ON 子句生成直方图表里存储的统计信息。

  - column-list

    指定如何定义一个 column-group-list。 该“列列表”可以同时代表单列组和多列组。

    Δ 单列组:

    column-name | (column-name) | column-name TO column-name | (column-name) TO (column-name)

    说明如何指定单独的列或一组单独的列。

    为了给单列生成统计信息，请将每列都列出。您可以列举每个单列的名称，有无括号均可。

    Δ 多列组:

    (column-name, column-name [,column-name]...)

    指定一个多列组。

    为了生成多列统计信息，请对括号中的列集进行分组。您不得在同一个列组里多次指定一个列名。

  可以为每个唯一列组生成一个直方图。

  重复组意味着同一个列组的任何排列都将被忽略不计，数据处理将继续

进行。

当再次为同一个用户表运行 UPDATE STATISTICS 语句时，该表的新数据将替换先前生成、并存储在该表直方图表里的数据。

ON 子句中没有指定的列组直方图将在直方图表里保持不变。更多信息，请参阅为列生成和清除统计信息。

- EVERY COLUMN

  EVERY COLUMN 关键字表明将为一个表的每个单独的列、以及任何构成主键和索引的多列生成直方图统计信息。

  例如，一个表已经定义了列 A、B、C 和 D，其中 A 列、B 列和 C 列将构成主键。

  在这种情况下，ON EVERY COLUMN 选项可以为 A 列、B 列、C 列和 D 列生成单列直方图，此外它还可生成两个多列直方图(A, B, C)和(A, B)。

  EVERY COLUMN 选项与 EVERY KEY 选项功能一致，并带有单列的附加统计信息。

- EVERY KEY

  EVERY KEY 关键字表明将为构成主键和索引的列生成直方图统计信息。

  例如，一个表已经定义了 A 列、B 列、C 列和 D 列，

  如果主键由 A 列和 B 列构成，则将为(A, B)、A 和 B 生成统计信息。如果主键由 A 列、B 列和 C 列构成，则将为 (A,B,C)、(A,B)、A、B、C 生成统计信息。

  如果主键由 A 列、 B 列、C 列和 D 列构成，则将为(A, B, C, D)、(A, B, C) 、(A, B)和 A、B、C、D 生成统计信息。

- EXISTING COLUMN[S]

  EXISTING COLUMN 关键字表明该表目前所有的直方图都要更新。必须事先捕获统计信息，以便建立现有的列。

- NECESSARY COLUMN[S]

  NECESSARY COLUMN[S]关键字为直方图生成统计信息，该直方图是优化器请求的并不存在的。

  必须针对 NECESSARY COLUMN[S]启用更新统计自动化，以便生成统计信息。更多关于启用自动化的信息，请参阅更新统计自动化。

- histogram-option
  - GENERATE n INTERVALS

    UPDATE STATISTICS 语句的 GENERATE n INTERVALS 选项接受 1 和 10,000 之间的数值。提高每个直方图的时间间隔数量可能对编译时间造成负面影响。

    可以为带有小型数值集和较大的数值频率方差的列增加时间间隔的数量。例如， SALES 表里的列‘CITY’，存储了已售项目所在的城市代码，销售数据里的城市数量为 1538。

    将时间间隔数量设置为一个大于或等于城市数量的数字（即将时间间隔数量设置为 1600）可以保证生成的直方图能够捕获每个城市的行数)。

    如果指定的数值 n 超过了列中唯一值的数量，则系统将只能生成与唯一值数量相等的时间间隔。

  - SAMPLE [sample-option]

    该子句里指定的抽样用于收集表数据的子集。

    UPDATE STATISTICS 语句存储样本结果并生成直方图。

    如果您指定了不含附加选项的 SAMPLE 子句，则结果将取决于表里的行数。

    如果该表的行数不超过 1 万，则整个表都将被读取（无抽样)。

    如果该表的行数介于 1 万和 100 万之间，则将从该表里随机抽样 1 万行。

    如果该表的行数超过了 100 万，则可使用随机行样本读取该表行数的 1%，最多抽样 100 万行。

提示：默认抽样该表行数的 1%（最多抽样 100 万行）可以为优化器提供良好的统计信息，从而生成良好的（执行）计划。

如果没有指定 SAMPLE 子句，则该表的行数将低于指定值；另外，如果样本大小超过了系统限制，则 EsgynDB SQL 将从表里读取所有行。更多信息，请参阅 SAMPLE 子句。

  - sample-option
    - r rows

      行样本被用于读取表中的 r 行。数值 r 必须是一个大于零的整数（r >

0)。

Δ `RANDOM percent PERCENT`
指导 EsgynDB SQL 随机地选择表里的行。数值百分比必须是一个介于 0 和 100 之间的数值（0 < percent <= 100）。
此外，只有小数点右侧的前四位数才是有效位。例如：数值 0.00001 将被视为 0.0000，数值 1.23456 将被视为 1.2345。

Δ `PERIODIC size ROWS EVERY period ROW`
指导 EsgynDB SQL 从每个周期的行里选择前面 size 数量的行。
Size 值必须是一个大于零且小于或等于 period 值的整数（0 < size <= period）。通过该 period 可以指定周期行数的大小。Period 值必须是一个大于零的整数。

- `INCREMENTAL WHERE predicate`

指示 EsgynDB SQL 增量地更新统计信息。也就是说，EsgynDB SQL 将使用之前创建的持久性样本表，而不是获取整个表的新样本。 EsgynDB SQL 将更新持久性样本，用满足谓词的原始表中的新行样本来替换满足谓词的任何行。使用的采样率是创建持久性样本表时指定的百分比。然后从这个更新后的样本生成统计信息。更多信息，请参阅增量更新统计信息。

- `CREATE SAMPLE RANDOM percent PERCENT`

创建与此表关联的持久性样本表。该样本是使用占行的百分比的随即样本创建的。该表可用于后续的增量统计信息更新。

- `REMOVE SAMPLE`

删除与此表关联的持久性样本表。

- `LIKE sourceTable-name`

将 `sourceTable-name` 的对应 SB_HISTOGRAMS 和 SB_HISTOGRAM_INTERVALS 中的直方图数据拷贝给 *table*。如执行该操作

需满足以下两个条件:

- *table* 和 sourceTable-name 的列数要完全一致。
- 两个表的每一列的类型完全一致。

另外，"_DIVISION_1_"列，"SYSKEY"和"SALT"列的直方图信息也会拷贝给 *table*。

如果 sourceTable-name 没有统计信息，会报错：ERROR[9255] UPDATE STATISTICS: No histogram for TRAFODION.SCH.T2.

如果 sourceTable-name 已经有统计信息，会报错：ERROR[9254] UPDATE STATISTICS: TRAFODION.SCH.TT2 already has histograms.Please use UPDATE STATISTICS...CLEAR to clear the previously generated histograms first.

对于 SALT 列， 需要满足以下条件才会拷贝 SALT 列相关的统计信息:

1. 分区数一致。
2. SALT 对应的列的表达式一致。
3. SALT 列所在原表的位置一致。
4. 初始 region 数一致。

如果两张表仅仅只是 SALT 列不相同，还是可以拷贝除 SALT 列以外的统计信息。

3.8.2 UPDATE STATISTICS 注意事项

3.8.2.1 使用统计信息

使用 UPDATE STATISTICS 语句收集和保存列的统计信息。

SQL 编译器使用直方图统计信息确定谓词、索引和表的选择率。由于选择率直接影响访问计划的成本，常规的统计信息收集可以提高 EsgynDB SQL 选择有效访问计划的可能性。

在一个表上运行 UPDATE STATISTICS 语句时，该表处于活动状态，并可用于查询访问。通过显著更改一个用户表的数据或定义而改变这个表时，应对其重新执行 UPDATE STATISTICS 语句。

3.8.2.2 直方图统计信息

编译器使用直方图统计为一条给定的 SQL 查询产生最佳计划。如果这些直方图不可用，编译器将做出默认假设、由此产生的计划可能也表现不佳。对于这个问题，反映表中最新数据的直方图是最佳解决方案。

编译器不需要一个表所有列的直方图统计信息。例如，如果一列仅出现在选择列表里，则其直方图统计信息将无关紧要。

如果列出现在以下情况中，那么直方图统计信息将非常有价值：

- 一个谓词中
- 一个 GROUP BY 列中
- 一个 ORDER BY 子句中
- 一个 HAVING 子句中
- 或类似子句中

除了单列直方图统计信息以外，编译器还需要多列直方图统计信息，例如，当一个查询里出现 group by column-5, column-3, column-19 时，编译器就需要（column-5, column-3, column-19）组合的直方图统计信息。

3.8.2.3 必需的权限

执行一个 UPDATE STATISTICS 操作，以下条件之一必须成立：

- 您是 DB__ROOT（数据库根用户）。
- 您是目标表的所有者。
- 您拥有 SQL_OPERATIONS 组件的 MANAGE_STATISTICS 组件权限。

3.8.2.4 锁定

UPDATE STATISTICS 语句在操作期间暂时锁定用户表的定义（而非用户表自身）。UPDATE STATISTICS 语句为用户表使用 READ UNCOMMITTED（未提交读）隔离级别。

3.8.2.5 事务

执行 UPDATE STATISTICS 语句前不得启动一个事务。必要时，UPDATE STATISTICS 语句可以同时运行多个自身的事务。

在预先启动的事务中运行 UPDATE STATISTICS 语句可能导致事务自动中止时间超时。

3.8.2.6 为列生成和清除统计信息

为了给特定列生成统计信息，应对表里的每个列或一连串列里的首列和末列进行命名。例如，假设一个表有连续列 CITY、STATE、ZIP，该列表为您可以指

定的选项的例子:

单列组	括号里的单列组	多列组
ON CITY, STATE, ZIP	ON (CITY),(STATE),(ZIP)	ON (CITY, STATE)或 ON (CITY,STATE,ZIP)
ON CITY TO ZIP	ON (CITY) TO (ZIP)	
ON ZIP TO CITY	ON (ZIP) TO (CITY)	
ON CITY, STATE TO ZIP	ON (CITY), (STATE) TO (ZIP)	
ON CITY TO STATE, ZIP	ON (CITY) TO (STATE), (ZIP)	

当表中有很多列、并且您希望得到一个列的子集的直方图时，那么TO规范将有用。请勿混淆(CITY) TO (ZIP)和(CITY, STATE, ZIP)，后者引用了一个多列直方图。您可以按照任何指定的列组合清除统计信息，而不必通过创建统计信息时使用的 column-group-list 执行该操作。然而， 这些统计信息将被一直保留，直到清除它们为止。

3.8.2.7 “列列表”和访问计划

针对表数据访问计划里最常用的列生成统计信息，即定义在该表上的主键和索引、任何常在 WHERE 或 GROUP BY 子句（对表进行查询）中的谓词引用的列。使用 EVERY COLUMN 选项为构成主键和索引的单列或多列生成直方图。EVERY KEY 选项生成的直方图以构成主键和索引。

如果您经常在一个表的特定列上执行 GROUP BY 子句，则请使用 UPDATESTATISTICS 语句（由 GROUP BY 子句里的列组成）里的 multi-column lists 生成直方图统计信息，以使优化器能够选择一个更好的计划。同样，当一个查询通过两列或更多列联接两个表时，multi-column lists 能帮助优化器选择一个更好的计划。

3.8.2.8 增量更新统计信息

对于非常大的表，UPDATE STATISTICS 处理时间可能很长。减少时间的一种策略是仅为实际需要的列创建直方图（例如，使用 ON NECESSARY COLUMNS 的列组)。另一种策略是增量地更新统计信息。如果需要，这些策略可以一起使用。

要使用增量更新统计信息功能，必须首先为表创建统计信息并创建一个持久性样本表。一种方法是执行一个普通的 update statistics 命令，将关键字

PERSISTENT 添加到 *sample-option*（样本选项）。如果您已经拥有此表合理的最新统计信息，那么另一种方法则可以使用 CREATE SAMPLE 选项单独创建一个持久性样本表。

然后，您可以使用 on 子句中的 INCREMENTAL WHERE 谓词语法增量地执行更新统计信息。应选择谓词来描述自上次执行统计信息更新以来已更改的行集。例如，如果您的表包含一个有时间戳的列，给出了上次更新的日期和时间，则这个列在谓词中非常有用。

如果您稍后决定更改用于持久性样本表的采样率百分比，则可以通过删除持久性样本表（使用 REMOVE SAMPLE）并创建一个新表（使用 CREATE SAMPLE 选项）来实现。使用更高的百分比可以得到更精确的直方图，但要付出长期运行操作的代价。

3.8.2.9 更新统计自动化

为了启用更新统计自动化，请在运行更新统计操作的会话里设置控制查询默认（CQD）属性 USTAT_AUTOMATION_INTERVAL。例如：

```
control query default USTAT_AUTOMATION_INTERVAL '1440';
```

USTAT_AUTOMATION_INTERVAL 的数值拟定为自动化时间间隔（以分钟为单位）；但在 EsgynDB Release 1.0 里，该值并不作为定时间隔。相反，任何大于零的值都能启用更新统计自动化。

启用更新统计自动化后，对您希望优化的每条查询进行预处理。例如：

```
prepare s from select...;
```

PREPARE 语句使 EsgynDB SQL 编译器在没有执行的情况下能编译和优化一个查询。缺少优化器所需的任何直方图都将导致这些列被标记为“需要直方图”。

使用 ON NECESSARY COLUMN[S]针对每个表运行该 UPDATE STATISTICS 语句，以便生成所需的直方图：

```
update statistics for table table-name on necessary columns
sample;
```

3.8.3 UPDATE STATISTICS 例子

- 该例子可以为 EMPLOYEE 表的列 jobcode、empnum、deptnum 和(empnum, deptnum) 生成 4 个直方图。根据该表的大小和数据分布情况，每个直方图应该包含 10 个时间间隔。

```
UPDATE STATISTICS FOR TABLE employee
   ON (jobcode),(empnum, deptnum)
   GENERATE 10 INTERVALS;

--- SQL operation complete.
```

- 该例子使用 DEPT 表的 ON EVERY COLUMN 选项生成直方图统计信息。该语句将执行一次全表扫描，EsgynDB SQL 确定默认的时间间隔数量。

```
UPDATE STATISTICS FOR TABLE dept ON EVERY COLUMN;

--- SQL operation complete.
```

- 假设一家建筑公司有潜在场地的 ADDRESS 表和一个包含 ADDRESS 表某些列的 DEMOLITION_SITES 表。主键为 ZIP。通过两个公共列联接这两个表：

```
SELECT  COUNT(AD.number),  AD.street,  AD.city,  AD.zip,
AD.state
FROM address AD, demolition_sites DS
WHERE AD.zip = DS.zip AND AD.type = DS.type
GROUP BY AD.street, AD.city, AD.zip, AD.state;
```

  为了生成该查询特定的统计信息，输入下列语句：

```
UPDATE STATISTICS FOR TABLE address
ON (street), (city), (state), (zip, type);

UPDATE STATISTICS FOR TABLE demolition_sites
ON (zip, type);
```

- 该例子删除 DEMOLITION_SITES 表的所有直方图：

```
UPDATE STATISTICS FOR TABLE demolition_sites CLEAR;
```

- 该例子有选择性地删除 ADDRESS 表的 STREET 列的直方图:

```
UPDATE STATISTICS FOR TABLE address ON street CLEAR;
```

- 该例子生成统计信息，还会创建一个持久性样本表，用于增量更新统计信息:

```
UPDATE STATISTICS FOR TABLE address
ON (street), (city), (state), (zip, type)
SAMPLE RANDOM 5 PERCENT PERSISTENT;
```

- 该例子增量地更新统计信息。它假定已经创建了一个持久性样本表。

  WHERE子句中的谓词描述自上次更新统计信息以来已更改的行集。这里，我们假设只有与California有关的行发生了变化:

```
UPDATE STATISTICS FOR TABLE address
ON EXISTING COLUMNS
INCREMENTAL WHERE state = 'CA';
```

- 该例子使用LIKE将源表的直方图统计信息拷贝给目标表

```
>>create table tt3(c1 int,c2 int);

--- SQL operation complete.
>>insert into tt3 values(1,2),(2,2);

--- 2 row(s) inserted.
>>create table tt4 like tt3;

--- SQL operation complete.
>>update statistics for table tt3 on every column;

--- SQL operation complete.
>>showstats for table tt3 on every column;

Histogram data for Table TRAFODION.SCH.TT3
Table ID: 559250575550442674 8

   Hist ID # Ints Rowcount UEC Colname(s)
========== ====== =========== ===========
 935491952 2 2 2 SYSKEY
 935491947 2 2 2 C1
 935491942 1 2 1 C2
```

```
--- SQL operation complete.
>>update statistics for table tt4 like tt3;

--- SQL operation complete.
>>showstats for table tt4 on every column;

Histogram data for Table TRAFODION.SCH.TT4
Table ID: 5658933850029147350

   Hist ID # Ints Rowcount UEC Colname(s)
========== ====== =========== ===========
============================
 963081545 2 2 2 SYSKEY
 963081540 2 2 2 C1
 963081535 1 2 1 C2

--- SQL operation complete.
```

4. SQL 语言元素

SQL 语句语法包括 EsgynDB SQL 语言元素，例如数据类型、表达式、函数、标识符、字面值和谓词。本手册中的语句和命令部分支持本章的语言元素的语法和语义描述。

本章目录:

4.1 授权 ID
4.2 字符集
4.3 列
4.4 约束
4.5 相互关系名称
4.6 数据库对象
4.7 数据库对象名称
4.8 数据类型
4.9 表达式
4.10 标识符
4.11 标识列
4.12 索引
4.13 键
4.14 字面值
4.15 Null
4.16 谓词
4.17 权限
4.18 角色
4.19 Schemas
4.20 搜索条件
4.21 子查询
4.22 表
4.23 视图
4.24 ROWID 函数
4.25 ROWNUM 函数

4.1 授权 ID

授权 ID 适用于授权操作。授权流程是验证一个数据库用户是否有权执行一个特定的 SQL 操作。

从外部看，授权 ID 是一种不区分大小写字母的常规或分隔标识符，最多 128 个字符。更多信息，请参阅不区分大小写字母的分隔标识符。

从内部看，授权 ID 与一个 32 位数字相关联，该数字由数据库生成，并用于实现高效的访问和存储。

所有授权 ID 都共享相同的命名空间。授权 ID 可以是数据库用户名或角色名。因此，数据库用户和角色不能共享相同的名称。

一个授权 ID 可以是 PUBLIC 的授权 ID，它代表所有目前和未来的授权 ID。一个授权 ID 不能是 _SYSTEM，这是对象创建者权限的隐式授予者。

4.2 字符集

您可以指定 ISO88591 或 UTF8 定义字符列。使用 UTF8 允许您存储多种不同语言的字符。

4.3 列

列既是一个表的垂直分量，又是一条记录里字段的关系型表示。一个列包含表中每行的数据值。

列值是一个表里可选或可更新的最小数据单位。每个列都有一个名称，这是一种 SQL 标识符，并且在包含该列的表或视图里是唯一的。

4.3.1 列引用

限定的列名或列引用是通过列所属表的名称、视图的名称或相互关系名称限定的列的名称。

如果一个查询引用了分属不同表的同名列，则必须使用一个限定的列名以引用查询里的列。

此外，如果在一个查询里将一个表与自身相联接，则必须通过一个限定的列名以引用一个列，以比较表里的一行与其他行。

列引用或限定列名的语法如下：

```
{table-name | view-name | correlation-name}.column-name
```

如果为一条语句的 FROM 子句里的一个表定义了一个相互关系名称，且需要在

该语句内限定列名，则您必须使用该名称。
如果未在 FROM 子句里定义显式相互关系名称，则可以使用包含该列的表的名称或视图的名称限定该列名。更多信息，请参阅相互关系名称。

4.3.2 派生列名称

派生列是一种 SQL 数值表达式，它将作为一条 SELECT 语句的选择列表项出现。派生列的显式名称是一种与其相关联的 SQL 标识符。

派生列名的语法如下：

```
column-expression [[AS] column-name]
```

列表达式可以简单地是一个列引用，该表达式后面跟有 AS 关键字和派生列名。如果没有为这些派生列分配名称，则查询结果表里的未命名列的标题将作为(EXPR)出现。
如果选择列表里有多个派生列，则应使用 AS 子句分配一些对您有意义的名称，这一点非常重要。

4.3.3 派生列名称例子

以下例子都显示了派生列的名称。

- 第一个例子显示(EXP)作为 SELECT 结果表的列标题：

```
SELECT AVG (salary) FROM persnl.employee;

(EXPR)
----------------
49441.52

--- 1 row(s) selected.
```

- 第二个例子显示 AVERAGE SALARY 作为列标题。

```
SELECT AVG (salary) AS "AVERAGE SALARY"
FROM persnl.employee;

"AVERAGE SALARY"
----------------
49441.52

--- 1 row(s) selected.
```

4.3.4 列的默认设置

创建表时，您可以为列定义特定的默认设置。CREATE TABLE 语句可以定义表列的默认设置。当一条 INSERT 语句省略特定列的一个数值时，列的默认设置在于将数值插入行里。

4.4 约束

SQL 约束是一种可以保护一个表的数据的完整性的对象。这是通过指定表中特定列的所有数值或一组列都必须满足的条件来实现。

EsgynDB SQL 在 SQL 表上强制实施这些约束：

CHECK	**列或表约束，指定该表的每行都必须满足的条件。**
FOREIGN KEY	列或表约束，指定该表的一个引用约束；并声明表里的一列或一组列（外键）仅能包含 REFERENCES 子句指定的表的一列或多列的匹配数值。
NOT NULL	列约束指定该列不得包含空值。
PRIMARY KEY	列或表约束指定一列或一组列作为该表的主键。
UNIQUE	列或表约束指定在一列或一组列里不得多次出现同一个数值或同一组数值。

4.4.1 在 SQL 表上创建或添加约束

使用 NOT NULL、UNIQUE、CHECK、FOREIGN KEY 或 CREATE TABLE 语句的 PRIMARY KEY 子句创建一个 SQL 表时，可以在该表上创建约束。

更多关于 EsgynDB SQL 命令的信息，请参阅 CREATE TABLE 语句和 ALTER TABLE 语句。

4.4.2 约束名

创建一个约束时，您既可以为其指定一个名称，也可由 EsgynDB SQL 为其生成一个名称。您可以选择指定列名和表约束名。约束名是符合ANSI标准的逻辑名称。更多信息，请参阅数据库对象名称。

由于约束名与表和视图属于同一个命名空间，因此，约束名不能和表名或视图

名相同。您指定的名称既可以是完全限定的名称，也可以不是完全限定的名称。

如果您指定了该名称的 schema 部分，则它们必须匹配受影响的表的那些部分，并且它们在该 schema 的表、视图和约束名当中必须是唯一的；如果您省略了指定名称的 schema 部分，则 EsgynDB SQL 将使用该表所属的 schema 扩展该名称。如果您没有指定一个约束名，EsgynDB SQL 将会构造一个 SQL 标识符作为该约束名，并使用表的 schema 限定它。该标识符包括一个表名和连在一起的系统生成的唯一标识符。

4.5 相互关系名称

相互关系名称是一个可以关联表引用（一条 SELECT 语句里的表、视图或子查询）的名称，它可以：

- 区分一个表/视图和语句里引用的另一个表/视图
- 区分同一个表的不同用途
- 缩短查询

相互关系名称可以是显式名称或隐式名称。

4.5.1 显式相互关系名称

一个表引用的显式相互关系名称是一种 SQL 标识符，它与 SELECT 语句的 FROM 子句里的表引用相关联。更多信息，请参阅标识符。

相互关系名称在 FROM 子句里必须是唯一的。更多关于 FROM 子句、表引用和相互关系名称的信息，请参阅 SELECT 语句。

一条 FROM 子句里不同形式的表引用都具有相同的相互关系名称语法：

```
{table | view | (query-expression)} [AS]correlation-name
```

一个表或视图后面可以跟有 AS 关键字和相互关系名称。对于一个通过查询表达式求值得到的派生表，其后面必须跟有 AS 关键字和相互关系名称。只有通过相关的定义语句才能知道显式相互关系名称。您可以在另一个语句里使用相同的标识符作为相互关系名称。

4.5.2 隐式相互关系名称

一个没有显式相互关系名称的表或视图引用具有一个隐式相互关系名称。该名称是通过 schema 名称限定的表或视图名。

您不得为语句中具有显式相互关系名称的引用使用隐式相互关系名称。

4.5.3 相互关系名称例子

该查询引用了两个表 ORDERS 和 CUSTOMER，它们都包含一个名为 CUSTNUM 的列。在 WHERE 子句中，一个列引用通过隐式相互关系名称（ORDERS）限定；另一个列引用则通过显式相互关系名称（C）限定：

```
SELECT ordernum, custname FROM orders, customer c
WHERE orders.custnum = c.custnum AND orders.custnum = 543;
```

4.6 数据库对象

数据库对象是存在于命名空间内的 SQL 实体。SQL 语句可以访问 EsgynDB SQL 数据库对象。以下章节描述了这些 EsgynDB SQL 数据库对象，“约束”、“索引”、“表”和“视图”。

4.6.1 所有权

在 EsgynDB SQL 里，对象创建者将拥有 schema 里定义的对象，并具有该对象的所有权限。

此外，您还可以通过 GRANT 和 REVOKE 语句以授权某特定用户访问表或视图。更多信息，请参阅 GRANT 语句和 REVOKE 语句。

更多关于表和视图权限的信息，请参阅 CREATE TABLE 语句和 ALTER TABLE 语句。

4.7 数据库对象名称

DML 语句可以引用 EsgynDB SQL 数据库对象。为了在一条语句里引用数据库对象，请使用合适的数据库对象名。更多关于数据库对象类型的信息，请参阅数据库对象名称。

4.7.1 SQL 对象的逻辑名称

您可以使用一部分、两部分或三部分逻辑名称（ANSI 名称）引用 SQL 表、视图、约束、库、函数或存储过程。

catalog-name.schema-name.object-name

在这个三部分名称里，*catalog-name* 是 catalog 名称，对于映射到 HBase 表的 EsgynDB SQL 对象而言，该名称为 TRAFODION。*schema-name* 是 schema 名称，*object-name* 则是表、视图、约束、库、函数或存储过程的简易名称。该名称里的。每一项都是一个 SQL 标识符。更多信息，请参阅标识符。

EsgynDB SQL 能通过 schema 名称自动限定对象名，除非您使用对象名显式地指定 schema 名称。如果您没有使用 SET SCHEMA 语句设置该会话所用的 schema 名称，则默认 schema 将是 SEABASE，它存在于 TRAFODIONcatalog 中。更多信息，请参阅 SET SCHEMA 语句。采用“一部分名称”的对象名通过默认 schema 进行隐式限定。您可以使用一部分、两部分或三部分对象名或相互关系名称限定 EsgynDB SQL 语句里的列名。

4.7.2 SQL 对象的命名空间

EsgynDB SQL 对象以分层的方式组织编排。数据库对象存在于 schema 中，而 schema 则包含在一个名为 TRAFODION 的 catalog 里。Catalog 是一个由 schema 组成的集合，catalog 里的 schema 名称必须是唯一的。

如果多个同名对象分属于不同的命名空间，则他们可以共存。

EsgynDB SQL 支持以下命名空间:

- 索引
- 函数和存储过程
- 库
- Schema 标签
- 表值对象（表、视图、约束)

一个 schema 里的对象可以引用不同 schema 里的对象。给定命名空间里的对象必须在给定的 schema 里具有唯一名称。

4.8 数据类型

EsgynDB SQL 数据类型为字符型、日期时间型、时间间隔型或（精确或近似）

数字型：

EsgynDB SQL 数据类型	
字符串 数据类型	固定长度和可变长度的字符数据类型
日期时间数据类型	DATE, TIME 和 TIMESTAMP 数据类型
时间间隔数据类型	Year-month 型时间间隔（年和月）和 day-time 型时间间隔（日、小时、分、秒和几分之一秒）
LOB 数据类型	一组用于存储大量数据的大对象数据类型，提供对数据的随机和分段访问，如 BLOB 和 CLOB。
数字型数据类型	精确和近似数字型数据类型

表里的每一列都与一种数据类型相关联。可以使用 CAST 表达式将数据转换为您指定的数据类型，更多信息，请参阅 CAST 表达式。

下表总结了 EsgynDB SQL 数据类型：

类别	类型	SQL 名称	描述	大小或范围(1)
字符串数据类型	固定长度字符	CHAR[ACTER]	固定长度字符数据	1 到 16,777,216 字节 (2)
		NCHAR	采用预定义国家字符集的固定长度字符数据	1 到 16,777,216 字节(2) (3)
		NATIONAL CHAR[ACTER]	采用预定义国家字符集的固定长度字符数据	1 到 16,777,216 字节(2) (3)
	可变长度字符	VARCHAR	可变长度 ASCII 字符串	1 到 16,777,216 字节(2) (4)
		VARCHAR2	可变长度 ASCII 字符串	1 到 16,777,216 字节(2) (4)
		CHAR[ACTER] VARYING	可变长度 ASCII 字符串	1 到 16,777,216 字节(2) (4)
		NCHAR VARYING	可变长度 ASCII 字符串	1 到 16,777,216 字节(2) (3) (4)
		NATIONAL CHAR[ACTER] VARYING	可变长度 ASCII 字符串	1 到 16,777,216 字节(2) (3) (4)

类别	类型	SQL 名称	描述	大小或范围(1)
日期时间数据类型	Date-Time		时间点，使用公历和 24 小时制。五种受支持的名称如下所示	年份 0001-9999 月份 1-12 天数 1-31 通过月份和年份约束的天数 小时 0-23 分钟 0-59 秒数 0-59 小数秒 (n) 0-999999 这里的 n 是有效位数，1 到 6（默认值为 6；最小值为 1；最大值是 6）。 实际数据库存储大小按照以下字节递增： 年份：2 字节 月份：1 字节 天数：1 字节 小时：1 字节 分钟：1 字节 秒数：1 字节 小数秒：4 字节
		DATE	日期	格式为 YYYY-MM-DD; 实际数据库存储 大小是 4 个字节
		TIME	一天中的时间；24 小时制；没有时间精度	格式为 HH:MM:SS; 实际数据库存

				储大小为 3 个字节
		TIME (带有时间精度)	一天中的时间；24 小时制；带有时间精度	格式为 HH:MM:SS.FFFFFF; 实际数据库存储大小为 7 个字节
		TIMESTAMP	时间点；没有时间精度	格式为 YYYY-MM-DD HH:MM:SS.FFFFFF；实际数据库存储大小为 11 个字节
		TIMESTAMP (带有时间精度)	时间点；带有时间精度	格式为 YYYY-MM-DD HH:MM:SS.FFFFFF; 实际数据库存储大小为 7 个字节

类别	**类型**	**SQL 名称**	**描述**	**大小或范围(1)**
区间数据类型	Interval	INTERVAL	持续时间；数值位于 YEAR/MONTH（年/月）范围内或 DAY/HOUR/MINUTE/SECOND/FRACTION（日/小时/分/秒/小数秒）范围内	年份无约束(5) 月份 0-11 天数无约束小时 0-23 分钟 0-59 秒 0-59 小数秒 (n) 0-999999 这里的 n 是有效位数（默认为 6；最小值为 1；最大值为 6）；存储在 2、4 或 8 个字节里，具体根据位数而定（2）
LOB 数据类型	LOB	BLOB	二进制大对象	10G —— 无限制 目前，限制默认为 10G。这个限制可以根
			字符大对象	

				据需要使用 LOB_MAX_SIZE 配置。

类别	类型	SQL 名称	描述	大小或范围(1)
数字数据类型	Numeric (6) (7)	NUMERIC (1,scale) 到 + NUMERIC (128,scale)	带有可选标度的二进制字；有无符号均可；1-9 位	1 到 128 位；存储方式：1 到 4 位存储在 2 个字节里；5 到 9 位存储在 4 个字节里；10 到 128 位存储在 8-64 个字节里；具体基于精度而定。
		TINYINT	二进制整数；有符号或无符号	0 到 255 无符号；-128 到 +127 有符号；存储在 1 个字节里
		SMALLINT	二进制整数；有符号或无符号	0 到 65535 无符号；-32768 到 +32767 有符号；存储在 2 个字节里
		INTEGER	二进制整数；有符号或无符号	0 到 4294967295 无符号；-2147483648 到 +2147483647 有符号；存储在 4 个字节里
		LARGEINT	二进制整数；仅限有符号	-2^{63} 到 $+(2^{63})-1$；存储在 8 个字节里
	Numeric (扩展精度数字型)[7]	NUMERIC（精度为 19 到 128 位）	二进制整数；有符号或无符号均可	存储为多个 16 位整数 chunk（数据块）。最小存储长度为 8 字节
	浮点数[7]	FLOAT[(精度)]	浮点数（64 位）；指定精度为 1 到 52 位	+/- 2.2250738585072014e-308 到 +/- 1.7976931348623157e+308；存储在 8 个字节里

		REAL	浮点数（32 位）	+/- 1.17549435e-38 到 +/- 3.40282347e+38；存储在 4 个字节里
		DOUBLE PRECISION	浮点数（64 位）精度为 1 到 52 位（52 位二进制精度和 1 位指数位数）	+/- 2.2250738585072014e-308 到 +/- 1.7976931348623157e+308；存储在 8 个字节里
	Decimal number[8]	DECIMAL (1, scale) 到 DECIMAL (18, scale)	小数带有可选标度；存储为 ASCII 字符；1 到 9 位：有无符号均；10 位或更多位：必须有符号	1 到 18 位；字节长度等于位数。 符号被存储为最左侧字节的首位

提示：

1. 一个允许出现空值的列比定义的数据类型多出两个字节。
2. 最大字符数取决于字符集。

字符集	最大字符数
对于 8 位字符集，如 **ISO88591**	等于最大存储大小（以字节为单位）是 16,777,216，即 16,777,216 除以 1
对于 16 位字符集，如 **UCS2**	最大存储大小的一半（以字节为单位）是 8,388,608，即 16,777,216 除以

	2
对于 32 位字符集，如 **UTF8**	最大存储大小的四分之一（以字节为单位）是 4,194,304，即 16,777,216 除以 4

3. 注意：无论字符集在安装时被指定为 ISO88591 还是 UTF8，NCHAR 或 NATIONAL CHAR 都隐式地假设字符集是 UCS2，并且 NCHAR 或 NATIONAL CHAR 的字符集（UCS2）不能被指定或更改。 对于 ALTER 和 CREATE 语句都是如此。
4. 存储变长字符时，存储大小会减少 4 字节。
5. INTERVAL 值里的最大位数为 18，其中包含该值所有 INTERVAL 字段里的位数。任何作为起始字段的 INTERVAL 字段的位数可以多达 18 位。
6. 标度指的是小数点右侧的位数。
7. 精度指定允许出现的小数位数。

4.8.1 可比和兼容的数据类型

如果一种数据类型的数值可以与另一种数据类型的数值相比，则二者是可比的数据类型。

如果一种数据类型的数值可以赋给另一种数据类型的列，并且两种数据类型可通过算术运算结合起来，则这两种数据类型相兼容。兼容的数据类型也可进行比较。

赋值和比较是 EsgynDB SQL 的基本运算。赋值运算将在执行 INSERT 和 UPDATE 语句期间执行，比较操作则在执行包含谓词、聚合（或集合）函数、以及 GROUP BY、HAVING 和 ORDER BY 子句的语句期间执行。

基本的赋值和比较规则是操作数具有兼容的数据类型。在没有进行字符集转换之前，无法比较不同字符集的数据类型。然而， SQL 编译器通常会生成必要的代码，以便自动进行字符集转换。

4.8.1.1 字符串数据类型

同一种字符集的固定长度和可变长度字符数据类型的数值构成了所有字符串。这些字符串可以相互比较和相互赋值。

比较两个字符串时，将为较短的字符串创建临时副本，并右侧填充空格，使其与较长字符串具有相同的长度。

4.8.1.2 日期时间数据类型

只有当日期时间数据类型具有相同的日期时间字段时，它们的数值才可以相互

比较和相互赋值。一个 DATE、TIME 或 TIMESTAMP 值可以与另一个值进行比较（如果后者具有相同数据类型）。

所有的比较都是按时间顺序排列。例如：该谓词为真：

```
TIMESTAMP '2008-09-28 00:00:00' > TIMESTAMP '2008-06-26 00:00:00'
```

4.8.1.3 时间间隔数据类型

只有当 INTERVAL 的类型同时为 year-month（年-月型）时间间隔或 day-time（日-时间型）时间间隔时，二者的数值才能相互比较和相互赋值。例如：该谓词为真：

```
INTERVAL '02-01' YEAR TO MONTH > INTERVAL '00-01' YEAR TO MONTH
```

INTERVAL 的 field 组件可以不相同。例如：以下谓词也为真：

```
INTERVAL '02-01' YEAR TO MONTH > INTERVAL '01' YEAR
```

4.8.1.4 NUMERIC 数据类型

近似数据类型（FLOAT、REAL 和 DOUBLE PRECISION）的数值、以及精确数据类型（NUMERIC、DECIMAL、INTEGER、SMALLINT 和 LARGEINT）的数值构成了所有数字。它们之间既可以相互比较，也可以相互赋值。

如果将一种近似数据类型的数值赋给一个具有精确数据类型的列，则可能出现四舍五入的情况，并且小数部分可能会被截断。如果将一种精确数据类型的数值赋给一个具有近似数据类型的列，则结果可能不等于原始数字。

比较两个数字时，将根据定义的转换规则，使用其中一个数字的临时副本进行比较。例如：如果一个数字为 INTEGER，另一个数字为 DECIMAL，则将使用已转换为小数的整数的临时副本进行比较。

4.8.1.5 扩展精度 NUMERIC 型

EsgynDB SQL 为扩展精度数据提供支持。扩展精度数字型是 NUMERIC(x,y) 数据类型的扩展，这里不存在理论上的精度极限。

它是一种软件数据类型，这意味着底层硬件不支持它，并且所有计算工作都由软件执行。使用这种数据类型执行的计算可能不匹配硬件支持的其他数据类型的性能。

4.8.1.6 扩展精度 NUMERIC 型注意事项

- 成本开销可能超过其他数据类型选项。
- 这是一种软件数据类型。
- 无法与硬件支持的数据类型进行比较
- 如果您的应用程序需要扩展精度数字型数据的算术表达式，请在表的 DDL 语句里指定需要的精度，或对您的选择列表项进行扩展精度类型显式转换。默认系统行为是将用户指定的扩展精度表达式视为扩展精度值。相反，非用户特定（临时中间）扩展精度表达式可能会失去精度。

  在以下例子中，精度似乎丢失了一位，因为系统将两个 NUMERIC(18,4)类型的列的总和当作 NUMERIC(18,4)。NUMERIC(18)是最长的非扩展精度数字型。NUMERIC(19)则是最短的扩展精度数字型。系统实际上会将两个 NUMERIC(18,4)列的总和计算为一个扩展精度 NUMERIC(19,4)。但因为不存在用户指定的扩展精度列，所以系统会将转换后的总和返回给用户指定的 NUMERIC(18,4)类型。

```
CREATE TABLE T(a NUMERIC(18,4), B NUMERIC(18,4));
INSERT INTO T VALUES (1.1234, 2.1234);

>> SELECT A+B FROM T;

(EXPR)
--------------
3.246
```

  如果无法接受该行为，您可以使用以下选项之一：

  - 指定列类型为 NUMERIC(19,4)。例如：
    `CREATE TABLE T(A NUMERIC(19,4), B NUMERIC(19,4));` 或

  - 将总和转换为 NUMERIC(19,4)。例如：
    `SELECT CAST(A+B AS NUMERIC(19,4)) FROM T;` 或

  - 在表达式里使用一个扩展精度字面值。例如：
    `SELECT A+B*1.00000000000000000000 FROM T;`.

- 当类型改为 NUMERIC(19,4)时，请注意前述例子的结果。

```
SELECT CAST(A+B AS NUMERIC(19,4)) FROM T;

(EXPR)
------------
3.2468
```

在基于客户端工具的命令界面里显示输出结果时，可以将一个选择列表项转换为一个扩展精度数字型。然而，在检索应用程序主机变量里的扩展精度选择列表项时，您必须首先将扩展精度数字型转换为一种字符串数据类型。例如：

```
SELECT CAST(CAST(A+B AS NUMERIC(19,4)) AS CHAR(24))
FROM T;

(EXPR)
------------
3.2468
```

注意： 一种应用程序可以将字符串形式的具体扩展精度数值转换为其能够处理的数值值，但不能正确解释内部形式的扩展精度值。

4.8.1.7 扩展精度 NUMERIC 数据类型规则

这些规则适用于以下情况：

- 对于最大精度没有限制。
- 适用于所有支持常规 NUMERIC 数据类型的 DDL 和 DML 语句里仅允许作为散列分区表的部分键列。
- NUMERIC 类型的精度为 10 到 18。
  - 支持 UNSIGNED 作为扩展精度 NUMERIC 数据。
  - 支持 SIGNED 作为 64 位整数。
- CAST 函数允许进行常规 NUMERIC 数据类型和扩展精度 NUMERIC 之间的转换。
- SQL 查询里的参数支持扩展精度 NUMERIC 数据类型。

4.8.1.8 扩展精度 NUMERIC 数据例子

```
>>CREATE TABLE t( n NUMERIC(128,30));
```

```
--- SQL operation complete.

>>SHOWDDL TABLE t;

CREATE TABLE SCH.T (

     N  NUMERIC(128, 30) DEFAULT NULL

     );

--- SQL operation complete.
```

4.8.2 字符串数据类型

EsgynDB SQL 包含固定长度的字符数据和可变长度的字符数据。请勿混淆字符数据与数字型数据、日期时间数据或时间间隔数据。

```
character-type is:
CHAR[ACTER]   [(length   [unit])]   [char-set]   [UPSHIFT]
[[NOT]CASESPECIFIC]
| CHAR[ACTER] VARYING(length [unit]) [char-set] [UPSHIFT]
[[NOT]CASESPECIFIC]
| VARCHAR(length [unit]) [CHARACTERS] [char-set] [UPSHIFT]
[[NOT]CASESPECIFIC]
| VARCHAR2(length [unit]) [CHARACTERS] [char-set] [UPSHIFT]
[[NOT]CASESPECIFIC]
| NCHAR [(length)] [UPSHIFT] [[NOT]CASESPECIFIC]
| NCHAR VARYING (length) [UPSHIFT] [[NOT]CASESPECIFIC]
|     NATIONAL     CHAR[ACTER]     [(length)]     [UPSHIFT]
[[NOT]CASESPECIFIC]
|   NATIONAL   CHAR[ACTER]   VARYING   (length)   [UPSHIFT]
[[NOT]CASESPECIFIC]
```

CHAR、NCHAR 和 NATIONAL CHAR 都是固定长度的字符类型。CHAR VARYING、VARCHAR、NCHAR VARYING 和 NATIONAL CHAR VARYING 都是变长字符类型。

- *length*
  这是一个正整数，指定该列里允许出现的字符数。您必须指定一个长度值。

- *char-set*

指的是 CHARACTER SET char-set-name

- *char-set-name*
  这是字符集的名称，它既可以是 ISO88591，也可以是 UTF8 或 UCS2。
  - ISO88591 (ISO 8859-1) 是美国 ASCII 和西欧语言字符的单字节字符集。
  - UTF8 (UTF-8) 是 Unicode 字符的可变长度（1 到 4 个字节）编码，包括辅助平面中的字符。它与 US-ASCII 字符集兼容。
  - UCS2 (UCS-2) 是基本多语言平面（BMP）的 Unicode 字符的固定长度，2 字节编码。请注意，虽然不是 UCS2 的严格组成部分，但 EsgynDB 也允许 UCS2 列中的 UTF-16 替代（surrogate）对，但是这样的替代对被解释为两个单独的字符。
- CHAR[ACTER] [(*length* [unit])] [*char-set*] [UPSHIFT] [[NOT]CASESPECIFIC]
  指定一列，它带有固定长度的字符数据。

- CHAR[ACTER] VARYING (length)[unit][char-set] [UPSHIFT] [[NOT]CASESPECIFIC]
  指定一列，它带有 *varying-length*（变长）字符数据。VARYING 规定该列里存储的字符数可以小于列长度。
  一个被声明为 VARYING 的列的数值在逻辑上和物理上都可以小于最大长度，但 VARYING 列的最大内部尺寸实际比非 VARYING 的等效列的必需尺寸多出四个字节。

- VARCHAR (*length) [char-set*] [UPSHIFT] [[NOT]CASESPECIFIC]
  指定一列，它带有 *varying-length* 字符数据。VARCHAR 等效于数据类型 CHAR[ACTER] VARYING。

- VARCHAR2 (length) [unit] [char-set][UPSHIFT][[NOT]CASESPECIFIC]
  指定一列，它带有 *varying-length* 字符数据。VARCHAR2 等效于数据类型 CHAR [ACTER] VARYING。

注意： EsgynDB支持VARCHAR2作为VARCHAR的同义词。这提高了一些常用数据库的可移植性。EsgynDB中的VARCHAR2具有VARCHAR的标准语义，也就是说，空字符串是非空值。

- NCHAR [(*length*)] [UPSHIFT] [[NOT]CASESPECIFIC], NATIONAL CHAR[ACTER] [(*length*)] [UPSHIFT] [[NOT]CASESPECIFIC]
  指定一列，它带有预定义的国家字符集的数据。

- NCHAR VARYING [(*length*)] [UPSHIFT] [[NOT]CASESPECIFIC], NATIONAL CHAR[ACTER] VARYING (*length*) [UPSHIFT] [[NOT]CASESPECIFIC]
  指定一列，它带有预定义的国家字符集的 *varying-length* 数据。

4.8.2.1 字符串数据类型注意事项

- CHAR 和 VARCHAR 类型之间的差别
  您可以指定一个固定长度的字符列为 CHAR(n)，这里的 n 指的是您希望存储的字符数。然而，如果您在一个指定为 CHAR(10)的列里存储 5 个字符，则该列将存储 10 个字符，最右侧的 5 个字符为空格。
  如果不希望将空格添加到您的字符串里，您可以指定一个可变长度字符列 VARCHAR(n)，这里的 n 是您希望存储的最多字符数。
  如果您将 5 个字符存储在一个指定为 VARCHAR(10)的列里，则逻辑上只能存储 5 个字符，没有填充空格。

- VARCHAR 和 VARCHAR2 类型之间的差别
  1) 在当前版本中，如果 VARCHAR 字段或 VARCHAR2 字段不是主键字段或索引字段，则 VARCHAR 和 VARCHAR2 是等同的，例如，两者都不会忽略结尾的空格，即'A'不等于'A '。
     举例：
     ```
     create table test1(a char(10), b varchar(10), c varchar2(10));
     insert into test1 values('aaa','aaa','aaa');
     insert into test1 values('bbb ','bbb ','bbb ');
     select * from test1 where b='bbb'; --0rows
     select * from test1 where b='bbb '; --1rows
     select * from test1 where c='bbb'; --0rows
     select * from test1 where c='bbb '; --1rows
     ```

2) 如果 VARCHAR 字段和 VARCHAR2 字段是主键字段或索引字段，VARCHAR 和 VARCHAR2 是不同的，VARCHAR 会忽略尾部的空格，VARCHAR2 不会忽略。

   举例：

   a) VARCHAR 字段被定义为主键，尾部空格被忽略，第三条 INSERT 语句失败。

   ```
   create table test2(a char(10), b varchar(10), c varchar2(10), primary key (a,b));
   insert into test2 values('aaa','aaa','aaa');
   insert into test2 values('bbb ','bbb ','bbb ');
   insert into test2 values('bbb ','bbb','bbb '); --*** ERROR[8102] The operation is prevented by a unique constraint.
   ```

   b) VARCHAR2 字段被定义为主键，尾部空格不会被忽略，第三条 INSERT 语句插入成功。

   ```
   create table test3(a char(10), b varchar(10), c varchar2(10), primary key (a,c));
   insert into test3 values('aaa','aaa','aaa');
   insert into test3 values('bbb ','bbb ','bbb ');
   insert into test3 values('bbb ','bbb ','bbb'); --1 rows
   ```

   c) VARCHAR 字段尾部空格被忽略，在其上面建立索引失败。

   ```
   create table test4(a char(10), b varchar(10), c varchar2(10));
   insert into test4 values('aaa','aaa','aaa');
   insert into test4 values('bbb ','bbb ','bbb ');
   insert into test4 values('bbb ','bbb','bbb ');
   create unique index test4_uidx on test4(a,b); --*** ERROR[8110] Duplicate rows detected.
   ```

   d) VARCHAR2 字段尾部空格不会被忽略，建立索引成功。

   ```
   create table test5(a char(10), b varchar(10), c varchar2(10));
   insert into test5 values('aaa','aaa','aaa');
   insert into test5 values('bbb ','bbb ','bbb ');
   insert into test5 values('bbb ','bbb ','bbb');
   create unique index test5_uidx on test5(a,c); --1 rows
   ```

- SQL 表里的 NCHAR 列

  EsgynDB SQL 里的 NCHAR 类型规范等效于：

  - NATIONAL CHARACTER

- NATIONAL CHAR
- CHAR ... CHARACTER SET ...，这里的字符集指的是 NCHAR 字符集。

同样，您可以使用 NCHAR VARYING、NATIONAL CHARACTER VARYING、NATIONAL CHAR VARYING 和 VARCHAR ... CHARACTER SET ... ，这里的字符集为 NCHAR 字符集。安装 EsgynDB 时，确定使用 NCHAR 字符集。

- 支持 VARCHAR*VARCHAR

  为实现与 Oracle 的兼容，EsgynDB 支持对字符串进行加减乘除运算，只有当字符串里只包含数字字符的时候才能进行隐式转换，将字符串转换成数字，否则报错。该功能受 CQD MODE_COMPATIBLE_1 控制，只有 MODE_COMPATIBLE_1 被设为 ON 的时候才生效。

  例子:

```
SQL>cqd mode_compatible_1 'on';

SQL>select '123'*2 from dual;

(EXPR)
--------------------
                 246

--- 1 row(s) selected.

SQL>select '123'*'2' from dual;

(EXPR)
--------------------------------------------------------
246
--- 1 row(s) selected.

SQL>select '123'-'2' from dual;

(EXPR)
--------------------------------------------------------
121

--- 1 row(s) selected.

SQL>select '123'-2 from dual;

(EXPR)
--------------------
```

```
                  121

--- 1 row(s) selected.

SQL>select '123'+2 from dual;

(EXPR)
--------------------
                  125

--- 1 row(s) selected.

SQL>select '123'+'2' from dual;

(EXPR)
------------------------------------------------------------
125
--- 1 row(s) selected.

SQL>select '123'/2 from dual;

(EXPR)
--------------------
                   61

--- 1 row(s) selected.

SQL>select '123'*NULL from dual;

(EXPR)
------------------------------------------------------------
NULL

--- 1 row(s) selected.
```

4.8.3 日期时间数据类型

日期时间数据类型的数值代表一个基于公历 24 小时制的当地时间点。一个日期时间项可以代表一个日期、一个时间或一个日期时间。

如果针对一种数据类型进行数字值加/减操作，该数字值将被自动转换为一个 INTERVAL DAY 值。如果针对一种时间类型或时间戳类型进行数字值加/减操作，该数字值将被自动转换为一个 INTERVAL SECOND 值。更多有关 CAST 的信息，请参阅 CAST 表达式。

EsgynDB SQL 接受公历中省略的日期数据，例如：October 5 to 14, 1582（1582 年 10 月 5 日至 14 日）。该功能属于 EsgynDB SQL 的扩展功能。
日期时间值可以代表的时间范围为：January 1, 1 A.D., 00:00:00.000000（低值）December 31, 9999, 23:59:59.999999（高值）

EsgynDB SQL 有三种日期时间数据类型:

```
datetime-type is:
DATE
| TIME [(time-precision)]
| TIMESTAMP [(timestamp-precision)]
```

- DATE
  指定一个日期时间列。它包含一个外部日期形式 yyyy-mm-dd，并存储在 4 个字节里。
- TIME [(*time-precision*)]
  指定一个日期时间列。它没有可选的 *time-precision*（时间精度），包含一种外部时间形式 yyyy-mm-dd hh:mm:ss，并存储在 3 个字节里。
  time-precision 是一个无符号的整数，它指定小数部分的秒的位数，并存储在 4 个字节里。 *time-precision* 的默认值为 0，最大值为 6。

- TIMESTAMP
  指定一个 datetime（时间日期）列。
  外部时间戳形式为 yyyy-mm-dd hh:mm:ss.ffffff，默认精度为 6，存储大小为 11 个字节。

  例子:

```
SQL>SELECT CAST(CURRENT_TIMESTAMP AS TIMESTAMP) FROM
DUAL;
(EXPR)
--------------------------
2018-08-14 05:04:14.000591
--- 1 row(s) selected.

SQL>CREATE TABLE test1(c1 TIMESTAMP);
--- SQL operation complete.
SQL>SHOWDDL test1;
CREATE TABLE TRAFODION.SEABASE.TEST1
(
C1 TIMESTAMP(6) DEFAULT NULL NOT SERIALIZED
)
```

```
ATTRIBUTES ALIGNED FORMAT
;
-- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON
TRAFODION.SEABASE.TEST1 TO
DB__ROOT WITH GRANT OPTION;
--- SQL operation complete.
```

- TIMESTAMP [(timestamp-precision)]
  指定一个具有 *timestamp-precision* 的 datetime（日期时间）列。*timestamp-precision* 是一个无符号的整数，它指定小数部分的秒的位数，*timestamp-precision* 范围是 0 到 9，存储大小根据 *timestamp-precision* 而变化。

time-precision	**存储大小（字节）**
0	7
1~9	11

例子:

```
SQL>CREATE TABLE test2 (c1 TIMESTAMP, c2 TIMESTAMP(0), c3
TIMESTAMP(1), c4
TIMESTAMP(2), c5 TIMESTAMP(6), c6 TIMESTAMP(9));
--- SQL operation complete.
SQL>SELECT       COLUMN_NAME,       COLUMN_SIZE       from
"_MD_".columns_view WHERE TABLE_NAME =
'TEST2';
COLUMN_NAME COLUMN_SIZE
----------------------------------------
C1 11
C2 7
C3 11
C4 11
C5 11
C6 11
SYSKEY 8
--- 7 row(s) selected.
SQL>SHOWDDL test2;
CREATE TABLE TRAFODION.SEABASE.TEST2
(
C1 TIMESTAMP(6) DEFAULT NULL NOT SERIALIZED
, C2 TIMESTAMP(0) DEFAULT NULL NOT SERIALIZED
, C3 TIMESTAMP(1) DEFAULT NULL NOT SERIALIZED
, C4 TIMESTAMP(2) DEFAULT NULL NOT SERIALIZED
, C5 TIMESTAMP(6) DEFAULT NULL NOT SERIALIZED
, C6 TIMESTAMP(9) DEFAULT NULL NOT SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT
;
```

```
-- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON
TRAFODION.SEABASE.TEST2 TO
DB__ROOT WITH GRANT OPTION;
--- SQL operation complete.
```

4.8.3.1 日期时间数据类型注意事项

- 日期时间范围

DATE、TIME 或 TIMESTAMP 列中的单个字段的数值范围被指定为:

yyyy	年，0001 到 9999
mm	月，01 到 12
dd	日，01 到 31
hh	小时，00 到 23
mm	分，00 到 59
ss	秒，0 到 59
mssssss	微秒，000000 到 999999

如果您在 DML 语句里指定了 datetime_value（FORMAT 'string'），并且指定的格式为 'mm/dd/yyyy'、'MM/DD/YYYY、'yyyy/mm/dd' 或'yyyy-mm-dd'，则 datetime 类型将被自动转换。

4.8.4 时间间隔数据类型

时间间隔数据类型的数值代表 year-month（年-月型）为单位的持续时间（年和月），或以 day-time（日-时间型）为单位的持续时间（日、小时、分、秒和几分之一秒）。

```
interval-type is:
     INTERVAL[-] { start-field TO end-field | single-field }

start-field is:
     {YEAR | MONTH | DAY | HOUR | MINUTE} [(_leading-
precision_)]

end-field is:
     YEAR | MONTH | DAY | HOUR | MINUTE | SECOND
     [(_fractional-precision_)]

single-field is:
     _start-field_   |   SECOND   [(_leading-precision_,
```

fractional-precision)]

- INTERVAL[-] {*start-field* TO *end-field* |*single-field*}

  指定表示持续时间的列，具体为 year-month（年-月型）、day-time（日-时间型）或单个字段。可选的符号表明它是正整数还是负整数。如果您省略该符号，则它默认为正整数。

  如果时间间隔被指定为一个范围，则起始字段和终止字段必须是以下类别之一：

  - {YEAR|MONTH|DAY|HOUR|MINUTE} [(*leading-precision*)]
    指定 *start-field*（起始字段）。
    一个 *start-field* 的前导精度可以多达 18 位数（最大值取决于时间间隔内的字段数量）。*leading-precision* 是 *start-field* 里允许出现的位数。*leading-precision* 的默认值为 2。

  - YEAR|MONTH|DAY|HOUR|MINUTE|SECOND[(*fractional-precision)]*
    指定 *end-field*（终止字段）。如果 *end-field* 为 SECOND，则它的 *fractional-Precision*（小数部分的精度）可以多达 6 位数。*fractional-precision* 是小数点后的精度位数，默认值为 6。

  - *start-field*|SECOND [(*leading-precision, fractional-precision*)]
    指定 *single-field*（单个字段）。如果 *single-field* 为 SECOND，则 *leading-Precision* 将是小数点前的精度位数，*fractional-precision* 则是小数点后的精度位数；
    leading-precision 的默认值为 2，*fractional-precision* 的默认值则为 6；
    leading-precision 的最大值为 18，*fractional-precision* 的最大值仍为 6。

4.8.4.1 时间间隔数据类型注意事项

- 加/减不精确的时间间隔值

如果加/减的时间间隔是任何 YEAR（年）、MONTH（月）的倍数、或它们的组合，都会导致出现运行时错误。

例如：在 January 31, 2009（2009 年 1 月 31 日）里加入 1 MONTH 将会出现错误，因为 February 31 不存在，并且不确定该用户是否愿意四舍五入回到 2009 年 2 月 28 日、四舍五入至 2009 年 3 月 1 日、或将 1 MONTH 当成 30 天处理，这样得到的答案将是 March 2, 2009（2009 年 3 月 2 日）。

同样，从 February 29, 2008（2009 年 2 月 29 日）里减去 1 YEAR（一年）时也会出现错误。

更多关于如何加/减此类时间间隔而不收到运行时错误，请参阅 ADD_MONTHS 函数、DATE_ADD 函数、DATE_SUB 函数和 DATEADD 函数。

- 时间间隔前导精度

  leading-precision 的最大值取决于时间间隔内的字段数量和 *fractional-precision*（小数部分的精度）。

  最大值被计算为:

```
[[18 - _fractional-precision_ - 2 * (_n_ - 1)]]
_max-leading-precision_ = 18 - _fractional-precision_ -
2 * (_N_ - 1)
```

  这里的 N 是时间间隔内的字段数量。

  例如：数据类型为 INTERVAL YEAR TO MONTH 的列里的前导精度的最大位数被计算为：18 – 0 – 2 * (2 – 1) = 16

- 时间间隔范围

  在一个时间间隔范围（而非单个字段）的定义中，*start-field*（起始字段）和 *end-field*（终止字段）可以是任何指定的字段。

  具体使用限制如下:

  - 时间间隔范围既可以是 year-month（年-月型），也可以是 day-time（日-时间型）。即如果 *start-field* 为 YEAR，则 *end-field* 为 MONTH；如果 *start-field* 为 YDAY、HOUR 或 MINUTE，则 *end-field* 也将是一个时间字段。
  - 在层次结构中，*start-field* 必须位于 *end-field* 前面：YEAR、MONTH、DAY、HOUR、MINUTE 和 SECOND。

- 有符号的时间间隔

  为了将一个带引号的字符串包含在有符号的时间间隔数据类型里，该符号必须位于这个带引号的字符串的外部。它既可以位于整个字面值前面，也可以位于用引号括起来的 duration（持续时间）前面。

  例如：对于时间间隔 “minus (5 years 5 months)，这些格式是有效的：

  ```
  INTERVAL - '05-05'YEAR TO MONTH - INTERVAL '05-05' YEAR
  TO MONTH
  ```

- 溢出条件

  当您将一个分数值插入 INTERVAL 数据类型的字段时，如果该分数值为 0，则不会导致溢出。将数值 INTERVAL '1.000000' SECOND(6)插入 SECOND(0)字段不会导致数值损失。如果该数值匹配目标列，并且没有精度损失，EsgynDB SQL 将不会返回一个溢出错误。

  然而，如果该分数值大于 0，则会出现溢出。插入数值 INTERVAL '1.000001' SECOND(6)将会导致数值损失。

4.8.5 数字型数据类型

数字型数据类型既可以是精确数字型数据类型，也可以是近似数字型数据类型。一种数字型数据类型可以兼容任何其他的数字型数据类型，但不可兼容字符、日期时间或时间间隔数据类型。

```
exact-numeric-type is:
     NUMERIC [(precision [,scale])] [SIGNED|UNSIGNED]
     | TINYINT [SIGNED|UNSIGNED]
     | INT[EGER] [SIGNED|UNSIGNED]
     | LARGEINT
     | DEC[IMAL] [(precision [,scale])] [SIGNED|UNSIGNED]

approximate-numeric-type is:
     FLOAT [(precision)]
     | REAL
     | DOUBLE PRECISION
```

精确数字型数据类型是那些可以精确代表一个数值的数据类型：NUMERIC（数字型）、SMALLINT（小整型）、INTEGER（整型）、LARGEINT（大整型）和 DECIMAL（小数型）。

近似数字型数据类型是那些不一定精确代表一个数值的数据类型： FLOAT（浮点型）、REAL（实型）和 DOUBLE PRECISION（双精度型）。

EsgynDB SQL 表声明浮点数据类型的一列将被存储为 IEEE 浮点数格式，这里的所有计算都以此为基础进行。EsgynDB SQL 表可以仅包含 IEEE 浮点数据。

- NUMERIC *[(precision [,scale])]* [SIGNED|UNSIGNED]
  指定一个精确数字列——一个两字节的二进制数，SIGNED（有符号）或 UNSIGNED（无符号）均可。
  precision 指定一个不超过 128 的总位数。如果 precision 介于 10 和 18 之间，您必须使用有符号数值才能获得受支持的硬件数据类型。如果精度超过 18，将得到受支持的软件数据类型；如果精度类型介于 10 和 18 之间，并且您指定了 UNSIGNED 选项，将得到受支持的软件数据类型。标度可以指定小数点右侧的位数。
  默认值为 NUMERIC (9,0) SIGNED。

为兼容 Oracle，EsgynDB 将 NUMBER 和 NUMERIC 视为等同，其类型含义和用法都是相同的。

- TINYINT [SIGNED|UNSIGNED]
  指定一个精确数字列 ——一个一字节的二进制整数，SIGNED（有符号）或 UNSIGNED（无符号）均可。
  该列可存储的整数范围：无符号：0 到 255；有符号：-128 到+127。默认为 SIGNED。

- SMALLINT [SIGNED|UNSIGNED]
  指定一个精确数字列——一个两字节的二进制整数，SIGNED（有符号）或 UNSIGNED（无符号）均可。
  该列可存储的整数范围：无符号：0 到 65535；有符号：-32768 到+32767。默认为 SIGNED。

- INT[EGER] [SIGNED|UNSIGNED]
  指定一个精确数字列——一个 4 字节的二进制整数，SIGNED 或 UNSIGNED 均可。

该列可存储的整数范围：无符号： 0 到 4294967295；有符号： -2147483648 到+2147483647。

默认为 SIGNED。

- LARGEINT

  指定一个精确数字列——一个 8 字节、有符号的二进制整数。该列存储的整数范围是

  -2**63 to +2**63 -1（大约为 9.223 乘以 10 的 18 次方）。

- DEC[IMAL] *[(precision [,scale])]* [SIGNED|UNSIGNED]

  指定一个精确数字列——一个小数，SIGNED 或 UNSIGNED 均可，存储为 ASCII 字符。

  precision 指定总位数不得超过 18。如果 *precision* 大于或等于 10，则该数值必须为 SIGNED。符号将被存储为的最左边的字节的第一位。*scale* 指定小数点右侧的位数。

  默认值为 DECIMAL (9,0) SIGNED。

- FLOAT *[(precision)]*

  指定一个近似数字列。该列存储浮点数，指定的精度为 1 到 54 位；该列存储浮点数，范围是 +/- 2.2250738585072014e-308 到 +/1.7976931348623157e+308，存储在 8 个字节里。

  IEEE FLOAT 精度数据类型被存储为一个 IEEE DOUBLE，即存储在 8 个字节里，并具有指定的精度。

  默认精度为 52。

- REAL

  指定一个 4 字节的近似数字列。该列可以存储 32 位的浮点数；其中，二进制精度为 23 位，指数位为 8 位。

  最小值和最大值范围：+/- 1.17549435e-38 到+/ 3.40282347e+38。

- DOUBLE PRECISION

  指定一个 8 字节的近似数字列。

  该列存储 64 位浮点数，指定的精度从 1 到 52 位不等。

IEEE DOUBLE PRECISION 数据类型被存储在 8 个字节里；其中，二进制精

度为 52 位，指数位为 1 位。最小值和最大值范围是+/-2.2250738585072014e-308 到+/-1.7976931348623157e+308。

4.8.6 数据类型的隐式转换

EsgynDB 的所有函数都支持数据类型的隐式转换，具体规则如下：

1. 动态参数和 NULL 初始是没有数据类型的
2. 动态参数和 NULL 与一个有类型的数据比较，动态参数和 NULL 的类型会转换成与其比较的数据的数据类型。
3. 如果两个没有数据类型的值比较，如：动态参数和动态参数比较，动态参数和 NULL 比较，NULL 和 NULL 比较，其类型变为默认的 varchar(255)。
4. 如果一个函数有多个输出参数可作为其函数返回值，这些输出参数的类型应该是兼容的。

4.9 表达式

SQL 数值表达式的是求值结果为一个数值的表达式。EsgynDB SQL 支持以下表达式类型：

EsgynDB SQL 支持的表达式类型	
字符数值表达式	操作数可以和串联运算符 (||)结合使用，例如：'HOUSTON,' ||' TEXAS'
日期时间值表达式	通过特定的方式将操作数与算术运算符结合使用。 例如：CURRENT_DATE + INTERVAL '1' DAY
时间间隔数值表达式	通过特定的方式将操作数与加法/减法运算符结合使用。 例如：INTERVAL '2' YEAR - INTERVAL '3' MONTH
数字值表达式	通过特定的方式将操作数与算术运算符结合使用。 例如：SALARY * 1.10

一个表达式的数据类型是其数值的数据类型。

一个数值表达式可以是字符串字面值、数字字面值、动态参数或列名（指定表里一行的列值）。 一个数值表达式也可包含函数和标量子查询。

4.9.1 字符值表达式

字符值表达式的操作数被称作 character primaries（主要字符），它们可以与串联运算符 (||)结合使用。一种主要字符的数据类型是字符串。

```
character-expression is:
     character-primary
     | character-expression || character-primary

character-primary is:
     character-string-literal
     | column-reference
     | character-type-host-variable
     | dynamic parameter
     | character-value-function
     | aggregate-function
     | sequence-function
     | scalar-subquery
     | CASE- expression
     | CAST-expression
     | (character- expression)
```

字符（或字符串）值表达式可以通过以下操作数构建:

- 字符串字面值
- 字符串函数
- 带有字符值的列引用
- 动态参数
- 可以返回字符值的聚合函数、标量子查询、CASE 表达式或 CAST 表达式

4.9.1.1 字符值表达式例子

以下为字符值表达式的例子:

表达式	描述
'ABILENE'	字符串字面值
'ABILENE ' ||' TEXAS'	串联两个字符串字面值
'ABILENE ' ||' TEXAS ' || x’55 53 41'	将三个字符串字面值串联构成字面值
'Customer ' || custname	将一个字符串字面值与 CUSTNAME 列值串联
CAST (order_date AS CHAR(10))	在日期值上使用 CAST 函数

4.9.2 日期时间值表达式

通过特定的方式将日期时间值表达式的操作数与算术运算符结合使用。在这个语法图中，datetime primary 的数据类型为 DATE、TIME 或 TIMESTAMP。间隔期的数据类型为 INTERVAL。

```
datetime-expression is:
     datetime-primary
     | interval-expression + datetime-primary
     | datetime-expression + interval-term
     | datetime-expression - interval-term

datetime-primary is:
     datetime-literal
     | column-reference
     | datetime-type-host-variable
     | dynamic parameter
     | datetime-value-function
     | aggregate-function
     | sequence-function
     | scalar-subquery
     | CASE-expression
     | CAST-expression
     | (datetime-expression)

interval-term is:
     interval-factor
     | numeric-term * interval-factor
interval-factor is:
     [+|-] interval-primary

interval-primary is:
     interval-literal
     | column-reference
     | interval-type-host-variable
     | dynamic parameter
     | aggregate-function
     | sequence-function
     | scalar-subquery
     | CASE-expression
     | CAST-expression
     | (interval-expression)
```

4.9.2.1 日期时间值表达式可以通过以下操作数构建:

- 时间间隔值表达式
- 日期时间或时间间隔字面值
- 动态参数
- 带有日期时间或时间间隔值的列引用
- 动态参数
- 日期时间或时间间隔函数
- 任何可以返回时间间隔或时间间隔值的聚合函数、标量子查询、CASE 表达式或 CAST 表达式

4.9.2.2 日期时间值表达式注意事项

- 执行结果的数据类型

  一般而言，执行结果的数据类型即日期时间表达式的 datetime-primary（基本日期时间）部分的数据类型。例如：日期时间值表达式包含:

日期时间表达式	描述	执行结果的数据类型
CURRENT_DATE + INTERVAL '1' DAY	当前日期加上为期一天的时间间隔	DATE
CURRENT_DATE + est_complete	当前日期加上列 EST_COMPLETE里的时间间隔值	DATE
(SELECT ship_timestamp FROM WHERE projcode=1000) + INTERVAL ’07:04’DAY TO HOUR	指定项目、时间间隔值为 7 天 4 小时的运送时间戳	TIMESTAMP

第一个表达式里的基本日期时间是CURRENT_DATE，这是一个返回值为DATE数据类型的函数。因此，执行结果的数据类型为 DATE。

在最后一个表达式里的 datetime primary 指的是以下这条标量子查询:

```
(SELECT ship_timestamp FROM project WHERE projcode=1000)
```

前一个子查询可以返回一个 TIMESTAMP 数据类型的数值。因此，执行结果的

数据类型为 TIMESTAMP。

4.9.2.3 日期时间或时间间隔操作数的运算限制

您能在日期时间值表达式中使用日期时间和时间间隔操作数（带算术运算符）：

操作数 1	运算符	操作数 2	结果类型
Datetime	+ or -	Interval	Datetime
Interval	+	Datetime	Datetime

当将一个数字值与 DATE 类型相加或从一个 DATE 类型里减去该值时，该值都会被自动转换为一个 INTERVAL DAY 值。更多关于 CAST 表达式的信息，请参阅 CAST 表达式。

使用这些操作时，请注意：

- 将间隔月份与一个 DATE 值相加或从一个 DATE 值减去间隔月份时，得到的结果将是同一天加/减指定的月数后所得的数值。因为不同的月份长度各异，所以该结果是近似结果。

- 日期时间和时间间隔值的算术运算可能产生意外的结果，具体将取决于字段的使用方式。

  例如：执行该表达式（自左向右求值）时会返回一个错误：

  ```
  DATE '2007-01-30' + INTERVAL '1' MONTH + INTERVAL '7' DAY
  ```

  相比之下，该表达式（与前一个表达式相同的数值相加，但采用不同的顺序）可以正确地生成数值 2007-03-06：

  ```
  DATE '2007-01-30' + INTERVAL '7' DAY + INTERVAL '1' MONTH
  ```

  使用“ADD_MONTHS 函数”可以避免出现这些意外的结果。

4.9.2.4 日期时间值表达式的例子

PROJECT 表包含 5 列，它们分别使用数据类型 NUMERIC、VARCHAR、DATE、TIMESTAMP 和 INTERVAL DAY。

假设您已经将数值插入 PROJECT 表，例如：

```
INSERT INTO persnl.project
VALUES (1000,'SALT LAKE CITY',DATE '2007-04-10',
     TIMESTAMP '2007-04-21:08:15:00.00',INTERVAL '15' DAY);
```

以下例子将在 PROJECT 表里使用这些数值:

PROJCODE	START_DATE	SHIP_TIMESTAMP	EST_COMPLETE
1000	2007-04-10	2007-04-21 08:15:00.00	15
945	2007-10-20	2007-12-21 08:15:00.00	30
920	2007-02-21	2007-03-12 09:45:00.00	20
134	2007-1 -20	SHIP_TIMESTAMP	30

- 将一个通过 YEAR 限定的时间间隔值与一个日期时间值相加:

```
SELECT start_date + INTERVAL '1' YEAR
FROM persnl.project
WHERE projcode = 1000;

(EXPR)
----------
2008-04-10
--- 1 row(s) selected.
```

- 从一个日期时间值里减去通过 MONTH 限定的时间间隔值:

```
SELECT ship_timestamp - INTERVAL '1' MONTH
FROM persnl.project
WHERE projcode = 134;

(EXPR)
---------------------------
2007-12-01 00:00:00.000000

--- 1 row(s) selected.
```

执行结果是 2007-12-01 00:00:00.00。The YEAR 值按 1 递减，因为从 January 1 里减去一个月将导致日期回到去年。

- 将一列（数值为通过 DAY 限定的时间间隔值）与日期时间值相加:

```
SELECT start_date + est_complete
FROM persnl.project
```

```
WHERE projcode = 920;
 (EXPR)
----------
2007-03-12

--- 1 row(s) selected.
```

将 2008-02-21 加上 20 日，可以得到 2008-03-12。EsgynDB SQL 将 2008 年正确地作为闰年处理。

- 从一个日期时间值里减去一个时间间隔值（通过 HOUR TO MINUTE 限定）:

```
SELECT ship_timestamp - INTERVAL '15:30' HOUR TO MINUT
FROM persnl.project
WHERE projcode = 1000;

(EXPR)
---------------------------
2008-04-20 16:45:00.000000
```

从 2007-04-21 08:15:00.00 减去 15 小时 30 分的结果为 2007-04-20 16:45:00.00。

4.9.3 日期间隔值表达式

一个日期间隔值表达式可以通过特定的方式将操作数与加法和减法运算符结合使用。在这个语法图里，一个日期时间表达式的数据类型为 DATE、TIME 或 TIMESTAMP；一个间隔期或表达式的数据类型为 INTERVAL。

```
interval-expression is:
interval-term
| interval-expression + interval-term
| interval-expression - interval-term
| (datetime-expression - datetime-primary) [interval-
qualifier]

interval-term is:
interval-factor
| interval-term * numeric-factor


| interval-term / numeric-factor
```

```
      | numeric-term * interval-factor


interval-factor is:
[+|-] interval-primary

interval-primary is:
     interval-literal
      | column-reference
      | interval-type-host-variable
      | dynamic-parameter
      | aggregate-function
      | sequence-function
      | scalar-subquery
      | CASE-expression
      | CAST-expression
      | (interval-expression)

numeric-factor is:
      [+|-] numeric-primary
      | [+|-] numeric-primary ** numeric-factor
```

时间间隔值表达式可通过以下操作数构建:

- 整数
- 日期时间值表达式
- 时间间隔字面值
- 带有日期时间或时间间隔值的列引用
- 动态参数
- 日期时间或时间间隔值函数
- 返回时间间隔值的聚合函数、标量子查询、CASE 表达式或 CAST 表达式

更多有关 interval-term（间隔期）、datetime-expression（日期时间表达式）和 datetime-primary（基本日期时间）的信息，请参阅日期时间值表达式。

如果时间间隔表达式是两个日期时间表达式的差值，默认情况下，执行结果将被表示为该时间间隔的最低有效度量单位。
对于日期差值，时间间隔将被表示为天数。对于时间戳差值，时间间隔将被表示为几分之一秒。

如果时间间隔表达式是时间间隔操作数的差值或总和，则操作数的时间间隔限定符既可以是 year-month（年-月型）、也可以是 day-time（日-时间型）。

如果您正在更新或插入两个时间间隔限定符相加或相减所得的结果值，则执行结果的时间间隔限定符将取决于目标列的时间间隔限定符。

4.9.3.1 时间间隔值表达式注意事项

- 起始字段和终止字段

  在时间间隔范围的定义中，起始字段和终止字段可以是符合以下限制的任何指定的字段，具体使用限制如下：

  - 时间间隔既可以是 year-month（年-月型），也可以是 day-time（日-时间型）。如果 start-field 为 YEAR，则 end-field 为 MONTH；
    如果 start-field 为 DAY、HOUR 或 MINUTE，则 end-field 也应是一个时间字段
  - 在层次上，start-field 必须位于 end-field 前面：YEAR、MONTH、DAY、HOUR、MINUTE 和 SECOND。

  在一个时间间隔表达式的定义中，表达式所有操作数的 start -field 和 end-field 都必须为 year-month（年-月型）或 day-time（日-时间型）。

- 时间间隔限定符

  结果表达式里时间间隔限定符的确定规则各不相同。例如：时间间隔值表达式包含：

日期时间表达式	描述	结果数据类型
CURRENT_DATE - start_date	默认情况下，当前日期与START_DATE列值之差用天数表示。您不必指定时间间隔限定符	INTERVAL DAY (12)
INTERVAL '3' DAY - INTERVAL '2' DAY	两个时间间隔字面值之差；执行结果为 1 日	INTERVAL DAY (3)
INTERVAL '3' DAY +	两个时间间隔字面值之	INTERVAL DAY (3)

INTERVAL '2' DAY	和；执行结果为5日	
INTERVAL '2' YEAR – INTERVAL '3' MONTH	两个时间间隔字面值之差；执行结果为 1 年零 9 个月	INTERVAL YEAR (3) TO MONTH

- 运算限制

在一个时间间隔值表达式里，您只能以这些组合方式同时使用日期时间/时间间隔操作数与算术运算符：

操作数 1	运算符	操作数 2	结果类型
Datetime	-	Datetime	Interval
Interval	+或-	Interval	Interval
Interval	*或/	Numeric	Interval
Numeric	*	Interval	Interval

本表列出了日期时间和时间间隔算术运算符，以及结果数据类型的有效组合：

操作数	结果类型
Date + Interval 或 Interval + Date	Date
Date + Numeric 或 Numeric + Date	Date
Date - Numeric	Date
Date – Interval	Date
Date – Date	Interval
Time + Interval 或 Interval + Time	Time
Time + Numeric 或 Numeric + Time	Time
Time - Number	Time
Time – Interval	Time
Timestamp + Interval 或 Interval + Timestamp	Timestamp
Timestamp + Numeric 或 Numeric + Timestamp	Timestamp
Timestamp - Numeric	Timestamp
Timestamp – Interval	Timestamp
year-month Interval + year-month Interval	year-month Interval
day-time Interval + day-time Interval	day-time Interval
year-month Interval – year-month	year-month Interval

Interval	
day-time Interval – day-time Interval	day-time Interval
Time – Time	Interval
Timestamp – Timestamp	Interval
Interval * Number 或 Number * Interval	Interval
Interval / Number	Interval
Interval – Interval 或 Interval + Interval	Interval

使用这些运算时，请注意：

- 如果您希望从另一个日期时间值里减去一个日期时间值，两个值都必须具有相同的数据类型。为了得到这个结果，请使用 CAST 表达式。例如：
  `CAST (ship_timestamp AS DATE) - start_date`

- 如果您希望从另一个日期时间值里减去一个日期时间值，并且指定了时间间隔限定符，则必须允许执行结果里有精度所需的最大位数。例如：
  `(CURRENT_TIMESTAMP - ship_timestamp) DAY(4) TO SECOND(6)`

- 如果您正在更新两个字面值数值的相加或相减结果，并且源值不符合目标列的时间间隔字段范围，则会出现一个 SQL 错误。

  例如：这个表达式无法替换 INTERVAL DAY 列：
  `INTERVAL '1' MONTH + INTERVAL '7' DAY`
- 如果您将一个时间间隔值与一个数字值表达式相乘或相除，EsgynDB SQL 会把时间间隔值转换为最低有效的子字段，然后将其与数字值表达式相乘或相除。执行结果将与这个被乘或除的时间间隔具有相同的字段。例如：

  这个表达式将返回数值 5-02：
  `INTERVAL '2-7' YEAR TO MONTH * 2`

- 时间间隔值表达式的例子

  PROJECT 表包含 5 列，它们分别使用数据类型 NUMERIC、VARCHAR、DATE、TIMESTAMP 和 INTERVAL DAY。假设您已经将数值插入到 PROJECT 表里。

  例如：
  ```
  INSERT INTO persnl.project
  VALUES (1000,'SALT LAKE CITY',DATE '2007-04-10',
   TIMESTAMP '2007-04-21:08:15:00.00',INTERVAL '15' DAY);
  ```

以下例子在 PROJECT 表里使用这些数值:

PROJCODE	START_DATE	SHIP_TIMESTAMP	EST_COMPLETE
1000	2007-04-10	2007-04-21:08:15:00.0000	15
2000	2007-06-10	2007-07-21:08:30:00.0000	30
2500	2007-10-10	2007-12-21:09:00:00.0000	60
3000	2007-08-21	2007-10-21:08:10:00.0000	60
4000	2007-09-21	2007-10-21:10:15:00.0000	30
5000	2007-09-28	2007-10-28:09:25:01.1 1	30

- 假设 CURRENT_TIMESTAMP 为 2000-01-06 1 :14:41.748703。
  当前时间戳和 PROJECT 表里的 SHIP_TIMESTAMP 之间差值，天数、小时、分、秒和几分之一秒:

```
SELECT projcode,
(CURRENT_TIMESTAMP - ship_timestamp) DAY(4) TO
SECOND(6)
FROM samdbcat.persnl.project;

Project/Code   (EXPR)
------------   ---------------------
        1000   1355 02:58:57.087086
        2000   1264 02:43:57.087086
        2500   1111 02:13:57.087086
        3000   1172 03:03:57.087086
        4000   1172 00:58:57.087086
        5000   1165 01:48:55.975986
--- 6 row(s) selected.
```

4.9.4 数字值表达式

一个数字值表达式的可以通过特定的方式将操作数与算术运算符结合使用。在这个语法图里，term、factor 或 numeric primary 的数据类型均为数字型。

numeric-expression is:

```
      numeric-term
      | numeric-expression + numeric-term
      | numeric-expression - numeric-term

numeric-term is:
      numeric-factor
      | numeric-term * numeric-factor
      | numeric-term / numeric-factor

numeric-factor is:
      [+|-] numeric-primary
      | [+|-] numeric-primary ** numeric-factor

numeric-primary is:
      unsigned-numeric-literal
      | column-reference
      | numeric-type-host-variable
      | dynamic parameter
      | numeric-value-function
      | aggregate-function
      | sequence-function
      | scalar-subquery
      | CASE-expression
      | CAST-expression
      | (numeric-expression)
```

如前述语法图里所示，数字值表达式可以通过以下操作数构建:

- 数字字面值
- 带有数字值的列引用
- 动态参数
- 数字值函数
- 返回数字值的聚合函数、标量子查询、CASE 表达式或 CAST 表达式

4.9.4.1 数字值表达式注意事项

求值顺序:

1. 括号内的表达式
2. 一元运算符
3. 求幂运算
4. 乘法和除法
5. 加法和减法

除了求幂运算以外，所有同一级别的运算符都是自左向右求值。同一级别的求幂运算符则自右向左求值。
例如：X + Y +Z 按照 (X + Y) + Z 进行求值，而 X ** Y ** Z 则按照 X ** (Y ** Z) 进行求值。

4.9.4.2 附加的算术运算规则

数字表达式基于这些附加规则求值：

- 如果任何一个操作数为空，则带有数字运算符的表达式的计算结果都将为空。
- 除以 0 时将出现错误。
- 仅可采用数字型数据执行求幂运算。如果第一个操作数为 0，则第二个操作数必须大于 0，并且结果将为 0。如果第二个操作数为 0，则第一个操作数不能为 0，并且结果将为 1。如果第一个操作数是负数，则第二个操作数必须是一个精确数字型数值、标度为零。
- 求幂运算存在舍入误差。求幂运算的结果应被视为近似值。

4.9.4.3 算术结果的精度、量级和标度

算术表达式求值期间将计算精度、量级和标度。精度是表达式里的最大位数。量级指的是小数点左侧的位数。标度指的是小数点右侧的位数。
例如：一个声明为 NUMERIC (18, 5)的列的精度为 18、量级为 13、标度为 5。
作为另一个例子，字面值 12345.6789 的精度为 9、量级为 5、标度为 4。
精确数字型数据的最大精度为 128 位，REAL 数据类型的最大精度约为 7 位十进制数字， DOUBLE PRECISION 数据类型的最大精度约为 16 位。
当 EsgynDB SQL 在一个表达式里遇到一个算术运算符时，它会应用这些规则（包含以下使用限制：如果精度大于 18，则所得精度将被设为 18，所得的标度为 0 和 (18- [所得的精度-所得的标度])的最大值。

- 如果运算符为+或-，则所得的标度为操作数标度的最大值，所得的精度为（操作数量级的最大值+结果的标度+1）。
- 如果运算符为*，则所得的标度为操作数标度的总和，所得的精度为所有操作数的量级和结果标度的总和。
- 如果运算符为/，则所得的标度为分子标度和分母量级的总和，所得的量级为分子量级和分母标度的总和。

例如：如果分子为 NUMERIC (7, 3)、分母为 NUMERIC (7, 5)，则所得的标度为

3+2（或5），所得的量级为4+5（或9）。表达式结果的数据类型为NUMERIC(14, 5)。

4.9.4.4 算术运算的数字类型转换

EsgynDB SQL 可以自动转换浮点数字类型（REAL 和 DOUBLE PRECISION）和其他数字类型。表达式里的所有数字值将被首先转换为二进制数字，并且具有求值所需的最大精度。

4.9.4.5 数字值表达式的例子

以下是数字值表达式的例子:

`-57`	数字字面值
`salary * 1.10`	一个 SALARY 列值和一个数字字面值的乘积
`unit_price * qty_ordered`	UNIT_PRICE 和 QTY_ORDERED 列值的乘积
`12 * (7 - 4)`	操作数为数字字面值的表达式
`COUNT (DISTINCT city)`	适用于一列数值的函数

4.10 标识符

SQL 标识符指的是标识表、视图、列和其他 SQL 实体所用的名称。存在两种类型的标识符：常规标识符和限定标识符。

限定标识符用双引号（"）括起来。不区分大小写字母的限定标识符只能用于用户名和角色名。常规标识符、限定标识符或大小写字母敏感性限定标识符最多可包含 128 个字符。

4.10.1 常规标识符

常规标识符以一个字母（A 到 Z、a 到 z）开头，也可以包含数字（0 到 9）或下划线（_）。常规标识符不区分大小写。

您不能将保留字用作常规标识符。

4.10.2 限定标识符

限定标识符是出现在双引号字符(")中的字符串，由字母数字字符组成，包括下划线(_)或破折号 (-)。与常规标识符不同，限定标识符需要区分大小写字母。

考虑到底层 HBase 文件系统的约束，EsgynDB SQL 不支持限定标识符包含空格

或特殊字符。您可以使用保留字作为限定标识符。

4.10.3 不区分大小写字母的限定标识符

用户名和角色所用的不区分大小写字母的分隔标识符是出现在双引号字符（"）内的字符串，它包含字母数字字符（A 到 Z、a 到 z)、数字（0 到 9)、下划线（_)、破折号（-)、句号（.)、@符号（@）和正斜杠（/)，但前导@符号（@）或前导正斜杠（/）字符除外。与其他限定标识符不同，这种限定标识符不区分大小写，并且在插入 SQL 元数据之前都会升档。因此，无论您指定一个用户名为"Penelope.Quan@company.com"、还是"penelope.quan@company.com"，元数据里存储的数值都将是 PENELOPE.QUAN@COMPANY.COM。

您可以将保留字用作不区分大小写的限定标识符。

4.10.4 标识符例子

- 以下是常规标识符:

  ```
  mytable
  SALES2006
  Employee_Benefits_Selections
  CUSTOMER_BILLING_INFORMATION
  ```

  由于常规标识符不区分大小写，所以 SQL 将这些标识符视为 mytable 的替代性表示:

  ```
  mytable MYTABLE   MyTable  mYtAbLe
  ```

- 以下是限定标识符:

  ```
  "mytable" "table"
  "CUSTOMER-BILLING-INFORMATION"
  ```

  由于分隔标识符区分大小写字母，所以 SQL 认为标识符"mytable"不同于"MYTABLE"或"MyTable"。

  您可以将保留字作为限定标识符. 例如：不得将 table 作为常规标识符，但可以将"table"作为限定标识符。

4.11 标识列

标识列是一个自动增长列，它被定义为具有标识属性的表的列，用于在每次向表中插入行时自动生成列的递增或递减顺序数值。

与独立于表列的序列不同，标识列被绑定到表列，并且只能由表列访问。更多信息，请参阅 CREATE SEQUENCE 语句。

```
GENERATED [ALWAYS | BY DEFAULT] AS IDENTITY
(
[START WITH integer]
[INCREMENT BY integer]
[MAXVALUE integer | NOMAXVALUE]
[MINVALUE integer]
[CYCLE | NO CYCLE]
[CACHE integer | NO CACHE]
[DATA TYPE]
)
```

4.11.1 标识列语法

- ALWAYS
  指示将行插入表中时，始终为该列生成一个值。

- BY DEFAULT
  指示当向表中插入行时，如果未指定列的值，则将为该列分配默认值。如果在列上放置唯一索引，则该值可以是唯一的。

上述的选项与创建序列时的作用相同。更多信息，请参阅 CREATE SEQUENCE 语句。

4.11.2 标识列例子

该例子显示了如何使用标识列自动生成 *employee_id* 列的递增顺序数值。

```
CREATE TABLE identity_employee
(
employee_id LARGEINT GENERATED ALWAYS AS IDENTITY
(
START WITH 1
INCREMENT BY 3
MAXVALUE 9999
MINVALUE 1
NO CYCLE
CACHE 30
),
```

```
employee_name VARCHAR(40)
);
--- SQL operation complete.

INSERT INTO identity_employee (employee_name)
VALUES
('Kevin'),
('Andy'),
('Eason'),
('Magic'),
('William'),
('Aven');
--- 6 row(s) inserted.


SELECT * FROM identity_employee;
EMPLOYEE_ID EMPLOYEE_NAME
-------------------- ------------------------------------
----
1 Kevin
4 Andy
7 Eason
10 Magic
13 William
16 Aven
--- 6 row(s) selected.
```

ALWAYS 例子。

```
CREATE TABLE identity_employee
(
id LARGEINT GENERATED ALWAYS AS IDENTITY,
description VARCHAR(40)
);
INSERT INTO identity_employee (description) VALUES ('Just
DESCRIPTION');
--- 1 row(s) inserted.


INSERT INTO identity_employee (id, description) VALUES
(NULL, 'ID=NULL and
DESCRIPTION');
*** ERROR[4122] NULL cannot be assigned to NOT NULL
column
TRAFODION.SEABASE.IDENTITY_EMPLOYEE.ID.


INSERT INTO identity_employee (id, description) VALUES
```

```
(999, 'ID=999 and
DESCRIPTION');
*** ERROR[3428] IDENTITY column ID defined as GENERATED
ALWAYS cannot accept values
specified by the user.
```

BY DEFAULT 例子。

```
CREATE TABLE identity_employee
(
id LARGEINT GENERATED BY DEFAULT AS IDENTITY,
description VARCHAR(40)
);

INSERT INTO identity_employee (description) VALUES ('Just
DESCRIPTION');
--- 1 row(s) inserted.

INSERT INTO identity_employee (id, description) VALUES
(NULL, 'ID=NULL and
DESCRIPTION');
*** ERROR[4122] NULL cannot be assigned to NOT NULL
column
TRAFODION.SEBASE.IDENTITY_EMPLOYEE.ID.


INSERT INTO identity_employee (id, description) VALUES
(999, 'ID=999 and
DESCRIPTION');
--- 1 row(s) inserted.
```

4.12 索引

索引是表内各行指针的有序集合。每个索引都基于一列或多列的数值而定，索引对于 DML 语法而言是透明的。

索引行和基表行之间始终存在一对一的对应关系。

4.12.1 SQL 索引

EsgynDB SQL 索引里的每一行都包含:

- CREATE INDEX 语句里指定的列
- 底层表的聚集键（用户定义的聚集键）

索引名是一种 SQL 标识符，一个 schema 里的所有索引都有自身的命名空间，因此索引名可能与表名或约束名相同。然而，一个 schema 里的不会有相同的索引名。更多信息，请参阅 CREATE INDEX 语句。

4.13 键

EsgynDB SQL 支持这些类型的键:

- 聚集键
- SYSKEY
- 索引键
- 主键

4.13.1 聚集键

每个表都有一个聚集键，以确定磁盘上的行序的列集。

EsgynDB SQL 使用基于该聚集键的 B 树组织表或索引的记录。因此，聚集键值可以作为逻辑行的 ID。

4.13.2 SYSKEY

如果 STORE BY 子句指定了 *key-column-list* 子句，则 *key-column-list* 将追加一个名为 SYSKEY 的附加列。

SYSKEY（或系统定义的聚集键）是由 EsgynDB SQL（而非用户）定义的聚集键列。它的类型是 LARGEINT SIGNED（有符号的大整型）。当您在一个表中插入一条记录时，EsgynDB SQL 将为 SYSKEY 列自动生成一个数值。您不能指定该值。

您既无法在插入时指定 SYSKEY，也无法在 SYSKEY 生成后对其进行更新。为了查看所生成的 SYSKEY 值，请在选择列表里加入 SYSKEY 列:

```
SELECT *, SYSKEY FROM t4;
```

4.13.3 索引键

索引行和基表行之间存在一对一的对应关系。EsgynDB SQL 索引的每一行都包含:

- CREATE INDEX 语句里指定的列
- 底层表的聚集（主）键（用户定义的聚集键）

一个非唯一索引的聚集键由上述两项组成，且大小不得超过 2048 字节。因为该聚集键包含了表里的所有列，所以每一行也仅限于 2048 字节。

对于可变长度字符列而言，这些字节限制里提到的长度为定义的列长度，而非存储长度。存储长度为扩展长度，其包含两个额外字节，用于存储一个项目的

数据长度。更多信息，请参阅 CREATE INDEX 语句。

4.13.4 主键

主键指的是用于定义数据库表唯一性约束的一列或多列。这些列里不得包含空值，并且表里只能存在一个主键约束。

4.14 字面值

字面值是在表达式中、语句中、或作为参数值使用的常数。

SQL 字面值可以是以下数据类型之一：

字符串字面值	用单引号括起来的一连串字符。 例子：'Planning'
日期时间字面值	以关键字 DATE、TIME 或 TIMESTAMP 开头，后面紧跟一个字符串。 例子：DATE '2020-01-22'
时间间隔字面值	以关键字 INTERVAL 开头，后面紧跟一个字符串和一个时间间隔限定符。 例子：INTERVAL '2-7' YEAR TO MONTH
数字字面值	一个简单的数字字面值（不含指数）或科学计数法里的数字字面值。 例子：99E-2

4.14.1 字符串字面值

字符串字面值指的是用单引号括起来的一串字符。

您既可指定一个字符串、也可指定一组代表字符串里字符的十六进制代码值。

```
'string' [_character-set | N]'string'
| [_character-set | N] X'hex-code-value... '
| [_character-set | N] X'[space...]hex-code-value[[space...]hex-code-value...][space...]' character-set
```

指定字符集为 ISO88591 或 UTF8。

字符串字面值的_character-set 规范应与列定义的字符集- ISO88591 或 UTF8 相符。

如果您省略_character-set 规范，并且字符串字面值完全由 7 位 ASCII 字符组成，则 EsgynDB SQL 最初会假设字符集为 ISO88591，否则将假设字符集为 UTF8。

N

将字符串字面值与国家字符（NCHAR）数据类型的字符集关联起来。

安装 EsgynDB SQL 期间可以确定 NCHAR 的字符集。该值既可以是 UTF8（默认值），也可以是 ISO88591。

'string'

这是括在单引号里的一连串输入字符。

字符串中的单引号通过两个单引号（''）表示。如果您指定两个中间没有空格的单引号（''），则字符串的长度可以为零。

X

表示十六进制字符串。

'hex-code-value'

代表括在单引号里的一个 16 进制字符的代码值。 它必须包含偶数个 16 进制位。ISO88591 的每个数值都必须有两位数。UTF8 的每个数值都必须有 2、4、6 或 8 个 16 进制位。如果 hex-code-value 的格式不正确，它将包含一个无效的 16 进制位或奇数个 16 进制位，因此将返回一个错误。

space

指空格序列，它们可以添加到 16 进制代码值的前面或后面，用于提高可读性。交互式接口和编程接口的 SQL 模块字符集所用的空格编码必须为 TERMINAL_CHARSET。

4.14.1.1 字符串字面值注意事项

使用字符串字面值

一个字符串字面值可以与字符列具有相等的长度。更多信息，请参阅字符串数

据类型。

您也可以在字符串值表达式（例如：使用串联运算符(||)的表达式或使用可以返回字符串值的函数）里使用字符串字面值。

指定字符串字面值时：

- 不得在字符集限定符和字符串字面值之间置入空格。如果您在一条语句里使用一个字符串字面值，EsgynDB SQL 将返回一个错误。
- 在字符串字面值中指定单引号时，请使用两个连续的单引号。
- 指定一个长度超过一行的字符串字面值时，请将该字符串字面值分隔成几个较小的字面值，并使用串联运算符(||)将它们串联起来。
- 字符串字面值区分大小写。小写字母并不等效于相应的大写字母。
- 一个字符串字面值中的前导空格和尾随空格都是有效的。
- 另外，长度超过一行的字符串可被写成一个带有后随空格、CR 或制表符的字面值，后面紧跟另一个字符串字面值。

4.14.1.2 字符串字面值例子

- 通过可存储在列里的字面值例子显示这些列数据类型的规范。

字符串数据类型	字符串字面值的例子
CHAR (12) UPSHIFT	'PLANNING'
VARCHAR (18)	'NEW YORK'

- 以下是字符串字面值：

```
'This is a string literal.' 'abc^&*'
'1234.56'
'This literal contains '' a single quotation mark.'
```

- 以下是一个通过三行串联的字符串字面值：

```
'This literal is' ||
' in three parts,' ||
'specified over three lines.'
```

- 以下是一个 16 进制字符串字面值，它代表 ISO88591 字符串'Strauß'的 VARCHAR 样式：

```
_ISO88591 X'53 74 72 61 75 DF'
```

4.14.2 日期时间字面值

日期时间字面值是可以在表达式中、语句中、或作为参数值使用的 DATE、TIME 或 TIMESTAMP 常数。

日期时间字面值与相应的日期时间数据类型具有相同的有效值范围。

不能在一个日期时间字符串里（单引号内）使用前导空格或尾随空格。

日期时间字面值以 DATE、TIME 或 TIMESTAMP 关键字开头，默认情况下以美国或欧洲格式出现:

```
DATE 'date' | TIME 'time' | TIMESTAMP 'timestamp'

date is:
   yyyy-mm-dd Default
   | mm/dd/yyyy   USA
   | dd.mm.yyyy   European
time is:
   hh:mm:ss.mssssss Default
   | hh:mm:ss.mssssss [am | pm]USA
   | hh.mm.ss.mssssss  European

timestamp is:
   yyyy-mm-dd hh:mm:ss.mssssss Default
   | mm/dd/yyyy hh:mm:ss.mssssss [am | pm]USA
   | dd.mm.yyyy hh.mm.ss.mssssss European

date,time,timestamp
```

指定日期时间字面值字符串，其分量字段为:

yyyy	Year，0001 到 9999
mm	Month，01 到 12
dd	Day，01 到 31
hh	Hour，00 到 23
mm	Minute, 00 到 59
ss	Second，00 到 59
msssss	Microsecond，000000 到 999999
am	AM 或 am，表示午夜到中午前的一段时间
pm	PM 或 pm，表示中午到午夜前的一段时间

日期时间字面值例子

- 以下是默认的 DATE 字面值，它们分别采用美国和欧洲格式：
  DATE '2008-01-22' DATE '01/22/2008' DATE '22.01.2008'

- 以下是默认的 TIME 字面值，它们分别采用美国和欧洲格式：
  TIME '13:40:05'
  TIME '01:40:05 PM'
  TIME '13.40.05'

- 以下是默认的 TIMESTAMP 字面值，它们分别采用美国和欧洲格式：
  TIMESTAMP '2008-01-22 13:40:05'
  TIMESTAMP '01/22/2008 01:40:05 PM'
  TIMESTAMP '22.01.2008 13.40.05'

4.14.3 时间间隔字面值

时间间隔字面值是INTERVAL数据类型的一个常数，它可以将正或负持续时间表示为year-month（年-月型）或day-time（日-时间型）时间间隔；它以关键字 INTERVAL 开头，前面或后面可以带有负号（用于负持续时间）。您不能在一个时间间隔字符串里（单引号内）包含前导或尾随空格。

```
[-]INTERVAL [-]{'year-month' | 'day:time'} interval-
qualifier

year-month is:
     years [-months] | months

day:time is:
     days [[:]hours [:minutes [:seconds [.fraction]]]]
     | hours [:minutes [:seconds [.fraction]]]
     | minutes [:seconds [.fraction]]
     | seconds [.fraction]

interval-qualifier is:
     start-field TO end-field | single-field

start-field is:
     {YEAR | MONTH | DAY | HOUR | MINUTE} [(leading-
     precision)]

end-field is:
     YEAR | MONTH | DAY | HOUR | MINUTE | SECOND
```

```
      [(fractional-precision)]

single-field is:
      start-field        |        SECOND        [(leading-
      precision,fractional-precision)]
```

start-field TO end-field

必须为 year-month（年-月型）或 day-time（日-时间型）。您在字段名称列表里指定的 start-field 必须在 end-field 前面。

{YEAR | MONTH | DAY | HOUR | MINUTE} [(leading-precision)]

指定 start-field。

start-field 的前导精度可以多达 18 位（最大值取决于时间间隔里的字段数量）。前导精度指的是 start-field 里允许出现的位数，默认值为 2。

YEAR | MONTH | DAY | HOUR | MINUTE | SECOND [(fractional-precision)]

指定 end-field。

如果 end-field 为 SECOND，则其小数部分的精度可以多达 6 位。

小数部分的精度指的是小数点后的精度位数，默认值为 6。

start-field | SECOND [(leading-precision, fractional-precision)]

指定单个字段。

如果单个字段为 SECOND，则前导精度为小数点前的精度位数，小数部分的精度则为小数点后的精度位数。前导精度的默认值为 2，小数部分精度的默认值为 6；前导精度的最大值为 18，小数部分精度的最大值为 6。

'year-month' | 'day:time'

指定一个时间间隔字面值的日期和时间部分。

日和小时字段可以用空格或冒号分隔。

时间间隔字面值字符串包括:

years	这是一个无符号的整数，它指定年数，并具有多达 18 位或 16 位，（如果 months 是终止字段）。YEAR(18)或 YEAR(16) TO MONTH 可以在时间间隔限定符内指定前导精度的最大值。
months	这是一个无符号的整数，它指定月数。如果被用作起始字段，则 months 可以具有多达 18 位。前导精度的最大值由 MONTH(18)指定。如果被用作终止字段，则 months 范围是 0 到 11。
days	这是一个无符号的整数，它指定天数。如果不存在终止字段，则 days 可多达 18 位。如果 hours 是终止字段的话，则 days 可以具有多达 16 位。如果 minutes 是终止字段的话，则 days 可以具有多达 14 位。如果 seconds 是终止字段，则 days 可以具有多达 13-f 位，这里的 f 是一个小于或等于 6 的分数。这些最大值可通过 DAY(18)、DAY(16) TO HOUR、DAY(14) TO MINUTE 和 DAY(13-f) TO SECOND(f)指定。
hours	这是一个无符号的整数，它指定小时数。如果被用作起始字段，并且终止字段不存在时，hours 可以具有多达 18 位。如果 minutes 是终止字段，则 hours 可以具有多达 16 位。如果 seconds 是终止字段，则 hours 可以具有多达 14-f 位，这里的 f 是一个小于或等于 6 的分数。这些最大值可通过 HOUR(18)、HOUR(16) TO MINUTE 和 HOUR(14-f) TO SECOND(f)指定。如果被用作终止字段，则 hours 值的范围是 0 到 23。
minutes	这是一个无符号的整数，它指定分钟数。如果被用作起始字段，并且终止字段不存在时，minutes 可以具有多达 18 位。如果 seconds 是终止字段，则 minutes 可以具有多达 16-f 位，这里的 f 是一个小于或等于 6 的分数。这些最大值可通过 MINUTE(18)和 MINUTE(16-f) TO SECOND(f)指定。如果被用作终止字段，则 minutes 值的范围是 0 到 59。
seconds	这是一个无符号的整数，它可以指定秒数。如果被用作起始字段，则 seconds 可以具有多达 18 位，减去（一个小于或等于 6 的分数的）位数 f。该最大值可通过 SECOND(18-f, f)指定。seconds 值的范围是 0 到 59.9(n)，这里的 n 是 seconds 精度的指定位数。
fraction	这是一个无符号的整数，它可以指定几分之一秒。如果 seconds 被用作终止字段，fraction 将紧跟在 SECOND 关键字后面，并且

	位数限于 fractional-precision（小数部分的精度）指定的位数。

4.14.3.1 时间间隔字面值的注意事项

- **Year-Month（年-月型）和 Day-Time（日-时间型）字符串的长度**
  字符串里的一个时间间隔字面值可以包含多达 18 位，它紧跟在 INTERVAL 关键字后面，并带有一个连字符 plus（-），以分隔 year-month（年-月型）字段；以及冒号（:），以分隔 day-time（日-时间型）字段。此外，您还可以使用空格分隔日和小时。

- **时间间隔字面值的例子**

INTERVAL '1' MONTH	时间间隔为 1 个月
INTERVAL '7' DAY	时间间隔为 7 天
INTERVAL '2-7' YEAR TO MONTH	时间间隔为 2 年零 7 个月
INTERVAL '5:2:15:36.33' DAY TO SECOND(2)	时间间隔为 5 天零 2 小时 15 分钟 36.33 秒
INTERVAL - '5' DAY	减去 5 天后所剩的时间间隔
INTERVAL '100' DAY(3)	时间间隔为 100 天。该例子所需的显式前导精度为 3，因为默认值为 2
INTERVAL '364 23' DAY(3) TO HOUR	时间间隔为 364 天零 23 小时。day 和 hour 字段的分隔符可以是空格或冒号

4.14.4 数字字面值

数字字面值代表数字值。它既可以表示为一个精确数字字面值（不带指数），也可以通过使用科学记数法表示为一个近似数字字面值（带指数）。

```
exact-numeric-literal is:
  [+|-]unsigned-integer[.[unsigned-integer]]
| [+|-].unsigned-integer

approximate-numeric-literal is:
  mantissa{E|e}exponent

mantissa is:
  exact-numeric-literal

exponent is:
  [+|-]unsigned-integer

unsigned-integer is:
   digit. . .
```

- `exact-numeric-literal`

  这是一个精确数字值，包括一个可选的加号（+）或减号（-），多达 128 个数字（0 到 9）和一个代表小数点的可选句号（.）。

  前导零将不计入 128 个限位里，尾随零则将计入 128 个限位里。

  无符号的数字字面值是一个正数，不含小数点的精确数字字面值则是一个整数。每个精确数字字面值都具 NUMERIC 数据类型和表示该数值所需的最小精度。

- `approximate-numeric-literal`

  这是一个精确数字字面值，后面跟有采用大写字母 E 或小写字母 e 表示的指数、以及可选的有符号整数。

  如果指数前面的数字不超过 7 个，以科学计数法表示的数字值将被当作 REAL 数据类型。但如果包含 8 个或更多数字，它们将被当作 DOUBLE PRECISION 类型。

  正因如此，小数点后的尾随零有时可以提高数字字面值（用作 DOUBLE PRECISION 值）的精度。

  例如：如果 XYZ 表包含一个 DOUBLE PRECISION 列，则

  ```
  INSERT INTO XYZ VALUES (1.00000000E-10)
  ```

  插入值比

  ```
  INSERT INTO XYZ VALUES (1.0E-10)
  ```

  插入值的精度更高。

数字字面值例子

以下是数字字面值和它们的显示格式：

字面值	显示格式
477	477
580.45	580.45
+005	5
-.3175	-.3175
1300000000	1300000000
99.	99
-0.123456789012345678	-.123456789012345678
99E-2	9.9000000E-001
12.3e+5	1.2299999E+006

4.15 Null

Null（空值）是一种与数据类型无关的特殊符号，它代表一个未知数。

EsgynDB SQL 关键字 NULL 代表空值。Null 表示某项没有数值。

排序时，空值大于所有其他值。您不得使用 INSERT 或 UPDATE 语句将空值存储在一列里，除非该列允许出现空值。一个允许出现空值的列在任何行都可以为空。一个可空列在每一行里都有关联的额外字节。

存储在这些字节里的特殊值表明该列带有适用于该行的空值。

4.15.1 空值和默认值的使用情况对比

一个表行的特定列不含数值可能因为各种原因所致。 例如：

- 如果联系人未提供年龄，电话营销联系人数据库可能带有 AGE 空值字段。
- 一个订单记录可能带有空的 DATE_SHIPPED 列，直到订货被实际装运为止。
- 一位国际雇员的雇员记录可能不含社会安全号码。 您允许一列出现空值，如果您希望表明该列（例如：电话营销联系人的年龄）的值是未知的或不适用的（例如：一位国际雇员的社会安全号）。

在决定是否允许出现空值或使用默认值时，还应注意：

- 空值不同于空格。 两个空格可以进行等值比较，而两个空值的比较结果是不确定的。
- 空值与零不同。零可以参与算术运算，而空值则无法参与算术运算。

4.15.2 定义允许或禁止出现空值的列

CREATE TABLE 和 ALTER TABLE 语句可以定义表里各列的属性。一个列都允许出现空值，除非列定义包含 NOT NULL 子句或该列是表主键的一部分。

Null（而非 NOT NULL）是一个列的默认值，除非列定义包含一条 DEFAULT 子句（而非 DEFAULT NULL）或 NO DEFAULT 子句。当 INSERT 语句省略特定列的一个值时，一个列的默认值是 EsgynDB SQL 插入一个行里的值。

4.15.2.1 Null in DISTINCT、GROUP BY 和 ORDER BY 子句里的空值

在对 DISTINCT、GROUP BY 和 ORDER BY 子句求值时，EsgynDB SQL 会将所有空值视为相等。

使用这些子句的其他注意事项如下：

DISTINCT：空值被视为重复值；因此，执行结果最多含有一个空值。

GROUP BY：执行结果最多含有一个空值组。

ORDER BY：空值被视为大于非空值。

4.15.2.2 空值和表达式求值的比较

表达式类型	条件	结果
布尔运算符（AND、OR 和 NOT）	操作数为空。	对于 AND 而言，执行结果为空。对于 OR 而言，如果另一个操作数为真，则执行结果为真。如果另一个操作数为空或为假，则执行结果为空。对于 NOT 而言，执行结果为空。
算术运算符	一个操作数为空或两个操作数均为空。	执行结果为空值。
NULL 谓词	该操作数为空。	执行结果为真。
聚合（或集合）函数（COUNT 除外）	一些行带有空值列。消除空值后对该函数进行求值。	如果集合是空的，则执行结果为空值。
COUNT(*)	该函数没有消除空值。	执行结果为表里的行数，不管这些行是否为空。
COUNT COUNT DISTINCT	消除空值后对该函数进行求值。	如果集合是空的，则执行结果为零。
比较谓词：=, <>, <, >, <=, >=, LIKE	操作数为空。	执行结果为空。
IN 谓词	IN 数值列表里的一些表达式为空。	如果所有表达式都为空，则执行结果为空。
子查询	没有返回任何一行。	执行结果为空。

4.15.3 Null 对 Oracle 的兼容支持

4.15.3.1 支持 Null 作为四则运算的算子

Null 和任何数进行加/减/乘/除运算，结果都是 Null。该功能受 CQD MODE_COMPATIBLE_1 控制，只有 MODE_COMPATIBLE_1 被设为 ON 的时候才生效。

例子：

```
SQL>select NULL * 1 from dual;

(EXPR)
--------------------
                NULL

--- 1 row(s) selected.

SQL>select NULL + 1 from dual;

(EXPR)
--------------------
                NULL

--- 1 row(s) selected.

SQL>select NULL - 1 from dual;

(EXPR)
--------------------
                NULL

--- 1 row(s) selected.

SQL>select NULL/2 from dual;

(EXPR)
--------------------
                NULL

--- 1 row(s) selected.

SQL>select -NULL from dual;

(EXPR)
--------------------
                NULL

--- 1 row(s) selected.
```

4.15.3.2 支持 Null 作为逻辑运算的算子

支持 Null 作为逻辑运算符的算子，Null 和任何算子比较，结果都是 FALSE。该功能受 CQD MODE_COMPATIBLE_1 控制，只有 MODE_COMPATIBLE_1 被设为 ON 的时候才生效。

例子:

```
SQL>select 1 from dual where 1 = NULL;

--- 0 row(s) selected.

SQL>select 1 from dual where NULL = NULL;

--- 0 row(s) selected.

SQL>select 1 from dual where 1 > NULL;

--- 0 row(s) selected.

SQL>select 1 from dual where 1 < NULL;

--- 0 row(s) selected.

SQL>select 1 from dual where 1 <> NULL;

--- 0 row(s) selected.

SQL>select 1 from dual where '1' > NULL;

--- 0 row(s) selected.
```

4.16 谓词

一个谓词可以确定一个值或一组值的相关问题的答案。一个谓词可以返回 true（真）或 false（假），如果该问题无法回答，则返回 unknown（未知）。

在搜索条件建立使用这些谓词从表或视图里选择行。

BETWEEN 谓词	确定一个数值序列是否在数值序列范围内。
比较谓词（=, <>, <, >, <=, >=)	比较表达式序列的值或比较行值序列的值（行子查询的结果）。
EXISTS 谓词	确定是否可以通过子查询选择任何行。如果子查询发现至少一行满足其搜索条件，则谓词求值结果为 true。否则，如果子查询的结果表为空，则谓词为 false。
IN 谓词	确定一个数值序列是否等于一个序列列表里的任何数值序列。
LIKE 谓词	搜索匹配一种样式的字符串。
NULL 谓词	确定一个数值序列里的所有值是否都为空。
量化比较谓词（ALL、ANY、SOME）	对比表达式序列的值和通过表子查询选出的每一行里的数值。这种比较可通过 ALL、SOME 或 ANY 实现 量化。

更多信息，请参阅谓词。

4.16.1 BETWEEN 谓词

4.16.1.1 BETWEEN 谓词语法

该谓词确定一个数值序列是否在数值序列范围内。

```
row-value-constructor [NOT] BETWEEN
   row-value-constructor AND row-value-constructor

row-value-constructor is:
   (expression [,expression]...)
   | row-subquery
```

row-value-constructor

指定 BETWEEN 谓词的一个操作数。三个操作数可以是以下构造函数之一:

- *(expression [,expression]...)*
  这是一种 SQL 值表达式，以逗号分隔、并用括号括起来。expression 不能包含一个聚合函数，除非它位于一个 HAVING 子句中。expression 可以是一种标量子查询（即返回单行单列的子查询）。更多信息，请参阅表达式。

- *row-subquery*
  返回单行（包含一个数值序列）的子查询。更多信息，请参阅子查询 。

  BETWEEN 谓词指定的这三个 *row-value-constructors* 都必须包含相同数量的元素。即每个列表里的数值表达式的数量或一条行子查询所返回的数值的数量都必须相同。
  三个行值构造函数里各自数值的数据类型都必须可以比较。各自数值指的是两个列表里具有相同序号位置的数值。更多信息，请参阅可比和兼容的数据类型。

4.16.1.2 BETWEEN 谓词注意事项

- 使用 AND 和 OR 的逻辑等式
  谓词 *expr1* BETWEEN *expr2* AND *expr3* 为真，当且仅当该条件为真时:
  expr2 <= expr1 AND expr1 <= expr3

  谓词 *expr1* NOT BETWEEN *expr2* AND *expr3* 为真，当且仅当该条件

为真时:

expr2 > expr1 OR expr1 > expr3

- 按键降序列

  如果子句指定一个键列值介于 *expr2* 和 *expr3* 之间，则 *expr3* 必须大 *expr2*，即使表定义中已将该列指定为 DESCENDING（降序排列）。

4.16.1.3 BETWEEN 例子

- 该谓词为真，如果库存单元的总价格介于 1000 和 10000 美元之间:

```
qty_on_hand * price
BETWEEN 1000.00 AND 10000.00
```

- 该谓词为真，如果部件价格小于 5 美元或大于 800 美元:

```
partcost NOT BETWEEN 5.00 AND 800.00
```

- 该 BETWEEN 谓词可以选择部件号为 6400 的行:

```
SELECT * FROM partsupp
WHERE partnum BETWEEN 6400 AND 6700
AND partcost > 300.00;

 Part/Num Supp/Num Part/Cost Qty/Rec
-------- -------- ------------ ----------
6400            1 390.00          50
6401            2 500.00          20
6401            3 480.00          38
--- 3 row(s) selected.
```

- 查找 Jody Selby 和 Gene Wright 之间的人名:

```
(last_name, first_name) BETWEEN
('SELBY', 'JODY') AND ('WRIGHT', 'GENE')
The name Barbara Swift would meet the criteria; the name
Mike Wright would not.
SELECT empnum, first_name, last_name
FROM persnl.employee
WHERE (last_name, first_name) BETWEEN
   ('SELBY', 'JODY') AND ('WRIGHT', 'GENE');

EMPNUM FIRST_NAME      LAST_NAME
------ --------------- --------------------
```

```
43        PAUL              WINTER
72        GLENN             THOMAS
74        JOHN              WALKER
...
--- 15 row(s) selected.
```

4.16.2 比较谓词

该谓词比较表达式序列的数值或行值序列的数值（行子查询的结果）。

```
row-value-constructor comparison-op row-value-constructor

comparison-op is:
   = Equal
   | <> Not equal
   | < Less than
   | > Greater than
   | <= Less than or equal to
   | >= Greater than or equal to
row-value-constructor is:
   (expression [,expression]...)
   | row-subquery
```

row-value-constructor

指定一个比较谓词的操作数。两个操作数可以是以下构造函数之一：

- *(expression [,expression]...)*

  这是 SQL 值表达式，它以逗号分隔、并用括号括起来。

  expression 不能包含一个聚合函数，除非它位于一个 HAVING 子句里。

  expression 是一条标量子查询（返回单行单列的子查询）。更多信息，请参阅表达式。

- *row-subquery*

  返回单行（包含一个数值序列）的子查询。更多信息，请参阅子查询。

  这两个 *row-value-constructors* 必须包含相同数量的元素。即每个列表里的数值表达式的数量或一条行子查询所返回的数值的数量都必须相同。

  两个 *row-value-constructors* 里各自数值的数据类型都必须可以进行比较。各自数值指的是两个列表里具有相同序号位置的数值。更多信息，请参阅可比和兼容的数据类型。

4.16.2.1 比较谓词注意事项

- 一个比较谓词何时为真

  通过顺次比较相应位置的数值，直到发现首个不相等的对，EsgynDB SQL 可以确定一种关系为真还是为假。

  如果一个行值构造函数里的数值表达式为空，则不能在 WHERE 或 HAVING 子句里使用一个比较谓词（而应使用 IS NULL 谓词）去比较行值构造函数。

  假设存在带有多个分量的两行 X 和 Y:

  `X=(X1,X2,...,Xn), Y=(Y1,Y2,...,Yn).`

  如果对于所有 i=1,...,n 都存在 Xi=Yi，则谓词 X=Y 为真。对于该谓词，EsgynDB SQL 必须必须检查所有值。

  如果对于某些 i 存在 Xi<>Yi，则谓词 X=Y 为假。当 SQL 找到不相等的分量时，它会停下来，并且不会继续查看剩余的分量。

  如果 X=Y 为假，则谓词 X<>Y 为真。

  如果 X1<>Y1, 则 EsgynDB SQL 不会查看所有分量，它会停下来为 X=Y 谓词返回假值，为 X<>Y 谓词返回真值。

  如果 X=Y 为真或对于所有 i=1,...,n 都存在 Xi=Yi，则 X<>Y 为假。在这种情况下，EsgynDB SQL 必须检查所有分量。

  如果对于某些索引 m 存在 Xm>Ym，并且对于所有 i=1,..,m-1 都存在 Xi=Yi，则谓词 X>Y 为真。EsgynDB SQL 不会检查所有分量。当它找到第一个不相等分量（Xm<>Ym）时会停下来。如果 Xm>Ym，则该谓词为真；否则，该谓词为假。另外，如果所有分量都相等或 X=Y，则该谓词也为假。

  如果 X>Y 为真或 X=Y 为真，则谓词 X>=Y 为真。在这种情况下，EsgynDB SQL 可能需要检查所有分量，

  并返回“真”（如果它们都相等）。它将在第一个不相等分量处（Xm<>Ym）停下来。如果 Xm>Ym，则该谓词为真；否则，该谓词为假。

  如果对于某些索引 m 存在 Xm<Ym，并且对于所有 i=1,..,m-1 都存在 Xi=Yi，则谓词 X<Y 为真。EsgynDB SQL 不会仔细检查所有分量。当它找到第一个不相等分量（Xm<>Ym）时会停下来。如果 Xm<Ym，则该谓词为真；否则，该谓词为假。另外，如果所有分量都相等或 X=Y，则该谓词也为假。

  如果 X<Y 为真或 X=Y 为真，则谓词 X<=Y 为真。在这种情况下，EsgynDB SQL 可能需要检查所有分量，并返回“真”（如果它们都相等）。它将在第一个不相等分量处（Xm<>Ym）停下来。如果 Xm<Ym，则该谓词为真；否则，该谓词为假。

- 比较字符数据

  为了比较具有不同长度的两个字符串，应在较短字符串的右侧填充空格（HEX 20），直到它达到较长字符串的长度为止。固定长度和可变长度字符串都使用这种方式填充。

  例如：EsgynDB SQL 认为字符串'JOE'等于 CHAR 或 VARCHAR 数据类型列里存储的数值 JOE（宽度大于或等于 3）。

  同样，EsgynDB SQL 认为任何 CHAR 数据类型列里存储的数值 JOE 都等于任何 VARCHAR 数据类型列里存储的数值 JOE。

  相同序号位置的所有字符都相等的情况下，两个字符串才相等。这里需要区分大小写字母。

- 比较数字型数据

  求值之前，一个表达式里的所有数字值都将按照表达式各处所需的最大精度进行转换。

- 比较时间间隔数据

  为了比较 INTERVAL 值，EsgynDB SQL 将首先把时间间隔转换成一种通用单位。

  如果不存在通用单位，EsgynDB SQL 会报告一个错误。两个 INTERVAL 值必须都是 year-month（年-月型）时间间隔或 day-time（日-时间型）时间间隔。

- 比较多值

  请尽量使用多值谓词，在同等条件下，通常它们比其他非多值谓词更高效。

4.16.2.2 比较谓词例子

- 如果客户号码为 3210，则该谓词为真：

  ```
  custnum = 3210
  ```

- 如果工资大于所有雇员的平均工资，则该谓词为真：

  ```
  salary > (SELECT AVG (salary) FROM persnl.employee);
  ```

- 如果客户名是 BACIGALUPI，则该谓词为真：

  ```
  custname = 'BACIGALUPI'
  ```

- 对于包含 CUSTNUM 空值列的 CUSTOMER 或 ORDERS 表的任何一行而言，

该谓词的计算结果都是“未知”：

```
customer.custnum > orders.custnum
```

- 该谓词返回一个按姓氏的字母顺序排列（对于相同的姓氏，则按名的字母顺序排列）的列表里名字排在 MOSS.DUNCAN 后面的任何人员的相关信息：

```
(last_name, first_name) > ('MOSS', 'DUNCAN')
```

REEVES.ANNE 符合该标准，但 MOSS.ANNE 则不符合该标准。

这种多值谓词在该条件下等效于三个比较谓词：

```
(last_name > 'MOSS') OR
(last_name = 'MOSS' AND first_name > 'DUNCAN')
```

- 比较两个日期时间值 START_DATE 和 CURRENT_DATE 函数的执行结果

```
START_DATE < CURRENT_DATE
```

- 比较两个日期时间值 START_DATE 和 SHIP_TIMESTAMP:

```
CAST (start_date AS TIMESTAMP) < ship_timestamp
```

- 比较两个 INTERVAL 值：

```
JOB1_TIME < JOB2_TIME
```

假设被定义为 INTERVAL DAY TO MINUTE 的 JOB1_TIME 是两天零 3 小时，被定义为 INTERVAL DAY TO HOUR 的 JOB2_TIME 则为 3 天。

为了对该谓词进行求值，EsgynDB SQL 会将两个 INTERVAL 值转换为 MINUTE。该比较谓词为真。

- 以下例子在一个比较谓词里包含了一个子查询。每条子查询都运行在 EMPLOYEE 表单独的逻辑副本上。

内部查询先求值，外部查询基于内部查询求值的结果，再进行求值。

查找所有工资高于部门 1500 最高工资的员工：

```
SELECT first_name, last_name, deptnum, salary
       FROM persnl.employee
       WHERE salary > (SELECT MAX (salary)
                       FROM persnl.employee
                       WHERE deptnum = 1500);
```

```
  FIRST_NAME LAST_NAME DEPTNUM SALARY
```

```
---------- --------- ------- -----------
ROGER      GREEN     9000    175500.00
KATHRYN    HALL      4000    96000.00
RACHEL     MCKAY     4000    118000.00
THOMAS     RUDLOFF   2000    138000.40
JANE       RAYMOND   3000    136000.00
JERRY      HOWARD    1000    137000.10
--- 6 row(s) selected.
```

查找所有来自其他部门的工资低于最低工资（非部门 1500）的员工，此处的平均工资比部门 1500 的平均工资高：

```
SELECT first_name, last_name, deptnum, salary
   FROM persnl.employee
   WHERE deptnum <> 1500 AND
        salary < (SELECT MIN (salary)
             FROM persnl.employee
             WHERE deptnum <> 1500 AND
                  salary > (SELECT AVG (salary)
                            FROM persnl.employee
                            WHERE deptnum = 1500));

   FIRST_NAME LAST_NAME DEPTNUM SALARY
   ---------- --------- ------- -----------
   JESSICA    CRINER    3500    39500.00
   ALAN       TERRY     3000    39500.00
   DINAH      CLARK     9000    37000.00
   BILL       WINN      2000    32000.00
   MIRIAM     KING      2500    18000.00
   ...
   --- 35 row(s) selected.
```

该查询的第一个子查询确定工资高于部门 1500 平均工资的其他部门员工的最低工资。主查询找到不属于部门 1500、并且工资低于第二个子查询所确定的最低工资的员工。

4.16.3 EXISTS 谓词

4.16.3.1 EXISTS 谓词语法

该谓词确定任何行是否满足一个子查询。如果子查询发现至少一行满足其搜索条件，则该谓词的求值结果为真。否则，如果子查询的结果表是空的，则该谓词的求值结果为假。

```
[NOT] EXISTS subquery
```

Subquery
指定该谓词的操作数。*subquery* 是用括号括起来的查询表达式。
EXISTS-*subquery* 通常与外部查询相关联。

4.16.3.2 EXISTS 谓词例子

- 查找职位代码为 300 的员工所处的位置:

```
SELECT deptnum, location FROM persnl.dept D
WHERE EXISTS
   (SELECT jobcode FROM persnl.employee E
   WHERE D.deptnum = E.deptnum AND jobcode = 300);

   DEPTNUM LOCATION
   ------- -------------
   3000 NEW YORK
   3100 TORONTO
   3200 FRANKFURT
   3300 LONDON
   3500 HONG KONG

   --- 5 row(s) selected.
```

在前述例子里，EXISTS 谓词包含一个 subquery，它确定职位代码为 300 的员工所处的位置。该 subquery 取决于外部查询的 D.DEPTNUM 的数值，并针对 D.DEPTNUM 和 E.DEPTNUM 相等条件下的结果表的每一行进行求值。D.DEPTNUM 列是一个外部引用的例子。

- 搜索不含职位代码为 420 的员工的部门:

```
SELECT deptname FROM persnl.dept D
WHERE NOT EXISTS
   (SELECT jobcode FROM persnl.employee E
   WHERE D.deptnum = E.deptnum AND jobcode = 420);

DEPTNAME
------------
FINANCE
PERSONNEL
INVENTORY
...
```

```
--- 11 row(s) selected.
```

- 搜索库存单元少于 20 个的部件:

```
SELECT partnum, suppnum
FROM invent.partsupp PS
WHERE EXISTS
   (SELECT partnum FROM invent.partloc PL
   WHERE PS.partnum = PL.partnum AND qty_on_hand < 20);
```

```
PARTNUM SUPPNUM
------- -------
212       1
212       3
2001      1
2003      2
...
--- 18 row(s) selected.
```

4.16.4 IN 谓词

4.16.4.1 IN 谓词语法

该谓词确定一个数值序列是否等于序列列表里的任何数值序列。NOT 运算符可以反转其真值。

例如：如果 IN 为真，则 NOT IN 为假。

```
row-value-constructor
     [NOT] IN {table-subquery| in-value-list}

row-value-constructor is:
     (expression [,expression]...)
     | row-subquery

in-value-list is:
     (expression [,expression]...)
```

row-value-constructor

指定 IN 谓词的第一个操作数。第一个操作数可以是以下之一:

- *(expression [,expression]...)*

  这是一种 SQL 值表达式序列，它以逗号分隔、并括在括号里。

  该 *expression* 不得包含一个聚合函数，除非它位于一个 HAVING 子句中。

  该 *expression* 可以是一种标量子查询（一种返回单行单列的子查询）。

  更多信息，请参阅表达式。

- *row-subquery*
  这是一种返回单行（包含一个数值序列）的子查询。

table-subquery
这类子查询以表的形式返回结果（由行和列构成）。
该表将指定各行中的各项值与 *row-value-constructor* 指定的行中的个值进行比较。*row-value-constructor* 的数值数量必须等于表子查询的结果表里的列数，并且这些数值的数据类型必须可以进行比较。

in-value-list
这是一种 SQL 值表达式序列，它以逗号分隔、并括在括号里。
该表达式不得包含按列定义的聚合函数。该 *expression* 可以是一个标量子查询（一个返回单行单列的子查询）。在这种情况下，*row-value-constructor* 的执行结果是一个单值。这些数值的数据类型必须可以进行比较。*in-value-list* 里表达式的数量至少有 5000 个。

4.16.4.2 IN 谓词注意事项

- 使用 ANY（或 SOME）的逻辑等式
  谓词 *expr IN (expr1, expr2, ...*)为真，当且仅当以下谓词为真：
  expr = ANY (expr1, expr2, ...)

- IN 谓词的执行结果
  IN 谓词为真，当且仅当以下条件之一为真：
  - *row-value-constructor* 的执行结果（一行或一个数值序列）等于该表子查询指定列值的任何行。
    表子查询是一种查询表达式，它可以被指定为简单表格形式。例如：指定为（VALUES 关键字+一个行值列表）。更多信息，请参阅 SELECT 语句。
  - *row-value-constructor* 的执行结果（一个单值）等于 in-value-list 表达式列表指定的任何值。
    在这种情况下，可以将表达式列表看成是一个单列表——表子查询的特殊情况。Row value constructor 的度等于表达式列表的度。

- 比较字符数据

如果相同序数位置的所有字符都相等，则两个字符串是相等的。大小写字母不被视为等同。为了比较具有不同长度的两个字符串，应在较短字符串的右侧填充空格（HEX 20），直到它达到较长字符串的长度为止。固定长度和可变长度字符串都使用这种方式填充。

例如：EsgynDB SQL 认为字符串'JOE'等于 CHAR 或 VARCHAR 数据类型列里存储的数值 JOE（宽度大于或等于 3）。

同样，EsgynDB SQL 认为任何 CHAR 数据类型列里存储的数值 JOE 都等于任何 VARCHAR 数据类型列里存储的数值 JOE。

- 比较数字型数据

  进行求值之前，一个表达式里的所有数字值都将被转换为表达式各处所需的最大精度。

- 比较时间间隔数据

  为了比较 INTERVAL 值，EsgynDB SQL 将首先把时间间隔转换为一种通用单位。如果不存在通用单位，EsgynDB SQL 会报告一个错误。两个 INTERVAL 值必须都是 year-month（年-月型）时间间隔或 day-time（日-时间型）时间间隔。

4.16.4.3 IN 谓词例子

- 查找 EMPNUM 为 39、337 或 452 的员工：

```
SELECT last_name, first_name, empnum
FROM persnl.employee
WHERE empnum IN (39, 337, 452);
LAST_NAME FIRST_NAME EMPNUM
--------- ---------- ------
CLARK     DINAH      337
SAFFERT   KLAUS      39
--- 2 row(s) selected.
```

- 查找部件号不在 PARTLOC 表里的 PARTS 表里的表项：

```
SELECT partnum, partdesc
FROM sales.parts
WHERE partnum NOT IN
   (SELECT partnum FROM invent.partloc);

PARTNUM PARTDESC
------- ------------------
```

```
186     186 MegaByte Disk

--- 1 row(s) selected.
```

- 在 PARTS 表里查找相应的项及其供应商，该供应商应存在于 PARTSUPP 表中：

```
SELECT P.partnum, P.partdesc, S.suppnum, S.suppname
FROM sales.parts P,
   invent.supplier S
WHERE P.partnum, S.suppnum IN
   (SELECT partnum, suppnum
   FROM invent.partsupp);
```

- 在 EMPLOYEE 表里查找姓氏和职位代码都匹配姓氏和职位代码列表的员工：

```
SELECT empnum, last_name, first_name
FROM persnl.employee
WHERE (last_name, jobcode) IN
   (VALUES ('CLARK', 500), ('GREEN', 200));
```

4.16.5 LIKE 谓词

该谓词搜索与指定的某种模式相匹配的字符串。

4.16.5.1 LIKE 谓词语法

match-value [NOT] LIKE *pattern* [ESCAPE *esc-char-expression*]

match-value

这是一种字符值表达式，给定一组将被搜索的目标字符串，该字符串将被确定是否含有指定的匹配模式。

pattern

这是一种字符值表达式，给定了需要匹配的字符串模式。

esc-char-expression

这是一种字符值表达式，它的计算结果必须是一个字符。转义字符值被用于关闭百分号（%）和下划线（_）的特殊含义。

4.16.5.2 LIKE 谓词注意事项

- 将数值与模式进行比较

  需要比较的值必须是字符串。大小写字母并不等同。为了使小写字母匹配大写字母，应使用 UPSHIFT 函数。空格的比较方式与任何其他字符相同。

- LIKE 谓词何时为真

  当您引用一个列时，如果样式匹配列值，则 LIKE 谓词为真。如果列引用的值为 NULL，则该行 LIKE 谓词的求值结果为未知。 如果需要比较的数值都是空串（即零长度的字符串），则 LIKE 谓词为真。

- 使用 NOT

  如果指定了 NOT，并且模式不匹配 match-value（匹配值）里的任何字符串或与此类字符串长度不同，则该谓词为真。例如：如果字符串没有占两个字符的长度或最后一个字符非 Z ，则 NAME NOT LIKE '_Z' 为真。在一个搜索条件里，谓词 NAME NOT LIKE '_Z'等效于 NOT (NAME LIKE '_Z')。

- 通配符

  通过仅指定样式的部分字符并结合这些通配符，您可以搜索类似的数值。

- 百分号（%）

  使用百分号表示任何类型的零个或多个字符。例如：'%ART%'匹配'SMART'、'ARTIFICIAL'和'PARTICULAR'，但不匹配'smart'。

- 下划线（_）

  使用下划线表示任何单个字符。例如：'BOO_' 匹配 'BOOK'和'BOOT' 、但不匹配'BOO'、'BOOKLET'或'book'。

- 转义字符

  为了搜索一个包含百分号（%）或下划线（_）的字符串，使用 ESCAPE esc-char-expression 定义一个转义字符，以便关闭百分号和下划线的特殊含义。

  为了将百分号或下划线包含在一个比较字符串里，请紧接在百分号或下划线前面键入转义字符。

  例如：为了查找数值 'A_B'，请输入：

  ```
  NAME LIKE 'A\_B' ESCAPE '\'
  ```

  为了将转义字符自身包含在比较字符串里，请键入两个转义字符。

  例如：为了查找'A_B\C%'，请输入

```
NAME LIKE 'A\_B\\C\%' ESCAPE '\'
```

转义字符必须只能位于百分号、下划线或转义字符前面。
例如：如果转义字符被定义为'\'，则样式 RA\BS 是一种无效的 LIKE 样式。如果这种样式被用于一条 SQL 查询里，则将返回错误 8410。

- 将样式与 CHAR 列进行比较
  数据类型为 CHAR 的列具有固定长度。将一个值插入 CHAR 列后，EsgynDB SQL 必要时会用空格填充该值。插入到 CHAR(4)列里的值'JOE'将变为'JOE '（3 个字符加上一个空格）。只有列值和比较值具有相同的长度时，LIKE 谓词才为真。列值 'JOE ' 不匹配'JOE'，但匹配'JOE%'。

- 将样式与 VARCHAR 列进行比较
  可变长度字符列不含尾随空格，除非输入数据时已指定了空格。例如：插入到 VARCHAR(4)列里的值'JOE'为不含尾随空格的'JOE'，该值匹配'JOE'和'JOE%'。
  如果您无法在可变长度字符列里找到一个值，可能因为该值插入表时已指定了尾随空格。例如：一个值 '5MB '（带有一个尾随空格）将不会被 LIKE '%MB'找到，但可以被 LIKE '%MB%'找到。

4.16.5.3 LIKE 谓词例子

- 查找以 ZE 开头的所有员工姓氏：
  ```
  last_name LIKE 'ZE%'
  ```

- 查找所有不是'FLOPPY_DISK'的部件描述：
  ```
  partdesc NOT LIKE 'FLOPPY\_DISK' ESCAPE '\'
  ```
  量化比较表明，'FLOPPY_DISK'里的下划线是要搜索的字符串的一部分，而不是通配字符。

4.16.6 REGEXP 谓词

对模式执行字符串表达式的模式匹配。模式可以是一个扩展正则表达式。
如果表达式与模式匹配，则返回 1；否则返回 0。如果表达式或模式为 NULL，则结果为 NULL。

4.16.6.1 REGEXP 谓词语法

```
match-value [NOT] REGEXP regular-expression
```

- *match-value*
  是一个字符值表达式，指定一组字符串来搜索与 *regular-expression* 匹配的字符串。

- NOT
  若指定 NOT，如果 *regular-expression* 与 *match-value* 中的任何字符串都不匹配，则谓词为真。

- *regular-expression*
  是一个指定正则表达式的字符值表达式。EsgynDB 正则表达式遵循 POSIX 正则表达式规则。

4.16.6.2 例子

- 查找有效的数字。
  ```
  col REGEXP '^[0-9]*\s*$'
  ```

- 查找有效的单词，不带数字。
  ```
  col REGEXP '^.[A-Za-z]+\s*$'
  ```

- 查找有效的电子邮件地址。
  ```
  col REGEXP '\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*\s*'
  ```

4.16.7 NULL 谓词

NULL 谓词确定一个序列里的所有表达式是否为空：

```
row-value-constructor IS [NOT] NULL

row-value-constructor is:
     (expression[,expression]...)
     | row-subquery
```

row-value-constructor
指定 NULL 谓词的操作数。该操作数可以是以下构造函数之一：

- *(expression [,expression]...)*
  一个 SQL 值表达式序列，它以逗号分隔并用括号括起来。*expression* 里不得包含一个聚合函数，除非该表达式在 HAVING 子句里。*expression* 可以是标量子查询（返回单行单列的子查询）。

- *row-subquery*

这是一条返回单行（包括数值序列）的子查询。

如果 *row-value-constructor* 里的所有表达式都为 null，则 IS NULL 谓词为真。否则，IS NULL 谓词为假。
如果 *row-value-constructor* 里没有一个表达式为 null，则 IS NOT NULL 谓词为真。否则，IS NOT NULL 谓词为假。

4.16.7.1 NULL 谓词注意事项

- **NULL 谓词的执行结果总结**

  让 *rvc* 作为 *row-value-constructor* 的数值。该表总结了 NULL 谓词的执行结果。一个 *rvc* 的度指的是 *rvc* 里数值的数量。

表达式	rcv IS NULL	rcv IS NOT NULL	NOT rcv is NULL	NOT rcv is NOT NULL
degree 1: null	TRUE	FALSE	FALSE	TRUE
degree 1: not null	FALSE	TRUE	TRUE	FALSE
degree>1: all null	TRUE	FALSE	FALSE	TRUE
degree>1: some null	FALSE	FALSE	TRUE	TRUE
degree>1: none null	FALSE	TRUE	TRUE	FALSE

rvc IS NOT NULL 谓词并不等效于 NOT *rvc* IS NULL。

- **NULL 谓词例子**

  在 SALARY 列里查找所有含有空值的行:

  ```
  salary IS NULL
  ```

  如果表达式(PRICE + TAX)的求值结果为 NULL，则该谓词的求值结果为真:

  ```
  (price + tax) IS NULL
  ```

4.16.8 OVERLAPS 谓词

OVERLAPS 谓词评估两个日期时间段是否相互重叠。日期时间段是一对日期时间表达式。

```
(start 1, end 1) OVERLAPS (start 2, end 2)
```

4.16.8.1 OVERLAPS 谓词语法

- *start 1, start 2*

指定日期时间段的开始，可以是 DATE（日期）、TIME（时间）或 TIMESTAMP（时间戳）值。

注意：*start 1, start 2* 不能为 NULL。

- *end 1, end 2*

指定日期时间段的结束，可以是 DATE（日期）、TIME（时间）、TIMESTAMP（时间戳）或 INTERVAL（时间间隔）值。

注意：*end 1, end 2* 不能为 NULL.

INTERVAL 值将日期时间段向前推进或向后递减。

例子

由于 TIMESTAMP'2011-11-02 01:00:00'比 TIMESTAMP'2011-11-02 02:00:01'早不到一小时，因此这两个日期时间段重叠。

```
SELECT 'Technical Writer' FROM DUAL
WHERE
(TIMESTAMP '2019-11-02 02:00:01', INTERVAL -'1' HOUR)
OVERLAPS
(TIMESTAMP '2019-11-02 01:00:00', INTERVAL '2' HOUR);
(EXPR)
----------------
Technical Writer
--- 1 row(s) selected.
```

4.16.8.2 OVERLAPS 谓词注意事项

- 如果两个日期时间段不是时间点，则当它们共享存在一个公共时间段时就会重叠，也就是说，它们的 TIME/TIMESTAMP 必须有一秒的重叠，或者 DATE/TIMESTAMP 必须有一天的重叠。

  注意：下图描述了每个时间段如何与 S1-E1 时间段重叠。这里，Sn-En 指（*start n, end n*）。

  ```
  Date-time Period Overlaps?
  ---------------|--------S1-E1-------|----------------
  |-S2-E2-| No
  |---S3-E3----| No
  ```

```
|--------S4-E4--------| Yes
|--------------S5-E5--------------| Yes
|--------------------S6-E6-----------------| Yes
|-S7-E7-| Yes
---------------|--------S1-E1-------|----------------
|--------S8-E8-------| Yes
|--------S9-E9-----------| Yes
|-S10-E10-| Yes
|---S11-E11--| Yes
|----S12-E12----| Yes
|-S13-E13-| No
|-S14-E14-| No
---------------|--------S1-E1-------|----------------
```

例子

- 这两个日期时间段有一秒的重叠。

```
SELECT 'Technical Writer' FROM DUAL
WHERE
(TIMESTAMP '2019-11-02 00:00:01', TIMESTAMP '2019-11-
02 00:00:03')
OVERLAPS
(TIMESTAMP '2019-11-02 00:00:02', TIMESTAMP '2019-11-
02 00:00:05');
(EXPR)
----------------
Technical Writer
--- 1 row(s) selected.
```

- 这两个日期时间段有一天的重叠。

```
SELECT 'Technical Writer' FROM DUAL WHERE
(TIMESTAMP '2019-11-03 00:00:01', TIMESTAMP '2019-11-
02 00:00:01')
OVERLAPS
(TIMESTAMP '2019-11-03 00:00:01', TIMESTAMP '2019-11-
01 00:00:01');
(EXPR)
----------------
Technical Writer
--- 1 row(s) selected.
```

- 如果两个日期时间段是瞬时的，则当它们完全相同时就会重叠。

例子

```
SELECT 'Technical Writer' FROM DUAL
WHERE
(TIMESTAMP '2019-11-02 00:00:01', TIMESTAMP '2019-11-
```

```
02 00:00:01')
OVERLAPS
(TIMESTAMP '2019-11-02 00:00:01', TIMESTAMP '2019-11-
02 00:00:01');
(EXPR)
----------------
Technical Writer
--- 1 row(s) selected.
```

- 如果两个日期时间段只有一个共同的端点，则它们不会重叠。

例子

- 由于'2019-11-02'这一常见的单个日期不构成重叠，因此不返回字面值。

```
SELECT 'Technical Writer' FROM DUAL
WHERE
(DATE '2019-11-01', DATE '2019-11-02')
OVERLAPS
(DATE '2019-11-02', DATE '2019-11-03');
--- 0 row(s) selected.
```

- 由于'2019-11-02 00:00:03'这一常见的单个时间戳不构成重叠，因此不返回字面值。

```
SELECT 'Technical Writer' FROM DUAL
WHERE
(TIMESTAMP '2019-11-02 00:00:01', TIMESTAMP '2019-11-
02 00:00:03')
OVERLAPS
(TIMESTAMP '2019-11-02 00:00:03', TIMESTAMP '2019-11-
02 00:00:05');
--- 0 row(s) selected.
```

- 无论是否按时间顺序指定该对的第一个和第二个参数，OVERLAPS 总是以较早的参数作为 *start*，以较晚的参数作为 *end*。

例子

- OVERLAPS 将 TIME '01：00：00'视为第一个范围的 *start*，将 TIME '19：00：00'视为 *end*，第一个范围完全包含第二个范围。

```
SELECT 'Technical Writer' FROM DUAL
WHERE
(TIME '19:00:00', TIME '01:00:00')
OVERLAPS
(TIME '01:00:01', TIME '18:59:59');
```

```
(EXPR)
----------------
Technical Writer
--- 1 row(s) selected.
```

- 日期时间段被视为一个开放的时间间隔（*start* <日期时间< *end*），也就是说，该期间不包括 *start* 和 *end* 值。

4.16.8.3 OVERLAPS 谓词例子

- 由于每两个周期都有重叠，因此返回表 t1、t2 和 t3 的行。

```
SELECT * FROM t1
WHERE
(DATE'2018-11-02', -INTERVAL '03-02' YEAR TO MONTH)
OVERLAPS
(date'2018-07-30', date'2019-11-02');
C1
-------------------------------------------------------
Technical writing is a highly valuable skill.



SELECT * FROM t2
WHERE
(TIME'00:00:01', INTERVAL '00:03:04' HOUR TO SECOND)
OVERLAPS
(TIME'03:04:05', -INTERVAL'03:01:01' HOUR TO SECOND);
C1
-------------------------------------------------------
Technical writing is not just about understanding
technical information and
recording it in a document.
SELECT * FROM t3
WHERE
(CURRENT_TIMESTAMP, INTERVAL -'2' HOUR)
OVERLAPS
(CURRENT_TIMESTAMP + INTERVAL '1' SECOND, CAST (('2018-
11-02 00:00:01') AS
TIMESTAMP));
C1
-------------------------------------------------------
Technical writing takes high-level information and
processes it into digestible
content for a specific audience.
--- 1 row(s) selected.
```

- OVERLAPS 与 WHERE 子句一起使用。

假设表 t4 是这样的：

```
CREATE TABLE t4 AS
SELECT '(yesterday, today) OVERLAPS (today, tomorrow)'
description,
SYSDATE-1 start1, SYSDATE end1, SYSDATE start2, SYSDATE+1
end2 FROM DUAL
UNION ALL
SELECT '(today, tomorrow) OVERLAPS (tomorrow, tomorrow)',
SYSDATE start1, SYSDATE+1 end1, SYSDATE+1 start2,
SYSDATE+1 end2 FROM DUAL
UNION ALL
SELECT '(today, today) OVERLAPS (today, tomorrow)',
SYSDATE start1, SYSDATE end1, SYSDATE start2, SYSDATE+1
end2 FROM DUAL
UNION ALL
SELECT '(today, today) OVERLAPS (today, today)'
description,
SYSDATE start1, SYSDATE end1, SYSDATE start2, SYSDATE
end2 FROM DUAL
--- 4 row(s) inserted.

SELECT * FROM t4;
DESCRIPTION START1 END1 START2
END2
------------------------------------------------ ---------
- ---------- ----------
----------
(today, today) OVERLAPS (today, today) 2019-02-15 2019-
02-15 2019-02-15
2019-02-15
(today, today) OVERLAPS (today, tomorrow) 2019-02-15
2019-02-15 2019-02-15
2019-02-16
(today, tomorrow) OVERLAPS (tomorrow, tomorrow) 2019-02-
15 2019-02-16 2019-02-16
2019-02-16
(yesterday, today) OVERLAPS (today, tomorrow) 2019-02-14
2019-02-15 2019-02-15
2019-02-16
--- 4 row(s) selected.
```

返回以下两行，因为：

条件	TRUE 或 FALSE
(yesterday, today) OVERLAPS (today, tomorrow)	FALSE
(today, tomorrow) OVERLAPS (tomorrow, tomorrow)	FALSE
(today, today) OVERLAPS (today, today)	TRUE
(today, today) OVERLAPS (today,	TRUE

tomorrow)	

```
SELECT * FROM t4 WHERE (start1,end1) OVERLAPS
(start2,end2);
DESCRIPTION START1 END1 START2
END2
------------------------------------------------ ---------
- ---------- ----------
----------
(today, today) OVERLAPS (today, today) 2019-02-15 2019-
02-15 2019-02-15
2019-02-15
(today, today) OVERLAPS (today, tomorrow) 2019-02-15
2019-02-15 2019-02-15
2019-02-16
--- 2 row(s) selected.
```

- OVERLAPS 有助于避免计划冲突。

假设 *employee* 表和 *tech_event_calendar_2019* 表如下所示。

employee 表:

```
SELECT * FROM employee;
NAME AVAILABLE_START AVAILABLE_END
-------------------- --------------- -------------
Aven 2019-01-02 2019-01-10
Alex 2019-02-28 2019-03-03
Andy 2019-04-09 2019-04-10
Eason 2019-05-02 2019-05-08
Kevin 2019-11-20 2019-12-10
Seth 2019-12-09 2019-12-16
--- 6 row(s) selected.
```

tech_event_calendar_2019 表:

```
SELECT * FROM tech_event_calendar_2019;
EVENT EVENT_START EVENT_END LOCATION
------------------------------ ----------- ---------- ---
-------
MWC Barcelona 2019-02-25 2019-02-28 Barcelona, Spain
Google Cloud Next 2019-04-09 2019-04-11 San Francisco, CA
Microsoft Build 2019-05-06 2019-05-08 Seattle, WA
Dreamforce 2019-11-19 2019-11-22 San Francisco, CA
--- 4 row(s) selected.
```

该语句找到了可以参加技术活动的员工。

```
SELECT name, event, event_start, event_end, location
FROM employee, tech_event_calendar_2019
```

```
WHERE
(available_start, available_end)
OVERLAPS
(event_start, event_end);
NAME EVENT EVENT_START EVENT_END LOCATION
-------------------- ------------------------------ -----
------ ----------
----------
Andy Google Cloud Next 2019-04-09 2019-04-11 San
Francisco, CA
Eason Microsoft Build 2019-05-06 2019-05-08 Seattle,
WA
Kevin Dreamforce 2019-11-19 2019-11-22 San
Francisco, CA
--- 3 row(s) selected.
```

4.16.9 量化比较谓词

量化比较谓词比较表达式序列的数值和通过表子查询选出的每一行的数值。

这种比较运算可通过逻辑量词 ALL、ANY 或 SOME 实现量化。

```
row-value-constructor comparison-op quantifier table-
subquery

row-value-constructor is:
     (expression [,expression]...)
     | row-subquery

comparison-op is:
      = Equal
     | <> Not equal
     | != Not equal
     | < Less than
     | > Greater than
     | <= Less than or equal to
     | >= Greater than or equal to

quantifier is:
     ALL | ANY | SOME
```

row-value-constructor

指定一个量化比较谓词的第一个操作数。第一个操作数可以是以下构造函数之一：

- *(expression [,expression]...)*
  这是一种 SQL 值表达式序列，它以逗号分隔、并括在括号里。

expression 不得包含一个聚合函数，除非它位于一个 HAVING 子句中。*expression* 可以是一种标量子查询（一种返回单行单列的子查询）。

- *row-subquery*
  这是一条可以返回单行（包含一个数值序列）的子查询。

ALL
如果比较结果对于 *table-subquery* 所选的每一行都为真（或如果 *table-subquery* 没有选择任何一行），则指定谓词为真。如果至少一行被选，则指定谓词为假。

ANY | *SOME*
如果比较结果对于 *table-subquery* 所选的至少一行为真，则指定谓词为真。比较结果对于所选的每一行都为假（或如果 *table-subquery* 没有选择任何一行），则指定谓词为假。

table-subquery
提供需要进行比较的数值。*row-value-constructor* 返回的数值数量必须等于 *table-subquery* 指定的数值数量。
row-value-constructor 返回的数值的数据类型和 *table-subquery* 返回的数值的数据类型必须可以进行比较。

4.16.9.1 ALL、ANY、SOME 注意事项

假设 R 是行值构造函数的执行结果，T 是表子查询的结果，RT 是 T 里的一行。

R comparison-op ALL T 的执行结果

如果 T 是空的，或如果 R comparison-op RT 对于 T 里的每一行 RT 都为真，则 comparison-op ALL 谓词为真。

如果 R comparison-op RT 对于 T 里的至少一行 RT 为假，则 comparison-op ALL 谓词为假。

Rcomparison-op ANY T 或 Rcomparison-op SOME T 的执行结果

如果 T 是空的或如果 R comparison-op RT 对于 T 里的每一行 RT 都为假，则 comparison-op ANY 谓词为假。

如果 R comparison-op RT 对于 T 里的至少一行 RT 为真，则 comparison-op ANY 谓词为真。

4.16.9.2 ALL、ANY、SOME 例子

- 该谓词为真，如果工资高于所有职位代码为 420 的员工工资：

```
salary > ALL (SELECT salary
                  FROM persnl.employee
                  WHERE jobcode = 420)
```

考虑在这条 SELECT 语句里使用前述谓词：

```
SELECT empnum, first_name, last_name, salary
FROM persnl.employee
WHERE salary > ALL (SELECT salary
FROM persnl.employee
WHERE jobcode = 420);
```

提供比较值的内部查询将产生这些结果：

```
SELECT salary
FROM persnl.employee WHERE jobcode = 420;
SALARY
-----------
33000.00
36000.00
18000.10
--- 3 row(s) selected.
```

使用这种内部查询的 SELECT 语句可以产生这些结果。所列的工资高于职位

代码为 420 的每位员工的工资—即所列工资高于 33,000 美元、36,000.00 美元和 18,000.10 美元。

```
SELECT empnum, first_name, last_name, salary
FROM persnl.employee
WHERE salary > ALL (SELECT salary
                      FROM persnl.employee
                      WHERE jobcode = 420);

EMPNUM FIRST_NAME LAST_NAME SALARY
------ ---------- --------- -----------
1      ROGER      GREEN     175500.00
23     JERRY      HOWARD    137000.10
29     JANE       RAYMOND   136000.00
...
343    ALAN       TERRY     39500.00
557    BEN        HENDERSON 65000.00
568    JESSICA    CRINER    39500.00
--- 23 row(s) selected.
```

- 如果这些部件号都具有 5 个以上的库存单元，则该谓词为真：

```
partnum = ANY (SELECT partnum
                 FROM sales.odetail
                 WHERE qty_ordered > 5)
```

考虑在该 SELECT 语句里使用前述谓词：

```
SELECT ordernum, partnum, qty_ordered
FROM sales.odetail
WHERE partnum = ANY (SELECT partnum
                      FROM sales.odetail
                      WHERE qty_ordered > 5);
```

- 提供比较值的内部查询将产生这些结果：

```
SELECT partnum
FROM sales.odetail
WHERE qty_ordered > 5;

Part/Num
--------
```

```
2403
5100
5103
6301
6500
....
--- 60 row(s) selected.
```

使用内部查询的 SELECT 语句将产生这些结果。这里列出的所有订单号的相应部件号（即：2403、5100、5103、6301、6500 等）都有 5 个以上的库存单元：

```
SELECT ordernum, partnum, qty_ordered
FROM sales.odetail
WHERE partnum = ANY (SELECT partnum
                     FROM sales.odetail
                     WHERE qty_ordered > 5);

Order/Num Part/Num Qty/Ord
---------- -------- ----------
100210     244       3
100210     2001      3
100210     2403      6
100210     5100     10
100250      244      4
100250     5103     10
100250     6301     15
100250     6500     10
...... .... ..
--- 71 row(s) selected.
```

4.17 权限

权限授权给特定的对象执行特定的操作；可以通过多种方式向用户或角色授予或撤消权限：

- 创建对象时，对象所有者将被授予隐式权限。该所有者将在对象的生命周期内一直保留隐式权限。
- 用户或角色可以被授予或撤销显式权限。数据库用户管理员、对象所有者、或一个被授予带有 WITH GRANT OPTION 选项权限的用户可以授予或撤销显式权限。
- 授予用户的权限可以有多种来源。用户既可以被直接授予这些权限，也可以通过一个角色继承这些权限。如：一个用户通过两个不同的角色获取表 T1

的 SELECT 权限。如果其中一个角色被撤销，该用户仍能通过另一被授予的 SELECT 权限的角色访问表 T1：select from T1。

- 被授予角色的用户将因此将被赋予该角色的所有权限。撤销任何此类权限的唯一方法为撤销该用户的角色。更多信息，请参阅角色。
- 对一个对象授予的权限可作用于该对象的所有列，也可作用于列的子集。只有以下权限子集适用于列级别：INSERT、REFERENCES、SELECT 和 UPDATE。

您可以通过使用 GRANT 和 REVOKE 语句管理权限。
更多有关 GRANT 的信息，请参阅：

- GRANT 语句
- GRANT COMPONENT PRIVILEGE 语句
- GRANT ROLE 语句

更多有关 REVOKE 的信息，请参阅：

- REVOKE 语句
- REVOKE COMPONENT PRIVILEGE 语句
- REVOKE ROLE 语句

4.18 角色

一个角色可以灵活地、隐式地分配一组权限（而非单个权限）给用户。一个用户可被授予一个或多个角色。
一个角色既可以被授予一个或多个用户，也可由数据库用户管理员、角色所有者或一个角色成员授予或撤销。一个角色将被授予多种权限。

当用户被授予一个角色时，该角色被授予的权限将提供给该用户。
如果该角色被授予新的权限，则这些权限将提供给所有已被授予该角色的用户。
如果撤销用户的一个角色，则该角色被授予的权限将不再提供给该用户。

在 EsgynDB Release 0.9 中，为了使任何权限变更生效，用户必须断开任何当前会话，再重新连接并建立新会话。
然而，从 EsgynDB Release 1.0 开始，权限变更情况将被自动传播给活动会话，并将被活动会话检测到。
因此用户无需断开再重新连接一个会话，即可看到更新后的权限集。更多有关权限的信息，请参阅权限。

角色名是一种授权 ID，它不能与已注册的数据库用户名相同。更多信息，请参阅授权 ID。

为了管理角色，请参阅以下 SQL 语句：
CREATE ROLE 语句
DROP ROLE 语句
GRANT ROLE 语句
REVOKE ROLE 语句

4.19 Schemas

ANSI SQL:1999 schema 名称是一种 SQL 标识符，它在给定的 ANSI 编目名称里是唯一的。EsgynDB SQL 使用当前默认的编目名称（TRAFODION）自动限定 schema 名称。
schema.object 形式的逻辑名称是一种 ANSI 名称。schema 部分表示 ANSI 标准定义的 schema。
为了遵循 ANSI SQL:1999 标准，EsgynDB SQL 为 ANSI 对象名称提供支持。
通过使用这些名称，您可以开发符合 ANSI 标准的应用程序访问所有 SQL 对象。
您也可以使用实际对象名称访问 EsgynDB SQL 对象。

4.19.1 创建和删除 schema

您既可以使用 CREATE SCHEMA 命令创建一个 schema，也可以使用 DROP SCHEMA 语句删除一个 schema。更多信息，请参考 CREATE SCHEMA 语句和 DROP SCHEMA 语句。

4.20 搜索条件

基于将条件应用到行的结果，搜索条件用来从表或视图里选择行。该条件是一种布尔表达式，它由谓词结合 OR、AND 和 NOT 运算符组成。
您可以在 SELECT、DELETE 或 UPDATE 语句的 WHERE 子句、SELECT 语句的 HAVING 子句、一个 CASE 表达式的搜索表单、以及涉及到联接操作、CHECK 约束的 ON 子句里使用一个搜索条件。

```
search-condition is:
     boolean-term | search-condition OR boolean-term
```

```
boolean-term is:
     boolean-factor | boolean-term AND boolean-factor

boolean-factor is:
     [NOT] boolean-primary

boolean-primary is:
     predicate | (search-condition)
```

OR
指定产生的搜索条件为真，当且仅当一个周边谓词或搜索条件为真。

AND
指定产生的搜索条件为真，当且仅当两个周边谓词或搜索条件都为真。

NOT
反转其操作数（以下谓词或搜索条件）的真值。

谓词
包含 BETWEEN、比较谓词、EXISTS、IN、LIKE、NULL 或量化比较谓词。
一个谓词指定被选中的行必须满足的条件。更多信息，请参阅谓词。

4.20.1 搜索条件注意事项

4.20.1.1 求值顺序

SQL 按照以下顺序对搜索条件进行求值：

1) 括号内的谓词
2) NOT
3) AND
4) OR

4.20.1.2 列引用

在一个搜索条件里，一个列引用能引用当前正通过搜索条件求值的行里的该列的值。

4.20.1.3 子查询

如果一个搜索条件包含一个子查询，并且该子查询没有返回任何数值，则谓词的求值结果为空。更多信息，请参阅子查询。

4.20.2 搜索条件例举例

- 通过使用一个由三个比较谓词（AND 运算符联接）构成的搜索条件选择行：

```
select O.ordernum, O.deliv_date, OD.qty_ordered
FROM sales.orders O,
   sales.odetail OD
WHERE qty_ordered < 9 AND deliv_date <= DATE '2008-11-
01'
   AND O.ordernum = OD.ordernum;

ORDERNUM DELIV_DATE QTY_ORDERED
-------- ---------- -----------
100210   2008-04-10 3
100210   2008-04-10 3
100210   2008-04-10 6
100250   2008-06-15 4
101220   2008-12-15 3
...
--- 28 row(s) selected.
```

- 通过使用一个由三个比较谓词构成的搜索条件选择行；其中两个谓词通过一个（括号内的）OR 运算符联接；这里，OR 的结果和第一个比较谓词通过 AND 运算符联接：

```
SELECT partnum, S.suppnum, suppname
FROM invent.supplier S,
   invent.partsupp PS
WHERE S.suppnum = PS.suppnum
   AND (partnum < 3000 OR partnum = 7102);
PARTNUM SUPPNUM SUPPNAME
------- ------- ------------------
    212       1 NEW COMPUTERS INC
    244       1 NEW COMPUTERS INC
    255       1 NEW COMPUTERS INC
    ...
    7102     10 LEVERAGE INC
   --- 18 row(s) selected.
```

4.21 子查询

子查询是一种用括号括起来的查询表达式。它的语法形式已经指定在 SELECT 语句的语法里。

子查询既可用于提供搜索条件里的 BETWEEN、比较谓词、EXISTS、IN 或量化比较谓词的值，也可用于指定 SELECT 语句的 FROM 子句里的派生表。

子查询可以是一个表、行或标量子查询。因此，其结果表可以包含多个行和列、带有多个列值的单行、或仅带有一个列值的单行。

4.21.1 SELECT 形式的子查询

子查询通常被指定为一种用括号括起来的特殊形式的 SELECT 语句，它可以通过查询（或选择操作）提供搜索条件里的数值或指定一个派生表作为表引用。
这种被指定为 SELECT 语句的子查询形式是一种 query-expr。
子查询里不得出现 ORDER BY 子句和[FIRST N] / [ANY N]子句。

4.21.2 使用子查询提供比较值

如果一个子查询被用于提供比较值，则包含该子查询的 SELECT 语句被称作外部查询，SELECT 语句中的子查询则被称作内部查询。
在这种情况下，SELECT 语句和 SELECT 形式的子查询有以下区别：

- 子查询始终用括号括起来。
- 子查询不得包含 ORDER BY 子句.
- 如果一个子查询不是 EXISTS、IN 或量化比较谓词的一部分，并且该子查询的求值结果为多行，则将出现运行时错误。

4.21.3 提供比较值的嵌套子查询

一条主 SELECT 语句的外部查询可能带有嵌套的子查询。同一个 WHERE 或 HAVING 子句里的子查询处于相同层次。例如：该查询具有一个嵌套层次：

```
SELECT * FROM TABLE1
     WHERE A = (SELECT P FROM TABLE2 WHERE Q = 1)
          AND B = (SELECT X FROM TABLE3 WHERE Y = 2)
```

然而，另一个子查询的 WHERE 子句中的子查询则处于不同的层次。因此，该查询具有两个嵌套层次：

```
SELECT * FROM TABLE1
     WHERE A = (SELECT P FROM TABLE2
          WHERE Q = (SELECT X FROM TABLE3
               WHERE Y = 2))
```

嵌套子查询的最高层次可能取决于：

- 子查询的复杂度
- 子查询是否相互关联；如果相互关联，子查询是否可以解套。
- 可用的内存量

其他因素也可能影响子查询的最高层级。

4.21.4 提供比较值时出现的相关子查询

在一个子查询里，当您引用一个外部查询里定义的任何表或视图的列时，这种引用被称为外部引用。 一条包含外部引用的子查询被称为相关子查询。

当您引用一个出现在多个外部查询里的列名时，您必须使用其所属的表或视图的相互关系名称限定该列名。同样，如果您引用一个出现在子查询、以及一个或多个外部查询里的列名时，您也必须使用其所属的表或视图的相互关系名称限定该列名。

该相互关系名称已被同一层次的其他子查询或内部查询知晓，但并未被外部查询知晓。如果您在不同的嵌套层次上使用同一个相互关系名称，则内部查询将使用来自最近的外层的相互关系名称。

4.22 表

表是数据的逻辑表述，其中一组记录以行序列呈现，所有行的公用字段集则被以列呈现。列是一组具有相同定义和相同数据类型的数值。行和列的交叉点表示特定记录里的特定字段的数据值。每个表都必须有一列或多列，但行数可以为零。表内不存在固有行序。

您可以使用 CREATE TABLE 语句创建一个 EsgynDB SQL 用户表。

语句里的用户表定义包含以下信息:

- 表名
- 表里各列的名称
- 可以存储在该表各列中的数据类型
- 关于表的其他信息，包括存储该表的文件的物理特性（例如：表内各行的存储顺序）

EsgynDB SQL 表被描述在 SQLschema 里，并被存储为一个 HBase 表。EsgynDB SQL 表有 TRAFODION catalog 里的常规 ANSI 名称。

EsgynDB SQL 表名可以是完全限定的 ANSI 名称，具体形式为 TRAFODION.schema-name.object-name。一个 EsgynDB SQL 表的元数据存储在 schema TRAFODION."_MD_"里。

由于 EsgynDB 为 EsgynDB SQL 表里的列值定义编码，所以这些表可以支持各种不同的 EsgynDB SQL 语句。更多信息，请参阅 HBase 表支持的 SQL 语句。

在内部，EsgynDB SQL 表使用单一 HBase 列族和缩短的列名以节省空间。它们的编码允许出现含有多个列的键，并可以保留由 S QL 定义的键值顺序。
底层的 HBase 列模型便于在 EsgynDB SQL 表里添加和删除列。未被记录到 EsgynDB 元数据里的 HBase 列将被忽略，缺失的列被视为空值。

4.22.1 基表

在一些 SQL 说明里，使用 CREATE TABLE 语句创建的表被称作基表，以此区分它们和视图（被称作逻辑表）。

4.22.1.1 基表例子

例如： EMPLOYEE 表是样本数据库里的基表：

EMPNUM	FIRST_NAME	LAST_NAME	DEPTNUM	JOBCODE	SALARY
1	ROGER	GREEN	9000	100	175500.00
23	JERRY	HOWARD	1000	100	137000.00
75	TIM	WALKER	3000	300	32000.00
..	...	...	...	...	...

这个样本视图里的列为 EMPNUM、FIRST_NAME、LAST_NAME、DEPTNUM、JOBCODE 和 SALARY。每个列里的值都具有相同的数据类型。

4.23 视图

视图是由查询规范定义的一种命名逻辑表，该规范使用一个基表、多个基表或其他视图。
视图为查看一个或多个表里的数据提供了一种替代方案。
视图是结果表的一种命名规范，该表指的是从一个或多个基表或其他视图选择或生成的行集。该规范涉及到 SELECT 语句，每当引用该视图时都会执行。
视图是一种逻辑表，它既可以通过 CREATE VIEW 语句创建，也可以通过投影列的子集、限制行的子集、或两种操作的组合从一个或多个基表或其他视图派生。

4.23.1 SQL 视图

一个视图名必须在其所属的 schema 里的表名和视图名里是唯一的。单表视图可

以更新，多表视图则不可更新。有关 SQL 视图的更多信息，请参考 CREATE VIEW 语句和 DROP VIEW 语句。

4.23.2 视图例子

您可以定义一个视图只显示一个表的部分数据。例如：该 EMPLIST 视图被定义为 EMPLOYEE 表的一部分：

EMPNUM	FIRST_NAME	LAST_NAME	DEPTNUM	JOBCODE
1	ROGER	GREEN	9000	100
23	JERRY	HOWARD	1000	100
75	TIM	WALKER	3000	300
..	...	...	...	...

该样本视图里的列为 EMPNUM、FIRST_NAME、LAST_NAME、DEPTNUM 和 JOBCODE。EMPLOYEE 表里的 SALARY 列不是 EMPLIST 视图的一部分。

4.24 ROWID

4.24.1 ROWID 语法

ROWID 返回一行数据在数据库中的地址。

在 EsgynDB 数据库中，ROWID 返回的是 16 进制的 Clustering-Key，对应 Hbase 中的 ROWKEY。所以在 EsgynDB 中，ROWID 的长度并不固定，是依赖于表定义的。

4.24.2 ROWID 例子

```
>>select rowid from vendor ;

ROWID
----------------
9980B3F9B5385D36
9980B3F9B538863F
9980B3F9B5388EE6
9980B3F9B53893E4
9980B3F9B5389D86

>>select vend_id from vendor where rowid =
'9980B3F9B5389D86';

VEND_ID
-----------

     115
```

4.25 ROWNUM

4.25.1 ROWNUM 语法

ROWNUM 是结果集的编号，从 1 开始。结果集第一行数据的 ROWNUM 值为 1，第二行值 2，以此类推。

4.25.2 ROWNUM 例子

```
SQL>SELECT ROWNUM, VEND_ID FROM VENDOR WHERE ROWNUM < 5;

ROWNUM                VEND_ID
--------------------- -----------
                 1         111
                 2         112
                 3         113
                 4         114
```

以下举例显示用户可以通过 ROWNUM 来实现分页显示的功能。

```
SQL>SELECT RN, VEND_ID FROM (
+>  SELECT ROWNUM AS RN, VEND_ID FROM VENDOR WHERE ROWNUM
< 8
+>) WHERE RN > 4;

RN                    VEND_ID
--------------------- -----------
                5         115
                6         116
                7         117
```

5. SQL 子句

EsgynDB SQL 语句使用这些子句指定默认值、抽样或排序方式、物理数据的存储方案和其他细节信息。

本章目录:

5.1 DEFAULT 子句	DEFAULT 子句为一个正在创建的列指定一个默认值。
5.2 FORMAT 子句	FORMAT 子句指定所用的格式。
5.3 SAMPLE 子句	SAMPLE 子句指定一种抽样方法，用于从 SELECT 语句的中间结果表里选择子集。
5.4 TRANSPOSE 子句	TRANSPOSE 子句为 SELECT 源表的每一行都生成一行，分别用于转置项列表里的各项。

5.1 DEFAULT 子句

5.1.1 DEFAULT 子句语法

CREATE TABLE 或 ALTER TABLE table-name ADD COLUMN 语句的 DEFAULT 选项可以为一个被创建的列指定默认值。该默认值将在一行插入表里时被使用。

```
DEFAULT default | NO DEFAULT

default is:
     literal
     | NULL
     | CURRENT_DATE
     | CURRENT_TIME
     | CURRENT_TIMESTAMP
```

NO DEFAULT
指定该列不含默认值。您不得在 ALTER TABLE 语句里指定 NO DEFAULT。

DEFAULT *literal*
这是一种与关联列的数据类型相兼容的数据类型的字面值。
对于一个字符列而言，字面值必须是一个不超过 240 个字符或列长度的字符串字面值（以较小者为准）。字符列默认值的最大长度为 240 字节（减去控制字符）或列长度（以较小者为准）。
控制字符包含字符集前缀和和出现在其文本里的单引号分隔符。
对于一个数字列而言，字面值不得超过定义的列长度。小数点右侧的位数不得超过列的标度，小数点左侧的位数不得超过长度数量（或长度减去标度，如果您指定了该列的标度）。
对于一个日期时间列而言，字面值必须是一个匹配该列精度的日期时间字面值。
对于一个 INTERVAL 列而言，字面值必须是一个 INTERVAL 字面值，其带有针对该列定义的 INTERVAL 字段范围。

DEFAULT NULL
指定 NULL 为默认值。该默认值只能在一个允许为空的列里出现。

DEFAULT CURRENT_DATE
当它为一个列赋值时，可以指定该列的默认值为 CURRENT_ DATE 函数的返回值。该默认值只能在列的数据类型为 DATE 时出现。

DEFAULT CURRENT_TIME
当它为一个列赋值时，可以指定该列的默认值为 CURRENT_TIME 函数的返回值。该默认值只能在列的数据类型为 TIME 时出现。

DEFAULT CURRENT_TIMESTAMP
当它为一个列赋值时，可以指定该列的默认值为 CURRENT_TIMESTAMP 函数的返回值。该默认值只能在列的数据类型为 TIMESTAMP 时出现。

为兼容 Oracle， DDL 可以将 Numeric 类型的 DEFAULT 值设置为字符串。

5.1.2 DEFAULT 子句例子

- 该例子在 CREATE TABLE 语句里使用 DEFAULT 子句指定默认列值:

```
CREATE TABLE items
( item_id CHAR(12) NO DEFAULT
,description CHAR(50) DEFAULT NULL
,num_on_hand INTEGER DEFAULT 0 NOT NULL );
```

- 该例子在 CREATE TABLE 语句里使用 DEFAULT 子句指定默认列值:

```
CREATE TABLE persnl.project
( projcode      NUMERIC (4) UNSIGNED
                NO DEFAULT
                NOT NULL
,empnum        NUMERIC (4) UNSIGNED
                NO DEFAULT
                NOT NULL
,projdesc       VARCHAR (18)
                DEFAULT NULL
,start_date    DATE
                DEFAULT CURRENT_DATE
,ship_timestamp TIMESTAMP
                DEFAULT CURRENT_TIMESTAMP
,est_complete INTERVAL DAY
```

```
                DEFAULT INTERVAL '30' DAY
  ,PRIMARY KEY (projcode) );
```

- 该例子将 Numeric 类型的 DEFAULT 值设置为字符串。

```
>>create table test1(a char(100), b numeric default '0');
--- SQL operation complete.
>>insert into test1(a) values('a');
--- 1 row(s) inserted.
>>select * from test1;

A
B
------------------------------------------------------------
----------------------------------
a
0

--- 1 row(s) selected.
```

5.2 FORMAT 子句

FORMAT 子句既可以指定 DATE 值的输出格式，也可以指定字符输出的长度、或指定使用冒号分隔整数输出数字。

日期格式:

```
(FORMAT 'format-string') |
(DATE, FORMAT 'format-string')

format-string for Date Formats is:
     YYYY-MM-DD
     MM/DD/YYYY
     YY/MM/DD
     YYYY/MM/DD
     YYYYMMDD
     DD.MM.YYYY
     DD-MM-YYYY
     DD-MMM-YYYY

Other Formats:
     (FORMAT 'format-string')

format-string for other formats is:
     XXX
     99:99:99:99
     -99:99:99:99
```

5.2.1 FORMAT 子句语法

YYYY-MM-DD
指定 FORMAT 子句的输出格式为 year-month-day（年-月-日）。

MM/DD/YYYY
指定 FORMAT 子句的输出格式为 month/day/year（月/日/年）。

YY/MM/DD
指定 FORMAT 子句的输出格式为 year/month/day（年/月/日）。
YYYY/MM/DD
指定 FORMAT 子句的输出格式为 year/month/day（年/月/日）。

YYYYMMDD
指定 FORMAT 子句的输出格式为 yearmonthday（年月日）。

DD.MM.YYYY
指定 FORMAT 子句的输出格式为 day.month.year（日. 月. 年）。

DD-MM-YYYY
指定 FORMAT 子句的输出格式为 day-month-year（日-月-年）。

DD-MMM-YYYY
指定 FORMAT 子句的输出格式为 day-month-year（日-月-年）。

XXX
指定 FORMAT 子句的输出格式为字符串。输入必须是一个数字值或字符串值。

99:99:99:99
指定 FORMAT 子句的输出格式为时间戳。输入必须是一个数字值。

-99:99:99:99
指定 FORMAT 子句的输出格式为时间戳。输入必须是一个数字值。

5.2.2 日期格式注意事项

(FORMAT " *format-string*')子句前面的表达式必须是一个 DATE（日期）值。

(DATE, FORMAT '*format-string*')子句前面的表达式必须是美国、欧洲或

DEFAULT（默认）日期格式的带引号的字符串。

5.2.3 其他格式注意事项

对于 XXX 而言，(FORMAT *'format-string'*)子句前面的表达式必须是一个数字值或字符串值。

对于 99:99:99:99 和-99:99:99:99 而言，(FORMAT *'format-string'*)子句前面的表达式必须是一个数字值。

5.2.4 FORMAT 子句例子

- 该例子里的字符串格式'XXX'将产生的样本结果为 abc:
  ```
  SELECT 'abcde' (FORMAT 'XXX') FROM (VALUES(1)) t;
  ```

- 该例子里的字符串格式'YYYY-MM_DD'将产生的样本结果为 2008-07-17:
  ```
  SELECT CAST('2008-07-17' AS DATE) (FORMAT 'YYYY-MM-DD')
  FROM (VALUES(1)) t;
  ```

- 该例子里的字符串格式'MM/DD/YYYY’将产生的样本结果为 07/17/2008:
  ```
  SELECT '2008-07-17' (DATE, FORMAT 'MM/DD/YYYY') FROM
  (VALUES(1)) t;
  ```

- 该例子里的字符串格式'YY/MM/DD’将产生的样本结果为 08/07/17:
  ```
  SELECT '2008-07-17'(DATE, FORMAT 'YY/MM/DD') FROM
  (VALUES(1)) t;
  ```

- 该例子里的字符串格式'YYYY/MM/DD’将产生的样本结果为 2008/07/17:
  ```
  SELECT '2008-07-17' (DATE, FORMAT 'YYYY/MM/DD') FROM
  (VALUES(1)) t;
  ```

- 该例子里的字符串格式'YYYYMMDD’将产生的样本结果为 20080717:
  ```
  SELECT '2008-07-17' (DATE, FORMAT 'YYYYMMDD') FROM
  (VALUES(1)) t;
  ```

- 该例子里的字符串格式'DD.MM.YYYY’将产生的样本结果为 17.07.2008:
  ```
  SELECT '2008-07-17' (DATE, FORMAT 'DD.MM.YYYY') FROM
  (VALUES(1)) t;
  ```

- 该例子里的字符串格式'DD-MMM-YYYY’将产生的样本结果为 17–JUL-2008:
  ```
  SELECT '2008-07-17' (DATE, FORMAT 'DD-MMM-YYYY') FROM
  ```

```
(VALUES(1)) t;
```

- 该例子里的字符串格式'99:99:99:99'将产生的样本结果为 12:34:56:78:

```
SELECT 12345678 (FORMAT '99:99:99:99') FROM (VALUES(1))
t;
```

- 该例子里的字符串格式'-99:99:99:99'将产生的样本结果为-12:34:56:78:

```
SELECT (-12345678) (FORMAT '-99:99:99:99') FROM
(VALUES(1)) t;
```

5.3 SAMPLE 子句

SELECT 语句的 SAMPLE 子句指定一种抽样方法，从 SELECT 语句的中间结果表中选择一个子集。中间结果表包含由 WHERE 子句返回的行或由 FROM 子句返回的行（如果 WHERE 子句不存在）。

SAMPLE 子句是 EsgynDB SQL 扩展子句。

```
SAMPLE sampling-method

sampling-method is:
     RANDOM percent-size
     | FIRST rows-size
          [SORT BY colname [ASC[ENDING] | DESC[ENDING]]
               [,colname [ASC[ENDING] | DESC[ENDING]]]...]
     | PERIODIC rows-size EVERY number-rows ROWS
          [SORT BY colname [ASC[ENDING] | DESC[ENDING]]
               [,colname [ASC[ENDING] | DESC[ENDING]]]...]

percent-size is:
     percent-result PERCENT [ROWS]
     | BALANCE WHEN condition THEN percent-result PERCENT
     [ROWS]
          [WHEN  condition  THEN  percent-result  PERCENT
          [ROWS]]...
          [ELSE percent-result PERCENT [ROWS]] END

rows-size is:
     number-rows ROWS
     | BALANCE WHEN condition THEN number-rows ROWS
          [WHEN condition THEN number-rows ROWS]...
          [ELSE number-rows ROWS] END
```

5.3.1 SAMPLE 子句语法

RANDOM *percent-size*

指导 EsgynDB SQL 随机地选择行（每一行都具备无偏性被选概率），结果表的行不可置换。抽样大小取决于 percent-size，具体定义如下：

```
percent-result PERCENT [ROWS]
| BALANCE WHEN condition THEN percent-result PERCENT [ROWS]
     [WHEN condition THEN percent-result PERCENT [ROWS]]...
     [ELSE percent-result PERCENT [ROWS]] END
```

通过使用结果表的百分比指定 RANDOM 抽样大小的数值。percent-result 的数值必须是一个数字型字面值。

您可以确定实际样本大小。假设中间结果表里有 N 行，每一行的被选概率为 r%，其中的 r 为样本大小，以 PERCENT（百分比）为单位。因此，所得样本的实际大小约为 N 的 r%。被选的行数遵循平均值为 r * N/100 的二项分布。

如果您指定的样本大小超过了 100%，EsgynDB SQL 将返回结果表里的所有行和重复行。基于指定的抽样方法从结果表中挑选出重复行，该技术为过度抽样。

ROWS

指定行抽样。默认值为行抽样。

BALANCE

如果您指定一个 BALANCE 表达式，则 EsgynDB SQL 将执行分层抽样。中间结果表基于 WHEN 条件划分为不相交的层次。使用抽样大小对每个层次独立抽样。一个给定的行所属的层次取决于其满足的第一个 WHEN 条件（如果真值条件存在）。如果真值条件不存在，则该行将属于 ELSE 层次。

```
FIRST rows-size [SORT BY colname [ASC[ENDING] |
DESC[ENDING]]
[,colname [ASC[ENDING] | DESC[ENDING]]]...]
```

指导 EsgynDB SQL 从结果表里选出前几行。

您可以指定要采样的行的顺序，否则，EsgynDB SQL 将任意选择一种顺序。

样本大小取决于 rows-size，具体定义如下：

```
number-rows ROWS
     | BALANCE WHEN condition THEN number-rows ROWS
          [WHEN condition THEN number-rows ROWS]...
          [ELSE number-rows ROWS] END
```

通过使用样本里的预期行数，指定 FIRST 抽样大小的数值，值 number-rows 必须是整数文字。并且我们可以决定样本的实际大小，假设中间结果表里有 N 行。如果将样本大小 s 被指定为行数，则所得样本的实际大小将是 s 和 N 的最小值。

```
PERIODIC rows-size EVERY number-rows ROWS
[SORT BY colname [ASC[ENDING] | DESC[ENDING]]
[,colname [ASC[ENDING] | DESC[ENDING]]]...]
```

指导 EsgynDB SQL 从每个块（或周期）的连续行里选出前几行。

该抽样方法等效于每个周期独立的 FIRST（首行）抽样，行的大小在 FIRST 抽样中被定义。周期大小被指定为一个行数。您可以指定样本的行序，否则，EsgynDB SQL 将任意选择一种顺序。

您可以确定实际样本大小；假设中间结果表里有 N 行。如果样本尺寸被指定为一个行数，周期大小 p 被指定为另一个行数，则所得样本的实际大小将被计算为：FLOOR (N/p) * s + *minimum* (MOD (N, p), s)

minimum 在该表达式里被简单地用作两个值中的数学最小值。

5.3.2 SAMPLE 子句注意事项

5.3.2.1 样本行

一般而言，当您使用 SAMPLE 子句时，每次执行同一个查询时将返回不同的行集。只有当您使用带有 SORT BY 选项的 FIRST 和 PERIODIC 抽样方法，并且这里指定的列组合里没有重复值需要排序时，才会返回相同的行集。

5.3.3 SAMPLE 子句例子

- 假设数据挖掘表 SALESPER、SALES 和 DEPT 已被创建为：

```
CREATE TABLE trafodion.mining.salesper
   ( empid NUMERIC (4) UNSIGNED NOT NULL
   , dnum NUMERIC (4) UNSIGNED NOT NULL
   , salary NUMERIC (8,2) UNSIGNED
   , age INTEGER
   , sex CHAR (6)
   , PRIMARY KEY (empid) );
CREATE TABLE trafodion.mining.sales
   ( empid NUMERIC (4) UNSIGNED NOT NULL
   , product VARCHAR (20)
   , region CHAR (4)
   , amount NUMERIC (9,2) UNSIGNED
   , PRIMARY KEY (empid) );
```

```
CREATE TABLE trafodion.mining.dept
   ( dnum NUMERIC (4) UNSIGNED NOT NULL
   , name VARCHAR (20)
   , PRIMARY KEY (dnum) );
```

假设样本数据已被插入到该数据库里。

- 返回 50 位最年轻的销售人员的 SALARY（工资）:

```
SELECT salary FROM salesperson
SAMPLE FIRST 50 ROWS SORT BY age;

SALARY
-----------
90000.00
90000.00
28000.00
27000.12
136000.00
37000.40
...

--- 50 row(s) selected.
```

- 返回 50 位销售人员的 SALARY（工资）。在这种情况下，该表将聚集在 EMPID 列上。如果优化器使用主要访问路径选择一个行访问计划，则执行结果将包含具有最小员工标识符的 50 位销售人员的工资。

```
SELECT salary FROM salesperson
SAMPLE FIRST 50 ROWS;

SALARY
-----------
175500.00
137000.10
136000.00
138000.40
75000.00
90000.00
...

--- 50 row(s) selected.
```

- 返回 5 个最年轻的销售人员的 SALARY，跳过此后的 15 行，再重复这个过

程，直到中间结果表里不存在更多的行为止。当您指定定期抽样时，样本大小不得超过周期数。

```
SELECT salary FROM salesperson
SAMPLE PERIODIC 5 ROWS EVERY 20 ROWS SORT BY age;

SALARY
-----------
90000.00
90000.00
28000.00
27000.12
136000.00
36000.00

...

--- 17 row(s) selected.
```

该例子的 SALESPERSON 表里有 62 行。从每 20 行组成的行集里选出前 5 行，最后一个行集包含 2 行，它们都将被选出。

- 计算随机抽取的 10%销售人员的平均工资。由于它基于随机样本，所以每次执行该查询时您都将得到不同的结果。

```
SELECT AVG(salary) FROM salesperson
SAMPLE RANDOM 10 PERCENT;

(EXPR)
--------------------
61928.57

--- 1 row(s) selected.
```

- 如果通过执行 WHERE 子句已经选出符合条件的行，该查询将对随后的抽样情况进行说明。该查询可以计算随机抽取的 10%的 35 岁以上销售人员的平均工资。由于它基于随机样本，所以每次执行该查询时您都将得到不同的结果。

```
SELECT AVG(salary) FROM salesperson WHERE age > 35
SAMPLE RANDOM 10 PERCENT;

(EXPR)
--------------------
```

```
58000.00

--- 1 row(s) selected.
```

- 计算随机抽取的 CORPORATE 部门 10%销售人员的平均工资。抽取的样本来自 SALESPERSON 和 DEPARTMENT 表的联接。 由于它基于随机样本，所以每次执行该查询时您都将得到不同的结果。

```
SELECT AVG(salary)
FROM salesperson S, department D
WHERE S.DNUM = D.DNUM
AND D.NAME = 'CORPORATE'
SAMPLE RANDOM 10 PERCENT;

(EXPR)
---------------------
106250.000

--- 1 row(s) selected.
```

- 该例子里首先对SALESPERSON表进行抽样，然后将其与DEPARTMENT表进行联接。该查询可以计算CORPORATE Department部门的所有销售人员的平均工资，这里抽取10%的销售员工作为随机样本。

```
SELECT AVG(salary)
FROM (SELECT salary, dnum
      FROM salesperson
      SAMPLE RANDOM 10 PERCENT ) AS S, department D
WHERE S.DNUM = D.DNUM
AND D.NAME = 'CORPORATE';

(EXPR)
--------------------
37000.000

--- 1 row(s) selected.
```

该查询的执行结果和前面查询的一些执行结果都可能返回 null:

```
SELECT AVG(salary)

FROM (SELECT salary, dnum
      FROM salesperson
      SAMPLE RANDOM 10 PERCENT ) AS S, department D
```

```
WHERE S.DNUM = D.DNUM
AND D.NAME = 'CORPORATE';

(EXPR)
---------------------
?

--- 1 row(s) selected.
```

为了执行该查询，嵌入式查询所返回的行数将受限于 SALESPERSON 表的总行数。因此，可能没有一行满足 WHERE 子句里的搜索条件。

- 在该例子中，首先对两个表进行抽样，再对它们进行联接操作。通过从所有销售人员里随机抽取 10%，并从所有销售事务里随机抽取 20%，该查询可以计算由此生成的平均工资和平均销售量。

```
SELECT AVG(salary), AVG(amount)

FROM (SELECT salary, empid
      FROM salesperson
      SAMPLE RANDOM 10 PERCENT ) AS S,
     (SELECT amount, empid FROM sales
      SAMPLE RANDOM 20 PERCENT ) AS T
WHERE S.empid = T.empid;

(EXPR)     (EXPR)
--------- ---------
45000.00  31000.00

--- 1 row(s) selected.
```

- 该例子对过度抽样进行了说明。该查询可以检索到 150%的金额超过 1000 美元的销售事务。在查询结果里，每一行都至少出现一次；并且 50%的行将先后两次随机选出。

```
SELECT *
FROM sales
WHERE amount > 1000
SAMPLE RANDOM 150 PERCENT;

EMPID PRODUCT              REGION AMOUNT
----- -------------------- ------ -----------
1     PCGOLD, 30MB             E 30000.00
```

```
23    PCDIAMOND, 60MB          W 40000.00
23    PCDIAMOND, 60MB          W 40000.00
29    GRAPHICPRINTER, M1       N 11000.00
32    GRAPHICPRINTER, M2       S 15000.00
32    GRAPHICPRINTER, M2       S 15000.00
... ... ... ...

--- 88 row(s) selected.
```

- BALANCE 选项启用分层抽样。检索 1000 位销售人员的年龄和工资，由此得到的结果中男女比例各占一半。

```
SELECT age, sex, salary FROM salesperson
SAMPLE FIRST
BALANCE WHEN sex = 'male' THEN 15 ROWS
WHEN sex = 'female' THEN 15 ROWS
END ORDER BY age;

AGE  SEX   SALARY
---- ----- ----------
22   male   28000.00
22   male   90000.00
22   female 136000.00
22   male   37000.40
... ... ...

--- 30 row(s) selected.
```

- 检索所有金额超过 10000 美元的销售记录，并从剩余记录的 10%中随机抽样：

```
SELECT *
FROM sales
SAMPLE RANDOM
BALANCE WHEN amount > 10000 THEN 100 PERCENT
ELSE 10 PERCENT
END;
EMPID PRODUCT              REGION AMOUNT
----- -------------------- ------ -----------
1     PCGOLD, 30MB             E 30000.00
23    PCDIAMOND, 60MB          W 40000.00
29    GRAPHICPRINTER, M1       N 11000.00
32    GRAPHICPRINTER, M2       S 15000.00
... ... ... ...
228 MONITORCOLOR, M2           N 10500.00
```

```
... ... ... ...

--- 32 row(s) selected.
```

- 该查询显示了一个分层抽样的例子，其中的条件不是相互排斥的：

```
SELECT *
FROM sales SAMPLE RANDOM
BALANCE WHEN amount > 10000 THEN 100 PERCENT
        WHEN product = 'PCGOLD, 30MB' THEN 25 PERCENT
        WHEN region = 'W' THEN 40 PERCENT
        ELSE 10 PERCENT
        END;
EMPID PRODUCT                REGION AMOUNT
----- -------------------- ------ -----------
1      PCGOLD, 30MB             E 30000.00
23     PCDIAMOND, 60MB          W 40000.00
29     GRAPHICPRINTER, M1       N 11000.00
32     GRAPHICPRINTER, M2       S 15000.00
39     GRAPHICPRINTER, M3       S 20000.00
75     LASERPRINTER, X1         W 42000.00
... ... ... ...

--- 30 row(s) selected.
```

5.4 TRANSPOSE 子句

5.4.1 TRANSPOSE 子句语法

SELECT 语句的 TRANSPOSE 子句可以为 SELECT 源表的每一行都生成转置项列表中每个项的行。TRANSPOSE 子句的结果表里带有源表的所有列（对于每个转置项列表，还有一列或多列值和一个可选的键列。

TRANSPOSE 子句是 EsgynDB SQL 扩展子句。

```
TRANSPOSE transpose-set [transpose-set]...
     [KEY BY key-colname]

transpose-set is:
     transpose-item-list AS transpose-col-list

transpose-item-list is:
     expression-list
     | (expression-list) [,(expression-list)]...

expression-list is:
```

```
     expression [,expression]...

transpose-col-list is:
     colname | (colname-list)

colname-list is:
     colname [,colname]...
```

transpose-item-list AS transpose-col-list

指定一个转置数据集，它可以将 transpose-item-list 与 transpose-col-list 相关联。transpose-item-list 既可以是一个表达式列表，也可以是一个用括号括起来的表达式列表。transpose-col-list 既可以是单个列名、也可以是用括号括起来的列名的列表。

例如：在转置数据集 TRANSPOSE (A,X) 、(B,Y) 、(C,Z) AS (V1,V2)里，transpose-item-list 里的项目为(A,X)、(B,Y)和(C,Z)，transpose-col-list 为(V1,V2)。各项中的表达式数量都必须等于“列”列表里的值列数量。

在 TRANSPOSE A,B,C AS V 例子中，项目是 A、B 和 C，值列是 V。这种形式是 TRANSPOSE (A),(B),(C) AS (V)更简短的书写方式。

transpose-item-list

指定项目列表。项目是一个数值表达式或括号括起来的数值表达式列表。

expression-list

指定一个以逗号分隔的 SQL 数值表达式列表。这些表达式必须具有兼容的数据类型。

例如：转置数据集 TRANSPOSE A,B,C AS V 里的表达式 A、B 和 C 具有兼容的数据类型。

(expression-list) [,(expression-list)]...

指定一个括号内表达式列表，后面跟有另一个括号内表达式列表，等等。每个列表的括号里的表达式数量必须相等。括号里相同序数位置的表达式必须具有兼容的数据类型。

例如：在转置数据集 TRANSPOSE (A,X),(B,Y),(C,Z) AS (V1,V2)里，表达式 A、B 和 C 具有兼容的数据类型，表达式 X、Y 和 Z 也有兼容的数据类型。

transpose-col-list

指定多个（包含项目列表中表达式求值结果的）列作为表达式应用到源表的行中。

`colname`

这是一种 SQL 标识符。它指定列名，并标识含有 expression-list 里数值的列。

例如：在转置数据集 TRANSPOSE A,B,C AS V 里，列 V 对应于表达式 A、B 和 C 的数值。

`(colname-list)`

指定用括号括起来的列名的列表。每个列都包含转置项列表括号内相同序数位置的表达式的数值。

例如：在转置数据集 TRANSPOSE (A,X),(B,Y),(C,Z) AS (V1,V2)里，列 V1 对应于表达式 A、B 和 C，列 V2 则对应于表达式 X、Y 和 Z。

KEY BY `key-colname`

可选择地通过在项目列表里所处的位置指定具体的表达式（与“转置列列表”里的数值相对应）。key-colname 是一种 SQL 标识符。该键列的数据类型为精确数字型，数值为 NOT NULL（非空）。

5.4.2 TRANSPOSE 子句注意事项

5.4.2.1 多个 TRANSPOSE 子句和集合

- 多个 TRANSPOSE 子句可以用在同一个查询里。例如：

```
SELECT KEYCOL1, VALCOL1, KEYCOL2, VALCOL2 FROM MYTABLE
TRANSPOSE A, B, C AS VALCOL1
   KEY BY KEYCOL1
TRANSPOSE D, E, F AS VALCOL2
   KEY BY KEYCOL2
```

- 一个 TRANSPOSE 子句可以包含多个转置数据集。例如：

```
SELECT KEYCOL, VALCOL1, VALCOL2 FROM MYTABLE
TRANSPOSE A, B, C AS VALCOL1
   D, E, F AS VALCOL2 KEY BY KEYCOL
```

5.4.2.2 TRANSPOSE 结果的度和列序

TRANSPOSE 结果的度等于源表（如果已指定，源表即为 FROM 子句和 WHERE 子句的表引用或引用所派生的结果表）的度加 1（如果指定了键列），再加上所有转置列列表的基数。

针对 TRANSPOSE 结果的列的排序操作从源表的列开始，其次是键列（如果已

指定)，再次是指定顺序的列名列表。

5.4.2.3 DTRANSPOSE 结果的数据类型

每个数值列的数据类型都是 transpose-item-list 里相应表达式的联合兼容型数据类型。表达式的数据类型必须在 transpose-item-list 里兼容。

例如：在 TRANSPOSE (A,X),(B,Y),(C,Z) AS (V1,V2)里，V1 的数据类型为 A、B、C 联合兼容型，V2 的数据类型为 X、Y、Z 联合兼容型。更多信息，请参阅可比和兼容的数据类型。

5.4.2.4 TRANSPOSE 结果的基数

每个 transpose-item-list 里的项目都从 1 列举到 N，这里的 N 是转置数据集所有项目列表里的项目总数。

在这个含有一个转置数据集的例子里，N 的数值为 3:

```
TRANSPOSE (A,X),(B,Y),(C,Z) AS (V1,V2)
```

在这个含有两个转置数据集的例子里，N 的数值为 5:

```
TRANSPOSE (A,X),(B,Y),(C,Z) AS (V1,V2) L,M AS V3
```

数值 1 到 N 为键值 ki。每个 transpose-item-list 里的项目都是表达式数值 vi。

TRANSPOSE 子句结果的基数等于源表的基数乘以 N（所有转置项列表里的项目总数)。

对于源表的每一行和键值 ki 里的每个值而言，TRANSPOSE 结果里都包含一行，其带有源表的所有属性、

键列里的键值 ki、相应转置数据集的数值列里的表达式数值 vi 和其他转置数据集的值列里的 NULL。

例如：思考一下这个 TRANSPOSE 子句:

```
TRANSPOSE (A,X),(B,Y),(C,Z) AS (V1,V2)
          L,M AS V3
     KEY BY K
```

N 的数值为 5。SELECT 源表的一行可以产生以下 TRANSPOSE 结果:

源列	K	V1	V2	V3
源行	1	value-of-A	value-of-X	NULL
源行	2	value-of-B	value-of-Y	NULL
源行	3	value-of-C	value-of-Z	NULL

源行	4	NULL	NULL	value-of-L
源行	5	NULL	NULL	value-of-M

5.4.3 TRANSPOSE 子句例子

- 假设创建表 MYTABLE 如下:

```
CREATE TABLE mining.mytable
( A INTEGER, B INTEGER, C INTEGER, D CHAR(2), E CHAR(2),
F CHAR(2) );
```

表 MYTABLE 具有列 A、B、C、D、E 和带相关数据的 F。列 A、B 和 C 均为 INTEGER（整型），列 D、E 和 F 均为 CHAR（字符型）。

A	B	C	D	E	F
1	10	100	d1	e1	f1
2	20	200	d2	e2	F2

- 假设表 MYTABLE 仅有前三列 A、B 和 C。TRANSPOSE 子句的执行结果里的行数是 MYTABLE 现有行数的三倍（因为转置项列表里有三个项目）:

```
SELECT * FROM mytable TRANSPOSE A, B, C AS VALCOL
KEY BY KEYCOL;
```

TRANSPOSE 查询的结果表为:

A	B	C	D	E	F	KEYCOL	VALCOL
1	10	100	d1	e1	f1	1	1
1	10	100	d1	e1	f1	2	10
1	10	100	d1	e1	f1	3	100
2	20	200	d2	e2	f2	1	2
2	20	200	d2	e2	f2	2	20
2	20	200	d2	e2	f2	3	200

- 该查询显示转置项列表里的项目可以是任何有效的标量表达式:

```
SELECT KEYCOL, VALCOL, A, B, C FROM mytable TRANSPOSE A
+ B, C + 3, 6 AS VALCOL
KEY BY KEYCOL;
```

TRANSPOSE 查询的结果表为:

KEYCOL	VALCOL	A	B	C
1	1	1	10	100
2	103	1	10	100
3	6	1	10	100
1	22	2	20	200
2	203	2	20	200
3	6	2	20	200

- 该查询显示如何同时使用 TRANSPOSE 子句和 GROUP BY 子句。

  该查询用于获取交叉表信息，这里的 A、B 和 C 是自变量，D 是因变量。

```
SELECT KEYCOL, VALCOL, D, COUNT(*) FROM mytable
TRANSPOSE A, B, C AS VALCOL
KEY BY KEYCOL
GROUP BY KEYCOL, VALCOL, D;
```

  TRANSPOSE 查询的结果表为:

KEYCOL	VALCOL	D	COUNT(*)
1	1	d1	1
2	10	d1	1
3	100	d1	1
1	2	d2	1
2	20	d2	1
3	200	d2	1

- 该查询显示如何将 COUNT 应用到 VALCOL。TRANSPOSE 查询的结果表显示 VALCOL 里不同值的数量

```
SELECT COUNT(DISTINCT VALCOL) FROM mytable TRANSPOSE A,
B, C AS VALCOL
KEY BY KEYCOL GROUP BY KEYCOL;

(EXPR)
--------------------
2
2
2

--- 3 row(s) selected.
```

- 该查询可以显示如何在一个查询里使用多个TRANSPOSE子句。该查询结果表里的行数是表MYTABLE现有行数的9倍:

```
SELECT KEYCOL1, VALCOL1, KEYCOL2, VALCOL2 FROM mytable
TRANSPOSE A, B, C AS VALCOL1
KEY BY KEYCOL1
TRANSPOSE D, E, F AS VALCOL2 KEY BY KEYCOL2;
```

TRANSPOSE查询的结果表为:

KEYCOL1	VALCOL1	KEYCOL2	VALCOL2
1	1	1	d1
1	1	2	e1
1	1	3	f1
2	10	1	d1
2	10	2	e1
2	10	3	f1
3	100	1	d1
3	100	2	e1
3	100	3	f1
1	2	1	d2
1	2	2	e2
1	2	3	f2
2	20	1	d2
2	20	2	e2
2	20	3	f2
3	200	1	d2
3	200	2	e2
3	200	3	f2

- 该查询显示一个 TRANSPOSE 子句如何包含多个转置数据集，即多个transpose-item-list AS transpose-col-list。其中的表达式A、B和C均为整型，表达式D、E和F则均为字符型。

```
SELECT KEYCOL, VALCOL1, VALCOL2 FROM mytable TRANSPOSE
A, B, C AS VALCOL1
D, E, F AS VALCOL2 KEY BY KEYCOL;
```

TRANSPOSE查询的结果表为:

KEYCOL	VALCOL1	VALCOL2

1	1	?
2	10	?
3	100	?
4	?	d1
5	?	e1
6	?	f1
1	2	?
2	20	?
3	200	?
4	?	d2
5	?	e2
6	?	f2

一个值列里出现问号（?）表明给定的 KEYCOL 里没有数值。

- 该查询可以显示前一个查询如何包含一个 GROUP BY 子句：

```
SELECT KEYCOL, VALCOL1, VALCOL2, COUNT(*) FROM mytable
TRANSPOSE A, B, C AS VALCOL1
D, E, F AS VALCOL2 KEY BY KEYCOL
GROUP BY KEYCOL, VALCOL1, VALCOL2;
```

TRANSPOSE 查询的结果表为：

KEYCOL	VALCOL1	VALCOL2	(EXPR)
1	1	?	1
2	10	?	1
3	100	?	1
1	2	?	1
2	20	?	1
3	200	?	1
4	?	d2	1
5	?	e2	1
6	?	f2	1
4	?	d1	1
5	?	e1	1
6	?	f1	1

- 该查询显示转置项列表里的一项如何包含一个表达式列表。这里可选用 KEY BY 子句也可不选：

```
SELECT * FROM mytable
```

```
TRANSPOSE (1, A, 'abc'), (2, B, 'xyz') AS (VALCOL1,
VALCOL2, VALCOL3);
```

TRANSPOSE 查询的结果表为:

A	B	C	D	E	F	VALCOL1	VALCOL2	VALCOL3
1	10	100	d1	e1	f1	1	1	abc
1	10	100	d1	e1	f1	2	10	xyz
2	20	200	d2	e2	f2	1	2	abc
2	20	200	d2	e2	f2	2	20	xyz

6. SQL 函数和表达式

本章描述在 EsgynDB SQL 语句里使用的特定函数和表达式的语法和语义。这些函数和表达式按照各自的功能分类。

本章目录:

6.1 类别
6.2 标准规范化
6.3 ABS 函数
6.4 AES_ENCRYPT 函数
6.5 ACOS 函数
6.6 ADD_MONTHS 函数
6.7 ASCII 函数
6.8 ASIN 函数
6.9 ATAN 函数
6.10 ATAN2 函数
6.11 AUTHNAME 函数
6.12 AVG 函数
6.13 BITAND 函数
6.14 BUFFERTOLOB 函数
6.15 CASE（条件）表达式
6.16 CAST 表达式
6.17 CEILING 函数
6.18 CHAR 函数
6.19 CHAR_LENGTH 函数
6.20 COALESCE 函数
6.21 CODE_VALUE 函数
6.22 CONCAT 函数
6.23 CONVERTTOHEX 函数
6.24 CONVERTTIMESTAMP 函数
6.25 COS 函数
6.26 COSH 函数
6.27 COUNT 函数
6.28 CRC32 函数
6.29 CURRENT 函数
6.30 CURRENT_DATE 函数

6.31 CURRENT_TIME 函数
6.32 CURRENT_TIMESTAMP 函数
6.33 CURRENT_USER 函数
6.34 DATE_ADD 函数
6.35 DATE_SUB 函数
6.36 DATEADD 函数
6.37 DATEDIFF 函数
6.38 DATEFORMAT 函数
6.39 DATE_PART 函数（时间间隔）
6.40 DATE_PART 函数（时间戳）
6.41 DATE_TRUNC 函数
6.42 DAY 函数
6.43 DAYNAME 函数
6.44 DAYOFMONTH 函数
6.45 DAYOFWEEK 函数
6.46 DAYOFYEAR 函数
6.47 DECODE 函数
6.48 DEGREES 函数
6.49 EMPTY_BLOB 函数
6.50 EMPTY_CLOB 函数
6.51 EXP 函数
6.52 EXPLAIN 函数
6.53 EXTERNALTOLOB 函数
6.54 EXTRACT 函数
6.55 FILETOLOB 函数
6.56.FIND_IN_SET 函数
6.57 FLOOR 函数
6.58 GROUP_CONCAT 函数
6.59 HOUR 函数
6.60 INSERT 函数
6.61 IS_IPV4 函数
6.62 IS_IPV6 函数
6.63 INET_ATION 函数
6.64 INET_NTOA 函数

6.65 ISNULL 函数
6.66 JULIANTIMESTAMP 函数
6.67 LCASE 函数
6.68 LEFT 函数
6.69 LISTAGG 函数
6.70 LOCATE 函数
6.71 LOG 函数
6.72 LOG10 函数
6.73 LOWER 函数
6.74 LPAD 函数
6.75 LTRIM 函数
6.76 MAX/MAXIMUM 函数
6.77 MD5 函数
6.78 MIN 函数
6.79 MINUTE 函数
6.80 MOD 函数
6.81 MONTH 函数
6.82 MONTHNAME 函数
6.83 NULLIF 函数
6.84 NULLIFZERO 函数
6.85 NVL 函数
6.86 NVL2 函数
6.87 OCTET_LENGTH 函数
6.88 OVERLAY 函数
6.89 PI 函数
6.90 PIVOT 函数
6.91 POSITION 函数
6.92 POWER 函数
6.93 QUARTER 函数
6.94 RADIANS 函数
6.95 RAND 函数
6.96 REGEXP_COUNT 函数
6.97 REGEXP_REPLACE 函数
6.98 REGEXP_SUBSTR 函数

6.99 REPEAT 函数
6.100 REPLACE 函数
6.101 REVERSE 函数
6.102 RIGHT 函数
6.103 ROLLUP 函数
6.104 ROUND 函数
6.105 RPAD 函数
6.106 RTRIM 函数
6.107 SECOND 函数
6.108 SHA 函数
6.109 SHA2 函数
6.110 SIGN 函数
6.111 SIN 函数
6.112 SINH 函数
6.113 SLEEP 函数
6.114 SPACE 函数
6.115 SPLIT_PART 函数
6.116 SQRT 函数
6.117 STDDEV 函数
6.118 STRINGTOLOB 函数
6.119 SUBSTRING/SUBSTR 函数
6.120 SUM 函数
6.121 SYSDATE 函数
6.122 SYSTIMESTAMP 函数
6.123 SYS_GUID 函数
6.124 TAN 函数
6.125 TANH 函数
6.126 TIMESTAMPADD 函数
6.127 TIMESTAMPDIFF 函数
6.128 TO_CHAR 函数
6.129 TO_DATE 函数
6.130 TO_TIME 函数
6.131 TO_TIMESTAMP 函数
6.132 TRANSLATE 函数

6.133 TRIM 函数
6.134 TRUNC 函数
6.135 UCASE 函数
6.136 UPPER 函数
6.137 UPSHIFT 函数
6.138 USER 函数
6.139 UUID 函数
6.140 UUID_SHORT 函数
6.141 UNIX_TIMESTAMP 函数
6.142 VARIANCE 函数
6.143 WEEK 函数
6.144 YEAR 函数
6.145 ZEROIFNULL 函数

6.1 类别

在一个 SQL 数值表达式里使用这些类型的函数:

聚合（集合）函数
字符串函数
日期时间函数
数学函数
加密函数
其他函数和表达式

更多有关 SQL 数值表达式的信息，请参阅表达式。

6.2 标准规范化

对于日期时间函数而言，标准规范化是指如果结果日期的结束日无效，则该日将被四舍五入到结果月份的最后一天。

6.2.1 聚合（集合）函数

一个聚合（集合）函数运行在通过 SELECT 语句检索到的一组或多组行上或存在聚合函数的子查询上。

AVG 函数	为函数表达式参数求值所派生的一组数字计算平均值。
COUNT 函数	对（使用*）得到的查询结果行进行计

	数，或对函数表达式参数所派生的单列表里包含不同值的行进行计数（可选不同的值）。
MAX/MAXIMUM 函数	从表达式参数求值所派生的一组数值中确定最大值。
MIN 函数	从表达式参数求值所派生的一组数值中确定最小值。
STDDEV 函数	从函数表达式参数求值所派生的一组数值中计算统计学标准差。这些数字可以加权处理。
SUM 函数	从函数表达式参数求值所派生的一组数值中计算总和。
VARIANCE 函数	从函数表达式参数求值所派生的一组数值中计算统计方差。这些数字可以加权处理。

一个聚合函数的参数可以是列和表达式。表达式不得包含聚合函数或子查询。一个聚合函数可以接受一个指定为 DISTINCT 的参数，这样可以在应用该函数之前消除重复值。更多信息，请参阅 DISTINCT 聚合函数。

如果您将一个 GROUP BY 子句包含在 SELECT 语句里，则您在选择列表里引用的列必须是分组列或一个聚合函数的参数。如果您没有包含一个 GROUP BY 子句，但您在选择列表里指定了一个聚合函数，则 SELECT 结果表里的所有行将构成唯一的组。更多信息，请参阅该函数的单个条目。

在 Oracle 的聚合函数中，如果聚合函数期望一个数值类型的输入，但是给的是一个字符类型的输入，那么 Oracle 会尝试把该字符类型隐式转换成数值类型，然后再进行后续的聚合操作。为兼容 Oracle，EsgynDB 在聚合函数中也支持字符串到数值的隐式转换。

例子:

```
>>select sum('100') from (values(1),(2)) ;

(EXPR)
-----------------------------------------------

200
--- 1 row(s) selected
```

6.2.2 字符串函数

这些函数可以操作字符串，使用一个字符值表达式作为一个参数或返回一个字符数据类型的结果。字符串函数将一个输入字符串里的单字节或多字节字符视为一个字符，不管该字符的字节长度如何。

ASCII 函数	返回一个字符值表达式的首字符的 ASCII 代码值。
CHAR 函数	返回一个字符集里的指定代码值。
CHAR_LENGTH 函数	返回一个字符串里的字符数。您也可使用 CHARACTER_LENGTH 函数。
CODE_VALUE 函数	返回一个无符号的整数，它是一个字符值表达式里的首字符的代码点，可以关联一种受支持的字符集。
CONCAT 函数	返回两个字符值表达式的串联作为一个字符串值。您也可使用串联运算符（||）。
INSERT 函数	返回一个字符串，其中指定数量的字符已被删除，并且另一个字符串被插入到指定的起始位置。
LCASE 函数	该函数将字符串中的大写字母改为小写字母。您也可使用 LOWER。
LEFT 函数	返回一个字符表达式最左侧的指定数量的字符。
LOCATE 函数	返回一个字符串里的指定子字符串所处的位置。您也可使用 POSITION 函数。
LOWER 函数	该函数将字符串中的大写字母改为小写字母。您也可使用 LCASE。
LPAD 函数	将一个字符表达式最左侧的指定数量的字符替换为一个填充字符。
LTRIM 函数	删除一个字符串里的前导空格。
OCTET_LENGTH 函数	返回一个字符串的长度，以字节为单位。
POSITION 函数	返回字符串中指定子字符串所处的位

	置。您也可使用 LOCATE 函数。
REPEAT 函数	返回一个字符串，其中包含按指定次数重复出现的字符表达式求值。
REPLACE 函数	将原始字符串里出现的所有指定字符串都替换为另一个字符串后，再返回由此产生的字符串。
RIGHT 函数	返回一个字符表达式最右侧指定数量的字符。
RPAD 函数	将一个字符表达式最右侧指定数量的字符替换为一个填充字符。
RTRIM 函数	删除一个字符串里的结尾空格。
SPACE 函数	返回指定数量的空格字符串。
SUBSTRING/SUBSTR 函数	从一个字符串里抽取一个子字符串。
TRANSLATE 函数	将一个字符串从源字符集转换为目标字符集。
TRIM 函数	删除一个字符串里的前导字符或尾随字符。
UCASE 函数	将字符串中的小写字母改为大写字母。您也可使用 UPSHIFT 或 UPPER。
UPPER 函数	升档字母数字字符。您也可使用 UPSHIFT 或 UCASE。
UPSHIFT 函数	升档字母数字字符。您也可使用 UPPER 或 UCASE。

更多信息，请参阅该函数的单个条目。

6.2.3 日期时间函数

这些函数使用一个 Datetime 值表达式作为参数或返回一个数据类型为 Datetime 的结果：

名称	描述	返回数据类型
ADD_MONTHS 函数	将 intr_expr 指定月份的整数值与 datetime_expr 相加，并将结果规范化。	DATE

CONVERTTIMESTAMP 函数	将一个儒略历时间戳转换为 TIMESTAMP 值。	TIMESTAMP
CURRENT 函数	返回当前时间戳。您也可使用 CURRENT_TIMESTAMP 函数。	TIMESTAMP
CURRENT_DATE 函数	返回当前日期。	DATE
CURRENT_TIME 函数	返回当前时间。	TIME
CURRENT_TIMESTAMP 函数	返回当前时间戳。您也可使用 CURRENT 函数。	TIMESTAMP
DATE_ADD 函数	将 interval_表达式指定的时间间隔与 datetime_expr 相加。	DATE
（一个 Interval 的）DATE_PART 函数	从 Interval 指定的时间间隔值里抽取文本指定的 Datetime 字段，并返回结果作为精确的数字值。	NUMERIC
（一个 Timestamp）的 DATE_PART 函数	从 Timestamp 指定的日期时间值里抽取文本指定的 Datetime 字段，并返回结果作为精确的数字值。	NUMERIC
DATE_SUB 函数	从 datetime_expr 里减去指定的 interval_expression。	DATE 或 TIMESTAMP
DATE_TRUNC 函数	返回一个带有截断日时间部分的日期。	TIMESTAMP
DATEADD 函数	将 datepart 和 num_expr 指定的时间间隔与 datetime_expr 相加。	DATE 或 TIMESTAMP
DATEDIFF 函数	返回一个整数值作为起始日期和结束日期之间的 datepart 时间单元数量。	INTEGER
DATEFORMAT 函数	格式化日期时间值（以便显示）。	CHAR
DAY 函数	返回一个整数值，代表该月的相应日，范围是 1 到 31。您也可使用 DAYOFMONTH 函数。	INTEGER (从 1 到 31)
DAYNAME 函数	从一个日期或时间戳表达式里返回一周中的一日的名称。	CHAR
DAYOFMONTH 函数	返回一个范围介于 1 和 31 之间的整数值，它代表一个月里	INTEGER (从 1 到 31)

	的相应日。您也可使用 DAY。	
DAYOFWEEK 函数	返回一个整数值，它代表一周中的相应日，范围是 1 到 7。	INTEGER (从 1 到 7)
DAYOFYEAR 函数	返回一个整数值，它代表一年中的相应日，范围是 1 到 366。	INTEGER (从 1 到 366)
EXTRACT 函数	从日期时间值表达式或时间间隔值表达式里返回指定的日期时间字段。	NUMERIC
HOUR 函数	返回一个范围介于 0 和 23 之间的整数值，它代表一日中的相应小时。	INTEGER (从 0 到 23)
JULIANTIMESTAMP 函数	将一个日期时间值转换为儒历时间戳。	TIMESTAMP
MINUTE 函数	返回一个范围介于 0 和 59 之间的整数值，它代表一小时中的相应分钟。	INTEGER (从 0 到 59)
MONTH 函数	返回一个范围介于 1 和 12 之间的整数值，它代表一年中的相应月份。	INTEGER (从 1 到 12)
MONTHNAME 函数	返回一个字符字面值，它代表一年中的月份名称（一月、二月等）。	CHAR
QUARTER 函数	返回一个范围介于 1 和 4 之间的整数值，它代表一年中的相应季度。	INTEGER (从 1 到 4)
SECOND 函数	返回一个范围介于 0 和 59 之间的整数值，它代表一分钟内的相应秒数。	INTEGER (从 0 到 59)
SYSDATE 函数	检索服务器的当前日期而不是会话的当前日期。	DATE
SYSTIMESTAMP 函数	检索服务器的当前日期和时间 (包括精度为六位数的小数秒)，而不是会话的当前日期和时间。	TIMESTAMP
TIMESTAMPADD 函数	将 interval-ind（时间间隔指示）和 num_expr 指定的时间间隔加入 datetime_expr。	DATE 或 TIMESTAMP

TIMESTAMPDIFF 函数	返回一个整数值作为起始日期和结束日期之间的 interval-ind 单元数量。	INTEGER
TO_CHAR 函数	将日期时间值转换为字符值。	CHAR
TO_DATE 函数	将字符值转换为日期或时间戳值。	DATE 或 TIMESTAMP
TO_TIME 函数	将字符值转换为时间值。	TIME
TO_TIMESTAMP 函数	将字符值转换为时间戳值。	TIMESTAMP
WEEK 函数	返回一个范围介于 1 和 54 之间的整数值，代表一年中相应的一周。	INTEGER (从 1 到 54)
UNIX_TIMESTAMP 函数	返回一个 Unix 时间戳	INTEGER
YEAR 函数	返回一个代表年份的整数值。	INTEGER

更多信息，请参阅该函数的单个条目。

6.2.4 LOB 函数

EsgynDB 提供以下 LOB 函数来处理 LOB 数据。

EMPTY_BLOB() 函数	创建一个类型为 BLOB 的虚拟 LOB 句柄。
EMPTY_CLOB() 函数	创建一个类型为 CLOB 的虚拟 LOB 句柄。
STRINGTOLOB 函数	将简单的字符串文字转换为 LOB 格式。
FILETOLOB 函数	将本地 linux/hdfs 文件中的数据转换为 LOB 格式。
BUFFERTOLOB_函数	获取输入缓冲区的地址和大小，并将该缓冲区的内容转换为 LOB 格式。
EXTERNALTOLOB_函数	将外部文件中的数据转换为 LOB 格式。
LOBTOSTRING	将 LOB 数据转换为简单的字符串文字。
LOBTOFILE	将 LOB 数据转换为 linux/hdfs 文件。
LOBTOBUFFER	将 LOB 数据转换为缓冲区。

6.2.5 数学函数

在一个 SQL 数字值表达式里使用以下数学函数:

ABS 函数	返回一个数字值表达式的绝对值。
ACOS 函数	用以弧度表示的角度作为一个数字值表达式的反余弦返回结果。
ASIN 函数	用以弧度表示的角度作为一个数字值表达式的反正弦返回结果。
ATAN 函数	用以弧度表示的角度作为一个数字值表达式的反正切返回结果。
ATAN2 函数	返回两个数字值表达式指定的 x 和 y 坐标的反正切作为一个以弧度表示的角度。
CEILING 函数	返回大于或等于一个数字值表达式的最小整数。
COS 函数	返回一个数字值表达式的余弦，这里的表达式是一个以弧度表示的角度。
CRC32 函数	返回 CRC32 校验和
COSH 函数	返回一个数字值表达式的双曲余弦，这里的表达式是一个以弧度表示的角度。
DEGREES 函数	将一个用弧度表示的数字值表达式转换为角度数。
EXP 函数	返回一个数字值表达式的指数值（以 e 为底）。
FLOOR 函数	返回小于或等于一个数字值表达式的最大整数。
LOG 函数	返回一个数字值表达式的自然对数。
LOG10 函数	返回一个数字值表达式的以 10 为底的对数。
MOD 函数	返回一个整数值表达式除以另一个整数值表达式所得的余数（模数）。
NULLIFZERO 函数	返回操作数的值（如果它为 0，则将返回 NULL）。
PI 函数	将 pi 常数值作为浮点值返回。
POWER 函数	返回一个数字值表达式的（整数值表

	达式）次方得到的数值。您还可使用指数运算符**。
RADIANS 函数	将一个用角度表示的数字值表达式转换为弧度数。
ROUND 函数	返回 numeric_expr 的数值，四舍五入到小数点右侧 num 位。
SIGN 函数	返回一个数字值表达式的符号标志。如果该数值小于 0，则将返回-1 作为标志。如果该数值等于 0，则将返回 0；如果该数值大于零，则将返回 1。
SIN 函数	返回一个数字值表达式的正弦，这里的表达式是一个以弧度表示的角度。
SINH 函数	返回一个数字值表达式的双曲正弦，这里的表达式是一个以弧度表示的角度。
SQRT 函数	返回一个数字值表达式的平方根。
TAN 函数	返回一个数字值表达式的正切，这里的表达式是用弧度表示的角度。
TANH 函数	返回一个数字值表达式的双曲正切，这里的表达式是一个以弧度表示的角度。
ZEROIFNULL 函数	返回操作数的值（它为 NULL，则将返回 0）。

更多信息，请参阅该函数的单个条目。

6.2.6 加密函数

在一个 SQL 值表达式里使用这些函数来执行数据加密或散列：

MD5 函数	返回 MD5 校验和
SHA 函数	返回 SHA-1 160 位校验和
SHA2 函数	返回 SHA-2 校验和

6.2.7 舍入函数

EsgynDB 提供了以下舍入函数，这些函数在某些情况下可以互换。

6.2.7.1 舍入函数的比较

函数	语法	描述	备注
CEIL 函数 (CEILING)	CEIL (*numericexpression*) CEILING (*numericexpression*)	将 *numericexpression* **向上**舍入为整数。它与 FLOOR 函数**相反**。	结果的刻度为 0。
FLOOR 函数	FLOOR (*numericexpression*)	将 *numericexpression* **向下**舍入为整数。它与 CEIL 函数**相反**。	
ROUND 函数	ROUND (*numericexpression*, *decimalplace*)	将小数位 decimalplace 上的 numericexpression 舍入。这是最复杂的四个函数之一。更多信息，请参阅 ROUND 函数。	结果与 *numericexpression* 具有相同的刻度。
TRUNC 函数 (TRUNCATE)	TRUNC (*numericexpression*, *decimalplace*) TRUNCATE (*numericexpression*, *decimalplace*)	截断小数位 *decimalplace* 上的 *numericexpression*，没有舍入。	在**减少小数位数**方面，TRUNC 函数与 ROUND 函数类似。 但是，TRUNC 函数不像 ROUND 函数那样执行任何舍入。

舍入小数

- **小数点右侧有多位数字**

函数	**负小数**	**正小数**
CEIL (CEILING)	CEIL (-0.123456789)=0	CEIL (0.123456789)=1
FLOOR	FLOOR (-0.123456789)=-1	FLOOR (0.123456789)=0
ROUND	ROUND (-0.123456789,0)=0.000000000	ROUND (0.123456789,0)=0.000000000

	ROUND (-0.123456789,1)=-0.100000000 ROUND (-0.123456789,2)=-0.120000000 ROUND (-0.123456789,3)=-0.123000000 ROUND (-0.123456789,4)=-0.123500000 ROUND (-0.123456789,5)=-0.123460000 ROUND (-0.123456789,6)=-0.123457000 ROUND (-0.123456789,7)=-0.123456800 ROUND (-0.123456789,8)=-0.123456790 ROUND (-0.123456789,9)=-0.123456789 ROUND (-0.123456789,10)=-0.123456789	ROUND (0.123456789,1)=0.100000000 ROUND (0.123456789,2)=0.120000000 ROUND (0.123456789,3)=0.123000000 ROUND (0.123456789,4)=0.123500000 ROUND (0.123456789,5)=0.123460000 ROUND (0.123456789,6)=0.123457000 ROUND (0.123456789,7)=0.123456800 ROUND (0.123456789,8)=0.123456790 ROUND (0.123456789,9)=0.123456789 ROUND (0.123456789,10)=0.123456789
TRUNC (TRUNCATE)	ROUND (-0.123456789,-1)=0.000000000 ROUND (-0.123456789,-2)=0.000000000 ROUND (-0.123456789,-3)=0.000000000 ROUND (-0.123456789,-4)=0.000000000 ROUND (-0.123456789,-5)=0.000000000 ROUND (-0.123456789,-6)=0.000000000 ROUND (-0.123456789,-7)=0.000000000 ROUND (-0.123456789,-8)=0.000000000 ROUND (-0.123456789,-9)=0.000000000 ROUND (-0.123456789,-10)=0.000000000 TRUNC (-0.123456789,0)=0 TRUNC (-0.123456789,1)=-0.1 TRUNC (-0.123456789,2)=-0.12 TRUNC (-0.123456789,3)=-0.123 TRUNC (-0.123456789,4)=-0.1234 TRUNC (-0.123456789,5)=-0.12345 TRUNC (-0.123456789,6)=-0.123456 TRUNC (-0.123456789,7)=-0.1234567 TRUNC (-0.123456789,8)=-0.12345678 TRUNC (-0.123456789,9)=-0.123456789 TRUNC (-0.123456789,10)=-0.1234567890	ROUND (0.123456789,-1)=0.000000000 ROUND (0.123456789,-2)=0.000000000 ROUND (0.123456789,-3)=0.000000000 ROUND (0.123456789,-4)=0.000000000 ROUND (0.123456789,-5)=0.000000000 ROUND (0.123456789,-6)=0.000000000 ROUND (0.123456789,-7)=0.000000000 ROUND (0.123456789,-8)=0.000000000 ROUND (0.123456789,-9)=0.000000000 ROUND (0.123456789,-10)=0.000000000 TRUNC (0.123456789,0)=0 TRUNC (0.123456789,1)=0.1 TRUNC (0.123456789,2)=0.12 TRUNC (0.123456789,3)=0.123 TRUNC (0.123456789,4)=0.1234 TRUNC (0.123456789,5)=0.12345 TRUNC (0.123456789,6)=0.123456 TRUNC (0.123456789,7)=0.1234567 TRUNC (0.123456789,8)=0.12345678 TRUNC (0.123456789,9)=0.123456789 TRUNC (0.123456789,10)=0.1234567890

- **小数点左侧有多位数字**

函数	负小数	正小数
CEIL (CEILING)	CEIL (-123456789.1)= -123456789	CEIL (123456789.1)=123456790
FLOOR	FLOOR (-123456789.1)= -123456790	FLOOR (123456789.1)=123456789
ROUND	ROUND (-123456789.1,0)=-123456789.0 ROUND (-123456789.1,1)=-123456789.1 ROUND (-123456789.1,2)=-123456789.1 ROUND (-123456789.1,3)=-123456789.1 ROUND (-123456789.1,4)=-123456789.1 ROUND (-123456789.1,5)=-123456789.1 ROUND (-123456789.1,6)=-123456789.1 ROUND (-123456789.1,7)=-123456789.1 ROUND (-123456789.1,8)=-123456789.1 ROUND (-123456789.1,9)=-123456789.1 ROUND (-123456789.1,10)=-123456789.1 ROUND (-123456789.1,-1)=-123456790.0 ROUND (-123456789.1,-2)=-123456800.0 ROUND (-123456789.1,-3)=-123457000.0 ROUND (-123456789.1,-4)=-123460000.0 ROUND (-123456789.1,-5)=-123500000.0 ROUND (-123456789.1,-6)=-123000000.0 ROUND (-123456789.1,-7)=-120000000.0 ROUND (-123456789.1,-8)=-100000000.0 ROUND (-123456789.1,-9)=0.0 ROUND (-123456789.1,-10)=0.0	ROUND (123456789.1,0)=123456789.0 ROUND (123456789.1,1)=123456789.1 ROUND (123456789.1,2)=123456789.1 ROUND (123456789.1,3)=123456789.1 ROUND (123456789.1,4)=123456789.1 ROUND (123456789.1,5)=123456789.1 ROUND (123456789.1,6)=123456789.1 ROUND (123456789.1,7)=123456789.1 ROUND (123456789.1,8)=123456789.1 ROUND (123456789.1,9)=123456789.1 ROUND (123456789.1,10)=123456789.1 ROUND (123456789.1,-1)=123456790.0 ROUND (123456789.1,-2)=123456800.0 ROUND (123456789.1,-3)=123457000.0 ROUND (123456789.1,-4)=123460000.0 ROUND (123456789.1,-5)=123500000.0 ROUND (123456789.1,-6)=123000000.0 ROUND (123456789.1,-7)=120000000.0 ROUND (123456789.1,-8)=100000000.0 ROUND (123456789.1,-9)=0.0 ROUND (123456789.1,-10)=0.0
TRUNC (TRUNCATE)	TRUNC (-123456789.1,0)=-123456789 TRUNC (-123456789.1,1)=-123456789.1	TRUNC (123456789.1,0)=123456789 TRUNC (123456789.1,1)=123456789.1

- **舍入整数（零）**

函数	零
CEIL (CEILING)	CEIL (0)=0

FLOOR	FLOOR (0)=0
ROUND	ROUND (0)=0
TRUNC (TRUNCATE)	TRUNC (0)=0

- **舍入整数（向下舍入）**

函数	负整数	正整数
CEIL (CEILING)	CEIL (-333)=-333	CEIL (333)=333
FLOOR	FLOOR (-333)=-333	FLOOR (333)=333
ROUND	ROUND (-333,0)=-333 ROUND (-333,1)=-333 ROUND (-333,2)=-333 ROUND (-333,3)=-333 ROUND (-333,4)=-333 **ROUND (-333,-1)=-330** **ROUND (-333,-2)=-300** **ROUND (-333,-3)=0** ROUND (-333,-4)=0	ROUND (333,0)=333 ROUND (333,1)=333 ROUND (333,2)=333 ROUND (333,3)=333 ROUND (333,4)=333 **ROUND (333,-1)=330** **ROUND (333,-2)=300** **ROUND (333,-3)=0** ROUND (333,-4)=0
TRUNC (TRUNCATE)	TRUNC (-333,0)=-333 TRUNC (-333,1)=-333	TRUNC (333,0)=333 TRUNC (333,1)=333

- **舍入整数（向上舍入）**

函数	负整数	正整数
CEIL (CEILING)	CEIL (-888)=-888	CEIL (888)=888
FLOOR	FLOOR (-888)=-888	FLOOR (888)=888
ROUND	ROUND (-888,0)=-888 ROUND (-888,1)=-888 ROUND (-888,2)=-888 ROUND (-888,3)=-888 ROUND (-888,4)=-888	ROUND (888,0)=888 ROUND (888,1)=888 ROUND (888,2)=888 ROUND (888,3)=888 ROUND (888,4)=888
TRUNC (TRUNCATE)	TRUNC (-888,0)=-888 TRUNC (-888,1)=-888	TRUNC (888,0)=888 TRUNC (888,1)=888

6.2.8 其他函数和表达式

在一个 SQL 值表达式里使用以下其他函数和表达式:

AUTHNAME 函数	返回与特定授权 ID 号相关联的授权名称。
BITAND 函数	在两个操作数的相应位上执行'AND'运算。
CASE (条件) 表达式	一个条件表达式。两种形式的 CASE 表达式分别为 CASE 简单表达式和 CASE 搜索表达式。
CAST 表达式	将一个值从一种类型转换为另一种数据类型。
COALESCE 函数	返回不含 NULL 值的列表里的第一个表达式的值；另外如果所有的表达式都具有 NULL 值，该函数也将返回一个 NULL 值。
CONVERTTOHEX 函数	将指定的值表达式转换为 16 进制数值，以便进行显示。
CURRENT_USER 函数	返回调用该函数的当前用户的数据库用户名。
DECODE 函数	按照所提供的顺序将 expr 与每个 test_expr 的数值逐

	一进行比较。
EXPLAIN 函数	生成一个结果表，其描述 SELECT、INSERT、DELETE 或 UPDATE 语句的访问计划。
FIND_IN_SET 函数	返回某个字符串在指定字符串集中的位置。
ISNULL 函数	返回第一个参数，如果它不为空；否则它将返回第二个参数。
IS_IPV4 函数	如果参数是一个被指定为字符串的有效 IPv4 地址，则返回 1，否则返回 0。
IS_IPV6 函数	如果参数是一个被指定为字符串的有效 IPv6 地址，则返回 1，否则返回 0。
INET_ATON 函数	给定一个以字符串表示的 IPv4 网络地址的点分表示形式，返回一个整数，该整数以网络字节顺（大端）表示地址的数值。如果 INET_ATON()不理解其参数，则返回 NULL。
INET_NTOA 函数	给定一个以网络字节序的数字 IPv4 网络地址，将该地址的点分字符串表示形式返回为连接字符集中的一个非二进制字符串。如果 INET_NTOA()不理解其参数，则返回 NULL。
NULLIF 函数	如果两个操作数不相等，则返回第一个操作数的值；否则返回 NULL。
NVL 函数	返回第一个操作数的值。如果它为 NULL，则将返回第二个操作数的值。
USER 函数	返回调用该函数的当前用户的数据库用户名或与指定的用户 ID 号相关联的数据库用户名。
SYS_GUID 函数	返回一个全局唯一标识符，类似于 Oracle 的 SYS_GUID 函数。
UUID 函数	返回一个全局唯一标识符。
UUID_SHORT 函数	将“短”通用标识符作为一个 64 位无符号整数返回。
SLEEP 函数	休眠几秒钟

更多信息，请参阅该函数的单个输入项。

6.2.9 标准规范化

对于日期时间函数而言，标准规范化是指如果结果日期的结束日无效，则该日将被四舍五入到结果月份的最后一天。

6.3 ABS 函数

ABS 函数返回一个值表达式的绝对值。它是一种 EsgynDB SQL 扩展函数。

ABS (numeric-expression)

numeric-expression

这是一种 SQL 数字值表达式，它指定 ABS 函数的参数值.

如果参数的精度小于 10，返回的结果将是一个无符号的数字型数值；如果参数的精度大于或等于 10，返回的结果将是一个 LARGEINT（大整型）。

6.3.1 ABS 函数例子

- 该函数返回数值 8:

  ```
  ABS (-20 + 12)
  ```

6.4 AES_ENCRYPT 函数

AES_ENCRYPT 函数使用一种指定的运用 AES（高级加密标准）算法的加密密钥对字符串进行加密，并返回一个二进制字符串。

该函数可防止敏感数据对公众可见。例如，使用 AES_ENCRYPT 函数来加密个人身份证号码或保护用户机密性。

AES_ENCRYPT (*string*, *key*)

6.4.1 AES_ENCRYPT 语法

- *string*

是要加密的数据，它可以包含大写字母和小写字母的混合，并包含数字和特殊字符。

注意：此参数区分大小写。

- *key*

是用来加密字符串的加密密钥，它可以包含大写字母和小写字母的混合，并包含数字和特殊字符。

注意：此参数区分大小写。

可以使用加密散列函数（如 SHA、SHA2 和 MD5）对此密钥进行散列处理。

例子

- 该例子演示了 AES_ENCRYPT 函数如何与 SHA 函数一起使用。

```
SELECT AES_ENCRYPT ('hello',SHA ('world')) FROM DUAL;
(EXPR)
----------------
[}âXéò0Êw«æÝAÆc
--- 1 row(s) selected.
```

- 该例子演示了 AES_ENCRYPT 函数如何与 SHA2 函数一起使用。

SQL 参考手册

```
SELECT AES_ENCRYPT ('hello',SHA2 ('world',512)) FROM DUAL;
(EXPR)
----------------
¤÷èx¼ó=L*Ì-µO*A
--- 1 row(s) selected.
```

- 该例子演示了 AES_ENCRYPT 函数如何与 MD5 函数一起使用。

```
SELECT AES_ENCRYPT ('hello',MD5 ('world')) FROM DUAL;
(EXPR)
```

```
----------------
C¡ïcåsí[1Ø
--- 1 row(s) selected.
```

注意：CQD BLOCK_ENCRYPTION_MODE 控制基于块的加密算法的模式。默认模式是 aes-128-ecb 表示使用长度为 128 位的密钥和 ECB（电子密码本）模式进行加密——对称加密方案，用密文块替换明文块。

数值	模式
0	aes-128-ecb
1	aes-192-ecb
2	aes-256-ecb

例子

以下例子显示了基于 CQD BLOCK_ENCRYPTION_MODE 的值从 AES_ENCRYPT 函数返回的不同结果。

- 模式 aes_192_ecb 生效。

```
CQD BLOCK_ENCRYPTION_MODE '1';
--- SQL operation complete.
SELECT AES_ENCRYPT ('Technical_Writer','Coder') FROM DUAL;
(EXPR)
---------------------------------
4}\îî¢Øã^ûh<á¼Æ7õé³ê!cH^Ð
```

该例子使用 CONVERTTOHEX 函数将加密的字符串转换为十六进制字符串来显示。

```
 SELECT              CONVERTTOHEX              (AES_ENCRYPT
('Technical_Writer','Coder')) FROM DUAL;
(EXPR)
------------------------------------------------------------
-------
2DB19032AD86EFEC2D37AC670D1993DAE2E9FA6CC08B044C9989BB260
0007B7B
--- 1 row(s) selected.
```

- 模式 aes_256_ecb 生效。

```
CQD BLOCK_ENCRYPTION_MODE '2';
--- SQL operation complete.
SELECT AES_ENCRYPT ('Technical_Writer','Coder') FROM DUAL;
(EXPR)
---------------------------------
Ô_8ÆðÑ=Uú2g1¥@°Ô3yä
--- 1 row(s) selected.
```

```
SELECT              CONVERTTOHEX              (AES_ENCRYPT
('Technical_Writer','Coder')) FROM DUAL;
(EXPR)
------------------------------------------------------------
-------
2DB19032AD86EFEC2D37AC670D1993DAE2E9FA6CC08B044C9989BB260
0007B7B
--- 1 row(s) selected.
```

6.4.2 AES_ENCRYPT 注意事项

两个参数都不能为 NULL。

6.4.3 AES_ENCRYPT 例子

- 该例子使用密钥*'888'*对字符串*'2019-11-02 00:00:00'*进行加密。

```
SELECT AES_ENCRYPT('2019-11-02 00:00:00', '888') FROM
DUAL;
(EXPR)
---------------------------------
°Mµ-b(úaP\ìqXqsnTç<`®
--- 1 row(s) selected.
```

- 该例子使用密钥'Germany-10:06'对字符串'Oct 9, 2013 was my first day in Muenster'进行加密，并将加密后的字符串插入表 t1。

```
INSERT INTO t1 VALUES (AES_ENCRYPT('Oct 9, 2013 was my
first day in
Muenster','Germany-10:06'), 'Anniversary');
--- 1 row(s) inserted.

SELECT * FROM t1;
C1
C2
------------------------------------------------------
------------------------------
---------------
------------------------------------------------------
------------------------------
---------------
lnWEÂªç¸
}öÓ/Å±ÄÖ
P'5]û-S°±[Üý6)Òâ(
Anniversary
--- 1 row(s) selected.
```

6.5 ACOS 函数

ACOS 函数返回以弧度表示的角度作为一个数字值表达式的反余弦。

ACOS 是 EsgynDB SQL 扩展 ACOS（numeric-expression）数字表达式。

这是一种 SQL 数字值表达式，它指定 ACOS 函数的参数值。参数值的范围是-1 到+1。更多信息，请参阅字符值表达式。

6.5.1 ACOS 函数例子

- ACOS 函数返回数值 3.49044274380724416E-001 或约 0.3491 弧度（20 度）。

  ```
  ACOS (0.9397)
  ```

- 该函数返回数值 0.3491。ACOS 函数是 COS 函数的反函数。

  ```
  ACOS (COS (0.3491))
  ```

6.6 ADD_MONTHS 函数

ADD_MONTHS 函数将 int_expr 指定的整数与 datetime_expr 相加，并对结果进行规范化。

ADD_MONTHS 是 EsgynDB SQL 扩展函数。

```
ADD_MONTHS (datetime_expr, int_expr [, int2 ])
```

datetime_expr

该表达式的求值结果为 DATE 或 TIMESTAMP 类型的日期时间值，返回值与 *datetime_expr* 具有相同的数据类型。更多信息，请参阅日期时间值表达式。

int_expr

这是一种 SQL 数字值表达式，数据类型为 SMALLINT 或 INTEGER，它指定月数。更多信息，请参阅字符值表达式。

int2

这是一个无符号的整型常数。

如果 int2 被省略或为字面值 0，则这种规范化即为标准规范化；如果 int2 为字面值 1，则这种规范化将包括标准规范化；如果起始日（*datetime_expr* 的日部分）是起始月份的最后一日，则结束日（结果值的日部分）将被设置为结果月份的最后一个有效日。更多信息，请参阅标准规范化和字符值表达式。

6.6.1 ADD_MONTHS 函数例子

- 该函数返回 DATE 值 '2007-03-31':
  ```
  ADD_MONTHS(DATE '2007-02-28', 1, 1)
  ```
- 该函数返回 DATE 值 '2007-03-28':
  ```
  ADD_MONTHS(DATE '2007-02-28', 1, 0)
  ```
- 该函数返回 DATE 值 '2008-03-28':
  ```
  ADD_MONTHS(DATE '2008-02-28', 1, 1)
  ```
- 该函数返回时间戳'2009-02-28 00:00:00':
  ```
  ADD_MONTHS(timestamp'2008-02-29 00:00:00',12,1)
  ```

6.7 ASCII 函数

ASCII 函数返回一个整数，即 ISO8891 字符集或 UTF8 字符集所关联的字符串表达式里的首字符的 ASCII 代码。

ASCII 函数是 EsgynDB SQL 扩展函数。

```
ASCII (character-expression)
character-expression
```

这是一种 SQL 字符值表达式，它指定由多个字符组成的字符串。更多信息，请参阅字符值表达式。

6.7.1 ASCII 函数注意事项

ASCII 函数以不同方式处理字符串字面值和列：

- 对于一个字符串来说，此字符的值或字符串中每个字符的值不能大于 255，否则将返回以下错误：

  ```
  *** ERROR[4106] The character set for the operand of
  function ASCII must be ISO88591
  ```

  例子

  ```
  SQL>SELECT ASCII('ÿ') FROM DUAL;
  (EXPR)
  ----------
  255
  --- 1 row(s) selected.

  SQL>SELECT ASCII('u') FROM DUAL;
  (EXPR)
  ----------
  117
  --- 1 row(s) selected.

  SQL>SELECT ASCII('ÿu') FROM DUAL;
  (EXPR)
  ----------
  255
  --- 1 row(s) selected.

  SQL>SELECT ASCII('¬') FROM DUAL;
  *** ERROR[4106] The character set for the operand of
  function ASCII must be
  ISO88591. [2018-05-07 06:01:55]

  SQL>SELECT ASCII('ÿ¬') FROM DUAL;
  *** ERROR[4106] The character set for the operand of
  function ASCII must be
  ```

```
ISO88591. [2018-05-08 08:21:34]
```

- 对于一个列来说，此列中的字符串表达式只能是单字节，否则将返回以下错误：

```
*** ERROR[8428] The argument to function ASCII is not valid.
```

- 对于一个被定义为 UTF8 的列中的字符串来说，字符串首字节的值不能大于 127。
- 对于一个被定义为 ISO88591 的列中的字符串来说，字符串首字节的值可以是 ISO88591 字符集中的所有字符。

例子

列定义	UTF8	UTF8
表结构	SQL>SHOWDDL t25; CREATE TABLE TRAFODION.SEABASE.T25 (C1 CHAR(4 CHARS) CHARACTER SET UTF8 COLLATE DEFAULT DEFAULT NULL NOT SERIALIZED) ATTRIBUTES ALIGNED FORMAT ; --- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON TRAFODION.SEABASE.T25 TO DB__ROOT WITH GRANT OPTION; --- SQL operation complete.	SQL>SHOWDDL t24; CREATE TABLE TRAFODION.SEABASE.T24 (C1 CHAR(4 CHARS) CHARACTER SET UTF8 COLLATE DEFAULT DEFAULT NULL NOT SERIALIZED) ATTRIBUTES ALIGNED FORMAT ; --- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON TRAFODION.SEABASE.T24 TO DB__ROOT WITH GRANT OPTION; --- SQL operation complete.
数据	SQL>SELECT * FROM t25; C1 ---- ¬u --- 1 row(s) selected.	SQL>SELECT * FROM t24; C1 ---- u¬ --- 1 row(s) selected.
结果	SQL>SELECT ASCII(c1) FROM t25; ***ERROR[8428] The argument to function ASCII is not valid. [2018-05-08 10:14:08]	SQL>SELECT ASCII(c1) FROM t24; (EXPR) ---------- 117 --- 1 row(s) selected.

列定义	UTF8	ISO99591
表结构	SQL>SHOWDDL t11; CREATE TABLE TRAFODION.SEABASE.T11 (C1 CHAR(4 CHARS) CHARACTER SET UTF8 COLLATE DEFAULT DEFAULT NULL NOT SERIALIZED) ATTRIBUTES	SQL>SHOWDDL t12; CREATE TABLE TRAFODION.SEABASE.T12 (C1 CHAR(4) CHARACTER SET ISO88591 COLLATE DEFAULT DEFAULT NULL NOT SERIALIZED) ATTRIBUTES

	ALIGNED FORMAT ; --- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON TRAFODION.SEABASE.T11 TO DB__ROOT WITH GRANT OPTION; --- SQL operation complete.	ALIGNED FORMAT ; --- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON TRAFODION.SEABASE.T12 TO DB__ROOT WITH GRANT OPTION; --- SQL operation complete.
数据	SQL>SELECT * FROM t11; C1 ---- ÿ --- 1 row(s) selected.	SQL>SELECT * FROM t12; C1 ---- ÿ --- 1 row(s) selected.
结果	SQL>SELECT ASCII(c1) FROM t1; ***ERROR[8428] The argument to function ASCII is not valid. [2018- 05-07 02:13:42]	SQL>SELECT ASCII(c1) FROM t2; (EXPR) ---------- 255 --- 1 row(s) selected.

列定义	**UTF8**	**ISO99591**
表结构	SQL>SHOWDDL t17; CREATE TABLE TRAFODION.SEABASE.T17 (C1 CHAR(4 CHARS) CHARACTER SET UTF8 COLLATE DEFAULT DEFAULT NULL NOT SERIALIZED) ATTRIBUTES ALIGNED FORMAT ; --- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON TRAFODION.SEABASE.T17 TO DB__ROOT WITH GRANT OPTION; --- SQL operation complete.	SQL>SHOWDDL t18; CREATE TABLE TRAFODION.SEABASE.T18 (C1 CHAR(4) CHARACTER SET ISO88591 COLLATE DEFAULT DEFAULT NULL NOT SERIALIZED) ATTRIBUTES ALIGNED FORMAT ; --- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON TRAFODION.SEABASE.T18 TO DB__ROOT WITH GRANT OPTION; --- SQL operation complete.
数据	SQL>SELECT * FROM t17; C1 ---- uÿ --- 1 row(s) selected.	SQL>SELECT * FROM t18; C1 ---- uÿ --- 1 row(s) selected.
结果	SQL>SELECT ASCII(c1) FROM t17; (EXPR) ---------- 117 --- 1 row(s) selected.	SQL>SELECT ASCII(c1) FROM t18; (EXPR) ---------- 117 --- 1 row(s) selected.

列定义	**UTF8**	**ISO99591**
表结构	SQL>SHOWDDL t22; CREATE TABLE TRAFODION.SEABASE.T22 (C1 CHAR(4 CHARS) CHARACTER SET UTF8 COLLATE DEFAULT DEFAULT NULL NOT SERIALIZED)	SQL>SHOWDDL t23; CREATE TABLE TRAFODION.SEABASE.T23 (C1 CHAR(4) CHARACTER SET ISO88591 COLLATE DEFAULT DEFAULT NULL NOT SERIALIZED)

	ATTRIBUTES ALIGNED FORMAT ; --- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON TRAFODION.SEABASE.T22 TO DB__ROOT WITH GRANT OPTION; --- SQL operation complete.	ATTRIBUTES ALIGNED FORMAT ; --- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON TRAFODION.SEABASE.T23 TO DB__ROOT WITH GRANT OPTION; --- SQL operation complete.
数据	SQL>SELECT * FROM t22; C1 ---- uÿÿ --- 2 row(s) selected.	SQL>SELECT * FROM t23; C1 ---- uÿÿ --- 2 row(s) selected.
结果	SQL>SELECT ASCII(c1) FROM t22; ***** ERROR[8428] The argument to function ASCII is not valid. [2018-05-08 10:01:29]	SQL>SELECT ASCII(c1) FROM t23; (EXPR) ---------- 117 255 --- 2 row(s) selected.

列定义	**UTF8**	**ISO99591**
表结构	SQL>SHOWDDL t20; CREATE TABLE TRAFODION.SEABASE.T20 (C1 CHAR(4 CHARS) CHARACTER SET UTF8 COLLATE DEFAULT DEFAULT NULL NOT SERIALIZED) ATTRIBUTES ALIGNED FORMAT ; ¬—¬GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON TRAFODION.SEABASE.T20 TO DB__ROOT WITH GRANT OPTION; --- SQL operation complete.	SQL>SHOWDDL t21; CREATE TABLE TRAFODION.SEABASE.T21 (C1 CHAR(4) CHARACTER SET ISO88591 COLLATE DEFAULT DEFAULT NULL NOT SERIALIZED) ATTRIBUTES ALIGNED FORMAT ; --- GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON TRAFODION.SEABASE.T21 TO DB__ROOT WITH GRANT OPTION; --- SQL operation complete.
数据	SQL>SELECT * FROM t20; C1 ---- uÿuÿ --- 2 row(s) selected.	SQL>SELECT * FROM t21; C1 ---- uÿuÿ --- 2 row(s) selected.
结果	SQL>SELECT ASCII(c1) FROM t20; (EXPR) ---------- 117 117 --- 2 row(s) selected	SQL>SELECT ASCII(c1) FROM t21; (EXPR) ---------- 117 117 --- 2 row(s) selected.

6.7.2 ASCII 函数例子

选择 JOBDESC 列，并返回 job description（职位描述）首字符的 ASCII 代码：

```
SELECT jobdesc, ASCII (jobdesc) FROM persnl.job;

JOBDESC            (EXPR)
----------------- --------
```

```
MANAGER           77
PRODUCTION SUPV   80
ASSEMBLER         65
SALESREP          83
...               ...
--- 10 row(s) selected.
```

6.8 ASIN 函数

ASIN 函数返回一个数字值表达式的反正弦作为一个以弧度表示的角度。

ASIN 函数是 EsgynDB SQL 扩展函数。

ASIN (numeric-expression)

numeric-expression

这是一种 SQL 数字值表达式，它指定 ASIN 函数的参数值。

参数值的范围是-1 到 +1。更多信息，请参阅字符值表达式。

6.8.1 ASIN 函数例子

- 该函数返回数值 3.49044414403046400E-001 或近似值 0.3491 弧度（即 20 度）:
  ```
  ASIN (0.3420)
  ```

- 该函数返回数值 0.3491。ASIN 函数是 SIN 函数的反函数。
  ```
  ASIN (SIN (0.3491))
  ```

6.9 ATAN 函数

ATAN 函数返回一个数字值表达式的反正切作为一个以弧度表示的角度。

ATAN 函数是 EsgynDB SQL 扩展函数。

ATAN (numeric-expression)

numeric-expression

这是一种 SQL 数字值表达式，它指定 ATAN 函数的参数值。更多信息，请参阅字符值表达式。

6.9.1 ATAN 函数例子

- 该函数返回数值 8.72766423249958272E-001 或近似值 0.8727 弧度（即 50 度）:
  ```
  ATAN (1.192)
  ```

- 该函数返回数值 0.8727。ATAN 函数是 TAN 函数的反函数。
  ```
  ATAN (TAN (0.8727))
  ```

6.10 ATAN2 函数

ATAN2 函数返回 x 和 y 坐标的反正切，两个数字值表达式指定作为一个以弧度表示的角度。

ATAN2 函数是 EsgynDB SQL 扩展函数。

```
ATAN2 (numeric-expression-x,numeric-expression-y)
numeric-expression-x, numeric-expression-y
```

这些都是 SQL 数字值表达式，它们指定 ATAN2 函数的 x 和 y 坐标参数值。更多信息，请参阅字符值表达式。

6.10.1 ATAN2 函数例子

- 该函数返回数值 2.66344329881899520E+000 或近似值 2.6634：
  ```
  ATAN2 (1.192,-2.3)
  ```

6.11 AUTHNAME 函数

AUTHNAME 函数返回与指定的授权 ID 号相关联的授权 ID 名称。

```
AUTHNAME (auth-id)
auth-id
```

这是一个与授权 ID 相关联的 32 位数。更多信息，请参阅授权 ID。

AUTHNAME 函数类似 USER 函数。

6.11.1 AUTHNAME 函数注意事项

- 该函数仅可指定在 SELECT 语句里。
- 返回值为字符串数据类型 VARCHAR(128)，它采用 ISO8859-1 编码。

6.11.2 AUTHNAME 函数例子

- 该例子显示与授权 ID 号 33333 相关联的授权名称：

```
>>SELECT AUTHNAME (33333) FROM (values(1)) x(a);
(EXPR)
-------------------------
DB__ROOT
--- 1 row(s) selected.
```

6.12 AVG 函数

AVG 是一个聚合函数，它返回一组数字的平均值。

```
AVG ([ALL | DISTINCT] expression)
```

`ALL | DISTINCT`

指定在计算一个表达式的 AVG 时是否包含重复值。

默认选项为 ALL，由此使重复值被包含在内。如果指定 DISTINCT 选项，那么在应用 AVG 函数之前，重复值将被消除。

expression

指定一个数字或时间间隔值表达式，它确定需要进行平均的数值。该表达式不得包含一个聚合函数或子查询。DISTINCT 子句指定 AVG 函数运行在表达式求值所派生的单列表的不同数值上。更多信息，请参阅字符值表达式和时间间隔值表达式。

6.12.1 AVG 函数注意事项

6.12.1.1 结果的数据类型

结果的数据类型取决于参数的数据类型。对于一个精确数字型参数而言，结果为 LARGEINT；对于一个近似数字型参数而言，该函数的结果将是 DOUBLE PRECISION；对于一个 INTERVAL 型参数而言，该函数的结果将是一个与它具有相同精度的 INTERVAL。

该函数的结果与参数具有相同的标度。如果该参数没有标度，则结果将被截断。

6.12.1.2 表达式的操作数

该表达式包含 SELECT 结果表的行里的列，但不得包含一个聚合函数。以下为有效表达式：

```
AVG (SALARY)
AVG (SALARY * 1.1)
AVG (PARTCOST * QTY_ORDERED)
```

6.12.1.3 空值

将该函数应用到这组数值以前必须消除所有空值。如果结果表为空， AVG 将返回 NULL。

6.12.1.4 AVG 函数例子

- 返回 SALARY 列的平均值：

```
SELECT AVG (salary) FROM persnl.employee;

(EXPR)
---------------------
49441.52

--- 1 row(s) selected.
```

- 返回 SALARY 列唯一值的平均值:

```
SELECT AVG(DISTINCT salary) AS Avg_Distinct_Salary
FROM persnl.employee;
AVG_DISTINCT_SALARY
---------------------
53609.89

--- 1 row(s) selected.
```

- 按部门返回平均工资:

```
SELECT deptnum, AVG (salary) AS "AVERAGE SALARY"
FROM persnl.employee
WHERE deptnum < 3000
GROUP BY deptnum;

Dept/Num "AVERAGE SALARY"
-------- ---------------------
1000     52000.17
2000     50000.10
1500     41250.00
2500     37000.00

--- 4 row(s) selected.
```

6.13 BITAND 函数

BITAND 函数在两个操作数的对应位上执行 AND 运算。如果两个位都是 1，则结果位也是 1；否则，结果位是 0。

```
BITAND (expression, expression)
```

expression

该函数的结果是一个二进制数字。根据不同的操作数精度，结果的数据类型可以是 INT（32 位整数）或 LARGEINT（64 位整数）。

如果一个操作数的最大精度大于 9，则将选择 LARGEINT（精度大于 9 的数字被表示为 LARGEINT）。否则将选择 INT。

如果两个操作数都是无符号数，则结果也是无符号数。否则，结果将是有符号数。

执行按位运算之前应把两个操作数都转换为结果的数据类型。

6.13.1 BITAND 函数注意事项

BITAND 可以用在一条 SQL 查询里使用表达式（其中包括 SELECT 列表、WHERE 谓词、VALUES 子句和 SET 语句等）的任何地方。

该函数可以返回一种数字数据类型，也可用在算术表达式里。

数字型操作数可以是正数或负数。所有的数字数据类型都允许出现 BITAND 函数的使用限制里所列的异常情况。

6.13.1.1 BITAND 函数使用限制

BITAND 函数的使用限制如下：

- 必须有两个操作数。
- 操作数必须为二进制或十进制精确数字。
- 操作数的标度必须为 0。
- 操作数不能是浮点数。
- 操作数不能是扩展精度数字型（Numeric 扩展数据类型的最大精度是 128 位）。

6.13.2 BITAND 函数例子

```
>>select bitand(1,3) from (values(1)) x(a);

(EXPR)
--------------
```

```
             1

--- 1 row(s) selected

>>select 1 & 3 from (values(1)) x(a);

(EXPR)
--------------
             1

--- 1 row(s) selected

>>select bitand(1,3) + 0 from (values(1)) x(a);

(EXPR)
--------------
             1

--- 1 row(s) selected
```

6.14 BUFFERTOLOB 函数

6.14.1 BUFFERTOLOB 函数语法

BUFFERTOLOB 函数获取输入缓冲区的地址和大小，并将该缓冲区的内容转换为 LOB。该函数可以在 INSERT 或 UPDATE 语句中使用。

BUFFERTOLOB 函数是 EsgynDB SQL 扩展函数。
更多信息，请参阅 EsgynDB SQL Larege Objects 指南。

```
BUFFERTOLOB(LOCATION lob source buffer address, LENGTH
lob length value)
```

- *lob source buffer address*

用户缓冲区地址的 int64 值。

- *lob length value*

用户指定的 lob 缓冲区的 int64 值。

6.14.2 BUFFERTOLOB 函数例子

- 该例子将 int64 值作为输入，它是缓冲区的地址和大小参数。缓冲区内容被转换为 LOB 格式并存储在 HDFS 中。

  ```
  insert into tlob1 values (1, buffertolob(LOCATION
  124647474, SIZE 2048));
  ```

- 在表 tlob1 中，该例子将 c2 更新（覆盖）在 1254674 处的缓冲区位置，长度为 4000 字节。

  ```
  update tlob1 set c2=buffertolob(LOCATION 12546474, SIZE
  4000);
  ```

6.15 CASE（条件）表达式

CASE 表达式是一种条件表达式，它有两种形式：CASE 简单表达式和 CASE 搜索表达式。

在一个 CASE 简单表达式里，EsgynDB SQL 会将一个数值与一个数值序列进行比较，并将 CASE 表达式设置为第一个匹配的关联数值（如果该匹配存在）；否则，EsgynDB SQL 将返回 ELSE 子句里指定的数值（可以为空）。

在 CASE 搜索表达式里，EsgynDB SQL 会对一个条件序列进行求值，并将该 CASE 表达式设置为第一个为真的条件的关联数值（如果存在一个为真的条件）；如果不存在为真的条件，EsgynDB SQL 将返回 ELSE 子句里指定的数值（可以为空）。

CASE 简单表达式为：

```
CASE case-expression
     WHEN expression-1 THEN {result-expression-1 | NULL}
     WHEN expression-2 THEN {result-expression-2 | NULL}
     ...
     WHEN expression-n THEN {result-expression-n | NULL}
     [ELSE {result-expression | NULL}]
END
```

```
Searched CASE is:
CASE
     WHEN condition-1 THEN {result-expression-1 | NULL}
     WHEN condition-2 THEN {result-expression-2 | NULL}
     ...
     WHEN condition-n THEN {result-expression-n | NULL}
     [ELSE {result-expression | NULL}]
END
```

case-expression

指定一个值表达式与 CASE 简单表达式的每条 WHEN 子句里的值表达式进行比较。WHEN 子句里的每个表达式的数据类型必须都能与 case-expression 的数据类型进行比较。

expression-1 ... expression-n

指定与每个与 result-expression 都相关联的一个数值。如果 WHEN 子句里一个表达式的数值匹配 case-expression 的数值，CASE 简单表达式返回所关联的 result-expression 值。如果不存在匹配，CASE 表达式将返回 ELSE 子句里指定的数值

表达式；如果没有指定 ELSE 值，CASE 表达式将返回 NULL。

result-expression-1 ... result-expression-n
指定 CASE 简单表达式的 WHEN 子句里的每个表达式所关联的结果值表达式或 CASE 搜索表达式的 WHEN 子句里的每个条件所关联的结果值表达式。
所有 result-expressions 都必须具有可比的数据类型，并且至少一个 result-expressions 必须返回非空值。

result-expression
紧跟在 ELSE 关键字后面，指定返回值（如果 CASE 简单表达式的 WHEN 子句里没有一个表达式与 CASE 表达式等值，或一个 CASE 搜索表达式的 WHEN 子句里没有一个条件为真）。如果没有指定 ELSE result-expression 子句，则 CASE 将返回 NULL。
result-expression 的数据类型必须可以与其他结果进行比较。

condition-1 ... condition-n
指定一个 CASE 搜索表达式里的待检验条件。
如果一个条件为真，则 CASE 表达式将返回相关联的结果表达式的数值。
如果没有条件为真，则 CASE 表达式将返回 ELSE 子句里指定的数值表达式。
如果没有指定 ELSE 值，则 CASE 表达式将返回 NULL。

6.15.1 CASE（条件）表达式注意事项

6.15.1.1 CASE 表达式数据类型

CASE 表达式结果的数据类型取决于结果表达式的数据类型，如果所有结果都具有相同的数据类型，则 CASE 表达式将采用该数据类型。如果这些结果具有可比、但不等效的数据类型，则 CASE 表达式将采用结果表达式的联合数据类型。综上所述，结果的数据类型可通过这些方式确定。

6.15.1.2 字符数据类型

如果结果表达式的任何数据类型都是可变长度字符串，则结果的数据类型也将是可变长度字符串，并且其最大长度等于结果表达式的最大长度。否则，如果没有一种数据类型是可变长度字符串，则结果的数据类型将是固定长度字符串，并且其长度等于结果表达式的最大长度。

6.15.1.3 数字数据类型

如果结果表达式的所有数据类型都是精确数字型，则结果的数据类型也将是精确数字型，并且其精度和标度等于结果表达式的最大精度和标度。

例如：如果 *result-expression-1* 和 *result-expression-2* 都具有数据类型 NUMERIC(5)，*result-expression-3* 具有数据类型 NUMERIC(8,5)，则结果的数据类型将为 NUMERIC(10,5)。

如果结果表达式里的任何数据类型都是近似数字型，则结果的数据类型也将是近似数字型，并且其精度等于结果表达式的最大精度。

6.15.1.4 日期时间数据类型

如果结果表达式的数据类型为日期时间型，则结果的数据类型也将是相同的日期时间型。

6.15.1.5 时间间隔数据类型

如果结果表达式的数据类型为时间间隔型，则结果的数据类型也将是相同的时间间隔型（year-month 或 day-time）；其中的起始字段为结果表达式的最高有效的起始字段，终止字段为结果表达式的最低有效的终止字段。

6.15.2 CASE（条件）表达式例子

- 如果 JOBCODE 不匹配任何列举值，使用一个 CASE 简单表达式对 JOBCODE 进行解码，并返回 NULL（空值）：

```
SELECT last_name, first_name, CASE jobcode
WHEN 100 THEN 'MANAGER'
WHEN 200 THEN 'PRODUCTION SUPV' WHEN 250 THEN 'ASSEMBLER'
WHEN 300 THEN 'SALESREP'
WHEN 400 THEN 'SYSTEM ANALYST' WHEN 420 THEN 'ENGINEER'
WHEN 450 THEN 'PROGRAMMER'
WHEN 500 THEN 'ACCOUNTANT'
WHEN 600 THEN 'ADMINISTRATOR ANALYST' WHEN 900 THEN
'SECRETARY'
ELSE NULL END
FROM persnl.employee;

LAST_NAME FIRST_NAME (EXPR)
--------- ----------  ----------
GREEN     ROGER       MANAGER
HOWARD    JERRY       MANAGER
RAYMOND   JANE        MANAGER
```

```
...
CHOU       JOHN         SECRETARY
CONRAD     MANFRED      PROGRAMMER
HERMAN     JIM          SALESREP
CLARK      LARRY        ACCOUNTANT
HALL       KATHRYN      SYSTEM ANALYST
...

--- 62 row(s) selected.
```

- 使用一个 CASE 搜索表达式返回 LAST_NAME、FIRST_NAME 和一个基于 SALARY（取决于 DEPTNUM 值）而定的值：

```
SELECT last_name, first_name, deptnum,
   CASE
        WHEN deptnum = 9000 THEN salary * 1.10
        WHEN deptnum = 1000 THEN salary * 1.12
        ELSE salary
   END
FROM persnl.employee;

LAST_NAME FIRST_NAME DEPTNUM (EXPR)
--------- ------------ ------- -------
GREEN     ROGER         9000 193050.0000
HOWARD    JERRY         1000 153440.1120
RAYMOND   JANE          3000 136000.0000
...
--- 62 row(s) selected.
```

6.16 CAST 表达式

CAST 表达式将数据转换为您指定的数据类型。

```
CAST ({expression | NULL} AS data-type)
```

expression | NULL

指定一个需要转换为数据类型 data-type 的操作数。

如果该操作数是一种表达式，则其数据类型将取决于该表达式的数据类型，并遵循 CAST 表达式有效转换中所列的规则。

无论您指定何种数据类型，如果该操作数为 NULL 或表达式的数值为空，则 CAST 表达式的结果也为 NULL。

data-type

指定 CAST 表达式的操作数所关联的一种数据类型。更多信息，请参阅数据类型。

当将数据转换为 CHAR 或 VARCHAR 数据类型时，所得到的数据值将向左对齐。否则，所得到的数据值将向右对齐。此外，如果您需要将数据转换为 CHAR 或 VARCHAR 数据类型，则必须指定目标值的长度。

6.16.1 CAST 表达式注意事项

当您使用 CAST 表达式将一个数字型数值转换为 INTERVAL 型数值时，小数部分将被丢弃。

根据您的文件设置方式，使用 CAST 表达式可能导致查询性能不佳，因为它会阻止优化器选择最有效的计划、并要求执行器执行全表或全索引扫描。

6.16.1.1 CAST 表达式有效转换

- 将一个精确或近似数字型数值转换为任何其他数字数据类型。
- 将一个精确或近似数字型数值转换为任何字符串数据类型。
- 将一个精确数字型数值转换为单个字段的 year-month（年-月型）或 day-time（日-时间型）时间间隔，诸如 INTERVAL DAY(2)。
- 将一个字符串转换为任何其他数据类型时，应遵循以下使用限制：
  待转换的字符串的内容必须与结果数据类型的含义一致。例如：如果您需要将一个字符串转换为 DATE 型，则该字符串的内容必须为 10 个字符，其中包含年份、一个连字符、月份、另一个连字符和日。
- 将一个时间值转换为一个字符串或 TIMESTAMP （EsgynDB SQL 会将

00:00:00.00 填入时间部分)。

- 将一个时间值转换为一个字符串或 TIMESTAMP (EsgynDB SQL 会将当前日期填入日期部分)。
- 将一个时间戳值转换为一个字符串、DATE、TIME 或另一个具有不同小数秒精度的 TIMESTAMP。
- 将一个 year-month 型时间间隔值转换为一个字符串、精确数字或另一个具有不同起始字段精度的 year-month 型 INTERVAL。
- 将一个 day-time 型时间间隔值转换为字符串、精确数字或另一个带有不同起始字段精度的 day-time 型 INTERVAL。

6.16.2 CAST 表达式例子

- 在该例子中，小数部分将被丢弃:

```
CAST (123.956 as INTERVAL DAY(18))
```

- 该例子返回两个时间戳之间的差值，以分钟为单位:

```
CAST((d.step_end - d.step_start) AS INTERVAL MINUTE)
```

- 假设您的数据库包含一个用户信息日志文件。这个例子将当前的时间戳转换为一个字符串，并将结果串联成一个字符字面值。必须指定长度。

```
INSERT INTO stats.logfile (user_key, user_info)
VALUES (001, 'User JBrook, executed at ' ||
   CAST (CURRENT_TIMESTAMP AS CHAR(26)));
```

6.17 CEILING 函数

该函数返回大于或等于一个数字值表达式的最小整数，它将被表示为 FLOAT 数据类型。

CEILING 函数是 EsgynDB SQL 扩展函数。

CEILING *(numeric-expression)*

这是一种 SQL 数字值表达式，它指定 CEILING 函数的参数值。更多信息，请参阅数字值表达式。

6.17.1 CEIL 语法

numeric-expression 是一种要四舍五入的 SQL 数字值表达式。

更多信息，请参阅数字值表达式。

- 如果 *numeric-expression* 是一个小数，则小数部分将被截断，结果的标度为 0。

  例子

  ```
  SELECT CEIL (-1.23), CEIL (1.23), CEIL (-12.3), CEIL
  (12.3) FROM DUAL;
  (EXPR) (EXPR) (EXPR) (EXPR)
  ------------- ------------ ------------ -----------
  -1 2 -12 13
  --- 1 row(s) selected.
  ```

- 如果 *numeric-expression* 是一个整数，则结果是其本身。

  例子

  ```
  SELECT CEIL (-99), CEIL (-1), CEIL (0), CEIL (1), CEIL
  (99) FROM DUAL;
  (EXPR) (EXPR) (EXPR) (EXPR) (EXPR)
  ------ ------ ------ ------ ------
  -99 -1 0 1 99
  --- 1 row(s) selected.
  ```

  •在两个相反的数中，正的*numeric-expression*结果的绝对值大于负的*numeric-expression*结果的绝对值。

  ```
  SELECT CEIL (-99.99), CEIL (99.99) FROM DUAL;
  (EXPR) (EXPR)
  ----------- ------
  -99 100
  --- 1 row(s) selected.
  ```

6.17.2 CEIL 注意事项

6.17.2.1 CEIL 与 FLOOR 的比较

FLOOR 函数与 CEIL 函数**相反**，它向下舍入 *numeric-expression* 并返回小于或等于 *numeric-expression* 的最大整数。

更多信息，请参阅舍入函数的比较。

```
            -1.5                                1.5
              ¬                                  ¬
---- -2 ----·---- -1 --------0-------- 1 ----·---- 2 ----
      ¬             ¬                   ¬            ¬
  FLOOR(-1.5) CEIL(-1.5)            FLOOR(1.5)   CEIL(1.5)
```

数字表达式	FLOOR	CEIL
0.3	0	1
1.8	1	2
2.37	2	3
3.84	3	4
4.123	4	5
5.999	5	6
-0.3	-1	0
-1.8	-2	-1
-2.37	-3	-2
-3.84	-4	-3
-4.123	-5	-4
-5.999	-6	-5

6.17.3 CEIL 例子

- 该例子使用 CEIL 函数将值向上舍入到下一个便士。

```
SELECT * FROM ORDERS;
CUSTOMER_NAME PRODUCT_NAME PRICE QUANTITY
--------------- ----------------- ----------------- ---
--------------
Kevin PC 7899.99 36
Andy Monitor 14566.78 3
Eason Keyboard 1099.73 45
Magic Memory 1634.74 57
William CPU 1940.98 75
Aven Hard Disk 2566.82 134
--- 6 row(s) selected.

SELECT  customer_name,  product_name,  CEIL  (price  *
quantity) AS total FROM orders
ORDER BY total;
CUSTOMER_NAME PRODUCT_NAME TOTAL
--------------- --------------- ---------------
Andy Monitor 43701
Eason Keyboard 49488
```

```
Magic Memory 93181
William CPU 145574
Kevin PC 284400
Aven Hard Disk 343954
--- 6 row(s) selected.
```

6.18 CHAR 函数

该函数返回一个具有指定代码值的字符 ，它必须是标度为 0 的精确数字型数据。CHAR 函数是 EsgynDB SQL 扩展函数。

```
CHAR(code-value, [,char-set-name])
```

code-value
这是正在使用的字符集里的有效代码值。

char-set-name
可以是 ISO88591 或 UTF8。所返回的字符必须与 *char-set-name* 指定的字符集相关联。*char-set-name* 的默认值为 ISO88591。

6.18.1 CHAR 函数注意事项

- 对于 ISO88591 字符集而言，返回类型将是 VARCHAR(1)。
- 对于 UTF8 字符集而言，返回类型将是 VARCHAR(1)。

6.18.2 CHAR 函数例子

- 选择列 CUSTNAME，并返回客户名首字符的 ASCII 代码和其 CHAR 值：

```
SELECT custname, ASCII (custname), CHAR (ASCII
(custname))
FROM sales.customer;

CUSTNAME           (EXPR)    (EXPR)
------------------ ------- -------
CENTRAL UNIVERSITY 67      C
BROWN MEDICAL CO   66      B
STEVENS SUPPLY     83      S
PREMIER INSURANCE  80      P
... ... ...
--- 15 row(s) selected.
```

6.19 CHAR_LENGTH 函数

CHAR_LENGTH 函数返回一个字符串里的字符数；另外，您也可以使用 CHARACTER_LENGTH 函数。包括多字节字符在内每个字符都将被计数为一个字符。

```
CHAR[ACTER]_LENGTH (string-value-expression)
```

`string-value-expression`
指定将一个字符值表达式并返回字符长度。EsgynDB SQL 返回的结果是一个标度为 0、两个字节的有符号整数。
如果 `string-value-expression` 为空，则 EsgynDB SQL 将返回一个空值长度。更多信息，请参阅字符值表达式。

6.19.1 CHAR_LENGTH 函数注意事项

6.19.1.1 CHAR 和 VARCHAR 操作数

对于一个被声明为固定 CHAR 的列而言，EsgynDB SQL 将返回该列的最大长度。对于一个 VARCHAR 列而言，EsgynDB SQL 将返回存储在该列里的字符串的实际长度。

6.19.2 CHAR_LENGTH 函数例子

- 该函数可以返回 12 作为结果。串联运算符为两条竖线（||）。
  ```
  CHAR_LENGTH ('ROBERT' || ' ' || 'SMITH')
  ```

- 字符串''是一个空串，该函数可以返回 0（零）:
  ```
  CHAR_LENGTH ('')
  ```

- DEPTNAME 列具有数据类型 CHAR(12)。因此，该函数将始终返回 12:
  ```
  CHAR_LENGTH (deptname)
  ```

- ROJECT 表里的 PROJDESC 列具有数据类型 VARCHAR(18)。由于它是一个 VARCHAR 值，所以该函数可以返回该列值的实际长度对于较短的字符串而言并非 18。
  ```
  SELECT CHAR_LENGTH (projdesc) FROM persnl.project;

  (EXPR)
  ----------
  ```

```
        14
        13
        13
        17
        9
        9
--- 6 row(s) selected.
```

6.20 COALESCE 函数

该函数返回列表里的第一个不含 NULL 的表达式数值。

如果所有表达式都包含 NULL，则该函数将返回 NULL。

```
COALESCE (expr1, expr2, ...)
```

expr1

一个需要进行比较的表达式

expr2

一个需要进行比较的表达式

6.20.1 COALESCE 函数例子

COALESCE 返回第一个不为 NULL（空值）的操作数的数值：

```
SELECT COALESCE (office_phone, cell_phone, home_phone, pager,
     fax_num, '411') from emptbl;
```

6.21 CODE_VALUE 函数

该函数返回一个无符号整数（INTEGER UNSIGNED），这是一个字符值表达式首字符的 ASCII 码值，它可以关联一种支持的字符集。

CODE_VALUE 函数是 EsgynDB SQL 扩展函数。

```
CODE_VALUE(character-value-expression)
     character-set
```

character-value-expression

是一个字符串。

6.21.1 CODE_VALUE 函数例子

- 该函数将返回 97 作为结果:

```
>>select code_value('abc') from (values(1))x;

(EXPR)
----------

          97
```

6.22 CONCAT 函数

该函数返回两个字符值表达式的串联作为一个字符串值。另外，您也可以使用串联运算符（||）。

CONCAT 函数是 EsgynDB SQL 扩展函数。

```
CONCAT (character-expr-1, character-expr-2)
```

character-expr-1, character-expr-2

这些都是 SQL 字符值表达式（数据类型 CHAR 或 VARCHAR），指定由多个字符组成的两个字符串。

这两个字符值表达式必须都是 ISO8859-1 字符表达式或 UTF8 字符表达式。

CONCAT 函数的结果是 *character-expr-1* 和 *character-expr-2* 的串联值。如果两个表达式都是 CHAR 类型，则结果的类型为 CHAR；如果两个表达式中的任何一个为 VARCHAR 类型，则结果的类型为 VARCHAR。更多信息，请参阅字符值表达式。

6.22.1 串联运算符 (||)

串联运算符被表示为两条竖线（||），它可以将两个字符串值串联起来构成一个新的字符串值。为了标明两个字符串已被串联，请用两条竖线（||）将它们连接起来。

```
character-expr-1 || character-expr-2
```

一个操作数可以是数据类型 CHAR 或 VARCHAR 的任何 SQL 值表达式。

6.22.2 CONCAT 函数注意事项

6.22.2.1 操作数

一个字符串值可以由任何字符值表达式（诸如字符串字面值、字符串函数、列引用、聚合函数、标量子查询、CASE 表达式或CAST 表达式）指定。操作数的值必须是 CHAR 或 VARCHAR 类型。

如果您使用了 CAST 表达式，则必须指定 CHAR 或 VARCHAR 的长度。

6.22.2.2 SQL 参数

您可以将一个 SQL 参数和一个字符值表达式串联起来。串联的参数将具备字符值表达式的数据类型属性。请思考下面这个例子，这里的?p 被赋予一个字符串值'5 March':

```
?p || ' 2002'
```

参数?p 的类型赋值将变为 CHAR(5)，与字符字面值'2002'具有相同的数据类型。因为您已经将一个包含 5 个字符以上的字符串值赋给?p，所以 EsgynDB SQL 将返回一个截断警告，串联结果为 5 Mar 2002。

为了指定该参数的类型赋值，请在它上面使用 CAST 表达式：

```
CAST(?p AS CHAR(7)) || '2002'
```

在这个例子中，该参数未被截断，串联结果为 5 March 2002。

6.22.3 CONCAT 函数例子

- 插入一条包含单个字符串的信息。使用 CONCAT 函数构造并插入数值：

```
INSERT INTO stats.logfile (user_key, user_info)
VALUES (001, CONCAT ('Executed at ',
   CAST (CURRENT_TIMESTAMP AS CHAR(26))));
```

- 使用串联运算符（||）构造并插入数值：

```
INSERT INTO stats.logfile (user_key, user_info)
VALUES (002, 'Executed at ' ||
   CAST (CURRENT_TIMESTAMP AS CHAR(26)));
```

6.23 CONVERTTOHEX 函数

该函数将指定的值表达式转换为 16 进制数用以显示。

CONVERTTOHEX 函数是 EsgynDB SQL 扩展函数。

CONVERTTOHEX (expression)

expression

指的是任何数字、字符、日期时间或时间间隔表达式。

CONVERTTOHEX 函数的主要目的在于从一个列里抽取精确值。该函数特别适用于字符表达式，其中的一些字符可能来自客户端语言环境不支持的字符集、也可能为控制代码或其他无法显示的字符。

6.23.1 CONVERTTOHEX 函数注意事项

虽然 CONVERTTOHEX 可用于日期时间和时间间隔表达式，但显示的输出将显示内部值，因此对于一般用户并没有特殊意义。未来的发行版本对此可能有更改。

CONVERTTOHEX 可以返回采用 ISO8859-1 编码的 ASCII 字符。

6.23.2 CONVERTTOHEX 函数例子

- 将一个 smallint、integer 和 largeint 数值显示为 16 进制数:

```
CREATE TABLE EG (S1 smallint, I1 int, L1 largeint);
INSERT INTO EG VALUES( 37, 2147483647,
2305843009213693951); SELECT CONVERTTOHEX(S1),
CONVERTTOHEX(I1), CONVERTTOHEX(L1) from EG;

(EXPR) (EXPR)   (EXPR)
------ -------- ----------------
0025   7FFFFFFF 1FFFFFFFFFFFFFFF
```

- 显示采用 UTF8 字符集的一个 CHAR(4)列、VARCHAR(4)列和 CHAR(4)列里的内容。与固定长度的列不同，varchar 列不带有尾随空格:

```
CREATE TABLE EG_CH (FC4 CHAR(4), VC4 VARCHAR(4), FC4U
CHAR(4) CHARACTER SET UTF8);
INSERT INTO EG_CH values('ABC', 'abc', _UTF8'abc');
SELECT CONVERTTOHEX(FC4), CONVERTTOHEX(VC4),
CONVERTTOHEX(FC4U) from EG_CH;
```

```
(EXPR)   (EXPR)     (EXPR)
-------- -------- ----------------
41424320 616263   0061006200630020
```

- 显示一个 DATE 列、TIME 列、TIMESTAMP(2)列和 TIMESTAMP(6)列的内部值:

```
CREATE TABLE DT (D1 date, T1 time,
                 TS1 timestamp(2), TS2 timestamp(6) );
INSERT INTO DT values(current_date, current_time,
                 current_timestamp,
                 current_timestamp);
SELECT CONVERTTOHEX(D1), CONVERTTOHEX(T1),
                 CONVERTTOHEX(TS1), CONVERTTOHEX(TS2)
                 from DT;
```

```
(EXPR) (EXPR) (EXPR)                (EXPR)
------- ----- --------------------- --------------
07D8040F 0E201E 07D8040F0E201E00000035
07D8040F0E201E00081ABB
```

- 显示一个 INTERVAL YEAR 列、INTERVAL YEAR(2) TO MONTH 列和 INTERVAL DAY TO SECOND 列的内部值:

```
CREATE TABLE IVT ( IV1 interval year, IV2 interval year(2)
to month,
                 IV3 interval day to second);
INSERT INTO IVT values( interval '1' year,
                 interval '3-2' year(2) to month,
                 interval '31:14:59:58' day to second);
SELECT CONVERTTOHEX(IV1), CONVERTTOHEX(IV2),
  CONVERTTOHEX(IV3) from IVT;
```

```
(EXPR) (EXPR)   (EXPR)
------ -------- -----------------------
0001    0026     0000027C2F9CB780
```

6.24 CONVERTTIMESTAMP 函数

该函数将儒略时间戳转换为 TIMESTAMP 数据类型的值。

CONVERTTIMESTAMP 函数是 EsgynDB SQL 扩展函数。

```
CONVERTTIMESTAMP (julian-timestamp)
```

julian-timestamp

该表达式的求值结果为儒略时间戳，这是一个 LARGEINT 值。

6.24.1 CONVERTTIMESTAMP 函数注意事项

julian-timestamp 的数值必须介于 14873163200000000 和 274927348799999999 之间。

6.24.1.1 CONVERTTIMESTAMP 函数与 JULIANTIMESTAMP 函数的关系

JULIANTIMESTAMP 函数的操作数是一个儒略时间戳，函数的结果是一个 TIMESTAMP 数据类型的值。

CONVERTTIMESTAMP 函数的操作数则是一个 TIMESTAMP 数据类型的数值，函数的结果则是一个儒略时间戳，即这两个函数存在逆相关关系。

6.24.1.2 使用 CONVERTTIMESTAMP 函数

您可以使用 JULIANTIMESTAMP 和 CONVERTTIMESTAMP 函数之间的逆相关关系将儒历时间戳列插入您的数据库里，并将这些列值显示为 TIMESTAMP 格式。

6.24.2 CONVERTTIMESTAMP 函数例子

- 假设 EMPLOYEE 表包含一个名为 HIRE_DATE 的列，其中包含每个雇员的聘用日期作为儒略时间戳。

  将儒略时间戳转换为一个 TIMESTAMP 值：

  ```
  SELECT CONVERTTIMESTAMP (hire_date) FROM
  persnl.employee;
  ```

- 以下例子说明 JULIANTIMESTAMP 函数和 CONVERTTIMESTAMP 函数之间的逆相关关系：

  ```
  SELECT CONVERTTIMESTAMP (JULIANTIMESTAMP
  (ship_timestamp))
  FROM persnl.project;
  ```

例如：如果 SHIP_TIMESTAMP 的值为 2008-04-03 21：05：36.143000，则 `CONVERTTIMESTAMP(JULIANTIMESTAMP(ship_timestamp))` 的结果也将是相同的数值 2008-04-03 21：05：36.143000。

6.25 COS 函数

该函数返回一个数字值表达式的余弦，这里的表达式是一个以弧度表示的角度。

COS 函数是 EsgynDB SQL 扩展函数。

```
COS (numeric-expression)
```

numeric-expression
这是一种 SQL 数字值表达式，它指定 COS 函数的参数值。更多信息，请参阅字符值表达式。

6.25.1 COS 函数例子

- 该函数返回数值 9.39680940386503680E-001 或近似值 0.9397-0.3491 弧度（即 20 度）的余弦：
  ```
  COS (0.3491)
  ```

6.26 COSH 函数

该函数返回数字值表达式的双曲余弦，这里的表达式是一个以弧度表示的角度。

COSH 函数是 EsgynDB SQL 扩展函数

```
COSH (numeric-expression)
```

numeric-expression
这是一种 SQL 数字值表达式，它指定 COSH 函数的参数值。更多信息，请参阅字符值表达式。

6.26.1 COSH 函数例子

- 该函数返回数值 1.88842387716101568E+000 或近似值 1.8884-1.25 弧度的双曲余弦：
  ```
  COSH (1.25)
  ```

6.27 COUNT 函数

该函数对一条查询所得的行数进行计数或对特定列里包含不同值的行数进行计数。

COUNT 函数的结果为数据类型 LARGEINT。该结果不得为空。

```
COUNT {(*) | ([ALL | DISTINCT] expression)}
```

```
COUNT (*)
```

返回一条包含 COUNT (*)的 SELECT 语句的 FROM 子句里指定表的行数。如果结果表为空（即该查询没有返回任何行），则 COUNT (*)函数可以返回 0。

```
ALL | DISTINCT
```

返回所有的行数或表达式求值所派生的单列表里的不同行数。默认选项是 ALL，由此使得重复值被包含在内。如果您指定 DISTINCT 选项，则将在应用 COUNT 函数之前消除重复值。

expression

指定一个数值表达式，确定被计数的数值。该表达式不得包含一个聚合函数或子查询。DISTINCT 子句指定 COUNT 函数运行在表达式求值所派生的单列表的不同值上。更多信息，请参阅表达式。

6.27.1 COUNT 函数注意事项

6.27.1.1 表达式的操作数

COUNT 函数的操作数既可以是 *、也可以是一种表达式，其包含带有 COUNT 函数的 SELECT 语句指定的结果表里的列。然而，该表达式不得包含一个聚合函数或子查询。以下这些表达式是有效的：

```
COUNT (*)
COUNT (DISTINCT JOBCODE)
COUNT (UNIT_PRICE * QTY_ORDERED)
```

6.27.1.2 空值

在消除操作数指定的单列表里的所有空值之后，对 COUNT 函数进行求值。如果一个表里没有任何行，则 COUNT 函数可以返回 0。COUNT(*)函数不会从 SELECT 语句的 FROM 子句指定的表里消除空值行。如果一个表里的所有行都为空，则 COUNT(*)将返回表里的行数。

6.27.2 COUNT 函数例子

- 对 EMPLOYEE 表的行数进行计数:

```
SELECT COUNT (*) FROM persnl.employee;

(EXPR)
-----------
        62

--- 1 row(s) selected.
```

- 对 EMPLOYEE 表里具有职位代码的雇员数量进行计数:

```
SELECT COUNT (jobcode) FROM persnl.employee;

(EXPR)
-----------
        56
--- 1 row(s) selected. SELECT COUNT(*)

FROM persnl.employee
WHERE jobcode IS NOT NULL;

(EXPR)
-----------
        56

--- 1 row(s) selected.
```

- 对 EMPLOYEE 表的不同部门的数量进行计数:

```
SELECT COUNT (DISTINCT deptnum)
FROM persnl.employee;

(EXPR)
-----------
        11

--- 1 row(s) selected.
```

6.28 CRC32 函数

计算循环冗余校验值并返回一个 32 位无符号值。如果参数为 NULL，则结果为 NULL。该参数应是一个字符串，如果不是，就当作一个字符串处理（可能的话）。

```
CRC32{ expression)}
```

expression

指定一个值表达式，用于确定要计数的值。表达式不能包含聚合函数或子查询。更多信息，请参阅表达式。

6.28.1 CRC32 函数例子

```
>>SELECT CRC32('Trafodion') from dual;
(EXPR)
----------
1960931967

>>SELECT CRC32(2016) from dual;
(EXPR)
----------
2177070256
```

6.29 CURRENT 函数

该函数基于本地当前日期和时间返回一个 TIMESTAMP 类型的数值。

当查询开始执行时，该函数将被求值一次，并且不会被再次求值（即使它是一条长时间运行的查询，也是如此）。

另外，您也可以使用“CURRENT_TIMESTAMP 函数”。

```
CURRENT [(precision)]
```

precision

这是一个整数，范围是 0 到 9；它可以在返回值里指定小数秒（小数位数）的精度，其默认值为 6。

例如：函数 CURRENT (2)返回当前日期和时间作为 TIMESTAMP 数据类型的数值。其中，小数秒的精度为 2。

例如：2008-06-26 09：01：20.89。返回值不是一个字符串值。

6.29.1 CURRENT 函数例子

- PROJECT 表包含一个 TIMESTAMP 数据类型的列 SHIP_TIMESTAMP。通过使用 CURRENT 值更新一行数据：

  ```
  UPDATE persnl.project
  SET ship_timestamp = CURRENT
  WHERE projcode = 1000;
  ```

6.30 CURRENT_DATE 函数

该函数返回本地当前日期作为一个 DATE 类型的值。

当查询开始执行时，该函数将被求值一次，并且不会被再次求值（即使它是一条长时间运行的查询，也是如此）。

```
CURRENT_DATE
```

CURRENT_DATE 函数返回当前日期诸如 2008-09-28。返回值是一个 DATE 型数值，而非字符串值。

6.30.1 CURRENT_DATE 函数例子

- 基于当前日期从 ORDERS 表里选择行:

  ```
  SELECT * FROM sales.orders
  WHERE deliv_date >= CURRENT_DATE;
  ```

- PROJECT 表里有一列 EST_COMPLETE，其类型为 INTERVAL DAY。如果当前日期为您项目的起始日期，则请确定预计的项目完成日期:

  ```
  SELECT projdesc, CURRENT_DATE + est_complete FROM persnl.project;

  Project/Description (EXPR)
  ------------------- ----------
  SALT LAKE CITY      2008-01-18
  ROSS PRODUCTS       2008-02-02
  MONTANA TOOLS       2008-03-03
  AHAUS TOOL/SUPPLY   2008-03-03
  THE WORKS           2008-02-02
  THE WORKS           2008-02-02
  --- 6 row(s) selected.
  ```

6.31 CURRENT_TIME 函数

CURRENT_TIME 函数返回本地当前时间作为一个 TIME 类型的值。

当查询开始执行时，该函数将被求值一次，并且不会被再次求值（即使它是一条长时间运行的查询，也是如此）。

```
CURRENT_TIME [(precision)]
```

precision

这是一个整数，范围是 0 到 9；它在返回值里指定小数秒（小数位数）的精度，其默认值为 6。

例如：函数 CURRENT_TIME (2)可以返回当前时间作为一个 TIME 数据类型的值。其中的小数秒的精度为 2。例如：14：01：59.30。返回值不是一个字符串值。

6.31.1 CURRENT_TIME 函数例子

- 使用 CURRENT_DATE 和 CURRENT_TIME 作为一个插入行里的数值：

```
INSERT INTO stats.logfile
(user_key, run_date, run_time, user_name)
VALUES (001, CURRENT_DATE, CURRENT_TIME, 'JuBrock');
```

6.32 CURRENT_TIMESTAMP 函数

该函数基于本地当前日期和时间返回一个 TIMESTAMP 类型的值。

当查询开始执行时，该函数将被求值一次，并且不会被再次求值（即使它是一条长时间运行的查询，也是如此）。

另外，您也可以使用“CURRENT 函数”。

```
CURRENT_TIMESTAMP [(precision)]
```

precision

这是一个整数，范围是 0 到 9；它在返回值里指定小数秒（放入的小数位数）的精度，其默认值为 6。

函数 CURRENT_TIMESTAMP (2)返回当前日期和时间作为一个 TIMESTAMP 数据类型的值。其中，小数秒的精度为 2；例如：2008-06-26 09：01：20.89。返回值不是一个字符串值。

6.32.1 CURRENT_TIMESTAMP 函数例子

PROJECT 表包含一个 TIMESTAMP 数据类型的列 SHIP_TIMESTAMP。通过使用 CURRENT_TIMESTAMP 值更新一行数据：

```
UPDATE persnl.project
SET ship_timestamp = CURRENT_TIMESTAMP WHERE projcode =
1000;
```

6.33 CURRENT_USER 函数

该函数返回调用该函数的当前用户的数据库用户名。 当前用户指的是启动会话、并且经过身份验证的用户。该数据库用户名在当前会话里用于验证 SQL 语句。

```
CURRENT_USER
```

CURRENT_USER 函数类似 USER 函数

6.33.1 CURRENT_USER 函数注意事项

- 该函数仅可指定在一条 SELECT 语句的顶层。
- The value 返回值为字符串数据类型 VARCHAR(128)，并且采用 ISO8859-1 编码。

6.33.2 CURRENT_USER 函数例子

- 该例子检索当前用户的数据库用户名:

```
SELECT CURRENT_USER FROM (values(1)) x(a);

(EXPR)
------------------------
TSHAW

--- 1 row(s) selected.
```

6.34 DATE_ADD 函数

该函数 interval_expression 指定的时间间隔与 datetime_expr 相加。如果指定的时间间隔以年或月为单位，DATE_ADD 函数将对结果进行规范化。更多信息，请参阅标准规范化。

该函数返回 datetime_expr 的类型，除非 interval_expression 包含所有包含任何时间间隔，在这种情况下，该函数返回一个时间戳。

DATE_ADD 函数是 EsgynDB SQL 扩展函数。

```
DATE_ADD (datetime_expr, interval_expression)
```

datetime_expr

该表达式的求值结果为一个 DATE 或 TIMESTAMP 类型的日期时间值。更多信息，请参阅日期时间值表达式。

interval_expression

该表达式通过特定的方式与加法运算符相结合。interval_expression 接受所有被 EsgynDB 数据库软件视为有效的时间间隔表达式类型。

6.34.1 DATE_ADD 函数例子

- 该函数返回 DATE 值'2007-03-07'

  ```
  DATE_ADD(DATE '2007-02-28', INTERVAL '7' DAY)
  ```

- 该函数返回 DATE 值'2008-03-06'

  ```
  DATE_ADD(DATE '2008-02-28', INTERVAL '7' DAY)
  ```

- 该函数返回时间戳'2008-03-07 00:00:00'

  ```
  DATE_ADD(timestamp'2008-02-29 00:00:00', INTERVAL '7'
  DAY)
  ```

- 该函数返回时间戳'2008-02-28 23：59：59'

  ```
  DATE_ADD(timestamp '2007-02-28 23：59：59', INTERVAL
  '12' MONTH)
  ```

注意：请将这里的最后一个例子与 DATE_SUB 下的最后一个例子进行比较。

6.35 DATE_SUB 函数

该函数从 datetime_expr 里减去指定的 interval_expression。如果指定的时间间隔以年或月为单位，DATE_SUB 函数将对结果进行规范化。更多信息，请参阅标准规范化。

该函数返回 datetime_expr 的类型，除非 interval_expression 包含所有包含任何时间间隔，在这种情况下，该函数返回一个时间戳。

DATE_SUB 函数是 EsgynDB SQL 扩展函数。

```
DATE_SUB (datetime_expr, interval_expression)
```

datetime_expr

该表达式的求值结果为一个 DATE 或 TIMESTAMP 类型的日期时间值。更多信息，请参阅日期时间值表达式。

interval_expression

该表达式通过特定的方式与减法运算符相结合。interval_expression 接受所有被 EsgynDB 数据库软件视为有效的时间间隔表达式类型。

6.35.1 DATE_SUB 函数例子

- 该函数返回 DATE 值 '2009-02-28'

  ```
  DATE_SUB(DATE '2009-03-07', INTERVAL'7' DAY)
  ```

- 该函数返回 DATE 值 '2008-02-29'

  ```
  DATE_SUB(DATE '2008-03-07', INTERVAL'7' DAY)
  ```

- 该函数返回时间戳 '2008-02-29 00:00:00'

  ```
  DATE_SUB(timestamp '2008-03-31 00:00:00', INTERVAL '31'
  DAY)
  ```

- 该函数返回时间戳 '2007-02-28 23：59：59'

  ```
  DATE_SUB(timestamp '2008-02-29 23：59：59', INTERVAL
  '12' MONTH)
  ```

6.36 DATEADD 函数

该函数将 datepart 和 num_expr 指定的时间间隔与 datetime_expr 相加。

如果指定的时间间隔以年或月为单位，DATEADD 函数将结果进行规范化。

更多信息，请参阅标准规范化。

该函数可以返回 datetime_expr 的类型，除非时间间隔表达式包含任何时间间隔，在这种情况下，该函数返回一个时间戳。

DATEADD 函数是 EsgynDB SQL 扩展函数。

```
DATEADD(datepart, num_expr, datetime_expr)
```

datepart

为 YEAR、MONTH、DAY、HOUR、MINUTE、SECOND、QUARTE、WEEK 或以下缩写之一：

YEAR	*YY* 和 *YYYY*
MONTH	*M* 和 *MM*
DAY	*D* 和 *DD*
HOUR	*HH*
MINUTE	*MI* 和 *M*
SECOND	*SS* 和 *S*
QUARTER	*Q* 和 *QQ*
WEEK	*WW* 和 *WK*

num_expr

这是一种 SQL 精确数字值表达式，它可以指定与 *datetime_expr* 相加的 datepart 时间单元数量。

如果 *num_expr* 带有小数部分，则该部分将被忽略。如果 num_expr 为负数，则 *datetime_expr* 返回值前面带有指定的时间量。更多信息，请参阅字符值表达式。

datetime_expr

该表达式的求值结果为一个 DATE 或 TIMESTAMP 类型的日期时间值。它返回 *datetime_expression* 的类型，除非时间间隔表达式包含任何时间间隔，在这种情况下，该函数返回一个时间戳。更多信息，请参阅日期时间值表达式。

6.36.1 DATEADD 函数例子

- 该函数将 start_date 指定的日期加上 7 天

  ```
  DATEADD (DAY, 7,start_date)
  ```

- 该函数返回 DATE 值 '2009-03-07'
  ```
  DATEADD (DAY, 7 , DATE '2009-02-28')
  ```

- 该函数返回 DATE 值 '2008-03-06'
  ```
  DATEADD(DAY, 7, DATE '2008-02-28')
  ```

- 该函数返回时间戳 '2008-03-07 00:00:00'
  ```
  DATEADD(DAY, 7, timestamp'2008-02-29 00:00:00')
  ```

6.37 DATEDIFF 函数

该函数返回一个整数值作为起始日期和结束日期之间的 datepart 时间单元数量。如果结束日期早于起始日期，则返回值将为负数或 0。

DATEDIFF 函数是 EsgynDB SQL 扩展函数。

```
DATEDIFF (datepart, startdate, enddate)
```

6.37.1 DATEDIFF 函数语法

datepart

是 YEAR、MONTH、DAY、HOUR、MINUTE、SECOND、QUARTE、WEEK 或以下缩写之一：

YEAR	*YY* 和 *YYYY*
MONTH	*M* 和 *MM*
DAY	*D* 和 *DD*
HOUR	*HH*
MINUTE	*MI* 和 *M*
SECOND	*SS* 和 *S*
QUARTER	*Q* 和 *QQ*
WEEK	*WW* 和 *WK*

startdate

为 DATE 或 TIMESTAMP 类型。更多信息，请参阅日期时间值表达式。

enddate

为 DATE 或 TIMESTAMP 类型。更多信息，请参阅日期时间值表达式。

跨越时间界限（诸如日、分钟和秒）的计数方法使得 DATEDIFF 函数得出的结果对于所有数据类型都能保持一致。该结果是一个有符号整数值，等于第一个日期和第二个日期所跨的 datepart 时间界限数量。

例如：1 月 4 日（周日）和 1 月 1 日（周日）之间的周数为 1。3 月 31 日和 4 月 1 日之间的月数也为 1，因为 3 月和 4 月相隔一个月的时间。如果结果超出了整数范围，则 DATEDIFF 函数将产生一个错误。最大秒数等于大约 68 年。

如果请求周数差值，并且两个日期中的一个早于 0001 年 1 月 7 日，则 DATEDIFF 函数也将产生一个错误。

6.37.2 DATEDIFF 注意事项

6.37.2.1 时间界限

跨越时间界限（诸如日、分钟和秒）的计数方法使得 DATEDIFF 函数得出的结果对于所有数据类型都能保持一致。该结果是一个有符号整数值，等于 `startdate` 和 `enddate` 所跨的 datepart 时间界限数量。

例如：

- 2017-12-31 23:59:59 和 2018-01-01 00:00:00 之间的差值仅为一秒。

但是，DATEDIFF的差值为1分钟、1小时、1天、0周1、1月、1季度或1月，具体取决于指定的datepart。

[1] 注意：2017-12-31（startdate）为周日，与 2018-01-01 为同一周。更多信息，请参阅一周的起始日。

- 同样，2017-01-01 和 2018-12-31 之间的差值将近两年。

但是，如果指定的 datepart 为 YEAR，则 DATEDIFF 的差值为 1 年。

6.37.2.2 一周的起始日

- 一周起始日为周天。不支持更改一周起始日。
- DATEDIFF 函数返回 startdate 和 enddate 之间的周天数。如果该函数落在周天，则计算 enddate，但不计算 startdate。

例如，

- 如果指定的 datepart 是 WEEK，则 2017-11-19 和 2017-11-26 之间的 DATEDIFF 差值为 1。
- 如果指定的 datepart 是 WEEK，则 2017-11-19 和 2017-11-25 之间的 DATEDIFF 差值为 0。

6.37.3 DATEDIFF 函数例子

6.37.3.1 SECOND 的日期差值

- 该函数返回数值 0，因为差值没有跨越 1 秒的时间界限。

```
DATEDIFF(    SECOND,
             TIMESTAMP '2006-09-12 11: 59: 58.999998',
             TIMESTAMP '2006-09-12 11: 59: 58.999999')
```

- 该函数返回数值 1，因为差值已经跨越了 1 秒的时间界限，即便两个时间戳仅仅相差 1 微秒也是如此。

```
DATEDIFF(    SECOND,
```

```
                TIMESTAMP '2006-09-12 11: 59: 58.999999',
                TIMESTAMP '2006-09-12 11: 59: 59.000000')
```

6.37.3.2 MINUTE 的日期差值

- 该函数返回数值 2，因为差值已经跨越了 2 分钟的时间界限。

```
SELECT DATEDIFF( MINUTE
, TIMESTAMP '2011-12-30 08:54:55'
, TIMESTAMP '2011-12-30 08:56:01'
)
FROM DUAL;
```

6.37.3.3 DAY 的日期差值

- 该函数返回数值-29。

```
SELECT DATEDIFF( DAY
, DATE '2008-03-01'
, DATE '2008-02-01'
)
FROM DUAL;
```

- 该语句计算买家必须等待的时间。

```
SELECT id, DATEDIFF( DAY, order_date, delivery_date ),
price
FROM orders
ORDER BY price DESC;

ID          (EXPR)      PRICE
----------- ----------- ------
     700300         145    926
     312203         147    621
     800660          23    568
     100350         160    543
     500450         148    324
     700510         141    229
     100210           3    228
     600480         151    197
     300380         154    183
     200490         227    123
     200320         153     91
     400410         158     65
     600250         143     32
   --- 13 row(s) selected.
```

6.37.3.4 WEEK 的日期差值

- 该函数返回数值 2，因为差值已经跨越了 1 周的时间界限。

```
SELECT DATEDIFF( WEEK
, DATE '2006-01-01'
, DATE '2006-01-09'
)
FROM DUAL;
```

6.37.3.5 QUARTER 的日期差值

- 该函数返回数值 3，因为差值已经跨越了四分之三的时间界限。

```
SELECT DATEDIFF( QUARTER
, DATE '2017-03-05'
, DATE '2017-11-17'
)
FROM DUAL;
```

6.37.3.6 YEAR 的日期差值

- 该函数返回数值 0。

```
DATEDIFF(    YEAR,
             TIMESTAMP '2006-12-31 23:59:59.999998',
             TIMESTAMP '2006-12-31 23:59:59.999999')
```

- 该函数返回数值 1，因为差值已经跨越了 1 年的时间界限。

```
DATEDIFF(    YEAR,
             TIMESTAMP '2006-12-31 23:59:59.999999',
             TIMESTAMP '2007-01-01 00:00:00.000000')
```

6.38 DATEFORMAT 函数

该函数返回一个日期时间值作为 DEFAULT、美国或欧洲格式的字符串字面值。该函数结果的数据类型为 CHAR。

DATEFORMAT 函数是 EsgynDB SQL 扩展函数。

```
DATEFORMAT (datetime-expression,{DEFAULT | USA | EUROPEAN})
```

datetime-expression

该表达式的求值结果为 DATE、TIME 或 TIMESTAMP 类型的日期时间值。更多信息，请参阅日期时间值表达式。

DEFAULT | USA | EUROPEAN

指定一个日期时间值的格式。更多信息，请参阅日期时间字面值。

6.38.1 DATEFORMAT 函数注意事项

DATEFORMAT 函数返回一个采用 ISO8859-1 编码的日期时间值。

6.38.2 DATEFORMAT 函数例子

- 将一个 DEFAULT 格式的日期时间字面值转换为美国格式的字符串：

  ```
  DATEFORMAT (TIMESTAMP '2008-06-20 14: 20: 20.00', USA)
  ```

  The 函数返回这个字符串字面值：

  ```
  '06/20/2008 02: 20: 20.00 PM'
  ```

- 将一个 DEFAULT 格式的日期时间字面值转换为欧洲格式的字符串：

  ```
  DATEFORMAT (TIMESTAMP '2008-06-20 14 : 20 : 20.00',
  EUROPEAN)
  ```

  该函数返回这个字符串字面值：

  ```
  '20.06.2008 14.20.20.00'
  ```

6.39 DATE_PART 函数（时间间隔）

该函数从 interval 指定的时间间隔值中抽取文本指定的日期时间字段，并返回一个精确数字型数值的结果。

DATE_PART 函数接受'YEAR'、'MONTH'、'DAY'、'HOUR'、'MINUTE'或'SECOND' 之类的规范作为文本。

DATE_PART 函数是 EsgynDB SQL 扩展函数。

```
DATE_PART (text, interval)
```

text

指定YEAR、MONTH、DAY、HOUR、MINUTE 或SECOND。该值必须用单引号引起来。

interval

interval 接受所有被 EsgynDB 数据库软件视为有效的时间间隔表达式类型。

*DATE_PART(text, interval)*等效于 *EXTRACT(text, interval)*，除了 DATE_PART 函数要求在文本规范两边加上单引号、而 EXTRACT 不允许使用单引号。

如果指定了 SECOND，则将返回一个秒的小数部分。

6.39.1 DATE_PART（时间间隔）函数例子

- 该函数返回数值 7

  ```
  DATE_PART('DAY', INTERVAL '07: 04' DAY TO HOUR)
  ```

- 该函数返回数值 6

  ```
  DATE_PART('MONTH', INTERVAL '6' MONTH)
  ```

- 该函数返回数值 36.33

  ```
  DATE_PART('SECOND', INTERVAL '5: 2: 15: 36.33' DAY TO SECOND(2))
  ```

6.40 DATE_PART 函数（时间戳）

该函数从 datetime_expr 指定的日期时间值中抽取文本指定的日期时间字段，并返回结果为一个精确数字型数值。

DATE_PART 函数接受'YEAR'、'YEARQUARTER'、'YEARMONTH'、'YEARWEEK'、'MONTH'、'DAY'、'HOUR、'MINUTE'或'SECOND'之类的规范作为文本。

时间戳的 DATE_PART 函数可以改为日期时间的 DATE_PART 函数，因为第二个参数可以是时间戳或日期表达式。

DATE_PART 函数是 EsgynDB 扩展函数。

```
DATE_PART(text, datetime_expr)
```

`text`

指定 YEAR、YEARQUARTER、YEARMONTH、YEARWEEK、MONTH、DAY、HOUR、MINUTE 或 SECOND。该值必须用单引号括起来。

- YEARMONTH：抽取年份和月份作为一个 6 位整数，形式为 yyyymm(100 * year + month) 。
- YEARQUARTER：抽取年份和季度作为一个 5 位整数，形式为 yyyyq, (10* year + quarter)；其中，第一季度的 q 为 1；第二季度的 q 为 2，依此类推。
- YEARWEEK：抽取年份和一年中的周数作为一个 6 位整数，形式为 yyyyww(100 * year + week) 周数与 WEEK 函数将采用相同的计算方式。

datetime_expr

该表达式的求值结果为 DATE 或 TIMESTAMP 类型的日期时间值。更多信息，请参阅日期时间值表达式。

DATE_PART(text, datetime_expr) 大体等效于 *EXTRACT(text, datetime_expr)*，除了 DATE_PART 函数要求在文本规范两边加上单引号、而 EXTRACT 不允许使用单引号。此外，您不能将 YEARQUARTER、YEARMONTH 和 YEARWEEK 与 EXTRACT 同时使用。

6.40.1 DATE_PART 函数例子

- 该函数返回数值 12

  ```
  DATE_PART('month', date'12/05/2006')
  ```

- 该函数返回数值 2006
  `DATE_PART('year', date'12/05/2006')`

- 该函数返回数值 31
  `DATE_PART('day', TIMESTAMP '2006-12-31 11 : 59 : 59.999999')`

- 该函数返回数值 2011 07
  `DATE_PART('YEARMONTH', date '2011-07-25')`

6.41 DATE_TRUNC 函数

该函数返回一个 TIMESTAMP 类型的值，其所有字段的精度都小于文本精度的 0（或对于月或日的精度设置为 1）。

DATE_TRUNC 函数是 EsgynDB SQL 扩展函数。

```
DATE_TRUNC(text, datetime_expr)
```

text

指定'YEAR'、'MONTH'、'DAY'、'HOUR'、'MINUTE'或'SECOND'。

DATE_TRUNC 函数也可以接受'CENTURY'或'DECADE'之类的规范。

datetime_expr

该表达式的求值结果为一个 DATE 或 TIMESTAMP 类型的日期时间值。

DATE_TRUNC 函数返回一个 TIMESTAMP 类型的值，其所有字段的精度都小于文本精度（设置为 0）或 1（月数或天数的精度）。更多信息，请参阅日期时间值表达式。

6.41.1 DATE_TRUNC 函数例子

- 该函数返回 TIMESTAMP 值 '2006-12-31 00:00:00'

  ```
  DATE_TRUNC('DAY', TIMESTAMP '2006-12-31 11: 59: 59')
  ```

- 该函数返回 TIMESTAMP 值 '2006-01-01 00:00:00'

  ```
  DATE_TRUNC('YEAR', TIMESTAMP '2006-12-31 11: 59: 59')
  ```

- 该函数返回 TIMESTAMP 值 '2006-12-01 00:00:00'

  ```
  DATE_TRUNC('MONTH', DATE '2006-12-31')
  ```

  限制：

  - DATE_TRUNC('DECADE', …)不得用在年数少于 10 的案例中。
  - DATE_TRUNC('CENTURY', …)不得用在年数少于 100 的案例中。

6.42 DAY 函数

该函数将一个 DATE 或 TIMESTAMP 表达式转换为 INTEGER（整型）值，范围是 1 到 31，表示相应月中的相应日。

DAY 函数与 DAYOFMONTH 函数具有相等的返回结果。

DAY 函数是 EsgynDB SQL 扩展函数。

```
DAY (datetime-expression)
```

datetime-expression

该表达式的求值结果为一个 DATE 或 TIMESTAMP 类型的日期时间值。更多信息，请参阅日期时间值表达式。

6.42.1 DAY 函数例子

- 返回一个整数，表示 PROJECT 表的 START_DATE 列里的相应月中的相应日：

```
SELECT start_date, ship_timestamp, DAY(start_date)
FROM persnl.project
WHERE projcode = 1000;
Start/Date Time/Shipped              (EXPR)
---------- -------------------------- ------
2008-04-10 2008-04-21 08:15:00.000000 10
```

6.43 DAYNAME 函数

该函数将一个 DATE 或 TIMESTAMP 表达式转换为一个字符字面值，具体为相应周中的相应日的星期名称（Sunday、Monday，等等）。

DAYNAME 函数是 EsgynDB SQL 扩展函数。

```
DAYNAME (datetime-expression)
```

datetime-expression

该表达式的求值结果为 DATE 或 TIMESTAMP 类型的日期时间值。更多信息，请参阅日期时间值表达式。

6.43.1 DAYNAME 函数注意事项

DAYNAME 函数返回采用 ISO8859-1 编码的相应日的星期名称。

6.43.2 DAYNAME 函数例子

- 返回 PROJECT 表的 START_DATE 列里的相应周中的相应日的星期名称：

```
SELECT start_date, ship_timestamp, DAYNAME(start_date)
FROM persnl.project
WHERE projcode = 1000;

Start/Date Time/Shipped              (EXPR)
---------- -------------------------- ---------
2008-04-10 2008-04-21 08:15:00.000000 Thursday
```

6.44 DAYOFMONTH 函数

该函数将一个 DATE 或 TIMESTAMP 表达式转换为 INTEGER（整型）值，范围是 1 到 31，表示相应月中的相应日。

DAYOFMONTH 函数与 DAY 函数具有相等的返回结果。

DAYOFMONTH 函数是 EsgynDB SQL 扩展函数。

```
DAYOFMONTH (datetime-expression)
```

datetime-expression

该表达式的求值结果为 DATE 或 TIMESTAMP 型的日期时间值。更多信息，请参阅日期时间值表达式。

6.44.1 DAYOFMONTH 函数例子

- 返回一个整数，表示 PROJECT 表的 START_DATE 列日期里的相应日（不返回年份和月份，仅返回日）：

```
SELECT start_date, ship_timestamp,
DAYOFMONTH(start_date)
FROM persnl.project
WHERE projcode = 1000;

Start/Date Time/Shipped              (EXPR)
---------- -------------------------- ------
2008-04-10 2008-04-21 08:15:00.000000 10
```

6.45 DAYOFWEEK 函数

该函数将一个 DATE 或 TIMESTAMP 表达式转换为 INTEGER（整型）值，范围是 1 到 7，表示一周中相应的第几天。数值 1 代表周日，数值 2 代表周一，依此类推。

DAYOFWEEK 函数是一个 EsgynDB SQL 扩展函数。

DAYOFWEEK (datetime-expression)

datetime-expression
该表达式的求值结果为一个 DATE 或 TIMESTAMP 类型的日期时间值。更多信息，请参阅日期时间值表达式。

6.45.1 DAYOFWEEK 函数例子

- 返回一个整数，表示 PROJECT 表的 START_DATE 列里的一周中的某一天：

```
SELECT start_date, ship_timestamp, DAYOFWEEK(start_date)
FROM persnl.project
WHERE projcode = 1000;
Start/Date Time/Shipped              (EXPR)
---------- -------------------------- ------
2008-04-10 2008-04-21 08:15:00.000000 5
```

  返回值为 5，代表周四。一周的起始日为周天。

6.46 DAYOFYEAR 函数

该函数将一个 DATE 或 TIMESTAMP 表达式转换为 INTEGER（整型）值，范围是 1 到 366，表示一年中相应的第几天。

DAYOFYEAR 函数是 EsgynDB SQL 扩展函数。

```
DAYOFYEAR (datetime-expression)
```

datetime-expression

该表达式的求值结果为一个 DATE 或 TIMESTAMP 类型的日期时间值。更多信息，请参阅日期时间值表达式。

6.46.1 DAYOFYEAR 函数例子

- 返回一个整数，表示 PROJECT 表的 START_DATE 列里的日期时间值是一年中的第几天：

```
SELECT start_date, ship_timestamp,
DAYOFYEAR(start_date)
FROM persnl.project
WHERE projcode = 1000;
Start/Date Time/Shipped              (EXPR)
---------- -------------------------- -----------------
2008-04-10 2008-04-21 08:15:00.000000 101
```

6.47 DECODE 函数

DECODE 函数按照所提供的顺序将 expr 与每个 test_expr 值逐一进行比较。如果 expr 与一个 test_expr 相等，则将返回相应的 retval。如果没有找到任何匹配，则将返回默认值。如果没有找到任何匹配，并且默认值也被省略，则返回 NULL（空值）。

DECODE 函数是一个 EsgynDB SQL 扩展函数。

```
DECODE (expr, test_expr, retval
          [, test_expr2, retval2 ... ]
          [ , default ] )
```

expr

这是一种 SQL 表达式。

test_expr, test_expr,..

它们均能和 expr 进行类型比较的 SQL 表达式类型。

retval

这是一种 SQL 表达式。

default, retval2,..

它们均能和 retval 进行类型比较的 SQL 表达式类型。

6.47.1 DECODE 函数注意事项

在一个 DECODE 函数里，两个空值被视为是等效的。如果 *expr* 为空，则返回值将是第一个也为空的 *test_expr* 的 *retval*。*expr*、*test_expr*、*retval* 和 *default*（默认）值 都从表达式里派生出来。

参数是任何数字类型或字符类型。然而， *expr* 和每个 *test_expr* 的数据类型之间都必须具有可比性。如果 *expr* 和 *test_expr* 为字符类型，则它们必须采用相同的字符集（具有可比类型）。

所有 *retval* 值和默认值（若存在），都必须具有类型可比性。

如果 *expr* 和一个 *test_expr* 的值都是字符数据，则将使用非填充比较语义进行比较。如果 *expr* 和一个 *test_expr* 的值都是数值类型数据，则应根据定义的转换规则使用其中一个数字的临时副本进行比较。

例如：如果一个数字为 INTEGER（整型），另一个数字为 DECIMAL（小数型），则将使用已转换为小数的整数的临时副本进行比较。

如果所有可能的返回值都为固定长度字符型，则返回值将是一个固定长度字符串，其大小等于所有可能的返回值类型大小的最大尺寸。

如果任何可能的返回值都为可变长度字符型，则返回值将是一个可变长度字符串，其大小等于所有可能的返回值类型的最大尺寸。

如果所有可能的返回值都为整型，则返回值将与所有可能的返回值的最大整型具有相同的类型。

如果返回值为 FLOAT 型，则其精度为所有可能的返回值的最大精度。

如果所有可能的返回值都是同一种非整数数字类型（REAL、FLOAT、DOUBLE PRECISION、NUMERIC 或 DECIMAL），则返回值将具有相同类型。

如果所有可能的返回值都为数字类型，但并非完全相同，至少一个为 REAL、FLOAT 或 DOUBLE PRECISION 型，则返回值将为 DOUBLE PRECISION 型。

如果所有可能的返回值都为数值类型，但并不都相同，没有一个是 REAL、FLOAT 或 DOUBLE PRECISION 类型，并且至少一个为 NUMERIC 型，则返回值将为 NUMERIC 型。

如果所有可能的返回值都为数值类型，但并不都相同，没有一个是 REAL、FLOAT 或 DOUBLE PRECISION 类型，并且至少一个为 DECIMAL 型，则返回值将为 DECIMAL 型。

如果返回值为 NUMERIC 或 DECIMAL 类型，则其精度将为以下数值的总和：

- 所有可能的返回值类型的最大标度
- 所有可能的返回值类型的(精度-标度)的最大值。然而，精度将不会超过 18，

返回值的标度为以下数值中的最小值:

- 所有可能的返回值类型的最大标度
- 18 - (所有可能的返回值类型的(精度-标度)的最大值)

除了对于 SQL 表达式大小的一般限制以外，DECODE 函数里的参数（包括 *expr*、*test_exprs*、*retvals* *和* *default*）数量没有限制。然而，表达式太多效果不佳。

```
DECODE (expr, test_expr, retval [, test_expr2,
retval2 ... ] [,default ] ):
```

在逻辑上等效于以下:

```
CASE WHEN (expr IS NULL AND test_expr IS NULL) OR
          expr = test_exprTHEN retval
     WHEN (expr IS NULL AND test_expr2 IS NULL) OR
          expr = test_expr2 THEN retval2
     ...
     ELSE default /* or ELSE NULL if default not */
END
```

除了一条 CASE 语句的通常操作以外，*expr*、*test_exprN* 或 *retvalN*,并没有执行特殊的转换。

6.47.2 DECODE 函数例子

- DECODE 函数例子:

```
SELECT emp_name,
decode( CAST (( yrs_of_service + 3) / 4 AS INT ) ,
        0,0.04,
        1,0.04,
        0.06) as perc_value
FROM employees;
SELECT supplier_name,
       decode(supplier_id, 10000, 'Company A',
                           10001, 'Company B',
                           10002, 'Company C',
                           'Company D') as result
FROM suppliers;
```

- 该例子显示了一种不同的空值处理方式，它将 NULL（空值）指定作为默认值处理（而非显式地指定为默认值）:

```
SELECT decode( (?p1 || ?p2), trim(?p1), 'Hi', ?p3, null )
  from emp;
```

```
..
*** ERROR[4049] A CASE expression cannot have a result
data type of both CHAR(2) and NUMERIC(18,6).
*** ERROR[4062] The preceding error actually occurred in
function DECODE((?P1 || ?P2),(‘ ’ TRIM ?P1), ‘Hi’, ?P3,
NULL)
*** ERROR[8822] The statement was not prepared.
```

最后一个 *ret-val* 为显式 NULL。当 EsgynDB SQL 遇到这种情况时，它会假设返回值的数据类型为 NUMERIC(18,6)。一旦 EsgynDB SQL 确定返回值都为数字型，即可确定所有可能的返回值都必须为数字型。如果一个 *ret-val* 位置

出现了‘Hi’，则将产生一个错误，因为 CHAR(2)型参数与 NUMERIC(18,6)型的返回值无法相比。

这是一条等价且不会产生错误的语句：

```
SELECT decode( (?p1 || ?p2), trim(?p1), ‘Hi’ ) from emp;
```

6.48 DEGREES 函数

该函数将以弧度表示的一个数字值表达式转换为角度数。

DEGREES 函数是一个 EsgynDB SQL 扩展函数。

```
DEGREES (numeric-expression)
```

numeric-expression
这是一种 SQL 数字值表达式，它指定 DEGREES 函数的参数值。更多信息，请参阅数值表达式。

6.48.1 DEGREES 函数例子

- 该函数返回数值 45.0001059971939008，该值以角度为单位：
  ```
  DEGREES (0.78540)
  ```
- 该函数返回数值 45。函数 DEGREES 是函数 RADIANS 的反函数。
  ```
  DEGREES (RADIANS (45))
  ```

6.49 EMPTY_BLOB 函数

6.49.1 EMPTY_BLOB 函数语法

EMPTY_BLOB函数创建一个BLOB类型的虚拟LOB句柄。
该函数可在INSERT或UPDATE语句中使用，将LOB列或属性初始化为空。

EMPTY_BLOB函数是EsgynDB SQL扩展函数。
更多信息，请参阅EsgynDB SQL Large Objects 指南。

```
EMPTY_BLOB()
```

6.49.2 EMPTY_BLOB 函数注意事项

- 如果执行 EMPTY_BLOB()，则一个虚拟 LOB 句柄也将被创建。
  - 尚未将数据与空 LOB 相关联，但稍后可以使用这些虚拟用新数据来填充 LOB。如果之前已有数据与 LOB 关联，则 LOB 将被删除。
  - 虚拟 LOB 句柄将获得与基础列相同的数据类型。

  例如，
  - 如果在创建表时将 LOB 列定义为“EXTERNAL”，则虚拟 LOB 句柄将获得“EXTERNAL”类型，并且只能处理外部 LOB 数据。
  - 如果在创建表时未将 LOB 列定义为“EXTERNAL”，则虚拟 LOB 句柄将被视为一个常规 LOB 虚拟句柄。如果此句柄用于执行插入或更新操作，则 LOB 数据将被保留在 EsgynDB 空间内，该空间是由 EsgynDB 维护的内部位置。

- 空的 LOB 与包含长度为零的字符串的 LOB 或空（null ）LOB 是不同的。

6.49.3 EMPTY_BLOB 函数例子

- 该例子使用 EMPTY_BLOB 插入一个空的 LOB 并创建一个虚拟 LOB 句柄。

```
insert into test1 values (1, empty_blob());
```

6.50 EMPTY_CLOB 函数

6.50.1 EMPTY_CLOB 函数语法

EMPTY_CLOB 函数创建一个 CLOB 类型的虚拟 LOB 句柄。
该函数可在 INSERT 或 UPDATE 语句中使用，将 LOB 列或属性初始化为空。
EMPTY_CLOB 函数是 EsgynDB SQL 扩展函数。
更多信息，请参阅 EsgynDB LOB Large Objects 指南。

```
EMPTY_CLOB()
```

6.50.2 EMPTY 注意事项

- 如果指定了EMPTY_CLOB()，则创建一个虚拟LOB句柄。
- 尚未将数据与空LOB相关联，但稍后可以使用这些虚拟LOB句柄用新数据来填充LOB。如果之前已有数据与LOB关联，则LOB将被删除。
- 虚拟LOB句柄将获得与基础列相同的数据类型。

例如，如果在创建表期间将LOB列定义为“EXTERNAL”，则LOB列将获得该类型。如果未定义，则将其视为一个常规LOB。

- 空的LOB与包含长度为零的字符串的LOB或空（null）LOB是不同的。

6.50.3 EMPTY_CLOB 函数例子

- 该例子使用 EMPTY_CLOB 插入一个空 LOB 并创建一个虚拟 LOB 句柄。
  ```
  insert into test2 values (1, empty_clob());
  ```

6.51 EXP 函数

该函数返回一个数字值表达式的指数值（以 e 为底）。

EXP 函数是一个 EsgynDB SQL 扩展函数。

```
EXP (numeric-expression)
```

numeric-expression

这是一种 SQL 数字值表达式，它指定 EXP 函数的参数值.。更多信息，请参阅数值表达式。

最小输入值必须介于-744.4400719 和-744.4400720 之间。

最大输入值必须介于 709.78271289338404 和 709.78271289338405 之间。

6.51.1 EXP 函数例子

- 该函数返回数值 3.49034295746184128E+000 或近似值 3.4903：
  ```
  EXP (1.25)
  ```

- 该函数返回数值 2.0。函数 EXP 为函数 LOG 的反函数：
  ```
  EXP (LOG(2.0))
  ```

6.52 EXPLAIN 函数

该函数是一种表值存储函数，它生成一个描述了 SELECT、INSERT、DELETE 或 UPDATE 语句的执行计划的结果表。更多信息，请参阅 EXPLAIN 函数的结果。

如果其前面有一个关键字 TABLE、并且用括号括了起来，则 EXPLAIN 函数可被指定为 SELECT 语句的 FROM 子句里的表引用。更多信息，请参阅 EXPLAIN 函数的结果。

```
EXPLAIN (module,' statement-pattern')

module is:
'module-name' | NULL
```

'module-name'
留作将来使用。
模块名称用单引号括起来，并区别大小写。如果一个模块名称为大写字母，则您在单引号中指定的值必须为大写字母。例如：'MYCAT.MYSCH.MYPROG'。

NULL
表示会话中前面刚 Prepared 语句。

'statement-pattern'
表示一个区别大小写大小写敏感性语句的样式，并用单引号括起来。该语句名必须为大写，除非您在一条 PREPARE 语句里将其分隔开。

6.52.1 EXPLAIN 函数注意事项

6.52.1.1 使用一种语句模式

语句模式的使用类似于 LIKE 模式的使用。您通过以下方式使用 LIKE 模式：

```
select * from table (explain(NULL,'S%'));
select * from table (explain(NULL,'S1'));
select * from table (explain(NULL,'%1'));
```

然而，不能以这种方式使用 LIKE 样式：

```
SELECT * FROM TABLE (EXPLAIN (NULL, '%'))
```

如下语句为名字以大写字母'S'开头的所有预处理语句返回 EXPLAIN 函数的结果：

```
SELECT * FROM TABLE (EXPLAIN (NULL,'S%'))
```

如果该语句样式没有找到任何匹配的语句名，则将和 SELECT 语句的结果一样不会返回任何行。

6.52.1.2 查询运行中获取一个 EXPLAIN 计划

EsgynDB SQL 能在运行带有 QID 选项的查询期间随时捕获 EXPLAIN 计划。默认情况下，这种行为在 EsgynDB 会话不开启。

注意： 开始准备和执行查询时，请启用此功能。

启用该功能后，请在一个EXPLAIN函数中使用以下语法获取一个正在运行的查询的执行计划：

```
SELECT * FROM TABLE (EXPLAIN(NULL, 'QID=qid'))
```

QID 是一个大小写区别标识符，表示查询 ID。例如：

```
'QID=MXID01001011194212103659400053369000000085905admin00
_2605_S1'
```

EXPLAIN 函数或语句在预处理一条查询时返回所生成的计划。QID 的 EXPLAIN 从正在执行的查询的原始计划里检索所有信息。查询执行完成，删除或是释放毕之前，该计划都可供使用。

6.52.1.3 EXPLAIN 函数的结果

EXPLAIN 函数的结果表描述了 SELECT、INSERT、DELETE 或 UPDATE 语句的访问计划。在EXPLAIN函数的结果描述里，在运算符树结构将访问计划里所用的运算符都表示为节点，树中的每个节点最多有一个父节点，并且只能有一个根节点。

运算符树的一个节点是树中的一个点，其表示计划里（涉及到运算符）的一个事件，每个节点都带有下级节点，即该计划里的每个事件都可能生成一个或多个下级事件。

列名	数据类型	描述
MODULE_NAME	CHAR(60)	留作将来使用。
STATEMENT_ NAME	CHAR(60)	语句名；如果长度超过 60 个字符，则将从右侧开始截断。
PLAN_ID	LARGEINT	EsgynDB SQL 自动分配的

		系统生成计划的唯一ID；该 ID 在编译期生成 （生成的是ID）。
SEQ_NUM	INT	运算符树中当前运算符的序号；表示其中生成运算符树的序列。
OPERATOR	CHAR(30)	当前运算符类型。
LEFT_CHILD_ SEQ_NUM	INT	当前运算符的第一个子运算符的序号；如果一个节点没有子运算符，则该序号为空。
RIGHT_CHILD_ SEQ_NUM	INT	当前运算符的第二个子运算符的序号；如果一个节点没有第二个子运算符，则该序号为空。
TNAME	CHAR(60)	对于扫描组里面的运算符，是基表的全名，如果长度超过列的限制，从右侧截断。如果其长度超过列的限制的话。如果相互关系名称不同于表名，则应在括号里依次列出相互关系名称和表名。
CARDINALITY	REAL	当前的运算符所返回的估算行数。基数将以某 EXPLAIN 输出的形式作为 ROWS/REQUEST 出现。对于一个嵌套联接的右侧子运算符而言，将基数乘以请求数量该运算符所产生的总行数。
OPERATOR_COST	REAL	与当前运算符执行相关的估算成本。
TOTAL_COST	REAL	与当前运算符执行相关的

		估算成本，其中包括运算符树的所有子树的成本。
DETAIL_COST	VARCHAR (200)	下表中将描述的 5 项成本向量。
DESCRIPTION	VARCHAR (3000)	关于运算符的其他信息。

EXPLAIN 函数结果的 DETAIL_COST 列包含这些成本因素：

CPU_TIME	处理器执行该运算符的指令所需的秒数估值；数值 1.0 代表 1 秒钟。
IO_TIME	执行该运算符的 I/O 操作（查找+数据传输）所需的秒数估值。
MSG_TIME	该运算符发送消息所需的秒数估值。该估值包括本地和远程消息的数量和数据发送量。
IDLETIME	等待一个事件发生的秒数估值。该估值包含打开一个表或启动一个 ESP 进程所需的时间量。
PROBES	运算符将被执行的次数。通常，该数值为 1，但如需对嵌套循环联接进行内部扫描，该数值还可提高。

6.52.2 EXPLAIN 函数例子

- 显示预处理语句 REGION 运行 EXPLAIN 函数后的结果表里指定的列：

```
>>select seq_num, operator, operator_cost from table
(explain (null, 'REG'));

SEQ_NUM     OPERATOR                        OPERATOR_COST
----------- ------------------------------- -------------
          1 TRAFODION_SCAN                   0.43691027
          2 ROOT 0.0

--- 2 row(s) selected.
>>log;
```

该例子仅显示了 EXPLAIN 函数的部分结果表。它首先使用 EXPLAIN 函数生成该表，然后选择所需的列。

6.53 EXTERNALTOLOB 函数

6.53.1 EXTERNALTOLOB 函数语法

EXTERNALTOLOB 函数将外部文件中的数据转换为 LOB 格式。该函数可用于 INSERT 或 UPDATE 语句。

EXTERNALTOLOB 函数是一个 EsgynDB SQL 扩展函数。

更多信息，请参阅 EsgynDB SQL Large Objects 指南。

```
EXTERNALTOLOB('external lob source file name')


external lob source file name is:
hdfs:///{local hdfs file name}
```

6.53.2 EXTERNALTOLOB 函数注意事项

- 当通过 EXTERNALTOLOB('external lob source file name')指定一个外部 LOB 时，与外部 HDFS 文件关联的数据不会被传输到 EsgynDB LOB 中。 相反，EsgynDB 存储外部文件的文件路径/句柄。

  例如，如果您有一个图片目录，则可以为该函数指定每个图片文件的完整 hdfs 路径，该路径将存储在 EsgynDB 表中。 在稍后的检索过程中，文件名将用于转到实际文件以检索数据。

6.53.3 EXTERNALTOLOB 函数例子

- 该例子使用不同的函数将字符串、文件、外部 lob 转换为 LOB 数据。EXTERNALTOLOB 函数接受外部文件。

  ```
  insert into tlob130ext values(1, stringtolob('first
  lob'),
  filetolob('hdfs:///lobs/lob_input_a1.txt'),
  externaltolob('hdfs:///lobs/lob_input_a1.txt'));
  ```

- 在表 clob130ext 中，c1 为 2，该例子将 c4 更新（覆盖）为存储在 hdfs:///lobs/ 中的 lob_input_d1.txt。

  ```
  update tlob130ext set
  c4=externaltolob('hdfs:///lobs/lob_input_d1.txt') where
  c1=2;
  ```

6.54 EXTRACT 函数

该函数从一个日期时间或时间间隔值表达式里抽取一个日期时间字段，它返回一个精确数字型数值。

```
EXTRACT (datetime-field FROM extract-source)

datetime-field is:
YEAR | MONTH | DAY | HOUR | MINUTE | SECOND
```

6.54.1.1 Extract 函数语法

- *datetime-field* 和 *extract-source* 是:

datetime-field	**描述**	**支持的** *extract-source*
CENTURY	世纪。	* *datetime-expression* **DATE** TIMESTAMP
DAY	* *datetime-expression* 日期。* *interval-expression* *intervalexpression*中的天数。	* *datetime-expression* **DATE** TIMESTAMP
DECADE	* *datetime-expression* 年份字段除以10。* *intervalexpression* *Intervalexpression*中的十年数。	* *datetime-expression* **DATE** TIMESTAMP * *interval-expression*
DOW	一周中的某一天(1-7)，其中1是周天，6是周六。 不可配置。	* *datetime-expression* **DATE** TIMESTAMP
DOY	一年中的某一天(1-366)。	* *datetime-expression* **DATE** TIMESTAMP
EPOCH	* *datetime-expression* 自1970-01-01 00:00:00开始的秒数。. 如果 *datetimeexpression* 早于 1970-01-01 00:00:00，则该值为负数。 * *interval-expression* *Intervalexpression*中	* *datetime-expression* **DATE** TIMESTAMP * *interval-expression*

	的秒数。	
HOUR	* *datetime-expression* 小时 (0-23). * *interval-expression* *intervalexpression*中的小时数。	* *datetime-expression* **TIME** TIMESTAMP * *intervalexpression*
MINUTE	* *datetime-expression* 分钟 (0-59). * *interval-expression* *intervalexpression*中的分钟数。	* *datetime-expression* **TIME** TIMESTAMP * *interval-expression*
MONTH	* *datetime-expression* 月份 (1-12). * *interval-expression* *intervalexpression*中的月份数。	* *datetime-expression* **DATE** TIMESTAMP * *interval-expression*
QUARTER	* *datetime-expression* 季度(1-4). * *intervalexpression* *intervalexpression*中的季度数。	* *datetime-expression* **DATE** TIMESTAMP * *interval-expression*
SECOND	* *datetime-expression* 秒数 (0-59). * *interval-expression* *intervalexpression*中的秒数。	* *datetime-expression* **TIME** TIMESTAMP * *intervalexpression*
WEEK	* *datetime-expression* 一年中的某一周。 **如果一年从周天开始，则对于当年前7天内发生的日期时间，返回值1。** **否则，对于第一个周天开始之前的星期内发生的日期时间，返回值1。**对于发生在一年中最后一个完整周或部分周的日期时间，返回值**53**。* 对于在闰年的最后一个完整周或部分周（从周六开始）中发生的日期时间，返回值**54**。	* *datetime-expression* **DATE** TIMESTAMP
WOM	* *datetime-expression*	* *datetime-*

	一月中的某一周。	*expression* **DATE** TIMESTAMP
YEAR	* *datetime-expression* 年份(0001-9999). * *interval-expression* *intervalexpression*中的年数。	* *datetime-expression* **DATE** TIMESTAMP * *interval-expression*

extract-source is:
datetime-expression | *interval-expression*
更多信息，请参阅日期时间值表达式和时间间隔值表达式。

6.54.2 EXTRACT 函数例子

6.54.2.1 EXTRACT (CENTURY)例子

- 该例子从一个 *DATE* 值里抽取世纪。
```
SQL>SELECT EXTRACT (CENTURY FROM DATE '0001-02-03')
FROM DUAL;
(EXPR)
------
1
--- 1 row(s) selected.
```

- 该例子从一个 *TIMESTAMP* 值里抽取世纪。
```
SQL>SELECT EXTRACT (CENTURY FROM TIMESTAMP '9899-12-31
23:59:59') FROM DUAL;
(EXPR)
------
99
--- 1 row(s) selected.
```

6.54.2.2 EXTRACT (DAY)例子

- 该例子从一个 *DATE* 值里抽取日期。
```
SQL>SELECT EXTRACT (DAY FROM DATE '2019-02-01') FROM
DUAL;
(EXPR)
------
     1
--- 1 row(s) selected.
```

- 该例子从一个 *TIMESTAMP* 值里抽取日期。
```
SELECT EXTRACT (DAY FROM TIMESTAMP '2019-12-31
11:43:00') FROM DUAL;
(EXPR)
------
```

```
    31
--- 1 row(s) selected.
```

- 该例子从一个 *interval-expression* 值里抽取天数。

```
SQL>SELECT EXTRACT (DAY FROM INTERVAL '1' DAY -
INTERVAL '99' DAY) FROM DUAL;
(EXPR)
------
   -98
--- 1 row(s) selected.
```

6.54.2.3 EXTRACT (DECADE)例子

- 该例子从一个 *DATE* 值里抽取十年。

```
SQL>SELECT EXTRACT (DECADE FROM DATE '0010-02-03') FROM
DUAL;
(EXPR)
------
1
--- 1 row(s) selected.
```

- 该例子从一个 *TIMESTAMP* 值里抽取十年。

```
 SQL>SELECT EXTRACT (DECADE FROM TIMESTAMP '9999-12-31
 23:59:59') FROM DUAL;
 (EXPR)
 ------
 999
 --- 1 row(s) selected.
```

- 该例子从一个 *interval-expression* 值里抽取十年的次数。

```
SQL>SELECT EXTRACT (DECADE FROM INTERVAL '1'YEAR -
INTERVAL '99'YEAR) FROM DUAL;
(EXPR)
------
-9
--- 1 row(s) selected.
```

6.54.2.4 EXTRACT (DOW)例子

- 该例子从一个 *DATE* 值里抽取 DOW。

```
SQL>SELECT EXTRACT (DOW FROM DATE '2017-12-31') FROM
DUAL;
(EXPR)
------
     1
--- 1 row(s) selected.
```

- 该例子从一个 *TIMESTAMP* 值里抽取 DOW。

```
SQL>SELECT EXTRACT (DOW FROM TIMESTAMP '2018-01-01
23:59:59') FROM DUAL;
(EXPR)
------
     2
--- 1 row(s) selected.
```

6.54.2.5 EXTRACT (DOY)例子

- 该例子从一个 *DATE* 值里抽取 DOY。

```
SQL>SELECT EXTRACT (DOY FROM DATE '2019-01-01') FROM
DUAL;
(EXPR)
------
     1
--- 1 row(s) selected.
```

- 该例子从一个 *TIMESTAMP* 值里抽取 DOY。

```
SQL>SELECT EXTRACT (DOY FROM TIMESTAMP '2020-12-31
11:36:28') FROM DUAL;
(EXPR)
------
   366
--- 1 row(s) selected.
```

6.54.2.6 EXTRACT (EPOCH)例子

- 该例子从一个 *DATE* 值里抽取自 1970-01-01 00:00:00 开始的秒。

```
SQL>SELECT EXTRACT (EPOCH FROM DATE '1970-01-02') FROM
DUAL;
(EXPR)
------------
86400
--- 1 row(s) selected.
```

- 该例子从一个 *TIMESTAMP* 值里抽取自 1970-01-01 00:00:00 开始的秒。

```
SQL>SELECT EXTRACT (EPOCH FROM TIMESTAMP '1969-12-31
23:59:59') FROM DUAL;
(EXPR)
------------
-1
--- 1 row(s) selected.
```

- 这是从 *interval-expression* 里抽取秒的第一个例子。

```
SQL>SELECT EXTRACT (EPOCH FROM INTERVAL '3' HOUR) FROM
DUAL;
(EXPR)
```

```
------------
10800
--- 1 row(s) selected.
```

- 这是从 *interval-expression* 里抽取秒的第二个例子。

```
SQL>SELECT EXTRACT (EPOCH FROM INTERVAL '1' MONTH -
INTERVAL '2' YEAR) FROM DUAL;
(EXPR)
------------
   -60069600
--- 1 row(s) selected.
```

6.54.2.7 EXTRACT (HOUR)例子

- 该例子从一个 *DATE* 值里抽取小时。

```
SQL>SELECT EXTRACT (HOUR FROM TIME '00:01:02') FROM
DUAL;
(EXPR)
------
     0
--- 1 row(s) selected.
```

- 该例子从一个 *TIMESTAMP* 值里抽取小时。

```
SQL>SELECT EXTRACT (HOUR FROM TIMESTAMP '1990-11-02
23:16:26') FROM DUAL;
(EXPR)
------
23
--- 1 row(s) selected.
```

- 该例子从一个 *interval-expression* 值里抽取小时数。

```
SQL>SELECT EXTRACT (HOUR FROM INTERVAL '1' HOUR -
INTERVAL '99' HOUR) FROM DUAL;
   (EXPR)
   ------
      -98
   --- 1 row(s) selected.
```

6.54.2.8 EXTRACT (MINUTE)例子

- 该例子从一个 *DATE* 值里抽取分钟。

```
SQL>SELECT EXTRACT (MINUTE FROM TIME '01:00:02') FROM
DUAL;
(EXPR)
------
```

```
     0
--- 1 row(s) selected.
```

- 该例子从一个 *TIMESTAMP* 值里抽取分钟。

```
SQL>SELECT EXTRACT (MINUTE FROM TIMESTAMP '1990-11-02
23:59:16') FROM DUAL;
(EXPR)
------
    59
--- 1 row(s) selected.
```

- 该例子从一个 *interval-expression* 值里抽取分钟数。

```
SQL>SELECT EXTRACT(MINUTE FROM INTERVAL '1' MINUTE -
INTERVAL '99' MINUTE) FROM
DUAL;
(EXPR)
------
-98
--- 1 row(s) selected.
```

6.54.2.9 EXTRACT (MONTH)例子

- 该例子从一个 *DATE* 值里抽取月份。

```
SQL>SELECT EXTRACT (MONTH FROM DATE '2019-01-02') FROM
DUAL;
(EXPR)
------
1
--- 1 row(s) selected.
```

- 该例子从一个 *TIME* 值里抽取月份。

```
SQL>SELECT EXTRACT (MONTH FROM TIMESTAMP '2019-12-31
23:59:59') FROM DUAL;
(EXPR)
------
12
--- 1 row(s) selected.
```

- 该例子从一个 *TIMESTAMP* 值里抽取月份数。

```
SQL>SELECT EXTRACT(MONTH FROM INTERVAL '1' MONTH -
INTERVAL '99' MONTH) FROM DUAL;
(EXPR)
------
-98
--- 1 row(s) selected.
```

6.54.2.10EXTRACT (QUARTER)例子

- 该例子从一个 *DATE* 值里抽取季度。
```
SQL>SELECT EXTRACT (QUARTER FROM DATE '2019-01-01')
FROM DUAL;
(EXPR)
------
1
--- 1 row(s) selected.
```

- 该例子从一个 *TIMESTAMP* 值里抽取季度。
```
SQL>SELECT EXTRACT (QUARTER FROM TIMESTAMP '1990-11-02
08:16:26') FROM DUAL;
(EXPR)
------
4
--- 1 row(s) selected.
```

- 该例子从一个 *interval-expression* 值里抽取季度数。
```
SQL>SELECT EXTRACT (QUARTER FROM INTERVAL '97' MONTH)
FROM DUAL;
(EXPR)
------
33
--- 1 row(s) selected.
```

6.54.2.11EXTRACT (SECOND)例子

- 该例子从一个 *DATE* 值里抽取秒。
```
SQL>SELECT EXTRACT (SECOND FROM TIME '01:02:00') FROM
DUAL;
(EXPR)
------
0
--- 1 row(s) selected.
```

- 该例子从一个 *TIMESTAMP* 值里抽取秒。
```
SQL>SELECT EXTRACT (SECOND FROM TIMESTAMP '1990-11-02
23:16:59') FROM DUAL;
(EXPR)
------
59
--- 1 row(s) selected.
```

- 该例子从一个 *interval-expression* 值里抽取秒数。
```
SQL>SELECT EXTRACT(SECOND FROM INTERVAL '1' SECOND -
INTERVAL '99' SECOND) FROM
DUAL;
(EXPR)
-----------
```

```
-98.000000
--- 1 row(s) selected.
```

6.54.2.12EXTRACT (WEEK)例子

- 该例子从一个 *DATE* 值里抽取周。

```
SQL>SELECT EXTRACT (WEEK FROM DATE '2019-01-01') FROM
DUAL;
(EXPR)
------
1
--- 1 row(s) selected.
```

- 该例子从一个 *TIMESTAMP* 值里抽取周。

```
SQL>SELECT EXTRACT (WEEK FROM TIMESTAMP '2000-12-31
23:59:59') FROM DUAL;
(EXPR)
------
54
--- 1 row(s) selected.
```

6.54.2.13EXTRACT (WOM)例子

- 该例子从一个 *DATE* 值里抽取 WOM。

```
SQL>SELECT EXTRACT (WOM FROM DATE '2019-01-01') FROM
DUAL;
(EXPR)
------
1
--- 1 row(s) selected.
```

- 该例子从一个 *TIMESTAMP* 值里抽取 WOM。

```
SQL>SELECT EXTRACT (WOM FROM TIMESTAMP '2020-02-29
01:02:30') FROM DUAL;
(EXPR)
------
5
--- 1 row(s) selected.
```

6.54.2.14EXTRACT (YEAR)例子

- 该例子从一个 *DATE* 值里抽取年份。

```
SQL>SELECT EXTRACT (YEAR FROM DATE '0001-02-03') FROM
DUAL;
(EXPR)
------
```

```
     1
--- 1 row(s) selected.
```

- 该例子从一个 *TIMESTAMP* 值里抽取年份。

```
SQL>SELECT EXTRACT (YEAR FROM TIMESTAMP '9999-12-31
23:59:59') FROM DUAL;
(EXPR)
------
9999
--- 1 row(s) selected.
```

- 该例子从一个 *interval-expression* 值里抽取年份。

```
SQL>SELECT EXTRACT (YEAR FROM INTERVAL '01-09' YEAR TO
MONTH) FROM DUAL;
(EXPR)
------
1
--- 1 row(s) selected.
```

- 该例子从一个 *interval-expression* 值里抽取年份数。

```
SQL>SELECT EXTRACT(YEAR FROM INTERVAL '1' YEAR -
INTERVAL '99' YEAR) FROM DUAL;
(EXPR)
------
-98
--- 1 row(s) selected.
```

6.55 FILETOLOB 函数

FILETOLOB 函数将本地 Linux/hdfs 文件中的数据转换为 LOB 格式。该函数可在 INSERT 或 UPDATE 语句中使用。

FILETOLOB 函数是 EsgynDB SQL 扩展函数。

更多信息，请参阅 EsgynDB SQL Large Objects 指南。

```
FILETOLOB('lob source file name')
```

lob 源文件名为:

```
hdfs:///{local hdfs file name} |
{local linux file name} |
{file:///linux file name}
```

6.55.1 FILETOLOB 函数例子

- 该例子将本地文件中的数据转换为 LOB 格式，并将所有数据存储到与该列/行关联的 HDFS 中。

  ```
  insert into tlob130txt1
  values(1,filetolob('lob_input_a1.txt'));
  ```

- 在 tlob1 表中，c1 为 3，该例子将 c2 更新（附加）到存储在 hdfs:///lobs/中的 lob_update.txt。

  ```
  update tlob1 set
  c2=filetolob('hdfs:///lobs/lob_update.txt', append)
  where c1 = 3;
  ```

6.56 FIND_IN_SET 函数

该函数是MySQL兼容函数，该函数返回某个字符串在指定字符串集中的位置。

```
FIND_IN_SET(str, strlist)
```

6.56.1 FIND_IN_SET 函数语法

str
查询的字符串

strlist
字段或字符串集合，如果是字符串集则以”,”分隔如 ('a,b,c,d')

该函数查询字段(strlist)中包含(str) 的结果，返回结果为 null 或。

如果字符串 str 位于由 N 个子字符串组成的字符串列表 strlist 中，则返回 1 到 N 之间的值。字符串列表是一个由逗号分隔的子字符串组成的字符串。如果 str 不在 strlist 中或 strlist 是空字符串，则返回 0。如果任一参数为 NULL，则返回 NULL。如果第一个参数包含逗号（,）字符，则此函数出错。

6.56.2 FIND_IN_SET 函数例子

- 该函数返回 b 在 'a,b,c,d'中的位置，位置从 1 开始计数。

```
>>SELECT FIND_IN_SET('b', 'a,b,c,d') from dual;

(EXPR)
----------

         2

--- 1 row(s) selected.
```

- 该函数返回'abc'在列 c1 中的位置。

```
>>select  find_in_set('abc',c1),c1 from
table_find_in_set;

(EXPR)      C1
----------  ---------------

         1  abc,bcd,def,efg
         0  abc?bcd?def?efg
         0  abc%bcd%def%efg
         0  aaa
```

--- 4 row(s) selected.

6.57 FLOOR 函数

该函数返回小于或等于一个数字值表达式的最大整数，它被表示为 FLOAT 数据类型。

FLOOR 函数是 EsgynDB SQL 扩展函数。

```
FLOOR (numeric-expression)
```

numeric-expression

这是一种 SQL 数字值表达式，它指定 FLOOR 函数的参数值。更多信息，请参阅字符值表达式。

更多信息，请参阅数字值表达式。

- 如果数值表达式是一个小数，则小数部分会被截断，结果的标度为 0。

例子

```
SELECT FLOOR (-1.23), FLOOR (1.23), FLOOR (-12.3), FLOOR
(12.3) FROM DUAL;
(EXPR) (EXPR) (EXPR) (EXPR)
------------- ------------ ------------ -----------
-2 1 -13 12
--- 1 row(s) selected.
```

- 如果数值表达式是一个整数，则结果是其本身。

例子

```
SELECT FLOOR (-99), FLOOR (-1), FLOOR (0), FLOOR (1),
FLOOR (99) FROM DUAL;
(EXPR) (EXPR) (EXPR) (EXPR) (EXPR)
------ ------ ------ ------ ------
-99 -1 0 1 99
--- 1 row(s) selected.
```

- 在两个相反的数中，负的 *numeric-expression* 结果的绝对值大于正的 *numeric-expression* 结果的绝对值。

例子

```
SELECT FLOOR (-99.99), FLOOR (99.99) FROM DUAL;
(EXPR) (EXPR)
----------- ------
-100 99
```

```
--- 1 row(s) selected.
```

6.57.1 FLOOR 注意事项

6.57.1.1 FLOOR 和 CEIL 的比较

CEIL (CEILING)函数与 FLOOR 函数**相反**，它向下舍入 *numeric-expression*，并返回大于或等于 *numeric-expression* 的最小整数。

```
          -1.5                                  1.5
            ¬                                    ¬
---- -2 ----·---- -1 --------0-------- 1 ----·---- 2 ----
      ¬            ¬                    ¬            ¬
   FLOOR(-1.5) CEIL(-1.5)           FLOOR(1.5)   CEIL(1.5)
```

numeric-expression	**FLOOR**	**CEIL**
0.3	0	1
1.8	1	2
2.37	2	3
3.84	3	4
4.123	4	5
5.999	5	6
-0.3	-1	0
-1.8	-2	-1
-2.37	-3	-2
-3.84	-4	-3
-4.123	-5	-4
-5.999	-6	-5

6.57.2 FLOOR 例子

- 该例子使用 FLOOR 函数清晰简洁地显示了城市的经度和纬度。

```
SELECT * FROM city_info;
CITY                 LONGITUDE         LATITUDE
-------------------- ----------------- ------------------
----
Shanghai             121.522179        31.267401
Muenster             7.626100          51.960700
Amsterdam            4.903600          52.368000
Paris                2.352200          48.856600
Vienna               16.373800         48.208200
Helsinki             24.938400         60.169900
--- 6 row(s) selected.

SELECT city,
FLOOR (longitude) AS Floor_Longitude,
FLOOR (latitude) AS Floor_Latitude
FROM city_info
ORDER BY longitude DESC;
```

```
CITY                 FLOOR_LONGITUDE   FLOOR_LATITUDE
-------------------- ----------------- ------------------
----
Shanghai             121.0             31.0
Helsinki             24.0              60.0
Vienna               16.0              48.0
Muenster             7.0               51.0
Amsterdam            4.0               52.0
Paris                2.0               48.0
--- 6 row(s) selected.
```

6.58 FROM_UNIXTIME 函数

该函数将时间戳转换成日期时间;

6.58.1 FROM_UNIXTIME 例子

SELECT FROM_UNIXTIME(UNIX_TIMESTAMP()) FROM DUAL;
(EXPR)

2019-03-11 10:24:22
--- 1 row(s) selected.

6.59 GROUP_CONCAT 函数

该函数返回一个字符串结果，其中包含组中连接的非空值。如果没有非空值，则返回空值。语法如下:

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name |
expr}
                 [ASC | DESC] [,col_name ...]]
             [SEPARATOR str_val])
```

获取表达式组合的串联值。要消除重复值，使用 DISTINCT 子句。要对结果中的值排序，使用 ORDER BY 子句。要按相反的顺序排序，将 DESC（降序）关键字添加到 ORDER BY 子句中要排序的列的名称中。默认为升序，可使用 ASC 关键字显式指定。组中值之间的默认分隔符是逗号（,）。若要显式指定分隔符，使用 SEPARATOR，后跟应在插入到组值之间的字符串字面值。若要完全消除分隔符，则需指定 SEPARATOR ''。

6.59.1 GROUP_CONCAT 例子

以下例子返回每个学生 test_score 列的连接字符串。

```
>> SELECT student_name,
         GROUP_CONCAT(DISTINCT test_score
                   ORDER BY test_score DESC SEPARATOR '
')
         FROM student
         GROUP BY student_name;
STUDENT_NAME   (EXPR)
-------------- --------------
scott          91 90 80 56
tom            91 77 43
```

6.60 HOUR 函数

该函数将一个 TIME 或 TIMESTAMP 表达式转换为 0 到 23 之间的一个 INTEGER 值，表示一天的对应的某一小时。

HOUR 函数是 EsgynDB SQL 扩展函数。

```
HOUR (datetime-expression)
```

datetime-expression

该表达式的求值结果为一个 TIME 或 TIMESTAMP 类型的日期时间值。

更多信息，请参阅日期时间值表达式。

6.60.1 HOUR 函数例子

- 返回一个整数，表示 PROJECT 表的 SHIP_TIMESTAMP 列里的一天中的某一小时：

```
SELECT start_date, ship_timestamp, HOUR(ship_timestamp)
FROM persnl.project
WHERE projcode = 1000;

Start/Date Time/Shipped                (EXPR)
---------- -------------------------- ------
2007-04-10 2007-04-21 08:15:00.000000 8
```

6.61 INSERT 函数

该函数返回一个字符串，其中已从指定的起始位置开始删除指定数量的字符，并且另一个字符串已被插入到起始位置。包含多字节字符在内的每个字符都将被视为一个字符。

INSERT 函数是 EsgynDB SQL 扩展函数。

INSERT (char-expr-1, start, length, char-expr-2)

char-expr-1, char-expr-2

这些都是（具有 CHAR 或 VARCHAR 数据类型的）SQL 字符值表达式，指定两个由多个字符组成的字符串。将字符串 *char-expr-2* 插入到字符串 *char-expr-1* 里。更多信息，请参阅字符值表达式。

start

指定 *char-expr-1* 里的起始位置，从这里开始删除的字符数长度。删除完毕后，从起始数字指定的起始位置开始，将字符串 *char-expr-2* 插入到字符串 *char-expr-1*， start 数字必须是一个大于 0、且标度为 0 的精确数字型数值。

length

指定 *char-expr-1* 里删除的字符数。*length* 数字必须是一个大于或等于 0、且标度为 0 的精确数字型数值。*length* 必须小于或等于 *char-expr-1* 的长度。

6.61.1 INSERT 函数例子

- 假设您的 JOB 表包含一个销售代表条目。请使用 INSERT 函数将 SALESREP 改为 SALES REP:

```
UPDATE persnl.job
SET jobdesc = INSERT (jobdesc, 6, 3, ' REP')
                 WHERE jobdesc = 'SALESREP';
```

现在检查一下您更新后的行:

```
SELECT jobdesc FROM persnl.job WHERE jobdesc = 'SALES
REP';

Job Description
------------------
SALES REP

--- 1 row(s) selected.
```

6.62 IS_IPV4 函数

对于一个给定的参数，如果它是一个有效的 IPV4 字符串，则 IS_IPV4()返回 1，否则返回 0。

```
IS_IPV4( expression )
```

expression

指定一个表达式，该表达式要确定将 IP 地址验证的值包含在内。表达式不能包含聚合函数或子查询。如果输入值为 NULL，则 IS_IPV4 返回 NULL。更多信息，请参阅表达式。

6.62.1 IS_IPV4 例子

该函数为第一个输入参数返回 1，因为它是一个有效的 IPV4 字符串；为第二个输入参数返回 0，因为它是一个无效的 IPV4 字符串。

```
>>SELECT IS_IPV4('10.0.5.9'), IS_IPV4('10.0.5.256') from
dual;
(EXPR) (EXPR)
------- -------
1       0
```

6.63 IS_IPV6 函数

如果参数是一个被指定为字符串的有效 IPv6 地址，则返回 1，否则返回 0。该函数不将 IPv4 地址视为有效的 IPv6 地址。

```
IS_IPV6( expression )
```

expression

指定一个表达式，该表达式要确定将 IP 地址验证的值包含在内。表达式不能包含聚合函数或子查询。如果输入值为 NULL，则 IS_IPV6 返回 NULL。更多信息，请参阅表达式。

6.63.1 IS_IPV6 例子

该函数为第一个输入参数返回 0，因为它是一个有效的 IPV6 字符串；为第二个输入参数返回 1，因为它是一个无效的 IPV6 字符串。

```
>>SELECT IS_IPV6('10.0.5.9'), IS_IPV6('::1') from dual;
(EXPR) (EXPR)
-------- -------
0        1
```

6.64 INET_ATON 函数

给定一个以字符串表示的 IPv4 网络地址的点分表示形式，该函数返回一个整数，该整数以网络字节顺（大端）表示地址的数值。如果 INET_ATON()不理解其参数，则返回 NULL。

```
INET_ATON( expression )
```

expression

指定一个表达式，该表达式要确定将 IP 地址转换的值包含在内。表达式不能包含聚合函数或子查询。如果输入值为 NULL，则 INET_ATON 返回 NULL。更多信息，请参阅表达式。

6.64.1 INET_ATON 例子

```
>>SELECT INET_ATON('10.0.5.9') from dual;
(EXPR)
-----------
151322634
```

6.65 INET_NTOA 函数

给定一个以网络字节序的数字 IPv4 网络地址，该函数将该地址的点分字符串表示形式返回为连接字符集中的一个非二进制字符串。如果 INET_NTOA()不理解其参数，则返回 NULL。

```
INET_NTOA( expression )
```

expression

指定一个表达式，该表达式要确定将数字转换为 IP 地址的值包含在内。表达式不能包含聚合函数或子查询。如果输入值为 NULL，则 INET_NTOA 返回 NULL。更多信息，请参阅表达式。

6.65.1 INET_NTOA 例子

该函数将把一个整数转换为 IP 地址的点分字符串表示形式。

```
>>SELECT INET_NTOA(167773449) from dual
(EXPR)
-------------
9.5.0.10
```

6.66 ISNULL 函数

第一个参数值不为 NULL 的情况下，ISNULL 函数将返回第一个参数值，否则将返回第二个参数值。两个表达式都必须具有可比类型。

ISNULL 函数是 EsgynDB SQL 扩展函数。

```
ISNULL(ck_expr, repl_value)
```

ck_expr

任何有效 SQL 数据类型的表达式。

repl_value

任何有效 SQL 数据类型的表达式，但必须具有和 ck_expr 可比的类型。

6.66.1 ISNULL 函数例子

- 如果数值为空，则该函数返回 0（而非空值）。
  ```
  ISNULL(value,0)
  ```
- 如果 date_col 为空，则该函数返回日期常数。
  ```
  ISNULL(date_col, DATE '2006-01-01')
  ```
- 如果字符串列 last_name 为空，则该函数返回 'Smith'。
  ```
  ISNULL(last_name, 'Smith')
  ```

6.67 JULIANTIMESTAMP 函数

该函数将一个日期时间值转换为 64 位儒略时间戳值，该值表示在公元前 4713, January 1, 00:00 和指定的日期时间值之间流逝的微秒数。JULIANTIMESTAMP 返回一个 LARGEINT 数据类型的值。

当一条查询开始执行时，该函数将被求值一次，并且不会被再次求值（即使它是一条长时间运行的查询，也是如此）。

JULIANTIMESTAMP 函数是 EsgynDB SQL 扩展函数。

```
JULIANTIMESTAMP (datetime-expression)
```

datetime-expression

该表达式的求值结果为一个 DATE、TIME 或 TIMESTAMP 类型的值。如果 *datetime-expression* 不包含从 YEAR 到 SECOND 的所有字段，EsgynDB SQL 将首先扩展该值，然后将其转换为一个儒略时间戳。Datetime 指定的日期时间值左侧的日期时间字段被设置为当前日期字段，日期时间值右侧的日期时间字段则被设置为 0。更多信息，请参阅日期时间值表达式。

6.67.1 JULIANTIMESTAMP 函数注意事项

The *datetime-expression* 的值必须为一个日期或时间戳值，范围从 0001 年年初到 9999 年年末。

6.67.2 JULIANTIMESTAMP 函数例子

PROJECT 表包含 5 列，它们分别使用数据类型 NUMERIC、VARCHAR、DATE、TIMESTAMP 和 INTERVAL。

- 将 TIMESTAM 值转换为一个儒略时间戳表达式:

```
SELECT ship_timestamp, JULIANTIMESTAMP (ship_timestamp)
FROM persnl.project
WHERE projcode = 1000;

SHIP_TIMESTAMP            (EXPR)
-------------------------- --------------------
2008-04-21 08:15:00.000000 212075525700000000

--- 1 row(s) selected.
```

- 将 DATE 值转换为一个儒略时间戳表达式:

```
SELECT start_date, JULIANTIMESTAMP (start_date)
FROM persnl.project
WHERE projcode = 1000;

START_DATE (EXPR)
---------- --------------------
2008-04-10 212074545600000000

--- 1 row(s) selected.
```

6.68 LCASE 函数

该函数转为小写字母数字字符。对于非字母数字字符而言，LCASE 返回相同的字符。LCASE 将出现在一条查询中使用一个数值的任何位置，例如选择列表、ON 子句、WHERE 子句、HAVING 子句、LIKE 谓词、一种表达式，也可在 UPDATE 或 INSERT 语句里限定一个新值。 LCASE 函数的返回结果等于“LOWER 函数”的返回结果。

LCASE 函数返回一个由固定长度或可变长度字符数据组成的字符串，根据输入字符串的数据类型而定。

LCASE 函数是 EsgynDB SQL 扩展函数。

```
LCASE (character-expression)
```

character-expression

这是一种 SQL 字符值表达式，它指定一个需要转为小写的由多个字符组成的字符串。更多信息，请参阅字符值表达式。

6.68.1 LCASE 函数例子

- 假设您的 CUSTOMER 表包含一个 Hotel Oregon（俄勒冈州旅馆）输入项。选择列 CUSTNAME，并使用 UCASE 和 LCASE 函数返回该列的大写字母和小写字母：

  ```
  SELECT custname,UCASE(custname),LCASE(custname) FROM
  sales.customer;
  (EXPR)              (EXPR)               (EXPR)
  ----------------- -------------------- -----------------
  ...                 ...                   ...

  Hotel Oregon        HOTEL OREGON         hotel oregon
  --- 17 row(s) selected.
  ```

  更多信息，请参阅 UCASE 函数。

6.69 LEFT 函数

LEFT 函数从一个字符表达式返回最左侧指定数量的字符。包含多字节字符在内的每个字符都被视为一个字符。

LEFTT 函数是 EsgynDB SQL 扩展函数。

```
LEFT (character-expr, count)
```

character-expr

指定一个源字符串，从中返回最左侧指定数量的字符。

这个源字符串是一种 SQL 字符值表达式，操作数为 character-expr 的求值结果。更多信息，请参阅字符值表达式。

count

指定从 character-expr 中返回的字符数。Count 必须是一个大于或等于 0、且标度为 0 的精确数值型数值。

6.69.1 LEFT 函数例子

- 返回 'Robert'：
  ```
  LEFT ('Robert John Smith', 6)
  ```

- 使用 LEFT 函数将“公司名”追加到“职位描述”后面：
  ```
  UPDATE persnl.job
  SET jobdesc = LEFT (jobdesc, 11) ||' COMNET';
  SELECT jobdesc FROM persnl.job;

  Job Description
  ------------------
  MANAGER COMNET
  PRODUCTION
  COMNET
  ASSEMBLER COMNET
  SALESREP COMNET
  SYSTEM ANAL COMNET
  ENGINEER COMNET
  PROGRAMMER COMNET
  ACCOUNTANT COMNET
  ADMINISTRAT COMNET
  SECRETARY COMNET
  ```

```
--- 10 row(s) selected.
```

6.70 LISTAGG 函数

该函数对列值进行排序后进行拼接，默认返回类型是 varchar(5000)。

EsgynDB 可以通过开关来控制返回字符串的长度，其取值范围是 50 - 4194303：
例如 cqd LISTAGG_MAX_LENGTH_IN_BYTES '1000';

```
LISTAGG(measure_expr [, 'delimiter'])
  WITHIN GROUP (order_by_clause)
```

对于指定的度量，LISTAGG 对 ORDER BY 子句中指定的每个组内的数据进行排序，然后连接度量列的值。

作为单集聚合函数，LISTAGG 对所有行进行操作并返回单个输出行。

作为一个组集聚合，该函数对 GROUP BY 子句定义的每个组进行操作并返回一个输出行。

measure_expr
可以是任何表达式。 度量列中的空值将被忽略。

delimiter_expr
指定用于分隔度量值的字符串。 此子句是可选的，默认为 NULL。

order_by_clause
确定返回连接值的顺序。 仅当 ORDER BY 列列表实现唯一排序时，该函数才是确定的。

6.70.1LISTAGG 函数的例子

- 以下示例列出了 staff 表中 Department 40 中的所有员工，按雇用日期和姓氏排序：

```
SELECT LISTAGG(last_name, '; ')
         WITHIN GROUP (ORDER BY hire_date, last_name)
"staff_list",
       MIN(hire_date) "Earliest"
  FROM staff
  WHERE department_id = 40;

staff_list                                        Earliest
---------------------------------------------------------
Raphaely; Khoo; Tobias; Baida; Himuro;          07-DEC-02
```

- 以下示例列出了 staff 表中每个部门 ID，该部门中的员工按其雇用日期的顺序排列：

```
SELECT department_id "DeptNo.",
       LISTAGG(last_name, '; ') WITHIN GROUP (ORDER BY
hire_date) "staff"
  FROM staff
  GROUP BY department_id
  ORDER BY department_id;

DeptNo. staff
------ ------------------------------------------------
------------
    10 Whalen
    20 Hartstein; Fay
    30 Raphaely; Khoo; Tobias; Baida; Himuro; Colmenares
    40 Mavris
    50 Kaufling; Ladwig; Rajs; Sarchand; Bell; Mallin
    60 Austin; Hunold; Pataballa; Lorentz; Ernst
    70 Poe
. . .
```

6.71 LOCATE 函数

该函数在一个字符串里搜索给定的子字符串。找到该子字符串后，EsgynDB SQL 将返回其在字符串里的字符位置。包括多字节字符在内的每个字符都被视为一个字符。LOCATE 函数返回的结果等于 POSITION 函数的返回结果。

LOCATE 是 EsgynDB SQL 扩展函数。

```
LOCATE (substring-expression, source-expression)
```

substring-expression

这是一种 SQL 字符值表达式，它指定需要在源表达式里搜索的子字符串。子字符串表达式不得为 NULL。更多信息，请参阅字符值表达式。

source-expression

这是一种 SQL 字符值表达式，它指定源字符串。*source-expression* 不得为 NULL。更多信息，请参阅字符值表达式。

EsgynDB SQL 的返回结果是一个标度为 0 的两个字节的有符号整数。如果 *source-expression* 里没有找到 *substring-expression*，EsgynDB SQL 将返回 0。

6.71.1 LOCATE 函数注意事项

6.71.1.1 LOCATE 函数结果

如果 *source-expression* 的长度为 0，而 *substring-expression* 的长度大于 0，则 EsgynDB SQL 将返回 0。如果 *substring-expression* 的长度为 0，则 EsgynDB SQL 将返回 1。

如果 *substring-expression* 的长度大于 *source-expression* 的长度，则 EsgynDB SQL 将返回 0。如果 *source-expression* 是一个空值，则 EsgynDB SQL 也将返回一个空值。

6.71.1.2 使用 UCASE

为了在搜索时忽略大小写字母的问题，请在 *substring-expression* 和 *source-expression* 中都使用 UCASE 函数（或 LCASE 函数）。

6.71.2 LOCATE 函数例子

- 返回数值 8，作为字符串中子字符串 'John' 的位置：
  ```
  LOCATE ('John','Robert John Smith')
  ```

- 假设 EMPLOYEE 表里有一个 EMPNAME 列，同时包含姓和名。该 SELECT 语句返回含有子字符串'SMITH'的 EMPLOYEE 表里的所有记录，无论该列值采用大写字符还是小写字符都是如此：
  ```
  SELECT * FROM persnl.employee
  WHERE LOCATE ('SMITH',UCASE(empname)) > 0 ;
  ```

6.72 LOG 函数

LOG 函数返回一个数字值表达式的自然对数。LOG 函数是一个 EsgynDB SQL 扩展函数。

```
LOG (numeric-expression)
```

numeric-expression

这是一种 SQL 数字值表达式，用来指定 LOG 函数的参数值。该参数值必须大于 0。更多信息，请参阅字符值表达式。

6.72.1 LOG 函数例子

- 该函数返回数值 6.93147180559945344E-001 或近似值 0.69315：
  ```
  LOG (2.0)
  ```

6.73 LOG10 函数

该函数返回一个数字值表达式以 10 为底的对数。 LOG10 函数是 EsgynDB SQL 扩展函数。

```
LOG10 (numeric-expression)
```

numeric-expression

这是一种 SQL 数字值表达式，用来指定 LOG10 函数的参数值。该参数值必须大于 0。更多信息，请参阅字符值表达式。

6.73.1 LOG10 函数例子

- 该函数返回数值 1.39794000867203776E+000 或近似值 1.3979：
  ```
  LOG10 (25)
  ```

6.74 LOWER 函数

LOWER 函数返回小写字母数字字符。对于非字母数字字符而言，LOWER 返回相同的字符。

LOWER 可以出现在一条查询中使用一个数值的任何位置，例如选择列表、ON 子句、WHERE 子句、HAVING 子句、LIKE 谓词和一个表达式，也可在 UPDATE 或 INSERT 语句里限定一个新值。LOWER 函数的返回结果等于 LCASE 函数的返回结果。

LOWER 函数返回一个由固定长度或可变长度字符数据的字符串，具体根据输入字符串的数据类型而定。

```
LOWER (character-expression)
```

character-expression

这是一种 SQL 字符值表达式，它指定一个需要返回小写由多个字符组成的字符串。更多信息，请参阅字符值表达式。

6.74.1 LOWER 函数注意事项

对于一个 UTF8 字符表达式而言，LOWER 函数将一个给定字符串里的所有大写或标题大写字符降档为小写字符，并返回一个与参数具有相同数据类型和字符集的字符串。

小写字符是一种具有 Unicode 2 标准里的“alphabetic”（字母）属性，并且 Unicode 名称中包含小写字母的字符。大写字符是一种具有 Unicode 2 标准里的“alphabetic”（字母）属性，并且 Unicode 名称中包含大写字母的字符。标题大写字符则是一种具有 Unicode “alphabetic”（字母）属性，并且 Unicode 名称中包含标题的字符。

6.74.2 LOWER 函数例子

假设您的 CUSTOMER 表包含一个 Hotel Oregon（俄勒冈州旅馆）输入项。选择列 CUSTNAME，并使用 UPPER 和 LOWER 函数返回该列的大写字母和小写字母：

```
SELECT custname,UPPER(custname),LOWER(custname) FROM
sales.customer;
(EXPR)            (EXPR)                (EXPR)
----------------- -------------------- ------------------
... ... ...
```

```
Hotel Oregon HOTEL OREGON hotel oregon
--- 17 row(s) selected.
```

更多信息，请参阅 UPPER 函数。

6.75 LPAD 函数

该函数使用指定的字符串填充一个字符串的左侧。该字符串里的每个字符（包含多字节字符在内）都被视为一个字符。

LPAD 函数是 EsgynDB SQL 扩展函数。

LPAD (str, len [,padstr])

str

是一种表达式。更多信息，请参阅字符值表达式。

len

标识需要返回的字符数。它是一种表达式，但必须是一个整数值。

如果 len 等于该字符串的长度，则无需进行更改。如果 len 小于字符串的大小，则需截断该字符串。

pad-character

既可以是表达式，也可以是字符串。

6.75.1 LPAD 函数例子

- 该函数返回' kite':
  `lpad('kite', 7)`

- 该函数返回 'ki':
  `lpad('kite', 2)`

- 该函数返回 '0000kite':
  `lpad('kite', 8, '0')`

- 该函数返回 'go fly a kite':
  `lpad('go fly a kite', 13, 'z')`

- 该函数返回 'John,John, go fly a kite":
  `lpad('go fly a kite', 23, 'John,')`

6.76 LTRIM 函数

该函数删除一个字符串的前导空格。如果您必须删除任何前导字符而非空格，则应使用 TRIM 函数、并指定字符值。更多信息，请参阅 TRIM 函数。

LTRIM 函数是 EsgynDB SQL 扩展函数。

```
LTRIM (character-expression)
```

character-expression

这是一种 SQL 字符值表达式，它指定一个需要去除前导空格的字符串。

更多信息，请参阅字符值表达式。

6.76.1 LTRIM 函数注意事项

6.76.1.1 LTRIM 函数结果

该结果始终为 VARCHAR 类型，并且其最大长度等于 *character-expression* 的固定长度或最大可变长度。

6.76.1.2 LTRIM 函数例子

- 返回 'Robert ':
  ```
  LTRIM (' Robert ')
  ```

更多信息，请参阅 TRIM 函数和 RTRIM 函数。

6.77 MAX/MAXIMUM 函数

该函数是一种聚合函数，它返回一组数值中的最大值。无论MAX函数出现在一条语句的何处，MAXIMUM 函数都和它等值。另外，该函数结果的数据类型与参数的数据类型相同。

```
MAX | MAXIMUM ([ALL | DISTINCT] expression)
```

ALL | DISTINCT

指定计算一个表达式的最大值时是否应将重复值包含在内。默认选项为 ALL，由此使得重复值被包含在内。如果您指定 DISTINCT 选项，则会在应用 MAX/MAXIMUM 函数之前消除重复值。

expression

指定一种表达式，以确定计算最大值时应包含的数值。

该表达式不得包含一个聚合函数或子查询。DISTINCT 子句指定 MAX/MAXIMUM 函数运行在表达式求值所派生的单列表的不同值上。将该函数应用到这组数值以前必须消除所有空值。如果结果表为空，MAX/MAXIMUM 函数将返回 NULL。

更多信息，请参阅表达式。

6.77.1 MAX/MAXIMUM 函数注意事项

6.77.1.1 表达式的操作数

该表达式包含 SELECT 结果表的行里的列，但不得包含一个聚合函数。以下这些表达式是有效的：

```
MAX (SALARY)
MAX (SALARY * 1.1)
MAX (PARTCOST * QTY_ORDERED)
```

6.77.2 MAX/MAXIMUM 函数例子

- 显示 SALARY 列的最大值：

```
SELECT MAX (salary) FROM persnl.employee;

(EXPR)
-----------
175500.00

--- 1 row(s) selected.
```

6.78 MD5 函数

计算字符串的MD5 128位校验和。 该值以一个32个十六进制数字的字符串返回，如果参数为NULL，则返回NULL。

```
MD5( _expression_)
```

expression

指定一个表达式，该表达式要确定将 MD5 计算的值包含在内。 表达式不能包含聚合函数或子查询。 如果输入值为 NULL，则 MD5 返回 NULL。 更多信息，请参阅表达式。

6.78.1 MD5 例子

返回值是连接字符集中的一个非二进制字符串。

```
>>SELECT MD5('testing') from dual;
(EXPR)
---------------------------------
'ae2b1fca515949e5d54fb22b8ed95575'
```

6.79 MIN 函数

该函数是一种聚合函数，它返回一组数值中的最小值。另外，该函数结果的数据类型与参数的数据类型相同。

```
MIN ([ALL | DISTINCT] expression)
```

ALL | DISTINCT

指定计算一个表达式的最小值时是否应将重复值包含在内。默认选项为 ALL，由此使得重复值被包含在内。如果您指定 DISTINCT 选项，则会在应用 MIN 函数之前消除重复值。

expression

指定一种表达式，以确定计算最小值时应包含的数值。

该表达式不得包含一个聚合函数或子查询。DISTINCT 子句指定 MIN 函数运行在表达式求值所派生的单列表的不同值上。将该函数应用到这组数值以前必须消除所有空值。如果结果表为空， MIN 函数将返回 NULL。更多信息，请参阅表达式。

6.79.1 MIN 函数注意事项

6.79.1.1 表达式的操作数

该表达式包含 SELECT 结果表的行里的列，但不得包含一个聚合函数。以下这些表达式是有效的：

```
MIN (SALARY)
MIN (SALARY * 1.1)
MIN (PARTCOST * QTY_ORDERED)
```

6.79.2 MIN 函数例子

- 显示 SALARY 列的最小值：

```
SELECT MIN (salary) FROM persnl.employee;

(EXPR)
-----------
17000.00

--- 1 row(s) selected.
```

6.80 MINUTE 函数

该函数将一个 DATE 或 TIMESTAMP 表达式转换为范围是 0 到 59 的 INTEGER（整型）值，表示一小时内的相应分钟数。

MINUTE 函数是 EsgynDB SQL 扩展函数。

```
MINUTE (datetime-expression)
```

datetime-expression

该表达式的求值结果为一个 DATE 或 TIMESTAMP 类型的日期时间值。更多信息，请参阅日期时间值表达式。

6.80.1 MINUTE 函数例子

- 返回一个整数，表示 PROJECT 表的 SHIP_TIMESTAMP 列里的一小时内的分钟数：

```
SELECT start_date, ship_timestamp,
MINUTE(ship_timestamp)
FROM persnl.project
WHERE projcode = 1000;

Start/Date Time/Shipped              (EXPR)
---------- -------------------------- ------
2008-04-10 2008-04-21 08:15:00.000000 15
```

6.81 MOD 函数

该函数返回一个整数值表达式除以另一个整数值表达式所得的余数（模数）。MOD 函数是 EsgynDB SQL 扩展函数。

```
MOD (integer-expression-1, integer-expression-1)
```

integer-expression-1

这是一种 SQL 数字值表达式，其数据类型为 SMALLINT、INTEGER 或 LARGEINT；指定 MOD 函数的被除数参数值。

integer-expression-2

这是一种 SQL 数字值表达式，其数据类型为 SMALLINT、INTEGER 或 LARGEINT；指定 MOD 函数的除数参数值。除数参数不得为 0。更多信息，请参阅字符值表达式。

从 EsgynDB 2.8.0 开始，支持使用运算符 % 实现 MOD 函数，两者功能相同。
以下是%的语法：
integer-expression-1% integer-expression-1

6.81.1 MOD 函数例子

- 该函数返回数值 2 作为余数或模：
  `MOD (11,3)`

- 使用%实现 MOD (11,3)
  `11%3`

6.82 MONTH 函数

该函数将一个 DATE 或 TIMESTAMP 表达式转换为范围是 1 到 12 的 INTEGER（整型）值，表示一年中的相应月份。

MONTH 函数是 EsgynDB SQL 扩展函数。

```
MONTH (datetime-expression)
```

datetime-expression

该表达式的求值结果为一个 DATE 或 TIMESTAMP 类型的日期时间值。更多信息，请参阅日期时间值表达式。

6.82.1 MONTH 函数例子

- 返回一个整数，表示 PROJECT 表的 START_DATE 列里的一年中的某个月份：

```
SELECT start_date, ship_timestamp, MONTH(start_date)
FROM persnl.project
WHERE projcode = 1000;

Start/Date Time/Shipped              (EXPR)
---------- --------------------------- ------
2008-04-10 2008-04-21 08:15:00.000000 4
```

6.83 MONTHNAME 函数

该函数将一个 DATE 或 TIMESTAMP 表达式转换为一个字符字面值，即一年中的某个月份的名称（一月、二月，等等）。

MONTHNAME 函数是 EsgynDB SQL 扩展函数。

```
MONTHNAME (datetime-expression)
```

datetime-expression

该表达式的求值结果为一个 DATE 或 TIMESTAMP 类型的日期时间值。更多信息，请参阅日期时间值表达式。

6.83.1 MONTHNAME 函数注意事项

The MONTHNAME 函数返回采用 ISO8859-1 编码的月份名称。

6.83.2 MONTHNAME 函数例子

- 返回一个字符字面值，表示 PROJECT 表的 START_DATE 列里的一年中的某个月份：

```
SELECT start_date, ship_timestamp, MONTHNAME(start_date)
FROM persnl.project
WHERE projcode = 1000;

Start/Date Time/Shipped              (EXPR)
---------- -------------------------- ---------
2008-04-10 2008-04-21 08:15:00.000000 April
```

6.84 NULLIF 函数

该函数比较两个表达式的值。这两个表达式必须是可比的类型。如果二者相等，则返回值为 NULL；如果二者不等，则返回值为第一个表达式的值。

NULLIF*(expr1, expr2)*

expr1
一种需要进行比较的表达式。

expr2
一种需要进行比较的表达式。

NULLIF(*expr1, expr2*) 等效于：

```
CASE WHEN expr1 = expr2
     THEN NULL
     ELSE expr1
END
```

如果两个参数相等，则 NULLIF 将返回 NULL（空值）；如果两个参数不等，则将返回第一个参数值。

6.84.1 NULLIF 函数例子

- 如果数值等于 7，则该函数将返回一个空值；如果数值不等于 7，则将返回第一个参数值。
  NULLIF(value,7)

6.85 NULLIFZERO 函数

如果该值不为零，则 NULLIFZERO 函数返回一个表达式的数值。否则，它将返回 NULL（空值）。

```
NULLIFZERO (expression)
```

expression
指定一个值表达式。它必须是一种数字数据类型。

6.85.1 NULLIFZERO 函数例子

- 如果该列值不为零，则该函数将为每一行都返回 Salary 列的值；如果该列值为零，则它将为表的每一行都返回 NULL（空值）。
  ```
  SELECT NULLIFZERO (salary) from employee_tab;
  ```

- 该函数将为表的每一行都返回数值 1：
  ```
  SELECT NULLIFZERO(1) from employee_tab;
  ```

- 该函数将为表的每一行都返回 NULL（空值）：
  ```
  SELECT NULLIFZERO(0) from employee_tab;
  ```

6.86 NVL 函数

该函数确定一个选定列是否具有空值，如果是空值，它将返回新操作数的值；如果不是空值，它将返回原操作数的值。

NVL *(operand, new-operand)*

operand
指定一个值表达式。

new-operand
指定一个值表达式。*operand* 和 *new-operand* 必须是可比的数据类型。
如果 *operand* 为空，NVL 函数将返回 *new-operand*。
如果 *operand* 不为空，NVL 函数将返回 *operand*。
operand 和 *new-operand* 是一个列名、子查询、EsgynDB SQL 字符串函数、数学函数或常数值。

为兼容 Oracle，如果两个参数的数据类型不同，EsgynDB 不会报错。

6.86.1 NVL 函数例子

- 该函数返回一个数值 z:
```
select nvl(cast(null as char(1)), 'z') from (values(1))
x(a);

(EXPR)
------

"z"
--- 1 row(s) selected.
```

- 该函数返回一个数值 1:
```
select nvl(1, 2) from (values(0)) x(a)
(EXPR)
-------
        1

--- 1 row(s) selected.
```

- 对于列 a1 里的空值，该函数将返回一个数值 9999999：

```
select nvl(a1, 9999999) from t1;
(EXPR)
-------
   123
   34
9999999
--- 3 row(s) selected.

select * from t1;
A1
-------
   123
   34
   ?

--- 3 row(s) selected.
```

- 如果两个参数的数据类型不同，不会报错。

```
SQL>select nvl('abc',123) from dual;

(EXPR)
------
abc

--- 1 row(s) selected.

SQL>select nvl(123, '456') from dual;

(EXPR)
------
   123

--- 1 row(s) selected
```

6.87 NVL2 函数

NVL2 函数允许您根据指定表达式是否为空来确定查询返回的值。如果 operand 不为空，则 NVL2 函数返回 expr2。如果 operand 为空，则 NVL2 函数返回 expr3。

```
NVL2(operand, expr2, expr3)
```

operand
指定一个值表达式。

expr2
指定一个值表达式。operand 和 expr2 必须是可比的数据类型。

expr3
指定一个值表达式。operand 和 expr3 必须是可比的数据类型。

6.87.1 NVL2 函数例子

- 该函数返回‘b’

```
>>select nvl2('a','b','c') from dual;
(EXPR)
------
b
--- 1 row(s) selected
```

- 该函数返回‘c’

```
>>select nvl2(null,'b','c') from dual;
(EXPR)
------
c
--- 1 row(s) selected.
```

6.88 OCTET_LENGTH 函数

OCTET_LENGTH 函数返回以字节为单位的字符串长度,。

```
OCTET_LENGTH (string-value-expression)
```

string-value-expression
指定一个需要返回字节长度的字符串值表达式。EsgynDB SQL 返回结果为一个标度为 0、两个字节的有符号整数。如果 *string-value-expression* 为空，则 EsgynDB SQL 将返回零长度。更多信息，请参阅字符值表达式。

6.88.1 OCTET_LENGTH 函数注意事项

6.88.1.1 CHAR 和 VARCHAR 操作数

如果一个列被声明为固定 CHAR，则 EsgynDB SQL 将返回该列的长度作为最大存储字节数。而对于一个 VARCHAR 列，EsgynDB SQL 将返回该列里存储的字符串的长度实际存储字节数。

6.88.1.2 类似于 CHAR_LENGTH 函数

OCTET_LENGTH 函数类似于 CHAR_LENGTH 函数。OCTET_LENGTH 函数返回一个字符串里的字节数（而非字符数），这个区别对于多字节的实现非常重要。更多信息，请参阅 OCTET_LENGTH 函数例子。

6.88.2 OCTET_LENGTH 函数例子

- 如果按每个字符占两个字节的规则存储一个字符串，则该函数返回数值 12。否则，则函数返回 6:
  ```
  OCTET_LENGTH ('Robert')
  ```

6.89 OVERLAY 函数

OVERLAY 函数用 *replacement-string* 替换 *source-string* 的某些字符（或整个 *source-string*）。

OVERLAY函数类似于INSERT函数（更多信息，请参阅INSERT函数），只是参数的顺序不同，*length*是可选的。

OVERLAY是EsgynDB SQL扩展函数。

OVERLAY (*source-string* PLACING *replacement-string* FROM *start* [FOR *length*])

6.89.1 OVERLAY 语法

- *source-string*

是要部分（或全部）覆盖的字符串。

- *replacement-string*

是要覆盖的字符串。

- *start*

是开始覆盖的位置。

注意：*start* 必须是一个大于或等于 1 的整数。

如果*start*大于*source-string*的长度，则只需附加*replacement-string*。

例子

```
SELECT OVERLAY ('writer' PLACING '_' FROM 10) FROM DUAL;
(EXPR)
-------------
writer_
--- 1 row(s) selected.
```

- *length*

指定要从 *source-string* 替换的字符数。

注意：如果未指定 *length*，则默认为 *replacement-string* 的长度。

例子

```
SELECT OVERLAY ('writer' PLACING '_' FROM 2) FROM DUAL;
(EXPR)
-------------
w_iter
--- 1 row(s) selected.
```

```
SELECT OVERLAY ('writer' PLACING '_' FROM 2 FOR 1) FROM
DUAL;
(EXPR)
-------------
w_iter
--- 1 row(s) selected.
```

```
SELECT OVERLAY ('writer' PLACING '_' FROM 2 FOR 2) FROM
DUAL;
(EXPR)
-------------
w_ter
--- 1 row(s) selected.
```

注意：如果*length*为0，则将*replacement-string*插入到*source-string*中，而不删除*source-string*中的任何字符。

例子

```
SELECT OVERLAY ('writer' PLACING '_' FROM 2 FOR 0) FROM
DUAL;
(EXPR)
-------------
w_riter
--- 1 row(s) selected.
```

6.89.2 OVERLAY 例子

- 以下示例演示如何使用 OVERLAY 函数并返回相应的结果。

语句	结果
SELECT OVERLAY ('writer' PLACING '_' FROM 2) FROM DUAL	w_iter
SELECT OVERLAY ('writer' PLACING '_' FROM 4) FROM DUAL	wri_er
SELECT OVERLAY ('writer' PLACING '_' FROM 7) FROM DUAL	writer_
SELECT OVERLAY ('writer' PLACING '_' FROM 10)	writer_

FROM DUAL	
SELECT OVERLAY ('writer' PLACING '_' FROM 2 FOR 0) FROM DUAL	w_riter
SELECT OVERLAY ('writer' PLACING '_' FROM 2 FOR 1) FROM DUAL	w_iter
SELECT OVERLAY ('writer' PLACING '_' FROM 2 FOR 3) FROM DUAL	w_er
SELECT OVERLAY ('writer' PLACING '_' FROM 2 FOR 5) FROM DUAL	w_
SELECT OVERLAY ('writer' PLACING '_' FROM 2 FOR 7) FROM DUAL	w_
SELECT OVERLAY ('writer' PLACING '_' FROM 1 FOR 6) FROM DUAL	_

语句	结果
SELECT OVERLAY ('writer' PLACING '' FROM 2) FROM DUAL	writer
SELECT OVERLAY ('writer' PLACING '' FROM 2 FOR 0) FROM DUAL	writer
SELECT OVERLAY ('writer' PLACING '' FROM 2 FOR 1) FROM DUAL	writer
SELECT OVERLAY ('writer' PLACING '' FROM 2 FOR 3) FROM DUAL	wer
SELECT OVERLAY ('writer' PLACING '' FROM 2 FOR 5) FROM DUAL	w
SELECT OVERLAY ('writer' PLACING '' FROM 2 FOR 7) FROM DUAL	w
SELECT OVERLAY ('writer' PLACING '' FROM 1 FOR 6) FROM DUAL	

语句	结果
SELECT OVERLAY ('' PLACING 'writer' FROM 2) FROM DUAL	writer
SELECT OVERLAY ('' PLACING 'writer' FROM 2 FOR 0) FROM DUAL	writer
SELECT OVERLAY ('' PLACING 'writer' FROM 2 FOR 1) FROM DUAL	writer
SELECT OVERLAY ('' PLACING 'writer' FROM 2 FOR 3) FROM DUAL	writer
SELECT OVERLAY ('' PLACING 'writer' FROM 2 FOR 5) FROM DUAL	writer
SELECT OVERLAY ('' PLACING 'writer' FROM 2 FOR 7) FROM DUAL	writer

- 该例子将单词 *stackoverflow* 替换为单词 *reddit*（5 位，长度为 13 个字符）。

```
SELECT OVERLAY ('www.stackoverflow.com' PLACING 'reddit'
FROM 5 FOR 13) FROM DUAL;
(EXPR)
--------------------------------------------------
www.reddit.com
--- 1 row(s) selected.
```

- 该例子将明斯特市的道路名 *Dunantstraße* 更改为 *Dunantstrasse*。

```
SELECT OVERLAY ('Dunantstraße' PLACING 'ss' FROM 11 FOR
1) FROM DUAL;
(EXPR)
---------------------------
Dunantstrasse
--- 1 row(s) selected.
```

6.90 PI 函数

PI 函数将常量值 Pi 以浮点类型返回。PI 函数是 EsgynDB SQL 扩展函数。

```
PI()
```

6.90.1 PI 函数例子

- 该常数函数返回数值 3.14159260000000000E+000:
  ```
  PI()
  ```

6.91 PIVOT 函数

PIVOT 函数将一个表的行转换为列，并在转换过程中聚合数据。

PIVOT 是 EsgynDB SQL 扩展函数。

```
PIVOT (source-column[, DELIMITER 'delimiter'])
```

6.91.1 PIVOT 语法

- *source-column*

是一个包含要转换的行的列。

- *delimiter*

是一个用于分隔列中的值的分隔符。
注意：如果省略 DELIMITER '*delimiter*'，则默认分隔符为逗号。

例子

vendor 表如下：

```
SELECT * FROM vendor;
VENDOR_ID VENDOR_NAME VENDOR_EMAIL VENDOR_EMAIL_ID
--------- -------------- ---------------------- ---------
------
       111 Tom           tom@hotmail.com        1
       232 Jerry         jerry@hotmail.com      2
       232 Jerry         jerry@gmail.com        3
       367 Aven          aven@hotmail.com       4
       367 Aven          aven@gmail.com         5
       367 Aven          aven@outlook.com       6
--- 6 row(s) selected.

SELECT vendor_id, PIVOT(vendor_email) FROM vendor GROUP
BY vendor_id;
VENDOR_ID (EXPR)
------------------------------------------------------------
-----
      111 tom@hotmail.com
      232 jerry@gmail.com,jerry@hotmail.com
      367
aven@gmail.com,aven@hotmail.com,aven@outlook.com

--- 3 row(s) selected.
```

注意：空格是有效的分隔符。

例子

```
SELECT PIVOT(vendor_email, delimiter ' ') FROM vendor
GROUP BY vendor_id;
(EXPR)
------------------------------------------------------------
--------------
tom@hotmail.com
jerry@gmail.com jerry@hotmail.com
aven@gmail.com aven@hotmail.com aven@outlook.com
--- 3 row(s) selected.
```

注意：*delimiter* 只能是一个字符。如果它包含多个字符，则只能使用第一个字符作为分隔符。

例子

```
SELECT PIVOT(vendor_email, delimiter '|~|') FROM vendor
GROUP BY vendor_id;
(EXPR)
------------------------------------------------------------
-----------------
tom@hotmail.com
jerry@gmail.com|jerry@hotmail.com
aven@gmail.com|aven@hotmail.com|aven@outlook.com
--- 3 row(s) selected.
```

6.91.2 PIVOT 例子

- 该例子根据商店名称将产品水平对齐到逻辑集。

saleorder 表如下：

```
SELECT * FROM salesorder;
SHOP       PRODUCT
---------- --------------------
Shop1      A
Shop1      B
Shop1      C
Shop2      T
Shop2      E
Shop2      C
Shop2      H
Shop2      N
Shop2      I
Shop2      C
```

```
Shop2      A
Shop2      L
Shop3      W
Shop3      R
Shop3      I
Shop3      T
Shop3      E
Shop3      R
--- 18 row(s) selected.

SELECT shop, PIVOT(product, delimiter '-') as product
FROM salesorder GROUP BY shop;
SHOP PRODUCT
---------- ----------------------
Shop1 A-B-C
Shop2 T-E-C-H-N-I-C-A-L
Shop3 W-R-I-T-E-R
--- 3 row(s) selected.
```

6.92 POSITION 函数

该函数在一个字符串里搜索给定的字符串。如果找到了该子串，EsgynDB SQL 将返回该子串在这个字符串中的位置。包含多字节字符在内的每个字符都被视为一个字符。POSITION 函数与 LOCATE 函数作用相同。

POSITION(*substring-expression IN source-expression*)

substring-expression
这是一种 SQL 字符值表达式，指定需要在 *source-expression* 里搜索的子串。*substring-expression* 不得为 NULL 值。更多信息，请参阅字符值表达式。

source-expression
这是一种 SQL 字符值表达式，指定源字符串。*source-expression* 不得为 NULL 值。更多信息，请参阅字符值表达式。
EsgynDB SQL 返回结果为一个标度为 0 的两个字节的有符号整数。如果没有找到 *substring-expression*，则 EsgynDB SQL 将返回 0。

6.92.1 POSITION 函数注意事项

6.92.1.1 POSITION 函数结果

如果 *source-expression* 的长度为零、而 *substring-expression* 的长度大于零，则 EsgynDB SQL 将返回 0。

如果 *substring-expression* 的长度为零，则 EsgynDB SQL 将返回 1。

如果 *substring-expression* 的长度大于 *source-expression* 的长度，EsgynDB SQL 将返回 0。

如果 *source-expression* 是一个空值，EsgynDB SQL 将返回空值。

6.92.1.2 使用 UPSHIFT 函数

如果想要在搜索时忽略大小写问题，请为 *substring-expression* 和 source-*expression* 使用 UPSHIFT 函数。

6.92.2 POSITION 函数例子

- 该函数返回数值 8 作为子串‘John’在字符串中的位置:
  ```
  POSITION (‘John' IN 'Robert John Smith’)
  ```

- 假设 EMPLOYEE 表带有一个 EMPNAME 列，它同时包含（雇员的）姓和名。POSITION 函数可以返回 EMPLOYEE 表里包含子串'Smith'的所有记录，忽略大小写:
  ```
  SELECT * FROM persnl.employee
  WHERE POSITION('SMITH' IN UPSHIFT(empname)) > 0 ;
  ```

6.93 POWER 函数

该函数返回 NUMERIC 型数值表达式的整数值表达式次方后得到的数值，也可以使用指数运算符**。

POWER 函数是 EsgynDB SQL 扩展函数。

```
POWER (numeric-expression-1,numeric-expression-2)
```

numeric-expression-1, numeric-expression-2

它们都是 SQL NUMERIC 型数值表达式，指定 POWER 函数的底数和指数参数值。

如果底数 *numeric-expression-1* 为 0，则指数 *numeric-expression-2* 必须大于 0。

如果该指数为 0、而底数不得为 0、那么结果为 1。如果底数为负数，则指数必须是一个标度为 0 的精确 NUMERIC 型数值。

6.93.1 POWER 函数例子

- 返回数值 15.625:
  ```
  POWER (2.5,3)
  ```

- 返回数值 27。POWER 函数取二次方后可以得到 SQRT 函数的反函数:
  ```
  POWER (SQRT(27),2)
  ```

6.94 QUARTER 函数

该函数将一个DATE或TIMESTAMP表达式转换为一个1到4范围内的INTEGER值，它表示一年中的相应季度序号。第一季度指的是 1 月 1 日至 3 月 31 日，依此类推。

QUARTER 函数是 EsgynDB SQL 扩展函数。

QUARTER *(datetime-expression)*

datetime-expression

该表达式表示一个 DATE 或 TIMESTAMP 类型的日期时间值。更多信息，请参阅日期时间值表达式。

6.94.1 QUARTER 函数例子

- 返回一个整数，它表示 PROJECT 表的 START_DATE 列里日期时间对应的一年中的季度序号：

```
SELECT start_date, ship_timestamp, QUARTER(start_date)
FROM persnl.project
WHERE projcode = 1000;

Start/Date Time/Shipped                (EXPR)
---------- -------------------------- ------
2008-04-10 2008-04-21 08:15:00.000000      2
```

6.95 RADIANS 函数

该函数将 NUMERIC 型数值表达式（以度数表示）转换为弧度数。

RADIANS 函数是 EsgynDB SQL 扩展函数。

```
RADIANS(numeric-expression)
```

numeric-expression

这是 SQL NUMERIC 型数值表达式，指定 RADIANS 函数的参数值。

6.95.1 RADIANS 函数例子

- 返回数值 7.85398150000000000E-001 或近似值 0.78540 度：
  ```
  RADIANS (45)
  ```
- 返回数值 45 度。DEGREES 函数是 RADIANS 函数的反函数。
  ```
  DEGREES (RADIANS (45))
  ```

6.96 RAND 函数

RAND 函数使用整数作为可选的种子值，生成均匀分布在 0 到 2^{32}（=4294967296）范围内的伪随机数。

注意：RAND 函数并不总是以预测的方式运行。它是不确定的，可能会导致意想不到的副作用。它只计算一次或每行，具体取决于优化器。因此它只能在极少数地方使用，请不要在谓词中使用它。

```
RAND ([numeric-expression])
```

6.96.1 RAND 函数语法

- *numeric-expression*

是一个指定 RAND 函数参数值的 SQL 数字值表达式。更多信息，请参阅数字值表达式。

6.96.2 RAND 函数注意事项

- **使用一个种子值进行调用时，种子值应是一个大于或等于 0 的整数，且不能为 NULL。**

例子

```
SQL>SELECT RAND(1-100) FROM DUAL;
*** ERROR[8432] A negative value cannot be converted to
an unsigned numeric
datatype. Source Type:SMALLINT SIGNED(MBIN16S) Source
Value:-99 Target
Type:LARGEINT(IBIN64S) Max Target Value:0.
Instruction:RANGE_LOW_S16S64
Operation:RANGE_LOW. [2018-04-18 04:44:37]

SQL>SELECT RAND(NULL) FROM DUAL;
*** ERROR[4097] A NULL operand is not allowed in function
(RANDOMNUM(NULL)). [2018-
```

```
04-18 04:44:48]
```

- RAND 函数将使用该值对随机数生成器进行种子设定。
- 每次为生成器设定种子时，RAND 函数都会生成相同的数字序列，并为程序的每次运行使用相同的参数值。
- 在没有种子值的情况下进行调用时，RAND 函数将自动以系统时钟值作为种子。
- 即使在查询中连续调用 RAND 函数，它也会生成不同的数字序列。

例子

```
SQL>SELECT RAND(), RAND(), RAND() FROM DUAL;
(EXPR) (EXPR) (EXPR)
---------- ---------- ----------
1958129868 1958398780 1958482815
--- 1 row(s) selected.
```

- 要在指定范围内获取所需的数字，可以使用内置函数、运算符和子句的组合。

例子

```
SQL>SELECT POWER(RAND(1)*10,3) FROM DUAL;
(EXPR)
------------------------
4.747561509943E15
--- 1 row(s) selected.
```

6.96.3 RAND 函数例子

- 该例子演示了如何将 RAND 函数与内置函数(ABS 函数)一起使用。

```
SQL>SELECT RAND(ABS(1-100)) FROM DUAL;
(EXPR)
----------
1663893
--- 1 row(s) selected.
```

- 该例子显示了 RAND 函数与内置函数(MOD 函数) 一起使用，以生成从 1 到 5 的数字，并获取按 order by 子句排序的随机化结果。对于每个数字，c1 的

100,000 行几乎是均匀分布的。

```
SQL>SELECT COUNT(*) FROM t1;
(EXPR)
--------------------
100000
--- 1 row(s) selected.

SQL>SHOWDDL t1;

CREATE TABLE TRAFODION.SEABASE.T1
(
     IDX                              INT DEFAULT NULL
NOT SERIALIZED
)
ATTRIBUTES ALIGNED FORMAT
;
--- SQL operation complete.

SQL>CREATE TABLE t2(idx int,c1 int);
--- SQL operation complete.

SQL>UPSERT USING LOAD INTO t2 SELECT idx,
MOD(RAND(),5)+1 FROM t1;
--- 100000 row(s) inserted.

SQL>SELECT c1, COUNT(*) FROM t2 GROUP BY c1 ORDER BY
c1;
C1 (EXPR)
----------- --------------------
1 19837
2 19951
3 20023
4 19913
```

```
5 20276
--- 5 row(s) selected.
```

此时，RAND 函数还可以在 ORDER BY 子句中与 LIMIT 子句一起用于输出行。

```
SQL>SELECT c1, COUNT(*) FROM t2 GROUP BY c1 ORDER BY c1
LIMIT 3;
C1 (EXPR)
----------- --------------------
1 19837
2 19951
3 20023
--- 3 row(s) selected.
```

6.97 REGEXP_COUNT 函数

REGEXP_COUNT 返回某种模式在源字符串中出现的次数。

```
REGEXP_COUNT (instr, pattern [, position])
```

instr
输入字符串，字符类型（如果启用了 mode_compatible_1，则可以是数字类型）。

pattern
pattern 是正则表达式，字符类型。

position
position 是一个正整数，表示 EsgynDB 应该开始搜索的输入字符串的字符。默认值为 1，表示 EsgynDB 在 instr 的第一个字符处开始搜索。 正整数类型（可以为 null）。

6.97.1REGEXP_COUNT 函数例子

- 返回从第一个位子开始查找到的’23’出现的次数。
  ```
  >>select regexp_count('123123123','23',1) from dual;

  (EXPR)
  ----------

        3

  --- 1 row(s) selected.
  ```
- 大小写敏感匹配列子。
  ```
  >>select regexp_count('Abcabcabc','ab',1) from dual;

  (EXPR)
  ----------

        2

  --- 1 row(s) selected.
  ```

6.98 REGEXP_REPLACE 函数

REGEXP_REPLACE 数返回一个字符串。

REGEXP_REPLACE 函数通过在字符串中搜索正则表达式模式，从而将符合条件的字符串替换掉。

```
REGEXP_REPLACE(instr,pattern,replace_str)
```

instr
输入字符串，字符类型（如果启用了 mode_compatible_1，则可以是数字类型）。

pattern
Pattern 是正则表达式，字符类型。

replace_str
replace_str 是要替换成的字符串。

6.98.1REGEXP_REPLACE 函数例子

- 将字符串中的':'替换成'.'。

```
>>select regexp_replace('foo:bar', ':', '.') from dual ;

(EXPR)
--------------

foo.bar

--- 1 row(s) selected.
```

- 将字符串中一个或多个空格替换成逗号。

```
>>select regexp_replace('happy    go lucky', '\s+', ',')
from dual ;

(EXPR)
--------------------------------

happy,go,lucky

--- 1 row(s) selected.
```

6.99 REGEXP_SUBSTR 函数

REGEXP_SUBSTR 函数返回一个字符串。
REGEXP_SUBSTR 函数通过在字符串中搜索正则表达式模式来扩展 SUBSTR 函数的功能。

```
REGEXP_SUBSTR(instr,pattern[,position[,occurrence]])
```

instr
输入字符串，字符类型（如果启用了 mode_compatible_1，则可以是数字类型）。

pattern
Pattern 是正则表达式，字符类型。

position
position 是一个正整数，表示 EsgynDB 应该开始搜索的输入字符串的字符。默认值为 1，表示 EsgynDB 在 instr 的第一个字符处开始搜索。 正整数类型（可以为 null）。

occurrence
occurrence 是一个正整数，表示输入字符串中 EsgynDB 应该搜索哪种模式。 默认值为 1，表示 EsgynDB 搜索第一次出现的模式。 正整数类型（可以为 null）。

6.99.1REGEXP_SUBSTR 函数例子

- 返回第一个不是’,’的子字符串。

```
>> select regexp_substr('a,b,c', '[^,]+') from dual;

(EXPR)
---------------------------------

a
--- 1 row(s) selected.
```

- 从给定字符串的第二位开始查找不是’,’的子字符串。

```
>>select regexp_substr('123,456,789,0','[^,]+',2,2)
from dual;

(EXPR)
-------------

456

--- 1 row(s) selected.
```

- 大小写敏感匹配

```
>> select regexp_substr('1A2b3c4d','[a-z]+') from dual;

(EXPR)
--------

b

--- 1 row(s) selected.
```

6.100 REPEAT 函数

该函数返回一个字符串， 该字符串是由字符表达式重复指定次数的评估组成的。REPEAT 函数是 EsgynDB SQL 扩展函数。

```
REPEAT (character-expr, count)
```

character-expr
指定从中返回重复指定次数的字符串的源字符串。源字符串是 SQL 字符值表达式，操作数是 *character-expr* 的求值结果。更多信息，请参阅字符值表达式。

count
指定源字符串 *character-expr* 的重复次数。*count* 数字必须是一个标度为 0、且大于或等于 0 的精确 NUMERIC 数据类型。

6.100.1 REPEAT 函数例子

- 返回 “Act 5, Scene 3, of King Lear” （《李尔王》的第 5 幕、第 3 场）里的引述：

  ```
  REPEAT ('Never,', 5)

  Never,Never,Never,Never,Never,
  ```

6.101 REPLACE 函数

该函数返回一个字符串，其中所有出现在原始字符串里的指定字符串都要被替换为另一个字符串。所有三种字符值表达式都必须为可比类型，返回值为 VARCHAR 类型。

REPLACE 函数是 EsgynDB SQL 扩展函数。

```
REPLACE (char-expr-1, char-expr-2, char-expr-3)
```

char-expr-1, char-expr-2, char-expr-3

这些都是 SQL 字符值表达式，操作数为字符表达式的求值结果。所有出现在 *char-expr-1* 里面的 *char-expr-2* 都将被替换为 *char-expr-3*。更多信息，请参阅字符值表达式。

6.101.1 REPLACE 函数例子

- 使用 REPLACE 函数更改“职位描述”，以便更新其中出现的公司名：

```
SELECT jobdesc FROM persnl.job;

Job Description
------------------
MANAGER COMNET
PRODUCTION COMNET
ASSEMBLER COMNET
SALESREP COMNET
SYSTEM ANAL COMNET
...
--- 10 row(s) selected.

UPDATE persnl.job
SET jobdesc = REPLACE (jobdesc, 'COMNET', 'TDMNET');

Job Description
------------------
MANAGER TDMNET
PRODUCTION TDMNET
ASSEMBLER TDMNET
SALESREP TDMNET
SYSTEM ANAL TDMNET
...

--- 10 row(s) selected.
```

6.102 REVERSE 函数

REVERSE 函数以相反的顺序返回 *format-string*。

REVERSE 是 EsgynDB SQL 的扩展函数。

```
REVERSE (format-string)
```

6.102.1 REVERSE 语法

format-string
是要颠倒顺序的字符串。

注意: format-string 不能为空值。

例子

```
SQL>SELECT REVERSE(NULL) FROM DUAL;
*** ERROR[4097] A NULL operand is not allowed in function
(REVERSE(NULL)). [2019-03-
12 03:23:40]
```

注意: *format-string* 中的尾随空格会成为结果中的前导空格。

例子

```
SELECT REVERSE ('TechnicalWriter ') FROM DUAL;
(EXPR)
--------------------
retirWlacinhceT
--- 1 row(s) selected.
```

6.102.2 REVERSE 例子

- 该例子颠倒了字符串'2019-11-02'。

  ```
  SELECT REVERSE ('2019-11-02') from dual;
  (EXPR)
  ----------
  20-11-9102
  --- 1 row(s) selected.
  ```

- 该例子颠倒了 product 表里的产品名称。

  ```
  SELECT REVERSE (prod_name) FROM product;
  (EXPR)
  --------------------------------
  torraC
  hsiF
  ```

```
feeB
pmirhS
   --- 4 row(s) selected.
```

- 这个例子显示了 REVERSE 函数与 LEFT 函数一起使用。

```
SELECT REVERSE(LEFT(REVERSE('TechnicalWriter'), 6))
FROM DUAL;
(EXPR)
------
Writer
--- 1 row(s) selected.
```

6.103 RIGHT 函数

该函数返回一个字符表达式最右侧指定数量的字符。包含多字节字符在内的每个字符都被视为一个字符。

RIGHT 函数是 EsgynDB SQL 扩展函数。

```
RIGHT (character-expr, count)
```

character-expr

指定一个需要返回最右侧指定数量字符的源字符串。这个源字符串是 SQL 字符值表达式。其操作数是 *character-expr* 的求值结果。更多信息，请参阅字符值表达式。

count

指定从 *character-expr* 中返回的字符数。该计数必须是一个标度为 0 的精确 NUMERIC 型数值。

6.103.1 RIGHT 函数例子

- 返回 'Smith':
  ```
  RIGHT ('Robert John Smith', 5)
  ```

- 假设一个 6 字符的公司字面量被串联起来作为 JOB 表的职位描述字段里的前 6 个字符。使用 RIGHT 函数可以从职位描述字段里删除这个公司字面量:
  ```
  UPDATE persnl.job
  SET jobdesc = RIGHT (jobdesc, 12);
  ```

6.104 ROLLUP 函数

ROLLUP 函数通过逗号分隔的列列表计算从右到左聚合的多个级别的小计，并提供一个总计。

ROLLUP 是 GROUP BY 子句的扩展。GROUPING 函数等相关功能）可与 ORDER BY 一起使用，以控制汇总结果的位置。

```
SELECT…GROUP BY ROLLUP (column 1, [column 2,]…[column n])
```

ROLLUP 生成 n+1 个级别的小计，包括一个总计，其中 n 是所选列的数目。

例如，一个包含三个汇总列的查询会返回以下行：

- 第一级：不使用 ROLLUP，由 GROUP BY 子句计算的通常聚合值。
- 第二级：针对第 1 列和第 2 列的每个组合，在第 3 列中汇总小计。
- 第三级：针对每个第 1 列，在第 2 列和第 3 列汇总小计。
- 第四级：总计行。

注意：EsgynDB 不支持 CUBE 函数，其功能与 ROLLUP 略有不同。

6.104.1 ROLLUP 注意事项

6.104.1.1结果集为 NULL

- 在表示小计或总计的超级聚合行中，较低级别的分组列将被替换为空值。

- 所选列中的空值被视为相等，并在结果集中被划分为一个空值组。

6.104.1.2使用颠倒列顺序进行 ROLLUP

ROLLUP 在每个步骤中删除最右侧的列，因此结果集随逗号分隔列表中指定列的顺序而变化。

如果列顺序为国家、州、城市和名称，ROLLUP 将返回以下分组。	如果列顺序为名称、城市、州和国家，ROLLUP 将返回以下分组。
country, state, city 和 name	name, city, state 和 country
country, state 和 city	name, city 和 state
country 和 state	name 和 city
country	name
总计	总计

6.104.2 ROLLUP 例子

6.104.2.1按一个或多个汇总列分组例子

假设我们有一个这样的 *sales1* 表：

```
SELECT * FROM sales1;
DELIVERY_YEAR REGION PRODUCT REVENUE
------------- ------ ------------------------------- ---
--------
2016 A Dress 100
2016 A Dress 200
2016 A Pullover 300
2016 B Dress 400
2017 A Pullover 500
2017 B Dress 600
2017 B Pullover 700
2017 B Pullover 800
--- 8 row(s) selected.
```

- 这是按一个汇总列分组的例子。

```
SELECT delivery_year, SUM (revenue) AS total_revenue
FROM sales1
GROUP BY ROLLUP (delivery_year);

DELIVERY_YEAR TOTAL_REVENUE
------------- --------------------
2016 1000
2017 2600
NULL 3600
--- 3 row(s) selected.
```

- 这是按两个汇总列分组的例子。
  ROLLUP 首先在最低级别（区域）进行汇总，然后将这些聚合汇总到下一级别（*delivery_year*），最后在这两个级别上生成总计。

```
SELECT delivery_year, region, SUM (revenue) AS
total_revenue
FROM sales1
GROUP BY ROLLUP (delivery_year, region);

DELIVERY_YEAR REGION TOTAL_REVENUE
------------- ------ --------------------
2016 A 600
2016 B 400
2016 NULL 1000
2017 A 500
2017 B 2100
2017 NULL 2600
NULL NULL 3600
--- 7 row(s) selected.
```

- 这是按三个汇总列分组的例子。

```
SELECT delivery_year, region, product, SUM (revenue) AS
total_revenue
FROM sales1
GROUP BY ROLLUP (delivery_year, region, product);
```

```
DELIVERY_YEAR REGION PRODUCT                          TOTAL_REVENUE
------------- ------ -------------------------------- --------------------
         2016 A      Dress                                             300
         2016 A      Pullover                                          300
         2016 A      NULL                                              600
         2016 B      Dress                                             400
         2016 B      NULL                                              400
         2016 NULL   NULL                                             1000
         2017 A      Pullover                                          500
         2017 A      NULL                                              500
         2017 B      Dress                                             600
         2017 B      Pullover                                         1500
         2017 B      NULL                                             2100
         2017 NULL   NULL                                             2600
         NULL NULL   NULL                                             3600

--- 13 row(s) selected.
```

- 第一级：**蓝色**标记的行是每年（2016 年和 2017 年）每个地区（A 和 B）每个产品（*Dress* 和 *Pullover*）的总收入，它们是按 GROUP BY 而不是 ROLLUP 计算的。
- 第二级：**红色**标记的行按 *product* 提供给定 *delivery_year* 和 *region* 的总收入。
  这些行的 *product* 列被设置为空值。
- 第三级：**黄色**标记的行表示每年（2016 年和 2017 年）的总收入。
  这些行的 *region* 和 *product* 列被设置为空值。
- 第四级：**紫色**标记的行合计 *delivery_year*, *region* 和 *product* 列的所有行。

此行的 *delivery_year*、*region* 和 *product* 列被设置为空值。

6.104.2.2NULL 例子

下面的例子演示了 ROLLUP 如何处理所选列中的空值并为超级聚合行生成空值。假设我们有一个这样的 *sales2* 表：

```
SELECT * FROM sales2;
DELIVERY_YEAR REGION PRODUCT REVENUE
------------- ------ ------------------------------- ---
--------
NULL A Dress 100
NULL A Dress 200
2016 A Pullover 300
```

```
2016 B Dress 400
2017 A Pullover 500
2017 B Dress 600
NULL B Pullover 700
NULL B Pullover 800
--- 8 row(s) selected.

SELECT delivery_year, region, product, SUM (revenue) AS
total_revenue
FROM sales2
GROUP BY ROLLUP (delivery_year, region, product);



DELIVERY_YEAR REGION PRODUCT TOTAL_REVENUE
------------- ------ -------------------------------- ---
-----------------
2016 A Pullover 300
2016 A NULL 300
2016 B Dress 400
2016 B NULL 400
2016 NULL NULL 700
2017 A Pullover 500
2017 A NULL 500
2017 B Dress 600
2017 B NULL 600
2017 NULL NULL 1100
NULL A Dress 300
NULL A NULL 300
NULL B Pullover 1500
NULL B NULL 1500
NULL NULL NULL 1800
NULL NULL NULL 3600
--- 16 row(s) selected.
```

6.104.2.3使用颠倒列顺序进行 ROLLUP 例子

假设我们有相同的 sales1 表，如“按一个或多个汇总列分组例子”中所示。

- 以下例子的列顺序是 *delivery_year,region* 和 *product*。

  ```
  SELECT delivery_year, region, product, SUM (revenue) AS
  total_revenue
  FROM sales1
  GROUP BY ROLLUP (delivery_year, region, product);

  DELIVERY_YEAR REGION PRODUCT TOTAL_REVENUE
  ------------- ------ -------------------------------- -
  -------------------
  2016 A Dress 300
  2016 A Pullover 300
  ```

```
2016 A NULL 600
2016 B Dress 400
2016 B NULL 400
2016 NULL NULL 1000
2017 A Pullover 500
2017 A NULL 500
2017 B Dress 600
2017 B Pullover 1500
2017 B NULL 2100
2017 NULL NULL 2600
NULL NULL NULL 3600
--- 13 row(s) selected.
```

- *以下例子的列顺序是 product*,、*region* 和 *delivery_year*，输出与上面的结果集不同。

```
SELECT product, region, delivery_year, SUM (revenue) AS
total_revenue
FROM sales1
GROUP BY ROLLUP (product, region, delivery_year);

PRODUCT REGION DELIVERY_YEAR TOTAL_REVENUE
-------------------------------- ------ ------------- -
-------------------
Dress A 2016 300
Dress A NULL 300
Dress B 2016 400
Dress B 2017 600
Dress B NULL 1000
Dress NULL NULL 1300
Pullover A 2016 300
Pullover A 2017 500
Pullover A NULL 800
Pullover B 2017 1500
Pullover B NULL 1500
Pullover NULL NULL 2300
NULL NULL NULL 3600
--- 13 row(s) selected.
```

6.104.2.4使用 ORDER BY 进行 ROLLUP 例子

假设我们有相同的 *sale1* 表，如“按一个或多个汇总列分组例子”中所示。

该例子使用 ROLLUP 和 ORDER BY 子句对结果进行排序。

```
SELECT delivery_year, product, SUM (revenue) AS
total_revenue
FROM sales1
GROUP BY ROLLUP (delivery_year, product)
ORDER BY total_revenue;
```

```
DELIVERY_YEAR PRODUCT TOTAL_REVENUE
------------- ------------------------------- ----------
----------
2016 Pullover 300
2017 Dress 600
2016 Dress 700
2016 NULL 1000
2017 Pullover 2000
2017 NULL 2600
NULL NULL 3600
--- 7 row(s) selected.
```

6.105 ROUND 函数

ROUND 函数返回 numeric_expr 的数值，它被四舍五入到小数点右侧的 num 位。ROUND 函数是 EsgynDB SQL 扩展函数。

ROUND *(numeric_expr [, num])*

numeric_expr
这是 SQL NUMERIC 型数值表达式。

num
指定小数点右侧的位数，然后进行四舍五入运算。如果 *num* 是一个负数，则小数点右侧的所有位数和小数点左侧的 *num* 位数都将被归零。如果 *num* 未指定或为 0，则小数点右侧的所有位数都将被归零。
对于任何精确 NUMERIC 型数值而言，*numeric_expr* 值都将向远离 0 的方向四舍五入（例如：如果 x.5 为正数，则四舍五入到 x+1；如果 x.5 为负数，则四舍五入到 x-1)。对于非精确 NUMERIC 型数值（real、float 和 double）而言，*numeric_expr* 值都将四舍五入到最接近的偶数。

6.105.1 ROUND 函数例子

- 该函数可以返回数值 123.46。
  ROUND(123.4567,2)

- 该函数可以返回数值 123。
  ROUND(123.4567,0)

- 该函数可以返回数值 120。
  ROUND(123.4567,-1)

- 该函数可以返回数值 0。
  ROUND(999.0,-4)

- 该函数可以返回数值 1000。
  ROUND(999.0.-3)

- 该函数可以返回数值 2.0E+000。
  ROUND(1.5E+000,0)

- 该函数可以返回数值 2.0E+00。

```
ROUND(2.5E+000,0)
```

- 该函数可以返回数值 1.0E+00。

```
ROUND(1.4E+000,0)
```

6.106 RPAD 函数

该函数使用指定的字符串填充一个字符串的右侧。字符串里的每个字符（包括多字节字符）都被视为一个字符。

RPAD 函数是 EsgynDB SQL 扩展函数。

```
RPAD (str, len [,padstr])
```

str

是一个表达式。更多信息，请参阅字符值表达式。

len

标识预期返回的字符数。它可以是一种表达式，但其结果必须是一个整数值。如果 len 等于字符串的长度，则无需进行更改。如果 len 小于字符串的长度，则字符串将被截断。

pad-character

既可以是一种表达式，也可以是一个字符串。

6.106.1 RPAD 函数例子

- 该函数返回'kite':
  ```
  RPAD('kite', 7)
  ```
- 该函数返回'ki':
  ```
  RPAD('kite', 2)
  ```
- 该函数返回'kite0000':
  ```
  RPAD('kite', 8, '0')
  ```
- 该函数返回'go fly a kite':
  ```
  RPAD('go fly a kite', 13, 'z')
  ```
- 该函数返回'go fly a kitez':
  ```
  RPAD('go fly a kite', 14, 'z')
  ```
- 该函数返回'kitegoflygoflygof':
  ```
  RPAD('kite', 17, 'gofly' )
  ```

6.107 RTRIM 函数

该函数删除一个字符串里的尾随空格。如果必须删除任何前导字符而非空格，请使用 TRIM 函数并指定字符值。更多信息，请参阅 TRIM 函数。

RTRIM 函数是 EsgynDB SQL 扩展函数。

```
RTRIM (character-expression)
```

character-expression

这是 SQL 字符值表达式，并指定一个需要去除尾随空格的字符串。更多信息，请参阅字符值表达式。

6.107.1 RTRIM 函数注意事项

6.107.1.1RTRIM 函数结果

该函数的结果始终为 VARCHAR 类型，其最大长度等于 *character-expression*（字符表达式）的固定长度或最大可变长度。

6.107.2 RTRIM 函数例子

- 返回 'Robert'：
  ```
  RTRIM (' Robert ')
  ```

更多信息，请参阅 TRIM 函数和 LTRIM 函数。

6.108 SECOND 函数

SECOND 函数将 TIME 或 TIMESTAMP 表达式转换为一个 0 到 59 范围内的 INTEGER 值，它表示一小时内的相应秒数。SECOND 函数是 EsgynDB SQL 扩展函数。

```
SECOND (datetime-expression)
```

datetime-expression

该表达式的求值结果为一个 TIME 或 TIMESTAMP 类型的日期时间值。更多信息，请参阅日期时间值表达式。

6.108.1 SECOND 函数例子

- 返回一个 NUMERIC 值，它表示 SHIP_TIMESTAMP 列里的一小时内的秒数：

```
SELECT start_date, ship_timestamp,
SECOND(ship_timestamp)
FROM persnl.project
WHERE projcode = 1000;

Start/Date Time/Shipped (EXPR)
---------- -------------------------- -----------
2008-04-10 2008-04-21 08:15:00.000000 .000000
```

6.109 SHA 函数

计算字符串的 SHA-1 160 位校验和，如 RFC3174（安全散列算法）中所述。该值以一个 40 位十六进制数字的字符串返回，如果参数为 NULL，则返回 NULL。

6.109.1 SHA 例子

```
>>SELECT SHA1('abc') from dual;
(EXPR)
-----------------------------------------
'a9993e364706816aba3e25717850c26c9cd0d89d'
```

6.110 SHA2 函数

计算 SHA-2 系列的散列函数（SHA-224、SHA-256、SHA-384 和 SHA-512）。第一个参数是要散列的明文字符串。第二个参数指示结果所需位长度，其值必须为 224、256、384、512。如果任一参数为空或散列长度不是允许值之一，则返回值为空。否则，函数结果是包含所需位数的散列值。请参阅本节开头有关有效存储散列值的说明。

6.110.1 SHA2 例子

```
>>SELECT SHA2('abc', 224) from dual;
(EXPR)
------------------------------------------------------------
'23097d223405d8228642a477bda255b32aadbce4bda0b3f7e36c9da7
'
```

6.111 SIGN 函数

SIGN 函数返回 NUMERIC 型数值表达式的符号指示符。如果数值小于 0，则该函数将返回-1 作为指示符； 如果数值为 0，则该函数将返回 0；如果数值大于 0，则该函数将返回 1。SIGN 函数是 EsgynDB SQL 扩展函数。

```
SIGN(numeric-expression)
```

numeric-expression
这是一个 SQL NUMERIC 型数值表达式，它指定 SIGN 函数的参数值。详情参考数字值表达式。

6.111.1 SIGN 函数例子

- 将返回数值 – 1:
  ```
  SIGN (-20 + 12)
  ```
- 将返回数值 0:
  ```
  SIGN (-20 + 20)
  ```
- 将返回数值 1:
  ```
  SIGN (-20 + 22)
  ```

6.112 SIN 函数

该函数返回 NUMERIC 型数值表达式的正弦，这里的表达式是一个以弧度表示的角度。

SIN 函数是 EsgynDB SQL 扩展函数。

```
SIN(numeric-expression)
```

numeric-expression

这是一个 SQL NUMERIC 型数值表达式，它指定 SIN 函数的参数值。详情参考数字值表达式。

6.112.1 SIN 函数例子

- 该函数可以返回数值 3.42052233254419840E-001 或近似值 0.3420，即 0.3491 弧度(20 度）的正弦：
  ```
  SIN(0.3491)
  ```

6.113 SINH 函数

该函数返回 NUMERIC 型数值表达式的双曲正弦，这里的表达式是一个以弧度表示的角度。

SINH 函数是 EsgynDB SQL 扩展函数。

```
SINH(numeric-expression)
```

numeric-expression

这是一个 SQL NUMERIC 型数值表达式，它指定 SINH 函数的参数值。详情参考数字值表达式。

6.113.1 SINH 函数例子

- 该函数返回数值 1.60191908030082560E+000 或近似值 1.6019，也就是 1.25 弧度的双曲正弦：
  ```
  SINH (1.25)
  ```

6.114 SLEEP 函数

在持续时间参数给定的秒数内休眠（暂停），然后返回 1。

```
SLEEP( seconds )
```

6.114.1 SLEEP 例子

- 在给定的秒数内休眠（暂停）

```
SELECT SLEEP(5) AS ID FROM dual;
(EXPR)
-----------
          1
--- 1 row(s) selected.
```

6.115 SPACE 函数

该函数返回一个字符串，它包含指定数量的空格，其中每个空格（的 ASCII 码）都是 0x20 或 0x0020，具体根据所选的字符集而定。

SPACE 函数是 EsgynDB SQL 扩展函数。

```
SPACE (length [,char-set-name])
```

length

指定将返回的字符数。计数必须是一个大于或等于 0、并且标度为 0 的精确 NUMERIC 型数值。对于 ISO8859-1 或 UTF8 字符集而言，*length* 则不得超过 32768。

char-set-name

是 ISO88591 或 UTF8。如果没有指定第二个参数，则默认值将是默认字符集。返回的字符串将具有 *char-set-name* 指定的字符集所关联的 VARCHAR 数据类型。

6.115.1 SPACE 函数例子

- 返回三个空格:
  ```
  SPACE (3)
  ```

6.116 SPLIT_PART 函数

SPLIT_PART 函数返回在定义的 *delimiter* 和 *field-position* 上拆分的第 n 个子字符串。
SPLIT_PART 是 EsgynDB SQL 扩展函数。

```
SPLIT_PART (source-string, delimiter, field-position)
```

6.116.1 SPLIT_PART 语法

- *source-string*

是一个要拆分的字符串。

注意：*source-string* 必须是字符，不能为空。

- *delimiter*

是一个用于拆分 *source-string* 的分隔符。

delimiter 可以包含多个字符，而不仅仅是单个字符。多个字符被视为一个整体，而不是单个的字符。例如，如果 *delimiter* 是“coder”，则 *source-string* 围绕“coder”进行拆分，而不是围绕“c”、“o”、“d”、“e”和“r”。

注意：*delimiter* 不能为空。

如果 *delimiter* 在 *source-string* 中出现多次，则函数从第一个 *delimiter* 开始计数。

例子

```
SQL>SELECT SPLIT_PART ('technical writer','t',2) FROM
DUAL;
(EXPR)
------------------------
echnical wri
--- 1 row(s) selected.
```

注意：如果 *source-string* 中不存在 *delimiter*，则函数返回一个空字符串。

例子

```
SQL>SELECT SPLIT_PART ('technical_writer','@_@',1) FROM
DUAL;
(EXPR)
----------------------
--- 1 row(s) selected.
```

注意：如果 *source-string* 以 *delimiter* 开头或结尾，则 *delimiter* 前后的空格会作为有效部分返回。

例子

```
SQL>SELECT SPLIT_PART ('|t|e|c|h|n|i|c|a|l', '|', 1),
SPLIT_PART ('|w|r|i|t|e|r|',
'|', 8) FROM DUAL;
(EXPR) (EXPR)
------------------ -------------
--- 1 row(s) selected.
```

• *field-position*

是要返回的 *source-string* 的第 n 个部分。

注意：*field-position* 必须是一个大于零的整数。

field-position 不能为空。

如果 *field-position* 超过拆分后的 *sourcestring* 部分的数量，该函数返回一个空字符串。

例子

```
SQL>SELECT SPLIT_PART ('technical writer',' ',3) FROM
DUAL;
(EXPR)
----------------
--- 1 row(s) selected.
```

6.116.2 SPLIT_PART 例子

- 该例子返回源字符串中所需部分。

```
SQL>SELECT SPLIT_PART ('a*^¬^*technical writer',
'*^¬^*', 2), SPLIT_PART ('is|a
professional', '$', 1), SPLIT_PART ('&-information&-
communicator', '&-', 4) FROM
DUAL;
(EXPR) (EXPR) (EXPR)
---------------------------------------------- -----------
-------
---------------------------
technical writer
--- 1 row(s) selected.
```

- 该例子将 *deliv_date* 列拆分为年、月和日。

```
SQL>SELECT orders.deliv_date,
SPLIT_PART (orders.deliv_date, '-', 1) AS
```

```
delivery_year,
SPLIT_PART (orders.deliv_date, '-', 2) AS
delivery_month,
SPLIT_PART (orders.deliv_date, '-', 3) AS delivery_day
FROM orders LIMIT 3;
DELIV_DATE DELIVERY_YEAR DELIVERY_MONTH DELIVERY_DAY
-------------------------------------------------------
-------------------------
2019-11-02 2019 11 02
2019-12-24 2019 12 24
2019-12-31 2019 12 31
--- 3 row(s) selected.
```

- SPLIT_PART 函数可以与其他函数一起使用。
  - 与 CHAR_LENGTH 和 TRIM 函数一起使用。

```
SQL>SELECT CHAR_LENGTH(TRIM(LEADING '"' FROM
(SPLIT_PART(' "Good technical writing
is concise, focused, easy to understand," ,free of
errors, and is audiencebased',',',
4)))) FROM DUAL;
(EXPR)
----------
1
--- 1 row(s) selected.

SQL>SELECT CHAR_LENGTH(trim(LEADING '"' FROM trim
(split_part(' "Good technical
writing is concise, focused, easy to understand," ,free
of errors, and is
audience-based',',',4)))) FROM DUAL;
(EXPR)
----------
0
--- 1 row(s) selected.
```

  - 与 ROW_NUMBER()函数一起使用。

    假设我们有以下两张表:

    t1:

```
SQL>SELECT * FROM t1;
C1 C2 C3
--- --- ----------------------------------------
----------------------
1 a technical~*~achievement
2 a ~*~writers~*~work across genres
3 a diagram~*~does not~*~convey~*~entity container
information
4 a ~*~minimally~*~serve as~*~technical~*~advisors
```

```
5 b concepts~*~of encoding
6 b can be~*~accurately~*~aggregated
7 b ~*~database~*~and~*~cloud
8 b supporting~*~learners~*~to
search~*~comprehensively~*~across databases
--- 8 row(s) selected.
```

t2:

```
SQL>SELECT * FROM t2;
C1
-----------
1
2
--- 2 row(s) selected.
```

- SPLIT_PART 函数与 ROW_NUMBER 函数一起使用。

```
SQL>SELECT * FROM (SELECT SPLIT_PART (t1.c3,
'~*~', ROW_NUMBER() OVER (PARTITION
BY t1.c2 ORDER BY t1.c1)) FROM t1) AS t3 CROSS
JOIN t2;
(EXPR) C1
------------------------- -----------
technical 1
technical 2
writers 1
writers 2
convey 1
convey 2
technical 1
technical 2
concepts 1
concepts 2
accurately 1
accurately 2
and 1
and 2
comprehensively 1
comprehensively 2
--- 16 row(s) selected.
```

6.117 SQRT 函数

该函数返回一型数值表达式的平方根。

SQRT 函数是 EsgynDB SQL 扩展函数。

```
SQRT(numeric-expression)
```

numeric-expression

这是 SQL NUMERIC 型数值表达式，它指定 SQRT 函数的参数值。该参数值不得为负数。详情参考数字值表达式。

6.117.1 SQRT 函数例子

- 该函数返回数值 5.19615242270663232E+000 或近似值 5.196：
  ```
  SQRT (27)
  ```

6.118 STDDEV 函数

该函数是聚合函数，它返回一组数字的标准差。

STDDEV 函数是 EsgynDB SQL 扩展函数。

```
STDDEV ([ALL | DISTINCT] expression [,weight])
```

`ALL | DISTINCT`

指定计算该表达式的 STDDEV 函数是否将重复值包含在内。默认选项为 ALL，这样会使所有重复值都被包含在内。如果指定了 DISTINCT，则须在应用 STDDEV 函数之前消除所有的重复值。如果指定了 DISTINCT，则不得指定权重。

expression

指定一个数值表达式，它能确定那些需要计算标准差的数值。该表达式不得包含聚合函数或子查询。DISTINCT 子句指定 STDDEV 函数对 expression 求值所得到的单列表的不同值操作。

weight

指定一个数值表达式，它能确定那些需要计算标准差的数值的权重。*weight* 不能包含聚合函数或子查询。*weight* 与表达式定义在相同的表上。*expression* 求值所得到的单列表和 *weight* 求值所得到的单列表必须具有相同的基数。

6.118.1 STDDEV 函数注意事项

6.118.1.1STDDEV 函数定义

一个数值表达式的标准差被定义为一个表达式方差的平方根。更多信息，请参阅 VARIANCE 函数。在方差定义中，N-1 是表达式的分母（若没有指定权重），所以，如果一个表或表里的一组的行数为 1，EsgynDB SQL 将返回一个系统定义的默认值 0（没有错误）。

6.118.1.2结果的数据类型

该函数结果的数据类型始终为 DOUBLE PRECISION。

6.118.1.3表达式的操作数

表达式包含 SELECT 结果表的行里的列，但不得包含聚合函数。以下这些表达式是有效的：

```
STDDEV (SALARY) STDDEV (SALARY * 1.1)
STDDEV (PARTCOST * QTY_ORDERED)
```

6.118.1.4空值

消除集合里的所有空值后对 STDDEV 函数进行求值。如果结果表为空，则 STDDEV 函数将返回 NULL 值。

6.118.1.5FLOAT(54) 和 DOUBLE PRECISION 数据

不得将大型 FLOAT(54)或 DOUBLE PRECISION 值用作 STDDEV 函数的参数。如果计算 STDDEV(x)期间 SUM (x* x) 超过了数值 1.15792089237316192e77，则将出现数值字溢出。

6.118.2 STDDEV 函数例子

- 计算当前雇员工资的标准差:

```
SELECT STDDEV(salary) AS StdDev_Salary FROM
persnl.employee;

STDDEV_SALARY
-------------------------
3.57174062500000000E+004

--- 1 row(s) selected.
```

- 计算当前库存中的部件成本的标准差:

```
SELECT STDDEV (price * qty_available) FROM sales.parts;

(EXPR)
-------------------------
7.13899499999999808E+006

--- 1 row(s) selected.
```

6.119 STRINGTOLOB 函数

6.119.1 STRINGTOLOB 函数语法

STRINGTOLOB 函数将一个简单字符串字面值转换为 LOB 格式。该函数可以在 INSERT 或 UPDATE 语句中使用。
STRINGTOLOB 函数是 EsgynDB SQL 扩展函数。
更多信息，请参阅 EsgynDB SQL Large Objects 指南。

```
STRINGTOLOB('string literal expression')
```

- STRINGTOLOB
  将一个简单字符串字面值转换为 LOB 格式。

- string literal expression （字符串文本表达式）
  是用单引号括起来的一系列字符。

6.119.2 STRINGTOLOB 函数例子

- 该例子在插入前将一个简单字符串字面值转换为 LOB 格式。

```
insert into tlob1 values(1,stringtolob('inserted row'));
```

6.120 SUBSTRING/SUBSTR 函数

SUBSTRING 函数从一个给定的字符表达式中截取子串，返回数据类型类型为 VARCHAR 的字符串，其最大长度为以下两个数值中的较小者：

- 输入字符串的固定长度（适用于 CHAR 型字符串）或最大可变长度（适用于 VARCHAR 型字符串）
- 参数长度值（指定一个常数时）或 32708 （指定一个非常数时）

  SUBSTR 函数等效于 SUBSTRING 函数。

SUBSTRING (*character-expr* FROM *start-position* [FOR *length*])

或

SUBSTRING (*character-expr,start-position[,length*])

character-expr

指定一个需要截取子串的源字符串。源字符串是一种 SQL 字符值表达式，操作数为 *character-expr* 的求值结果。更多信息，请参阅字符值表达式。

start-position

指定 *character-expr* 中的起始位置 *start-position*，随后将从此位置开始截取子串。*start-position* 必须是一个标度为 0 的精确 NUMERIC 型数值。

length

指定从 *character-expr* 中截取的字符数。包括多字节字符在内的每个字符都被计为一个字符。*length* 为截取子串的长度，它必须是一个大于或等于 0、且标度为 0 的精确 NUMERIC 型数值。长度字段是可选的，因此，如果不指定子串长度，则将返回从 *start-position* 开始到字符表达式末尾的所有字符。

6.120.1 替代形式

- SUBSTRING 函数将 SUBSTRING(string FOR int) 和 SUBSTRING (string FROM 1 FOR int)视为等效。EsgynDB 数据库软件支持以下 ANSI 标准形式：

  ```
  SUBSTRING(string FROM int [ FOR int ])
  ```

- SUBSTRING 函数将 SUBSTRING (string, Fromint) 和 SUBSTRING (string FROM Fromint)视为等效。EsgynDB 数据库软件支持 SUBSTRING (string, Fromint, Forint)等效于 ANSI 标准形式：

```
SUBSTRING(string FROM Fromint FOR Forint)
```

6.120.2 SUBSTRING/SUBSTR 函数注意事项

6.120.2.1expression、length 和 Start Position 的要求

- 子串长度和起始位置的数据类型都必须是标度为 0 的数字型，否则将返回一个错误。

- 如果起始位置和子串长度的总和大于字符表达式的长度，则将返回从起始位置到字符串末尾的子串。

- 如果起始位置大于字符表达式的长度，则将返回一个空串（‘‘）。

- 由此产生的子串始终具有 VARCHAR 类型。如果源字符串是一个完成大写转换的 CHAR 或 VARCHAR 字符串，则结果也将是完成大写转换的 VARCHAR 类型。

6.120.3 SUBSTRING/SUBSTR 函数例子

- 截取 'Ro':
  ```
  SUBSTRING(‘Robert John Smith' FROM 0 FOR 3)
  SUBSTR(‘Robert John Smith' FROM 0 FOR 3)
  ```

- 截取 'John':
  ```
  SUBSTRING (‘Robert John Smith' FROM 8 FOR 4)
  SUBSTR (‘Robert John Smith' FROM 8 FOR 4)
  ```

- 截取 'John Smith':
  ```
  SUBSTRING (‘Robert John Smith' FROM 8)
  SUBSTR (‘Robert John Smith' FROM 8)
  ```

- 截取 'Robert John Smith':
  ```
  SUBSTRING (‘Robert John Smith' FROM 1 FOR 17)
  SUBSTR (‘Robert John Smith' FROM 1 FOR 17)
  ```

- 截取 'John Smith':
  ```
  SUBSTRING (‘Robert John Smith' FROM 8 FOR 15)
  SUBSTR (‘Robert John Smith' FROM 8 FOR 15)
  ```

- 截取 'Ro':
  ```
  SUBSTRING (‘Robert John Smith' FROM -2 FOR 5)
  ```

```
SUBSTR ('Robert John Smith' FROM -2 FOR 5)
```

- 截取一个空串":

```
SUBSTRING ('Robert John Smith' FROM 8 FOR 0)
SUBSTR ('Robert John Smith' FROM 8 FOR 0)
```

6.120.1 SUBSTRING/SUBSTR 函数兼容 Oracle

SUBSTR 函数用来从给定的字符串里获取指定范围的子字符串，需要给定子字符串的起始位置和长度。

当给定的起始位置为负数的时候，Oracle 和 EsgynDB 的 SUBSTR 函数行为是不一致的:

Oracle 的 SUBSTR 给定起始位置为负数的时候，Oracle SUBSTR 会从给定字符串的末尾倒推，计算子字符串的起始位置;

EsgynDB 的 SUBSTR 给定起始位置为负数的时候，不会从源字符串的末尾倒推，而是直接从负数位置作为子字符串的起始位置。打开 CQD MODE_COMPAITLBE_1 ‘ON’；则可以在 EsgynDB 数据库中使用兼容 Oracle 行为的 SUBSTR 函数。

例子:

```
>>cqd mode_compatible_1 'off';
--- SQL operation complete.
>>select substr('1234567890', -5, 9) from dual ;
(EXPR)
---------
123
--- 1 row(s) selected.

>>cqd mode_compatible_1 'on';
--- SQL operation complete.
>>select substr('1234567890', -5, 9) from dual ; --oracle
like
(EXPR)
---------
67890
--- 1 row(s) selected.
```

6.121 SUM 函数

该函数是聚合函数，它返回一组数字的总和。

```
SUM ([ALL | DISTINCT] expression)
```

ALL | DISTINCT

指定计算一个表达式的 SUM 函数时是否将重复值都包含在内。默认选项为 ALL，这样会使所有重复值都被包含在内。如果指定 DISTINCT，则需在应用 SUM 函数之前消除所有重复值。

expression

指定一种数值或 INTERVAL 型数值表达式，以确定需要求和的数值。该表达式不得包含聚合函数或子查询。DISTINCT 子句指定 SUM 函数对 *expression* 求值所得到的单列表的不同值操作。将该函数应用到一组数值前，应消除所有空值。如果结果表为空，则 SUM 函数将返回 NULL。更多信息，请参阅表达式。

6.121.1 SUM 函数注意事项

6.121.1.1 结果的数据类型和标度

该函数结果的数据类型将取决于参数的数据类型。如果参数为精确数字型，则结果将为 LARGEINT 型。如果参数为近似数字型，则结果将为 DOUBLE PRECISION 型。如果参数为 INTERVAL 数据类型，则结果将为 INTERVAL 型、且其精度等于参数精度。另外，该函数结果的标度也等于参数标度。如果参数没有标度，则结果将被截断。

6.121.1.2 表达式的操作数

表达式包含 SELECT 结果表的行里的列，但不能包含聚合函数。以下为有效表达式：

```
SUM (SALARY)
SUM (SALARY * 1.1)
SUM (PARTCOST * QTY_ORDERED)
```

6.121.2 SUM 函数例子

- 计算当前库存中的部件的总价值：

  ```
  SELECT SUM (price * qty_available) FROM sales.parts;

  (EXPR)
  ```

```
---------------------
         117683505.96

--- 1 row(s) selected.
```

6.122 SYSDATE 函数

SYSDATE 函数（等效于 CURRENT_DATE 函数）检索服务器的当前日期，而不是会话的当前日期。

返回值为 DATE，默认格式为 YYYY-MM-DD。

例如，如果您于 2018-03-14 06:00:00 (UTC+8) 在位于上海的本地计算机上执行一个查询，而位于柏林的数据库服务器时间为 2018-03-13 23:00:00 (UTC+1)，则 SELECT SYSDATE FROM DUAL;结果为 2018-03-13，而不是 2018-03-14。

```
SYSDATE
```

6.122.1 SYSDATE 例子

- 该例子返回当前日期。

```
SQL>SELECT SYSDATE FROM DUAL;
(EXPR)
----------
2018-03-15
--- 1 row(s) selected.
```

- 该例子返回昨天、今天和明天的日期。

```
SQL>SELECT SYSDATE -1 AS yesterday,
SYSDATE AS today,
SYSDATE +1 AS tomorrow FROM DUAL;
YESTERDAY TODAY TOMORROW
---------- ---------- ----------
2018-03-14 2018-03-15 2018-03-16
--- 1 row(s) selected.
```

- 以下例子显示 SYSDATE 的值可以转换为字符值。

```
SQL>SELECT TO_CHAR (SYSDATE, 'DD-MON-YYYY') FROM DUAL;
(EXPR)
-----------
15-MAR-2018
--- 1 row(s) selected.

SQL>SELECT TO_CHAR (SYSDATE,'HH:MI:SS') FROM DUAL;
*** ERROR[4072] The operand of function TO_CHAR must be
a datetime containing a
time. [2018-03-15 11:49:22]

SQL>SELECT TO_CHAR (SYSDATE, 'MM/DD/YYYY HH24:MI:SS')
FROM DUAL;
(EXPR)
-------------------
```

```
03/15/2018 00:00:00
--- 1 row(s) selected.
```

- 该例子使用 SYSDATE 将天转换为分钟。

```
SQL>SELECT (SYSDATE-(SYSDATE-7))*1440 FROM DUAL;
(EXPR)
-----------------
10080
--- 1 row(s) selected.
```

- 该例子演示了 SYSDATE 函数在 SQL 语句中如何工作。
  假设我们有以下表:

```
SQL>SELECT * FROM orders;
ORDERNUM ORDER_DATE DELIV_DATE SALESREP CUSTNUM
-------- ---------- ---------- -------- -------
100210 2018-03-02 2018-04-10 220 127
100250 2018-01-23 2018-06-16 220 123
101220 2018-02-21 2018-12-15 221 156
200300 2018-02-06 2018-07-15 222 126
200320 2018-03-08 2018-07-20 223 121
200490 2018-02-19 2018-11-01 226 123
300350 2018-03-03 2018-08-10 231 123
300380 2018-01-19 2018-08-15 226 156
400410 2018-01-27 2018-09-14 227 154
500450 2018-03-12 2018-09-16 220 124
600480 2018-02-12 2018-10-14 226 123
700510 2018-02-01 2018-10-16 220 143
800660 2018-01-09 2018-11-01 229 100
--- 13 row(s) selected.
```

SYSDATE 是 2018-03-15。

```
SQL>SELECT SYSDATE FROM DUAL;
(EXPR)
----------
2018-03-15
--- 1 row(s) selected.
```

该语句使用 SYSDATE 函数返回符合条件的行。

```
SQL>SELECT * FROM orders
WHERE DAY(deliv_date) = DAY(sysdate+1)
AND ordernum <>100210
AND salesrep=220
ORDER BY order_date DESC;
```

```
ORDERNUM ORDER_DATE DELIV_DATE SALESREP CUSTNUM
-------- ---------- ---------- -------- -------
500450 2018-03-12 2018-09-16 220 124
700510 2018-02-01 2018-10-16 220 143
100250 2018-01-23 2018-06-16 220 123
--- 3 row(s) selected.
```

6.123 SYSTIMESTAMP 函数

SYSTIMESTAMP 函数（等效于 CURRENT_TIMESTAMP 函数）提供了比 SYSDATE 函数更高的粒度，并检索服务器的当前日期和时间（包括六位数精度的小数秒），而不是会话的当前日期和时间。

返回值为 TIMESTAMP，默认格式为 YYYY-MM-DD HH:MM:SS.FFFFFF。

例如，如果您于 2018-03-14 06:00:00 (UTC+8) 在位于上海的本地计算机上执行一个查询，而位于柏林的数据库服务器时间为 2018-03-13 23:00:00 (UTC+1)，SELECT SYSTIMESTAMP FROM DUAL;的结果为 2018-03-13 23:00:00，而不是 2018-03-14 06:00:00。

```
SYSTIMESTAMP
```

为实现与 Oracle 兼容，EsgynDB 支持 SYSTIMESTAMP(n)，n 为返回的毫秒位数。

6.123.1 SYSTIMESTAMP 例子

- 该例子使用 SYSTIMESTAMP 函数计算周年日期和时间。

```
SELECT SYSTIMESTAMP AS today, SYSTIMESTAMP + INTERVAL
'12' MONTH AS Annisversary
FROM DUAL;
TODAY ANNISVERSARY
-------------------------- --------------------------
2018-03-15 11:19:42.400382 2019-03-15 11:19:42.400382
--- 1 row(s) selected.
```

- 该例子演示如何将 SYSTIMESTAMP 的值插入到一列中。

```
SQL>CREATE TABLE test1 (C1 TIMESTAMP, C2 VARCHAR(40));
--- SQL operation complete.

SQL>INSERT INTO test1 VALUES (SYSTIMESTAMP, 'This is
the time that I insert
values');
--- 1 row(s) inserted.

SQL>SELECT * FROM test1;
C1 C2
-------------------------- ----------------------------
------------
2018-03-15 11:33:32.091057 This is the time that I
insert values
--- 1 row(s) selected.
```

- 该例子演示如何使用 SYSTIMESTAMP(n)获取四位毫秒。

```
>>select systimestamp(4) from dual;

(EXPR)
------------------------

2019-10-24 23:34:25.5990

--- 1 row(s) selected.
```

6.124 SYS_GUID 函数

SYS_GUID 返回一个全局唯一标识符。该标识符是一个被转换为十六进制格式的 16 字节值。该函数用于与 Oracle 兼容。

```
SYS_GUID ()
```

6.124.1 结果的数据类型

结果的数据类型始终是 CHAR(32)。

6.124.2 SYS_GUID 例子

- 返回一个全局唯一标识符

```
SELECT SYS_GUID() AS ID FROM dual;
ID
-------------------------
5D35B676r5F758E6E0E40F00F0835536
--- 1 row(s) selected.
```

6.125 TAN 函数

该函数返回 NUMERIC 型数值表达式的正切值，这里的表达式是一个以弧度表示的角度。

TAN 函数是 EsgynDB SQL 扩展函数。

```
TAN (numeric-expression)
```

numeric-expression

这是 SQL NUMERIC 型数值表达式，它指定 TAN 函数的参数值。详情参考数字值表达式。

6.125.1 TAN 函数例子

- 该函数返回数值 3.64008908293626880E-001 或近似值 0.3640，即 0.3491 弧度（20 度）的正切：

  ```
  TAN (0.3491)
  ```

6.126 TANH 函数

该函数返回 NUMERIC 型数值表达式的双曲正切，这里的表达式是一个以弧度表示的角度。

TANH 函数是 EsgynDB SQL 扩展函数。

```
TANH(numeric-expression)
```

numeric-expression

这是 SQL NUMERIC 型数值表达式，它指定 TANH 函数的参数值。详情参考数字值表达式。

6.126.1 TANH 函数例子

- 该函数返回数值 8.48283639957512960E-001 或近似值 0.8483，即 1.25 弧度的双曲正切：

  ```
  TANH (1.25)
  ```

6.127 TIMESTAMPADD 函数

该函数将 *interval-ind* 和 *num_expr* 指定的时间间隔与 *datetime_expr* 相加。

如果指定的时间间隔以年、月或季为单位，则生成的日期将不是一个有效日期，该日将被四舍五入到结果月份的最后一天。该表达式将返回 *datetime_expr* 的类型，除非 *interval-ind* 包含任何时间分量，在这种情况下，该表达式将返回一个 TIMESTAMP。

TIMESTAMPADD 函数是 EsgynDB SQL 扩展函数。

```
TIMESTAMPADD (interval-ind, num_expr, datetime_expr)
```

interval-ind

是 SQL_TSI_YEAR、SQL_TSI_MONTH、SQL_TSI_DAY、SQL_TSI_HOUR、SQL_TSI_MINUTE、SQL_TSI_SECOND、SQL_TSI_QUARTER 或 SQL_TSI_WEEK。

num_expr

这是一个确切的 SQL 型数值表达式，指定与 *datetime_expr* 相加的 *interval-ind* 时间单元数量。如果 *num_expr* 带有小数部分，则该部分将被忽略。如果 *num_expr* 为负数，则返回值将（在时间上）早于 *datetime_expr*，具体幅度为指定的时间量。

datetime_expr

该表达式的求值结果是一个 DATE 或 TIMESTAMP 类型的日期时间值。该表达式将返回 *datetime_expr* 的类型，除非 *interval-ind* 包含任何时间分量，在这种情况下，该表达式将返回一个 TIMESTAMP。

6.127.1 TIMESTAMPADD 函数例子

- 该函数将 start-date 里指定的日期加上 7 天：
  ```
  TIMESTAMPADD  (SQL_TSI_DAY, 7, start-date)
  ```

- 该函数返回数值 DATE '2008–03–06'：
  ```
  TIMESTAMPADD (SQL_TSI_WEEK, 1, DATE '2008-02-28’)
  ```

- 该函数返回数值 DATE '1999–02–28':
  ```
  TIMESTAMPADD (SQL_TSI_YEAR, -1, DATE '2000-02-29’)
  ```

- 该函数返回数值 TIMESTAMP '2003–02–28 13:27：35':
  ```
  TIMESTAMPADD (SQL_TSI_MONTH, -12, TIMESTAMP '2004-02-29
  13:27: 35’)
  ```

- 该函数返回数值 TIMESTAMP '2004–02–28 13:27：35':
  ```
  TIMESTAMPADD (SQL_TSI_MONTH, 12, TIMESTAMP '2003-02-28
  13:27: 35’)
  ```

- 该函数返回数值 DATE '2008–06–30':
  ```
  TIMESTAMPADD (SQL_TSI_QUARTER, -2, DATE '2008-12-31’)
  ```

- 该函数返回数值 TIMESTAMP '2008–06–30 23:59:55':
  ```
  TIMESTAMPADD (SQL_TSI_SECOND, -5, DATE '2008-07-01’)
  ```

6.128 TIMESTAMPDIFF 函数

该函数返回一个整数值作为起始日期和结束日期之间的 interval-ind 时间单元数量。如果结束日期早于起始日期，则返回值将为负数或 0。

TIMESTAMPDIFF *(interval-ind, startdate, enddate)*

interval-ind
是 SQL_TSI_YEAR、SQL_TSI_MONTH、SQL_TSI_DAY、SQL_TSI_HOUR、SQL_TSI_MINUTE、SQL_TSI_SECOND、SQL_TSI_QUARTER 或 SQL_TSI_WEEK。

startdate 和 enddate
均为 DATE 或 TIMESTAMP 类型。

这种跨越时间界限（例如日、分钟和秒）的计数方法使 TIMESTAMPDIFF 函数的结果对所有数据类型都可以保持一致。

TIMESTAMPDIFF 函数针对时间界限做出以下假设：

- 每年都从 1 月 1 日开始算起。
- 一个新季度分别从 1 月 1 日、4 月 1 日、7 月 1 日和 10 月 1 日开始算起。
- 每周从周日开始算起。
- 每天从午夜开始算起。

该函数的结果是一个有符号整数值，等于第一个日期和第二个日期所跨的 *interval-ind* 时间界限数量。
例如：1 月 4 日（周日）和 1 月 4 日（周日）之间的周数为 1。3 月 31 日和 4 月 1 日之间的月数也为 1，因为 3 月和 4 月相隔一个月的时间。
如果函数结果超出了整数范围，则 TIMESTAMPDIF 函数将产生一个错误。最大秒数约为 68 年。
如果收到关于周数差值的请求，并且两个日期中的一个早于 0001 年 1 月 7 日，则 TIMESTAMPDIFF 函数也将产生一个错误。

6.128.1 TIMESTAMPDIFF 函数例子

- 该函数将返回数值 1，因为差值已经超过了 1 秒的时间界限，即使两个时间

戳仅仅相差 1 微秒也是如此:

```
TIMESTAMPDIFF (    SQL_TSI_SECOND,
            TIMESTAMP '2006-09-12 11: 59:58.999999',
            TIMESTAMP '2006-09-12 11: 59:59.000000')
```

- 该函数将返回数值 0，因为差值没有超过 1 秒的时间界限:

```
TIMESTAMPDIFF (    SQL_TSI_YEAR,
            TIMESTAMP '2006-12-31 23:59:59.00000,
            TIMESTAMP '2006-12-31 23:59:59.999999')
```

- 该函数将返回数值 1，因为差值已经超过了 1 年的时间界限:

```
TIMESTAMPDIFF (    SQL_TSI_YEAR,
            TIMESTAMP '2006-12-31 23:59:59.999999',
            TIMESTAMP '2007-01-01 00:00:00.000000;)
```

- 该函数将返回数值 1，因为差值已经超过了 1 周的时间界限:

```
TIMESTAMPDIFF (SQL_TSI_WEEK, DATE '2006-01-01', DATE
'2006-01-09')
```

- 该函数可以返回数值-29:

```
TIMESTAMPDIFF (SQL_TSI_DAY, DATE '2004-03-01', DATE
'2004-02-01')
```

6.129 TO_CHAR 函数

TO_CHAR 函数将日期时间值转换为字符值。可选的第二个参数描述字符值的格式。

```
TO_CHAR(character-expression [,format-string])
```

为兼容 Oracle，TO_CHAR 返回类型由原来的 char 改成了 varchar。

6.129.1 TO_CHAR 语法

1. 日期时间格式模式语法

• *character-expression*

是一个给出日期时间值的表达式。

• *format-string*

是以下字符串字面值之一：

支持的 *characterexpression*	**format-string**	描述
* *datetime-expression* **DATE** TIMESTAMP	'CC'	两位数世纪。
* *datetime-expression* **DATE** TIMESTAMP	'D'	一周中的某一天(1-7)，其中 1 是周天，6 是周六。 不可配置。
* *datetime-expression* **DATE** TIMESTAMP	'DD'	一月中的某一天(01-31)。
* *datetime-expression* **DATE** TIMESTAMP	'DDD'	一年中的某一天(001-366)。
* *datetime-expression* **DATE** TIMESTAMP	'DY'	星期名称，是星期的三个大写字母的缩写。 返回以下值： * MON * TUE * WED * THU * FRI * SAT * SUN
* *datetime-expression* **DATE** TIMESTAMP	'DAY'	星期名称，它是 9 个字符长度的空白填充的完整大写名称。 返回以下值：

		* MONDAY * TUESDAY * WEDSDAY * THURSDAY * FRIDAY * SATURDAY * SUNDAY
* *datetime-expression* **TIME** TIMESTAMP	'HH'	24 小时制的小时 (00-23)。 'HH'与'HH24'相同。
* *datetime-expression* **TIME** TIMESTAMP	'HH12'	12 小时制的小时(01-12)。
* *datetime-expression* **TIME** TIMESTAMP	'HH24'	24 小时制的小时(00-23)。 HH24'与'HH'相同。
* *datetime-expression* **DATE** TIMESTAMP	'J'	儒略日（即自公元前 4713 年 1 月 1 日以来的天数）。
* *datetime-expression* **TIME** TIMESTAMP	'MI'	分钟(00-59)。
* *datetime-expression* **DATE** TIMESTAMP	'MM'	月份(01-12)。
* *datetime-expression* **DATE** TIMESTAMP	'MON'	月份，是月份的三个大写字母的缩写。 返回以下值： * JAN * FEB * MAR * APR * MAY * JUN * JUL * AUG * SEP * OCT * NOV * DEC
* *datetime-expression* **DATE** TIMESTAMP	'Q'	季度 (1-4)。
* *datetime-expression* **TIME** TIMESTAMP	'SS'	秒(00-59)。
* *datetime-expression* **DATE** TIMESTAMP	'W'	一月中的某一周。
* *datetime-expression* **DATE** TIMESTAMP	'WW'	一年中的某一周。 * 如果一年从周天开

		始，则对于当年前 7 天内发生的日期时间，返回值 1。 否则，对于第一个周天开始之前的部分星期内发生的日期时间，返回值1。* 对于发生在一年中最后一个完整周或部分周的日期时间，返回值 **53**。* 对于在闰年的最后一个完整周或部分周（从周六开始）中发生的日期时间，返回值 **54**。
* *datetime-expression* **DATE** TIMESTAMP	'Y'	年份最后一位(0-9)。
* *datetime-expression* **DATE** TIMESTAMP	'YY'	年份最后两位(00-99)。
* datetime-expression **DATE** TIMESTAMP	'YYY'	年份最后三位(000-999)。
* datetime-expression **DATE** TIMESTAMP	'YYYY'	四位数年份(0001-9999)。

除了上面列出的 `format-string` 之外，以下字符串字面值也有效:

• 'YYYY-MM-DD'
• 'MM/DD/YYYY'
• 'DD.MM.YYYY'
• 'YYYY-MM'
• 'MM/DD/YYYY'
• 'YYYY/MM/DD'
• 'YYYYMMDD'
• 'YY/MM/DD'
• 'MM/DD/YY'
• 'MM-DD-YYYY'
• 'YYYYMM'
• 'DD-MM-YYYY'
• 'DD-MON-YYYY'
• 'DDMONYYYY'
• 'HH:MI:SS'
• 'HH24:MI:SS'
• 'YYYYMMDDHH24MISS'
• 'DD.MM.YYYY:HH24.MI.SS'
• 'YYYY-MM-DD HH24:MI:SS'

• 'YYYYMMDD:HH24:MI:SS'
• 'MMDDYYYY HH24:MI:SS'
• 'MM/DD/YYYY HH24:MI:SS'
• 'DD-MON-YYYY HH:MI:SS'
• 'MONTH DD, YYYY, HH:MI'
• 'DD.MM.YYYY HH24.MI.SS'
• 'HH24MISS'
• 'HH24MISSFFN'
• 'FF'

在这里:

• YYYY 指四位数的年份。
• YY 指两位数的年份。
• MM 指两位数的月份。
• MON 指月份的三个字母缩写（'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP'）。
• MONTH 指拼写出来的月份。在输出值中，以大写形式显示月份缩写或月份。
• HH 和 HH24 指两位数的小时字段。
• MI 指两位数的分钟字段。
• SS 指两位数的秒字段。
• FF 指分秒，在 FF 之后使用数字 1 到 9 指定返回的日期时间值的小数部分中的位数，如果没指定则返回默认的 9 位。
• 如果省略 *format-string* 参数，则默认使用“YYYY-MM-DD”为日期和时间戳值，“HH:MI:SS”为时间值。

2. 数字格式模式语法

TO_CHAR 函数支持将值 NUMBER，BINARY_FLOAT 或 BINARY_DOUBLE 数据类型转换为 VARCHAR 数据类型。

• *character-expression*

是一个给出数值的表达式。

• *format-string*

数字格式元素及例子如下:

元素	例子	描述	限制
,(逗号)	select to_char(123456,'9,999,999,') from dual; 返回:1,23,456,	逗号返回在指定位置，可以在数字格式模型中指定多个逗号。	1.逗号元素不能以数字格式模型开头。2.逗号不能出现

			在数字格式模型的十进制字符或句点的右侧。
.(句号)	select to_char(12.345,'99.999') from dual; 返回:12.345	在指定位子返回小数点。	只能在数字格式模型中指定一个句点。
$	select to_char(123456,'$9999999') from dual; 返回:$123456	返回带有前导美元符号的值。	$只能出现一次
0	select to_char(123.456,'00999.9990') from dual; 返回:00123.4560	返回前导零。返回尾随零。	
9	select to_char(+123456,'999999999') from dual; 返回:123456	返回具有指定位数的值，如果是正数则前导为空格，如果为负数则为减号.前导零显示空白，除了零值本身，定点数的整数部分返回零。	
B	select to_char(0.12345,'B99999') from dual; 返回为'　　、	当整数部分为零时，返回的定点数的整数部分为空白。	空格长度取决于 format-string 的长度，如实例中是 6 个空格。
C	select to_char(99.99,'C99999999') from dual; 返回:CNY100	在ISO货币符号的指定位置返回	
D	select to_char(12.345,'99D999') from dual; 返回:12.345	在指定位置返回十进制字符。默认为句点 (.)。	只能在数字格式模型中指定一个十进制字符。
EEEE	select to_char(12345678,'99.99EEEE') from dual; 返回:1.23E+07	使用科学计数法返回值。	
G	select to_char(123456,'99G9G99G9') from dual;	在组分隔符的指定位置返回。您可以在数字格式模型中	组分隔符不能出现在数字格式模型中的小数字符

	返回:12,3,45,6	指定多个组分隔符。默认为,(逗号)	或句点的右侧。
MI	select to_char(-9999,'9999MI') from dual; 返回:9999-	使用尾随减号(-)返回负值。返回带有尾随空白的正值。	MI 格式元素只能出现在数字格式模型的最后位置。
PR	select to_char(-9999,'9999PR') from dual; 返回:<9999>	返回<尖括号>中的负值。返回带有前导和尾随空白的正值。	PR 格式元素只能出现在数字格式模型的最后位置。
RN rn	select to_char(3999,'rn') from dual; 返回:mmmcmxcix	以大写形式返回罗马数字的值。以小写形式返回罗马数字的值。	值可以是 1 到 3999 之间的整数。
S	select to_char(+123456,'S999999') from dual; 返回:+123456	使用前导减号(-)返回负值。使用前导加号(+)返回正值。使用尾随减号(-)返回负值。使用尾随加号(+)返回正值。	S 格式元素只能出现在数字格式模型的第一个或最后一个位置。
V	select to_char(123.555,'999v99') from dual; 返回:12356	返回一个乘以 10 的 n 次方的值(如果需要,将其四舍五入),其中 n 是 V 之后的 9 的数字。	

FM	select to_char(1.2,'FM99.9') from dual;
	返回:1.2
	select to_char(0.596,'FM999,999,999,990.00') from dual;
	返回:0.60
		返回没有前导或尾随空格的值。(删除导致产生 9 的空白)	

6.129.2 TO_CHAR 注意事项

如果 *format-string* 只包含时、分或秒字段,则输入数据类型必须是时间或

时间戳。

如果 *format-string* 只包含年、月或日字段，则输入数据类型必须是日期或时间戳。

如果 *format-string* 包含所有字段，且输入数据类型是日期，则结果中的时、分和秒字段将用零填充。

6.129.3 TO_CHAR 例子

6.129.3.1TO_CHAR (CC)例子

- 该例子将 DATE 值转换为世纪的字符值。

```
SQL>SELECT TO_CHAR (DATE '0001-11-02','CC') FROM DUAL;
(EXPR)
------
01
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为世纪的字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '9899-12-31
23:59:59','CC') FROM DUAL;
(EXPR)
------
99
--- 1 row(s) selected.
```

6.129.3.2TO_CHAR (D)例子

- 该例子将 DATE 值转换为一周中的第几天的字符值。

```
SQL>SELECT TO_CHAR (DATE '2018-01-01','D') FROM DUAL;
(EXPR)
------
2
--- 1 row(s) selected.
```

- 该例子将 TIMESTAMP 值转换为一周中的第几天的字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '2018-04-01
23:59:59','D') FROM DUAL;
(EXPR)
------
1
--- 1 row(s) selected.
```

6.129.3.3TO_CHAR (DD)例子

- 该例子将 DATE 值转换为一月中的第几天的字符值。

```
SQL>SELECT TO_CHAR (DATE '2018-01-01','DD') FROM DUAL;
(EXPR)
------
01
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为一月中的第几天的字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '2018-12-31
23:59:59','DD') FROM DUAL;
(EXPR)
------
31
--- 1 row(s) selected.
```

6.129.3.4TO_CHAR (DDD)例子

- 该例子将 *DATE* 值转换为一年中的第几天的字符值。

```
SQL>SELECT TO_CHAR (DATE '2018-01-01','DDD') FROM DUAL;
(EXPR)
------
001
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为一年中的第几天的字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '2020-12-31
23:59:59','DDD') FROM DUAL;
(EXPR)
------
366
--- 1 row(s) selected.
```

6.129.3.5TO_CHAR (DY)例子

- 该例子将 *DATE* 值转换为星期名称（缩写）的字符值。

```
SQL>SELECT TO_CHAR (DATE '2018-12-31','DY') FROM DUAL;
(EXPR)
------
MON
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为星期名称（缩写）的字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '2019-02-14
```

```
23:59:59','DY') FROM DUAL;
(EXPR)
------
THU
--- 1 row(s) selected.
```

6.129.3.1TO_CHAR (DAY) 例子

- 该例子将 *DATE* 值转换为星期名称的字符值。

```
SQL>SELECT TO_CHAR (DATE '2019-05-12','DAY') FROM DUAL;
(EXPR)
---------
SUNDAY
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为星期名称的字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '2019-12-24
23:59:59','DAY') FROM DUAL;
(EXPR)
---------
TUESDAY
--- 1 row(s) selected.
```

6.129.3.2TO_CHAR (HH)例子

- 该例子将 *TIME* 值转换为小时（24 小时制）的字符值。

```
SQL>SELECT TO_CHAR (TIME '00:00:01','HH') FROM DUAL;
(EXPR)
------
00
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为小时（24 小时制）的字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '2019-01-01
23:59:59','HH') FROM DUAL;
(EXPR)
------
23
--- 1 row(s) selected.
```

6.129.3.3TO_CHAR (HH12)例子

- 该例子将 *TIME* 值转换为小时（12 小时制）的字符值。

```
SQL>SELECT TO_CHAR (TIME '01:00:00','HH12') FROM DUAL;
(EXPR)
```

```
------
01
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为小时（12 小时制）的字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '2019-01-01
12:59:59','HH12') FROM DUAL;
(EXPR)
------
12
--- 1 row(s) selected.
```

6.129.3.4TO_CHAR (HH24)例子

- 该例子将 *TIME* 值转换为小时（24 小时制）的字符值。

```
SQL>SELECT TO_CHAR (TIME '00:00:01','HH24') FROM DUAL;
(EXPR)
------
00
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为小时（24 小时制）的字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '2019-01-01
23:59:59','HH24') FROM DUAL;
(EXPR)
------
23
--- 1 row(s) selected.
```

6.129.3.5TO_CHAR (J)例子

- 该例子计算了从儒略纪元开始的天数并将 *DATE* 值转换为字符值。

```
SQL>SELECT TO_CHAR (DATE '0001-01-01','J') FROM DUAL;
(EXPR)
-------
1721426
--- 1 row(s) selected.
```

- 该例子计算了从儒略纪元开始的天数并将 *TIMESTAMP* 值转换为字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '2018-11-02
23:59:59','J') FROM DUAL;;
(EXPR)
-------
2458425
--- 1 row(s) selected.
```

6.129.3.6TO_CHAR (MI)例子

- 该例子将 *TIME* 值转换为分钟的字符值。

```
SQL>SELECT TO_CHAR (TIME '01:00:02','MI') FROM DUAL;
(EXPR)
------
00
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为分钟的字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '2019-01-01
23:59:01','MI') FROM DUAL;
(EXPR)
------
59
--- 1 row(s) selected.
```

6.129.3.7TO_CHAR (MM)例子

- 该例子将 *DATE* 值转换为月份的字符值。

```
SQL>SELECT TO_CHAR (DATE '0001-01-01','MM') FROM DUAL;
(EXPR)
------
01
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为月份的字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '2019-01-01
23:59:59','MM') FROM DUAL;
(EXPR)
------
12
--- 1 row(s) selected.
```

6.129.3.8TO_CHAR (MON)例子

- 该例子将 *DATE* 值转换为月份的字符值。

```
SQL>SELECT TO_CHAR (DATE '0001-01-01','MON') FROM DUAL;
(EXPR)
------
JAN
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为月份的字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '2018-11-02
```

```
23:59:59','MON') FROM DUAL;
(EXPR)
------
NOV
--- 1 row(s) selected.
```

6.129.3.9TO_CHAR (QUARTER)例子

- 该例子将 *DATE* 值转换为季度的字符值。

```
SQL>SELECT TO_CHAR (DATE '0001-01-01','Q') FROM DUAL;
(EXPR)
------
1
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为季度的字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '2018-11-02
23:59:59','Q') FROM DUAL;
(EXPR)
------
4
--- 1 row(s) selected.
```

6.129.3.10 TO_CHAR (SS)例子

- 该例子将 *TIME* 值转换为秒的字符值。

```
SQL>SELECT TO_CHAR (TIME '01:02:00','SS') FROM DUAL;
(EXPR)
------
00
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为秒的字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '2019-01-01
23:01:59','SS') FROM DUAL;
(EXPR)
------
59
--- 1 row(s) selected.
```

6.129.3.11 TO_CHAR (W)例子

- 该例子将 *DATE* 值转换为一月中的某一周的字符值。

```
SQL>SELECT TO_CHAR (DATE '2019-01-01','W') FROM DUAL;
(EXPR)
------
```

```
1
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为一月中的某一周的字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '2018-04-30
23:59:59','W') FROM DUAL;
(EXPR)
------
5
--- 1 row(s) selected.
```

6.129.3.12 TO_CHAR (WW)例子

- 该例子将 *DATE* 值转换为一年中的某一周的字符值。

```
SQL>SELECT TO_CHAR (DATE '2016-01-03','WW') FROM DUAL;
(EXPR)
------
02
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为一年中的某一周的字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '2000-12-31
23:59:59','WW') FROM DUAL;
(EXPR)
------
54
--- 1 row(s) selected.
```

6.129.3.13 TO_CHAR (Y)例子

- 该例子将 *DATE* 值转换为年的字符值（最后一位数）。

```
SQL>SELECT TO_CHAR (DATE '1000-01-01','Y') FROM DUAL;
(EXPR)
------
0
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为年的字符值（最后一位数）。

```
SQL>SELECT TO_CHAR (TIMESTAMP '9999-12-31
23:59:59','Y') FROM DUAL;
(EXPR)
------
9
--- 1 row(s) selected.
```

6.129.3.14 TO_CHAR (YY)例子

- 该例子将 *DATE* 值转换为年的字符值（最后两位数）。

```
SQL>SELECT TO_CHAR (DATE '1000-01-01','YY') FROM DUAL;
(EXPR)
------
00
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为年的字符值（最后两位数）。

```
SQL>SELECT TO_CHAR (TIMESTAMP '9999-12-31
23:59:59','YY') FROM DUAL;
(EXPR)
------
99
--- 1 row(s) selected.
```

6.129.3.15 TO_CHAR (YYY)例子

- 该例子将 *DATE* 值转换为年的字符值（最后三位数）。

```
SQL>SELECT TO_CHAR (DATE '1000-01-01','YYY') FROM DUAL;
(EXPR)
------
000
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为年的字符值（最后三位数）。

```
SQL>SELECT TO_CHAR (TIMESTAMP '9999-12-31
23:59:59','YYY') FROM DUAL;
(EXPR)
------
999
--- 1 row(s) selected.
```

6.129.3.16 TO_CHAR (YYYY)例子

- 该例子将 *DATE* 值转换为年字符值。

```
SQL>SELECT TO_CHAR (DATE '0001-01-01','YYYY') FROM
DUAL;
(EXPR)
------
0001
--- 1 row(s) selected.
```

- 该例子将 *TIMESTAMP* 值转换为年字符值。

```
SQL>SELECT TO_CHAR (TIMESTAMP '9999-12-31
23:59:59','YYYY') FROM DUAL;
```

```
(EXPR)
------
9999
--- 1 row(s) selected.
```

6.129.3.17 TO_CHAR (Other)例子

- 该函数返回字符值'01MAR2016':
  ```
  TO_CHAR (DATE '2016-03-01', 'DDMONYYYY')
  ```
- 该函数返回字符值'01.03.2016 00:00:00':
  ```
  TO_CHAR (DATE '2016-03-01', 'DD.MM.YYYY HH24.MI.SS')
  ```
- 该函数返回字符值 '12:05:10':
  ```
  TO_CHAR (TIME '12:05:10')
  ```
- 该函数返回字符值'20160301120510':
  ```
  TO_CHAR (TIMESTAMP '2016-03-01
  12:05:10','YYYYMMDDHH24MISS')
  ```

6.129.3.18 TO_CHAR (FF)的例子

- • 该例子返回 9 位分秒。
  ```
  >>select to_char(systimestamp,'FF') FROM DUAL;

  (EXPR)
  ---------

  758168000
  ```
- 该例子返回 3 位分秒。

  ```
  >>select to_char(systimestamp,'FF3') FROM DUAL;

  (EXPR)
  ---------

  135

  --- 1 row(s) selected.
  ```

6.130 TO_DATE 函数

TO_DATE 函数将字符值转换为日期。可选的第二个参数描述字符值的格式。

```
TO_DATE(character-expression [,format-string])
```

- *character-expression*

是一个字符值的表达式。

- *format-string*

是以下字符串字面值之一:

• 'YYYY-MM-DD'
• 'MM/DD/YYYY'
• 'DD.MM.YYYY'
• 'YYYY-MM'
• 'MM/DD/YYYY'
• 'YYYY/MM/DD'
• 'YYYYMMDD'
• 'YY/MM/DD'
• 'MM/DD/YY'
• 'MM-DD-YYYY'
• 'YYYYMM'
• 'DD-MM-YYYY'
• 'DD-MON-YYYY'
• 'DDMONYYYY'
• 'YYYYMMDDHH24MISS'
• 'DD.MM.YYYY:HH24.MI.SS'
• 'YYYY-MM-DD HH24:MI:SS'
• 'YYYYMMDD:HH24:MI:SS'
• 'MMDDYYYY HH24:MI:SS'
• 'MM/DD/YYYY HH24:MI:SS'
• 'DD-MON-YYYY HH:MI:SS'
• 'MONTH DD, YYYY, HH:MI'
• 'DD.MM.YYYY HH24.MI.SS'
• 'HH24:MI:SS.FF'
• 'YYYYMMDDHH24MISSFF'
• 'DD-MM-YYYY HH24:MI:SS'
• 'DD-MM-YYYY HH24:MI:SS.FF'
• 'DD.MM.YYYY:HH24.MI.SS.FF'
• 'YYYY-MM-DD HH24:MI:SS.FF'
• 'YYYYMMDD:HH24:MI:SS.FF'
• 'MMDDYYYY HH24:MI:SS.FF'
• 'MM/DD/YYYY HH24:MI:SS.FF'
• 'DD-MON-YYYY HH:MI:SS.FF'
• 'DD.MM.YYYY HH24.MI.SS.FF'
• 'YYYY/MM/DD HH24:MI:SS.FF'
• 'YYYY-MM-DD:HH24:MI:SS.FF'

在这里，YYYY 指四位数的年份。YY 指两位数的年份。MM 指两位数的月份。MON 指月份的三个字母缩写（‘JAN’、‘FEB’、‘MAR’、‘APR’、‘MAY’、‘JUN’、‘JUL’、‘AUG’、‘SEP’、‘OCT’、‘NOV’ 或‘DEC’）。MONTH 指拼写出来的月份。可以用小写、大写或任何混合形式给出 MON 或 MONTH 形式的月份。

HH 和 HH24 指两位数的小时字段。MI 指两位数的分钟字段。SS 指两位数的秒字段。FF 指分秒，默认是 9 位数。

如果省略 *format-string* 参数，则默认使用’YYYY-MM-DD’。

6.130.1 TO_DATE 注意事项

如果 *format-string* 包含时、分或秒字段，则输出数据类型为时间戳。否则，输出数据类型为日期。

6.130.2 TO_DATE 例子

- 该函数返回日期值'2016-12-07':
  ```
  TO_DATE ('2016-12-07')
  ```
- 该函数返回日期值'2016-07-12':
  ```
  TO_DATE ('12-07-2016', 'DD-MM-YYYY')
  ```
- 该函数返回日期值'2016-12-07':
  ```
  TO_DATE ('07DEC2016', 'DDMONYYYY')
  ```
- 该函数返回时间戳值'2016-12-07 17:05:01.000000':
  ```
  TO_DATE ('07.12.2016:17.05.01',
  'DD.MM.YYYY:HH24.MI.SS')
  ```
- 该函数返回时间戳值'2018-03-01 12:05:10.123456009':
  ```
  TO_DATE
  ('20180301120510123456009','YYYYMMDDHH24MISSFF')
  ```

6.131 TO_TIME 函数

TO_TIME 函数将字符值转换为时间。第二个参数描述字符值的格式。

```
TO_TIME(character-expression ,format-string)
```

character-expression
是一个字符值的表达式。

format-string
是以下字符串字面值之一:
• 'HH:MI:SS'
• 'HH24:MI:SS'

在这里，HH 和 HH24 指两位数的小时字段。MI 指两位数的分钟字段。SS 指两位数的秒字段。

6.131.1 TO_TIME 例子

- 该函数返回时间值'17:05:01':

```
TO_TIME ('17:05:01', 'HH24:MI:SS')
```

6.132 TO_TIMESTAMP 函数

TO_TIMESTAMP 函数将字符值转换为时间戳。

```
TO_TIMESTAMP(character-expression)
```

character-expression
是一个提供字符值的表达式。假设表达式格式为 YYYY-MM-DD HH:MI:SS[.FFFFFF]，其中 YYYY 是 4 位数年份，MM 是 2 位数月份，DD 是 2 位数日期，HH 是 2 位数小时字段，MI 是 2 位数分钟字段，SS 是 2 位数秒字段，FFFFFF 是可选微秒字段。

6.132.1 TO_TIMESTAMP 例子

- 该函数返回时间戳值 '2016-12-07 10:01:00':
  ```
  TO_TIMESTAMP ('2016-12-07 10:01:00')
  ```

6.133 TRANSLATE 函数

该函数既可以将一个字符串从源字符集转换为目标字符集，该字符串既更改该字符串的数据类型也更改字符集编码。

TRANSLATE *(character-value-expression USING translation-name)*

character-value-expression
这是一个字符串。

translation-name
这是这些转换名称之一：

转换名称	源字符集	目标字符集	注释
ISO88591TOUTF8	ISO88591	UTF8	将 ISO8859-1 字符转换为 UTF8 字符。不会出现数据损失。
UTF8TOISO88591	UTF8	ISO88591	将 UTF8 字符转换为 ISO88591 字符。如果遇到一个无法转换为目标字符集的 Unicode 字符，EsgynDB SQL 将显示一个错误。

translation-name 标识转换、源和目标字符集。当您将源字符集转换为 UTF8 字符集时，不会出现数据损失。然而，当 EsgynDB SQL 转换一个 UTF8 *character-value-expression* 时，可能存在特定字符无法转换为目标字符集的情况。在这种情况下，EsgynDB SQL 会报告一个错误。

EsgynDB SQL 返回一个可变长度字符串，其字符指令表与目标字符集的指令表相同，最大长度则等于源 *character-value-expression* 的固定长度或最大可变长度。

如果输入一个非法的 *translation-name*，EsgynDB SQL 将返回一个错误。

如果 *character-value-expression* 的字符集有别于 *translation-name* 里指定的源字符集，EsgynDB SQL 将返回一个错误。

6.134 TRIM 函数

该函数删除一个字符串的前导和尾随字符。包括多字节字符在内的每个字符都被视为一个字符。

```
TRIM (([trim-type] [trim-char] FROM] trim-source)
```

trim-type is:
LEADING | TRAILING | BOTH

trim-type
指定是否去除 trim-source 的前导端（LEADING）、尾随端（TRAILING）或两端（BOTH）的字符。如果省略 trim-type，则默认值为 BOTH。

trim_char
这是一种 SQL 字符值表达式，它指定需要从 trim-Source 里去除的字符。trim_char 的最大长度为 1。如果省略 trim_char，SQL 将去除 trim-source 里的空格（“）。

trim-source
这是一种 SQL 字符值表达式，它指定一个需要去除字符的字符串。更多信息，请参阅字符值表达式。

6.134.1 TRIM 函数注意事项

6.134.1.1TRIM 函数结果

该函数的结果始终为 VARCHAR 类型，其最大长度等于 *trim-source* 的固定长度或最大可变长度。如果源字符串是完成大写转换的 CHAR 或 VARCHAR 字符串，则函数结果也将为完成大写转换的 VARCHAR 类型。

6.134.2 TRIM 函数例子

- 返回'Robert'：
  ```
  TRIM (' Robert ')
  ```

- EMPLOYEE 表将 FIRST_NAME 定义为 CHAR(15)，并将 LAST_NAME 定义为 CHAR(20)。该表达式使用 TRIM 函数返回一个不含额外空白的值 'Robert Smith'：
  ```
  TRIM (first_name) || ' ' || TRIM (last_name)
  ```

6.135 TRUNC 函数

TRUNC (TRUNCATE) 函数截断 *numeric-expression*.

```
TRUNC (numeric-expression[, decimal-place])
```

如果未提供 *decimal-place*，或者提供了 *decimal-place* 且为零，则将 *numeric-expression* 四舍五入为最接近零的整数。

如果提供了 *decimal-place* 且为正数，比如 n，则将 *numeric-expression* 四舍五入到最接近于零的 10_{-n} 的倍数。

在这两种情况下，小数点右侧的小数位被有效删除。

TRUNC 是 EsgynDB SQL 扩展函数。

6.135.1 TRUNC 语法

numeric-expression
是一个要截断的 SQL 数值表达式。
更多信息，请参阅数数字值表达式。

decimal-place
是一个指定截断精度的整数。

注意：*numeric-expression* 和 *decimal-place* 不能为 NULL。
目前，当**不支持负整数**时，*decimal-place* 可以是零或正整数。
如果省略 *decimal-place*，则将 *numeric-expression* 截断到小数点后 0 位，即小数点后的所有数字都将被删除。

例子

```
SELECT TRUNC (-1.2345), TRUNC (1.2345) FROM DUAL
(EXPR) (EXPR)
----------- -----------
-1 1
--- 1 row(s) selected.
```

注意：如果 *decimal-place* 是一个正整数，则将 *numeric-expression* 截断为小数点右侧的 *decimal-place* 小数位。

例子
1 用于截断为 0.1 的倍数
2 用于截断为 0.01 的倍数
3 用于截断为 0.001 的倍数

例子

```
SELECT TRUNC (1.2345,1), TRUNC (1.2345,2), TRUNC (1.2345,
3), TRUNC (1.2345, 4) FROM
DUAL;
(EXPR) (EXPR) (EXPR) (EXPR)
------------ ------------ ------------ ------------
1.2 1.23 1.234 1.2345
--- 1 row(s) selected.
```

注意：*decimal-place* 不能大于小数位数。

6.135.2 TRUNC 注意事项

- EsgynDB 提供了“CEIL 函数”、“FLOOR 函数”、“ROUND 函数”和 TRUNC 函数进行四舍五入，这些函数在特定情况下可以互换。

例子

条件	结果
TRUNC 函数的以下所有条件均为真： ¬ *numeric-expression* 大于零 ¬ *decimal-place* 未指定	**TRUNC** 函数 = **FLOOR** 函数
TRUNC 函数的以下所有条件均为真： ¬ *numeric-expression* 小于零 ¬ *decimal-place* 未指定	**TRUNC** 函数= **CEIL** 函数

- ***在减少小数位数方面***，TRUNC 函数与 ROUND 函数类似。

但是，TRUNC 函数舍入为零，而 ROUND 函数舍入为最接近的整数。

注意：
更多信息，请参阅舍入函数的比较。

6.135.3 TRUNC 例子

- 该例子演示了 TRUNC 函数如何与其他函数一起使用。

```
SELECT ROUND(TRUNC(ABS(-1.7256),3),2) FROM DUAL;
(EXPR)
---------------------
1.730
--- 1 row(s) selected.
```

6.136 UCASE 函数

UCASE 函数对字母数字字符进行大写转换。对于非字母数字字符而言，UCASE 将返回相同的字符。

UCASE 函数可以出现在一条查询语句中的任何位置，例如选择列表、ON 子句、WHERE 子句、HAVING 子句、LIKE 谓词或表达式，也可以在 UPDATE 或 INSERT 语句里限定一个新值。

UCASE 函数的返回结果等于 UPPER 函数或 UPSHIFT 函数的返回结果。

UCASE 函数返回一个由固定长度或可变长度字符数据组成的字符串，具体根据输入字符串的数据类型而定。

UCASE 函数是 EsgynDB SQL 扩展函数。

```
UCASE (character-expression)
```

character-expression

这是一种 SQL 字符值表达式，它指定一个需要进行大写转换的由多个字符组成的字符串。更多信息，请参阅字符值表达式。

6.136.1 UCASE 函数注意事项

对于一个 UTF8 字符表达式而言，UCASE 函数可以将所有的小写或标题大小写字符都转换为大写字符，并返回一个字符串。如果参数为 CHAR(n)或 VARCHAR(n)类型，则函数结果将为 VARCHAR(min(3n, 2048))类型。这里的 VARCHAR 最大长度至少为 3n 和 2048 中的较小值。

小写字符是一种具有 Unicode 2 标准里的 alphabetic 属性、并且 Unicode 名称中包含小写字母字符。大写字符是一种具有 Unicode 2 标准里的 alphabetic 属性、并且 Unicode 名称中包含大写字母字符。标题大小写字符则是一种具有 Unicode “alphabetic”属性、并且 Unicode 名称中包含标题字符。

6.136.2 UCASE 函数例子

- 假设 CUSTOMER 表包含一个 Hotel Oregon（俄勒冈州旅馆）输入项。选择列 CUSTNAME，并使用 UPPER 和 LOWER 函数返回完成大小写转换的该列值：

  ```
  SELECT custname,UPPER(custname),LOWER(custname)
  FROM sales.customer;
  ```

```
(EXPR)              (EXPR)               (EXPR)
----------------- -------------------- -------------------
...                 ...                  ...
Hotel Oregon        HOTEL OREGON         hotel oregon
--- 17 row(s) selected.
```

更多信息，请参阅 LCASE 函数和 UPSHIFT 函数。

6.137 UPPER 函数

该函数对字母数字字符进行大写转换。对于非字母数字字符而言，UCASE 将返回相同的字符。

UPPER 函数可以出现在一条查询语句中的任何位置，例如选择列表、ON 子句、WHERE 子句、HAVING 子句、LIKE 谓词或表达式，也可以在 UPDATE 或 INSERT 语句里限定一个新值。
UPPER 函数的返回结果等于“UCASE 函数”或“UPSHIFT 函数”的返回结果。
UPPER 函数返回一个由固定长度或可变长度字符数据组成的字符串，具体取决于输入字符串的数据类型。

```
UPPER (character-expression)
```

character-expression

这是一种 SQL 字符值表达式，它指定一个需要进行的大写转换的由多个字符组成的字符串。更多信息，请参阅字符值表达式。

6.137.1 UPPER 函数例子

- 假设 CUSTOMER 表包含一个 Hotel Oregon（俄勒冈州旅馆）输入项。选择列 CUSTNAME，并使用 UPPER 和 LOWER 函数返回完成大/小写转换的该列值：

```
SELECT custname,UPPER(custname),LOWER(custname)
FROM sales.customer;

(EXPR)              (EXPR)               (EXPR)
----------------- -------------------- -------------------
...                ...                  ...
Hotel Oregon       HOTEL OREGON         hotel oregon
--- 17 row(s) selected.
```

更多信息，请参阅 LCASE 函数和 UPSHIFT 函数。

6.138 UPSHIFT 函数

该函数对字母数字字符进行大写转换。对于非字母数字字符而言，UPSHIFT 函数将返回相同的字符。

UPSHIFT 函数可以出现在一条查询语句（可以使用一个数值）中的任何位置，例如选择列表、ON 子句、WHERE 子句、HAVING 子句、LIKE 谓词或表达式，也可以在 UPDATE 或 INSERT 语句里限定一个新值。

UPSHIFT 函数的返回结果等于 UCASE 函数或 UPPER 函数的返回结果。

UPSHIFT 函数返回一个由固定长度或可变长度字符数据组成的字符串，具体取决于输入字符串的数据类型。

UPSHIFT 函数是 EsgynDB SQL 扩展函数。

```
UPSHIFT (character-expression)
```

character-expression

这是一种 SQL 字符值表达式，它指定一个需要进行大写转换的由多个字符组成的字符串。更多信息，请参阅字符值表达式。

6.138.1 UPSHIFT 函数例子

- 假设 CUSTOMER 表包含一个 Hotel Oregon（俄勒冈州旅馆）输入项。选择列 CUSTNAME，并使用 UPSHIFT、UPPER 和 LOWER 函数返回完成大小写转换的该列值：

```
SELECT UPSHIFT(custname),UPPER(custname),UCASE(custname)
FROM sales.customer;

(EXPR)             (EXPR)              (EXPR)
----------------- ------------------- ------------------
...                ...                 ...
Hotel Oregon       HOTEL OREGON        hotel oregon
--- 17 row(s) selected.
```

- 为 DataSpeed 客户执行一次不区分大小写的搜索：

```
SELECT *
FROM sales.customer
WHERE UPSHIFT (custname) = 'DATASPEED';
CUSTNUM CUSTNAME   STREET                CITY       ...
------- ---------- -------------------- --------- ...
1234    DataSpeed  300 SAN GABRIEL WAY  NEW YORK   ...

--- 1 row(s) selected.
```

表里的名称可以采用大写字母、小写字母或大小写混合字母。

- 假设数据库包含两个部门表：DEPT1 和 DEPT2。该查询可以返回两个表里“部门名称”相同（不区分大小写）的所有行：

```
SELECT * FROM persnl.dept1 D1, persnl.dept2 D2
WHERE UPSHIFT (D1.deptname)  = UPSHIFT (D2.deptname) ;
```

6.139 USER 函数

该函数返回指定用户 ID 号所关联的数据库用户名或调用该函数的当前用户的数据库用户名。当前用户指的是启动会话、并且经过身份验证的用户。该数据库用户名用于授权当前会话里的 SQL 语句。

```
USER [ (user-id) ]
```

user-id

这是一个与数据库用户名相关联的 32 位数。

USER 函数类似 AUTHNAME 函数和 CURRENT_USER 函数 。

6.139.1 USER 函数注意事项

- 仅能在 SELECT 语句的顶层指定该函数。
- 返回值为字符串数据类型 VARCHAR(128)，并且采用 ISO8859-1 编码。

6.139.2 USER 函数例子

- 该例子显示了登录到该会话的当前用户的数据库用户名:
```
>>SELECT USER FROM (values(1)) x(a);
(EXPR)
-------------------------
TSHAW
--- 1 row(s) selected.
```
- 该例子显示了用户 ID 号 33333 所关联的数据库用户名:
```
>>SELECT USER (33333) FROM (values(1)) x(a);
(EXPR)
-------------------------
DB ROOT
--- 1 row(s) selected.
```

6.140 UUID 函数

返回一个根据 RFC 4122 生成的通用唯一标识符(UUID)。UUID 被设计为一个在空间和时间上全局唯一的数字。对 UUID()的两次调用将生成两个不同的值，即使这些调用是在两个互不连接的独立设备上执行的。

注意: UUID()行为是不确定的。因此，最好只在 SELECT 列表中使用它。例如，在 WHERE 子句中使用它可能导致非确定性 i 谓词求值。

```
UUID ()
```

6.140.1 结果的数据类型

结果的数据类型始终是 CHAR(32)。

6.140.2 UUID 例子

- 返回一个全局唯一标识符

```
SELECT UUID() AS ID FROM dual;
ID
-------------------------
5cdd380c-33ba-3216-1352-2c9d624042c3
--- 1 row(s) selected.
```

6.141 UUID_SHORT 函数

将“短”通用标识符作为一个 64 位无符号整数返回。

注意：UUID_SHORT()行为是不确定的。因此，最好只在 SELECT 列表中使用它。
例如，在 WHERE 子句中使用它可能导致 i 非确定性谓词求值。

```
UUID_SHORT ()
```

6.141.1 结果的数据类型

结果的数据类型始终是 UNSIGNED LARGEINT。

6.141.2 UUID_SHORT 例子

- 将全局唯一标识符作为一个 64 位无符号整数返回。

```
SELECT UUID_SHORT() AS ID FROM dual;
ID
-------------------------
4716501302154732154
--- 1 row(s) selected.
```

6.142 UNIX_TIMESTAMP 函数

如果不使用参数调用，则将 Unix 时间戳(自'1970-01-01 00:00:00' UTC 以来的秒数)作为一个无符号整数返回。

如果使用一个日期参数调用 UNIX_TIMESTAMP()，则返回自“1970-01-01 00:00:00”UTC 以来的参数值（秒）。

UNIX_TIMESTAMP 是 MySQL SQL 扩展函数。

```
UNIX_TIMESTAMP (datetime-expression
| datetime-expression,format-string
| emtpy)
```

- *datetime-expression*

是一个值为日期或时间戳类型的日期时间值的表达式。更多信息，日期时间值表达式。

- *format-string*

'YYYY-MM-DD'
'MM/DD/YYYY'
'DD.MM.YYYY'
'YYYY-MM'
'MM/DD/YYYY'
'YYYY/MM/DD'
'YYYYMMDD'
'YY/MM/DD'
'MM/DD/YY'
'MM-DD-YYYY'
'YYYYMM'
'DD-MM-YYYY'
'DD-MON-YYYY'
'DDMONYYYY'
'HH:MI:SS'
'HH24:MI:SS'
'YYYYMMDDHH24MISS'
'DD.MM.YYYY:HH24.MI.SS'
'YYYY-MM-DD HH24:MI:SS'

'YYYYMMDD：HH24:MI:SS'

'MMDDYYYY HH24:MI:SS'

'MM/DD/YYYY HH24:MI:SS'

'DD-MON-YYYY HH:MI:SS'

'MONTH DD,YYYY,HH:MI'

'DD.MM.YYYY HH24.MI.SS'

…

6.142.1 UNIX_TIMESTAMP 例子

- 该例子使用当前的时间戳生成 unix 时间戳。

```
SELECT UNIX_TIMESTAMP() FROM DUAL;
(EXPR)
--------------------
          1534672251
--- 1 row(s) selected.
```

- 该例子使用指定的时间戳生成 unix 时间戳。

```
SELECT UNIX_TIMESTAMP('2019-11-02 20:36:18') FROM DUAL;
(EXPR)
----------------------
            1572726978
--- 1 row(s) selected.
```

- 该例子使用指定的时间戳按照指定格式生成 unix 时间戳。

```
>>SELECT UNIX_TIMESTAMP ('2019-03-28 11:36:20', 'YYYY-
MM-DD HH24:MI:SS') FROM DUAL;

(EXPR)
--------------------

          1553772980

--- 1 row(s) selected.
```

6.143 VARIANCE 函数

该函数是聚合函数，它返回一组数字的统计方差。VARIANCE 函数是 EsgynDB SQL 扩展函数。

VARIANCE ([ALL | DISTINCT] *expression [,weight]*)

ALL | DISTINCT

指定计算该表达式的 VARIANCE 函数时是否将重复值包含在内。

默认选项为 ALL，这样会使所有重复值都被包含在内。如果指定了 DISTINCT，则需在应用 VARIANCE 函数之前消除所有的重复值。

如果指定了 DISTINCT，则不得指定权重。

expression

指定一种 NUMERIC 型数值表达式，它能确定参与方差计算的数值。该表达式不得包含聚合函数或子查询。DISTINCT 子句指定 VARIANCE 函数运行在表达式求值所派生的单列表的不同值上。

weight

指定一种 NUMERIC 型数值表达式，它能确定参与方差计算的数值的权重。*weight* 不得包含聚合函数或子查询。*weight* 与表达式定义在相同的表上。表达式求值所派生的单列表和 *weight* 求值所派生的单列表必须具有相同的基数。

6.143.1 VARIANCE 函数注意事项

6.143.1.1VARIANCE 函数定义

假设 vi 是表达式求值所派生的单列表里的数值，N 是将表达式应用到源表的每一行、并消除空值行后所得的单列表的基数。

如果已经指定权重， wi 是权重求值所派生的数值。N 指的是将表达式和权重应用到源表的每一行、并消除任何一列的空值行后所得的两列的表的基数。

6.143.1.2未指定权重时的定义

如果没有指定权重，则单列结果表里的数值的统计方差被定义为:

这里的 vi 是表达式的第 i 个值, v 是以常见数据类型表示的平均值，N 是结果表的基数。

在方差的定义里，N-1 是表达式的分母（如果未指定权重），所以，如果一个表

或一组表里的行数都为 1，则 EsgynDB SQL 将返回默认值 0（没有错误）。

6.143.1.3指定权重时的定义

如果已经指定了权重，则两列结果表里的数值的统计方差被定义为：

这里的 vi 是表达式的第 i 个值，wi 是权重的第 i 个值，w 是以常见数据类型表示的加权平均值，N 是结果表的基数。

6.143.1.4加权平均值

vi 和 wi 的加权平均值 vw 的定义是：

这里的 vi 是表达式的第 i 个值，wi 是权重的第 i 个值，N 是结果表的基数。

6.143.1.5函数结果的数据类型

函数结果的数据类型始终为 DOUBLE PRECISION（双精度型）。

6.143.1.6表达式的操作数

表达式包含 SELECT 结果表的行里的列，但不得包含聚合函数。以下为有效表达式：

```
VA VARIANCE (SALARY) VARIANCE (SALARY * 1.1)
VARIANCE (PARTCOST * QTY_ORDERED)
```

6.143.1.7空值

消除集合里的所有空值后，对 VARIANCE 函数进行求值。如果结果表为空，则 VARIANCE 将返回 NULL。

6.143.1.8FLOAT(54)和 DOUBLE PRECISION 数据

不得将大型 FLOAT(54)或 DOUBLE PRECISION 值用作 VARIANCE 函数的参数。如果计算 VARIANCE(x)期间 SUM (x* x)超过了数值 1.15792089237316192e77，则将出现数字溢出的情况。

6.143.2 VARIANCE 函数例子

- 计算当前雇员工资的方差：

```
SELECT VARIANCE(salary) AS Variance_Salary
FROM persnl.employee;

VARIANCE_SALARY
-------------------------
```

```
1.27573263588496116E+009

--- 1 row(s) selected.
```

- 计算当前库存中的部件成本的方差:

```
SELECT VARIANCE (price * qty_available)
FROM sales.parts;

(EXPR)
-------------------------
5.09652410092950336E+013

--- 1 row(s) selected.
```

6.144 WEEK 函数

该函数将 DATE 或 TIMESTAMP 表达式转换为一个 1 到 54 范围内的 INTEGER 值，它表示一年中的相应周号。如果一年从周日开始算起，则对于第一个 7 天里所出现的任何日期时间，该函数都将返回数值 1。否则，对于一年中的第一个周日开始前的部分周里所出现的任何日期时间，该函数也将返回数值 1；对于一年（从周六开始算起的闰年除外，在这里，12 月 31 日是第 54 个完整或部分周）中的最后一个完整周或部分周里所出现的任何日期时间，该函数都将返回数值 53。

WEEK 函数是 EsgynDB SQL 扩展函数。

WEEK *(datetime-expression)*

datetime-expression

该表达式的求值结果为一个 DATE 或 TIMESTAMP 类型的日期时间值。更多信息，请参阅日期时间值表达式。

6.144.1 WEEK 函数例子

- 返回一个整数，它表示 PROJECT 表的 START_DATE 列里的一年中的周号：

```
SELECT start_date, ship_timestamp,
WEEK(start_date)
FROM persnl.project
WHERE projcode = 1000;

Start/Date Time/Shipped                (EXPR)
---------- -------------------------- --------------
2008-04-10 2008-04-21 08:15:00.000000           15
```

6.145 YEAR 函数

该函数将 DATE 或 TIMESTAMP 表达式转换为一个 INTEGER 值，它代表年份。YEAR 函数是 EsgynDB SQL 扩展函数。

```
YEAR (datetime-expression)
```

datetime-expression

该表达式的求值结果为一个 DATE 或 TIMESTAMP 类型的日期时间值。

更多信息，请参阅日期时间值表达式。

6.145.1 YEAR 函数例子

- 返回一个整数，它表示 PROJECT 表的 START_DATE 列里的年份：

```
SELECT start_date, ship_timestamp, YEAR(start_date)
FROM persnl.project
WHERE projcode = 1000;

Start/Date Time/Shipped              (EXPR)
---------- --------------------------- ------
2008-04-10 2008-04-21 08:15:00.000000 2008
```

6.146 ZEROIFNULL 函数

如果表达式为 NULL，则 ZEROIFNULL 函数将返回零值。否则将返回表达式的值。

```
ZEROIFNULL (expression)
```

expression
指定一个表达式；它必须为 NUMERIC 数据类型。

6.146.1 ZEROIFNULL 函数例子

- 如果列值不为 NULL，ZEROIFNULL 函数都将返回 salary 列的值；否则如果列值为 NULL，该函数都将返回 0。

  ```
  ZEROIFNULL (salary)
  ```

7. OLAP 函数

本节描述了联机分析处理（OLAP）窗口函数的语法和语义。OLAP 窗口函数符合 ANSI 标准。

联机分析处理(OLAP)函数提供了将源数据划分成行组，对每个组中的行进行排序和计算一行在窗口框架内的聚合的功能。

通常，SQL 中的 OLAP 函数用两个组件<FUNCTION>和<OVER>表示：<FUNCTION >指定函数和参数，<OVER>指定分区、排序和窗口框架。<OVER>有时被称为窗口。OLAP 函数用于对存储在一组表中或从一组表中计算的数据集执行分析任务，例如移动平均值和排名。

本节描述了 OLAP 窗口函数的语法和语义。OLAP 窗口函数符合 ANSI 标准。

本章目录:

7.1 窗口函数注意事项
7.2 ORDER BY 子句支持 OLAP 函数表达式
7.3 窗口函数的使用限制
7.4 AVG 窗口函数
7.5 COUNT 窗口函数
7.6 DENSE_RANK 窗口函数
7.7 FIRST_VALUE 窗户函数
7.8 LAG 窗口函数
7.9 LEAD 窗口函数
7.10 LAST_VALUE 窗口函数
7.11 LISTAGG 窗口函数
7.12 MAX 窗口函数
7.13 MIN 窗口函数
7.14 RANK 窗口函数
7.15 ROW_NUMBER 窗口函数
7.16 STDDEV 窗口函数
7.17 STDDEV_POP 窗口函数
7.18 SUM 窗口函数
7.19 SYS_CONNECT_BY_PATH 窗口函数
7.20 VARIANCE 窗口函数

7.1 窗口函数注意事项

这些注意事项适用于所有窗口函数:

- *inline-window-specification*

inline-window-specification 定义的窗口包含 *window-frame-clause* 指定的行，它们被当前分区所限定。如果没有指定 PARTITION BY 子句，则分区被定义为中间结果表的所有行；如果指定了 PARTITION BY 子句，则分区指的是与 PARTITION 子句里指定的表达式数值相同的行集。

- *window-frame-clause*
  窗口函数不支持 DISTINCT。
  不支持使用一种 FOLLOWING 术语，使用它会导致出现错误。
  如果没有指定 *window-frame-clause*，则将假设“ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING”。由于涉及 FOLLOWING 术语，所以，该子句未获支持，将导致出现错误。
  “ROWS CURRENT ROW”等效于“ROWS BETWEEN CURRENT ROW AND CURRENT ROW”。
  “ROWS preceding-row”等效于“ROWS BETWEEN preceding-row AND CURRENT ROW”。

7.1.1 空值

将该函数应用到一组数值前，须消除所有的空值。如果窗口中包含所有 NULL 值，则该窗口函数的结果为 NULL 值。

如果一个特定行的指定窗口包含分区第一行之前的所有行（不含窗口中的行），则窗口函数的结果为 NULL 值。

7.2 ORDER BY 子句支持 OLAP 函数表达式

OLAP 函数的 ORDER BY 子句现在支持表达式，但不支持在同一条查询语句里同时使用多个不同表达式的 OLAP 函数。以下例子显示了这些表达式在 ORDER BY 子句里可行的使用方案。

```
SESELECT -1 * annualsalary neg_total,
RANK() OVER (ORDER BY -1 * annualsalary) olap_rank
FROM employee;
```

在 ORDER BY 子句里使用一种聚合函数:

```
SELECT num,
RANK() OVER (ORDER BY SUM(annualsalary)) olap_rank
FROM employee
GROUP BY num;
```

在 ORDER BY 子句里同时在多个函数里使用同一个表达式:

```
SELECT num, workgroupnum,
RANK() OVER (ORDER BY SUM (annualsalary)*num) olap_rank,
DENSE_RANK() OVER (ORDER BY SUM (annualsalary)*num)
olap_drank
ROW_NUMBER() OVER (ORDER BY SUM (annualsalary)*num)
olap_mum
FROM employee
GROUP BY num, workgroupnum, annualsalary;
```

在 ORDER BY 子句里同时在多个函数里使用同一个表达式:

```
SELECT num, workgroupnum, annualsalary,
SUM(AnnualSalary) OVER (ORDER BY SUM(annualsalary)*num
ROWS UNBOUNDED PRECEDING),
AVG(AnnualSalary) OVER (ORDER BY SUM(annualsalary)*num
ROWS UNBOUNDED PRECEDING),
MIN(AnnualSalary) OVER (ORDER BY SUM(annualsalary)*num
ROWS UNBOUNDED PRECEDING),
MAX(AnnualSalary) OVER (ORDER BY SUM(annualsalary)*num
ROWS UNBOUNDED PRECEDING),
VARIANCE(AnnualSalary) OVER (ORDER BY
SUM(annualsalary)*num
ROWS UNBOUNDED PRECEDING),
STDDEV(AnnualSalary) OVER (ORDER BY SUM(annualsalary)*num
ROWS UNBOUNDED PRECEDING),
```

```
COUNT(AnnualSalary) OVER (ORDER BY SUM(annualsalary)*num
ROWS UNBOUNDED PRECEDING),
FROM employee
GROUP BY num, workgroupnum, annualsalary;
```

7.3 窗口函数的使用限制

EsgynDB 支持许多符合 ANSI 标准的 OLAP 窗口函数。这些函数包括 AVG、COUNT,、DENSE_RANK、FIRST_VALUE、LAG、LAST_VALUE、LEAD、MAX、MIN、RANK、ROW_NUMBER、STDDEV、SUM 和 VARIANCE。

这些使用限制适用于所有窗口函数:

- EsgynDB 不支持 ANSI 标准的 *window-clause*，仅支持 *inline-window-specification*。如果尝试使用 ANSI 标准的 *window-clause* 将导致出现语法错误。

  注意: 窗口子句提供了一种在 SQL 查询中定义和引用窗口的机制，如以下例子所示。

  ```
  SELECT EmployeeID, Surname, Salary, State,
    AVG( Salary ) OVER Salary_Window
  FROM Employees
  WINDOW Salary_Window AS ( PARTITION BY State )
  ORDER BY State, Surname;
  ```

- *window-frame-clause* 不得显式或隐式地包含 FOLLOWING 术语。因为默认的 window frame 子句里含有隐式 FOLLOWING （“ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING”），所以 EsgynDB 默认不支持它。因此，*window-frame-clause* 实际上是不可选的。显式或隐式地使用 FOLLOWING 术语都将导致出现“4343”错误消息。

- window frame 单位只能是 ROWS。RANGE 未获 EsgynDB 支持。尝试使用 RANGE 将导致出现语法错误。
  两者的区别在于，带有 ROWS 单元的框架包含所有行，而不考虑 ORDER BY 子句中的值。带有 RANGE 单元的框架将包含 ORDER BY 子句中值相同的行。

- EsgynDB 不支持 ANSI *window-frame-exclusion-specification*。尝试使用 *window-frame-exclusion-specification* 将导致出现语法错误。

- EsgynDB 不支持在 SELECT 子句里使用多个 *inline-window-specification*。SELECT 子句中的每个窗口函数都必须具有相同的 ORDER BY 子句和 PARTITION BY 规范。 SELECT 子句中的窗框可以改变。在 SELECT 子句里尝试使用多个 *inline-window-specification* 将导致出现"4340"错误消息。

- EsgynDB 不支持 ORDER BY 子句中的 ANSI *null-ordering-specification*。
  在进行排序时，空值始终大于所有非空值，这与 NULLS LAST 函数的空值排序略有不同。尝试使用 *null-ordering-specification* 将导致出现语法错误。

- EsgynDB 不支持将 ANSI filter 子句用于窗口函数。*filter-clause* 适用于所有聚合函数（分组型和窗口型）。分组型聚合函数目前不支持 *filter-clause*，尝试使用 *filter-clause* 将导致出现语法错误。

- 不支持在一个窗口函数的 set-qualifier-clause（设置限定符子句）里使用 DISTINCT 值。仅支持在其中使用 ALL 值。在一个窗口函数里使用 DISTINCT 值将会导致出现“4341”错误消息。

7.4 AVG 窗口函数

AVG 是窗口函数，它为 inline-window specification 指定的当前窗口返回一个给定表达式的非空值的平均值。

```
AVG ([ALL] expression) OVER (inline-window-specification)
```

inline-window-specification 是:

```
     [PARTITION BY expression [, expression]...]
     [ORDER BY expression [ASC[ENDING] | DESC[ENDING]]
          [,expression [ASC[ENDING] | DESC[ENDING]]]...]
     [ window-frame-clause ]
```

window-frame-clause 是:

```
     ROWS CURRENT ROW
     | ROWS preceding-row
     | ROWS BETWEEN preceding-row AND preceding-row
     | ROWS BETWEEN preceding-row AND CURRENT ROW
     | ROWS BETWEEN preceding-row AND following-row
     | ROWS BETWEEN CURRENT ROW AND CURRENT ROW
     | ROWS BETWEEN CURRENT ROW AND following-row
     | ROWS BETWEEN following-row AND following-row
```

preceding-row 是:

```
     UNBOUNDED PRECEDING
     | unsigned-integer PRECEDING
```

following-row 是:

```
     UNBOUNDED FOLLOWING
     | unsigned-integer FOLLOWING
```

ALL

指定计算该表达式的 AVG 函数时是否将重复值包含在内。默认选项为 ALL，这样会使所有重复值都被包含在内。

expression

指定一个 NUMERIC 型数值或 INTERVAL 型数值表达式，它确定需要进行平均的数值。详细信息，请参考数字值表达式和日期间隔值表达式。

inline-window-specification
指定一个进行 AVG 函数计算的窗口。*inline-window-specification* 包含可选的 PARTITION BY 子句、可选的 ORDER BY 子句和可选的 window frame 子句。PARTITION BY 子句指定中间结果表的分区方案，而 ORDER BY 子句则指定各分区内的行排序方式。

window-frame-clause
指定分区内进行 AVG 函数计算的窗口。

7.4.1 AVG 窗口函数例子

- 返回 SALARY 列的运行平均值:
```
SELECT empnum, AVG(salary)
   OVER (ORDER BY empnum ROWS UNBOUNDED PRECEDING)
FROM persnl.employee;
```

- 返回各部门内的 SALARY 列的运行平均值:
```
SELECT deptnum, empnum, AVG(salary)

OVER (          PARTITION BY deptnum
                ORDER BY empnum
                ROWS UNBOUNDED PRECEDING)
FROM persnl.employee;
```

- 在最后 4 行的窗口上返回每个部门工资的移动平均线:
```
SELECT deptnum, empnum, AVG(salary)
   OVER (PARTITION BY deptnum ORDER BY empnum ROWS 3
   PRECEDING)
FROM persnl.employee;
```

7.5 COUNT 窗口函数

COUNT 是一种窗口函数，它为 inline-window specificatio 指定的当前窗口返回一个给定表达式的非空值的计数。

```
COUNT {(*) | ([ALL] expression) } OVER inline-window-
specification

inline-window-specification 是:
     [PARTITION BY expression [,expression]...]
     [ORDER BY expression [ASC[ENDING] | DESC[ENDING]]
          [,expression [ASC[ENDING] | DESC[ENDING]]]...]
     [ window-frame-clause  ]

window-frame-clause 是:
     ROWS CURRENT ROW
     | ROWS preceding-row
     | ROWS BETWEEN preceding-row AND preceding-row
     | ROWS BETWEEN preceding-row AND CURRENT ROW
     | ROWS BETWEEN preceding-row AND following-row
     | ROWS BETWEEN CURRENT ROW AND CURRENT ROW
     | ROWS BETWEEN CURRENT ROW AND following-row
     | ROWS BETWEEN following-row AND following-row

preceding-row 是:
     UNBOUNDED PRECEDING
     | unsigned-integer PRECEDING

following-row 是:
     UNBOUNDED FOLLOWING
     | unsigned-integer FOLLOWING
```

ALL

指定计算该表达式的 COUNT 函数时是否将重复值包含在内。默认选项为 ALL，这样会使所有重复值都被包含在内。

expression

指定一个将被计数的数值表达式。更多信息，请参阅表达式。

inline-window-specification

指定一个进行 COUNT 函数计算的窗口。*inline-window-specification* 包含可选的 PARTITION BY 子句、可选的 ORDER BY 子句和可选的 window

frame 子句。 PARTITION BY 子句指定中间结果表的分区方案，而 ORDER BY 子句则指定各分区内的行排序方式。

window-frame-clause
指定分区内进行 COUNT 函数计算的窗口。

7.5.1 COUNT 窗口函数例子

- 返回 SALARY 列的运行计数:
```
SELECT empnum, COUNT(salary)
OVER (ORDER BY empnum ROWS UNBOUNDED PRECEDING)
FROM persnl.employee;
```

- 返回各部门内的 SALARY 列的运行计数:
```
SELECT deptnum, empnum, COUNT(salary)
OVER (PARTITION BY deptnum ORDER BY empnum ROWS
UNBOUNDED PRECEDING)
FROM persnl.employee;
```

- 在最后的 4 行窗口中返回各部门内的工资移动计数:
```
SELECT deptnum, empnum, COUNT(salary)
OVER (PARTITION BY deptnum ORDER BY empnum ROWS 3
PRECEDING)
FROM persnl.employee;
```

- 返回各部门内的雇员运行计数:
```
SELECT deptnum, empnum, COUNT(*)
OVER (PARTITION BY deptnum ORDER BY empnum ROWS UNBOUNDED
PRECEDING)
FROM persnl.employee;
```

7.6 DENSE_RANK 窗口函数

DENSE_RANK 是一种窗口函数，它返回 inline-window-specification 指定的当前分区里的每一行的排名。该排名与 inline-window-specification 指定的排序有关。窗口第一行的 DENSE_RANK 函数的返回值从 1 开始。数值相同的给定表达式具有相同的排名；如果给定表达式发生变化，则 DENSE_RANK 函数的排名将向前移动一位。

```
DENSE_RANK() OVER (inline-window-specification)
```

inline-window-specification 是:

```
     [PARTITION BY expression [,expression]...]
     [ORDER BY expression [ASC[ENDING] | DESC[ENDING]]
          [,expression [ASC[ENDING] | DESC[ENDING]]]...]
```

inline-window-specification

指定一个进行 DENSE_RANK 函数计算的窗口。*inline-window-specification*（内联窗口规范）包含可选的 PARTITION BY 子句和可选的 ORDER BY 子句。PARTITION BY 子句指定中间结果表的分区方案，而 ORDER BY 子句则指定各分区内的行排序方式。

7.6.1 DENSE_RANK 窗口函数例子

- 基于雇员编号返回每位雇员的密集排名:

```
SELECT DENSE_RANK() OVER (ORDER BY empnum), *
FROM persnl.employee;
```

- 基于工资返回各部门内的每位雇员的密集排名:

```
SELECT DENSE_RANK()  OVER (PARTITION BY deptnum ORDER BY
salary), *
FROM persnl.employee;
```

7.7 FIRST_VALUE 窗口函数

FIRST_VALUE 窗口函数返回一个排序分区的第一个值。

```
FIRST_VALUE (expression)
```

expression
指定标量表达式或列，返回它们分区中第一行的值。
注意: EsgynDB 仅支持函数返回包含 null 的第一个值，这是默认行为。

7.7.1 FIRST_VALUE 窗口函数例子

以下查询返回的每一行都包含员工姓名、部门编号、工资和该部门最低工资的工资差额。

```
SELECT ename, deptno, sal,
       sal - FIRST_VALUE( sal ) OVER ( PARTITION BY deptno
   ORDER BY sal ) diff
   FROM emp
ORDER BY deptno, sal;
```

7.8 LAG 窗口函数

LAG 窗口函数返回一个分区中上一行表达式的值。如果不存在上一行，则将返回 NULL。

```
LAG (expression) [, offset] [, default]
     OVER inline-window-specification

inline-window-specification 是:
     [PARTITION BY expression [, expression]...]
     [ORDER BY expression [, expression] ...]
```

expression
指定一个表达式，以确定 MAX 函数计算的数值。更多信息，请参阅表达式。

offset
指定从目前行到前一行的正位移值，该值用于计算表达式。如果未指定，则默认值为 1。

default
当表达式在 *offset* 是 null 时，指定使用默认值。如果未指定默认值，则返回 NULL。

inline-window-specification
指定一个进行 LAG 函数计算的窗口。*inline-window-specification* 包含可选的 PARTITION BY 子句和可选的 ORDER BY 子句。PARTITION BY 子句指定中间结果表的分区方案，而 ORDER BY 子句则指定各分区内的行排序方式。如果没有指定 PARTITION BY 子句，ORDER BY 将对整张表进行排序。

7.8.1 LAG 窗口函数例子

以下查询中的 LAG 函数返回前一个计划的计划代码和第一个计划的' N/A '。

```
SELECT CustomerCode, PlanCode AS CurrentPlanCode,
LAG(PlanCode, 1, 'NA')
OVER (PARTITION BY CustomerCode ORDER BY StartDate ASC)
AS LastPlan
FROM CustomerPlan;
```

7.9 LAST_VALUE 窗口函数

LAST_VALUE 窗口函数与 FIRST_VALUE 函数类似，只是返回排序分区中最后一行的值。

7.10 LEAD 窗口函数

The LEAD 窗口函数返回一个分区中的当前行的下一行数值。如果不存在下一行，则将返回 NULL。

```
LEAD (expression [, offset] [, default])
     OVER inline-window-specification
```

inline-window-specification 是:

```
[PARTITION BY expression [, expression]...]
[ORDER BY expression [, expression] ...]
```

expression

指定一个表达式，用于确定MAX函数计算的数值。更多信息，请参阅表达式。

offset

指定从当前行到前一行的正偏移量，从中获取值以计算表达式。如果未指定，则默认值为 1。

default

当表达式在 offset 是 null 时，指定使用默认值。如果未指定默认值，则返回 NULL。

inline-window-specification

指定一个进行 LEAD 函数计算的窗口。*inline-window-specification* 包含可选的 PARTITION BY 子句和可选的 ORDER BY 子句。PARTITION BY 子句指定中间结果表的分区方案，而 ORDER BY 子句则指定各分区内的行排序方式。如果没有指定 PARTITION BY 子句，ORDER BY 将对整张表进行排序。

7.10.1 LEAD 窗口函数例子

以下查询中的 LEAD 函数返回下一个计划的开始日期，DATEADD 函数从该日期减去一天作为当前计划的结束日期。

```
SELECT *,
DATEADD(DAY, -1, LEAD(StartDate, 1,'01-Jan-2100')
     OVER (PARTITION BY CustomerCode ORDER BY StartDate
     ASC)) AS EndDate FROM CustomerPlan
```

7.11 LISTAGG 窗口函数

```
LISTAGG(measure_expr [, 'delimiter'])
  WITHIN GROUP (order_by_clause)
OVER (query_partition_clause)
```

对于指定的度量，LISTAGG 对 ORDER BY 子句中指定的每个组内的数据进行排序，然后连接度量列的值。

作为分析函数，LISTAGG 根据 query_partition_clause 中的一个或多个表达式将查询结果集划分为组。

measure_expr
可以是任何表达式。 度量列中的空值将被忽略。

delimiter_expr
指定用于分隔度量值的字符串。 此子句是可选的，默认为 NULL。

order_by_clause
确定返回连接值的顺序。 仅当 ORDER BY 列列表实现唯一排序时，该函数才是确定的。

query_partition_clause
使用 PARTITION BY 子句可将查询结果集基于一个或多个表达式的值分组。

7.11.1LISTAGG 窗口函数的例子

- 以下示例显示，对于 2019 年 9 月 1 日之前雇用的每位员工，员工的部门，雇用日期以及该部门中的所有其他员工也在 2019 年 9 月 1 日之前雇用：

```
SELECT department_id "Dept", hire_date "Date", last_name
"Name",
       LISTAGG(last_name, '; ') WITHIN GROUP (ORDER BY
hire_date, last_name)
         OVER   (PARTITION   BY   department_id)   as
"staff_list"
  FROM staff
  WHERE hire_date < '01-SEP-2019'
  ORDER BY "Dept", "Date", "Name";

 Dept Date      Name            staff_list
----- --------- --------------- -----------------------
```

```
----------------------
   30 07-DEC-12 Raphaely        Raphaely; Khoo
   30 18-MAY-13 Khoo            Raphaely; Khoo
   40 07-JUN-12 Mavris          Mavris
   50 01-MAY-13 Kaufling        Kaufling; Ladwig
   50 14-JUL-13 Ladwig          Kaufling; Ladwig
   70 07-JUN-12 Baer            Baer
   90 13-JAN-16 De Haan         De Haan; King
   90 17-JUN-17 King            De Haan; King
  100 16-AUG-17 Faviet          Faviet; Greenberg
  100 17-AUG-18 Greenberg       Faviet; Greenberg
```

7.12 MAX 窗口函数

MAX 是一种窗口函数，它为 inline-window-specification 指定的当前窗口返回给定表达式所有非空值中的最大值，

```
MAX ([ALL] expression MAX ([ALL] expression) OVER (inline-
window-specification)

inline-window-specification 是:
     [PARTITION BY expression [, expression]...]
     [ORDER BY expression [ASC[ENDING] | DESC[ENDING]]
          [,expression [ASC[ENDING] | DESC[ENDING]]]...]
     [ window-frame-clause  ]

window-frame-clause 是:
     ROWS CURRENT ROW
     | ROWS preceding-row
     | ROWS BETWEEN preceding-row AND preceding-row
     | ROWS BETWEEN preceding-row AND CURRENT ROW
     | ROWS BETWEEN preceding-row AND following-row
     | ROWS BETWEEN CURRENT ROW AND CURRENT ROW
     | ROWS BETWEEN CURRENT ROW AND following-row
     | ROWS BETWEEN following-row AND following-row

preceding-row 是:
     UNBOUNDED PRECEDING
     | unsigned-integer PRECEDING

following-row 是:
     UNBOUNDED FOLLOWING
     | unsigned-integer FOLLOWING
```

ALL

指定是否将重复值包含在该表达式的 MAX 函数的计算中。

默认选项为 ALL，这样会使所有重复值都被包含在内。

expression

指定一个表达式，它确定参与 MAX 函数计算的数值。更多信息，请参阅表达式。

inline-window-specification
指定一个进行 MAX 函数计算的窗口。*inline-window-specification* 包含可选的 PARTITION BY 子句、可选的 ORDER BY 子句和可选的 window frame 子句。PARTITION BY 子句指定中间结果表的分区方案，而 ORDER BY 子句指定各分区内的行排序方式。

window-frame-clause
指定分区内进行 MAX 函数计算的窗口。

7.12.1 MAX 窗口函数例子

- 返回 SALARY 列的最大运行值:

```
SELECT empnum, MAX(salary)
OVER (ORDER BY empnum ROWS UNBOUNDED PRECEDING)
FROM persnl.employee;
```

- 返回各部门内的 SALARY 列的最大运行值:

```
SELECT deptnum, empnum, MAX(salary)
OVER (PARTITION BY deptnum ORDER BY empnum ROWS
UNBOUNDED PRECEDING)
FROM persnl.employee;
```

- 在最后 4 行的窗口中返回各部门内的 SALARY 列的最大移动值:

```
SELECT deptnum, empnum, MAX(salary)
OVER (PARTITION BY deptnum ORDER BY empnum ROWS 3
PRECEDING)
FROM persnl.employee;
```

7.13 MIN 窗口函数

MIN 是一种窗口函数，它为 inline-window-specification 指定的当前窗口返回给定表达式所有非空值中的最小值。

```
MIN ([ALL] expression) OVER (inline-window-specification)

inline-window-specification 是:
[PARTITION BY expression [, expression]...]
[ORDER BY expression [ASC[ENDING] | DESC[ENDING]]
[,expression [ASC[ENDING] | DESC[ENDING]]]...]
[ window-frame-clause  ]

window-frame-clause 是:
ROWS CURRENT ROW
| ROWS preceding-row
| ROWS BETWEEN preceding-row AND preceding-row
| ROWS BETWEEN preceding-row AND CURRENT ROW
| ROWS BETWEEN preceding-row AND following-row
| ROWS BETWEEN CURRENT ROW AND CURRENT ROW
| ROWS BETWEEN CURRENT ROW AND following-row
| ROWS BETWEEN following-row AND following-row

preceding-row 是:
UNBOUNDED PRECEDING
| unsigned-integer PRECEDING

following-row 是:
UNBOUNDED FOLLOWING
| unsigned-integer FOLLOWING
```

ALL

指定是否将重复值包含在该表达式的 MIN 函数的计算中。

默认选项为 ALL，这样会使所有重复值都被包含在内。

expression
指定一个表达式，用于确定 MIN 函数计算的数值。更多信息，请参阅表达式。

inline-window-specification
指定一个进行 MIN 函数计算的窗口。*inline-window-specification* 包含可选的 PARTITION BY 子句、可选的 ORDER BY 子句和可选的 window frame 子句。PARTITION BY 子句指定中间结果表的分区方案，而 ORDER BY 子句则指定各分区内的行排序方式。

window-frame-clause
指定分区内进行 MIN 函数计算的窗口。

7.13.1 MIN 窗口函数例子

- 返回 SALARY 列的最小运行值：
```
SELECT empnum, MIN(salary)
OVER (ORDER BY empnum ROWS UNBOUNDED PRECEDING)
FROM persnl.employee;
```

- 返回各部门内的 SALARY 列的最小运行值：
```
SELECT deptnum, empnum, MIN(salary)
OVER (PARTITION BY deptnum ORDER BY empnum ROWS
UNBOUNDED PRECEDING)
FROM persnl.employee;
```

- 在最后 4 行的窗口中返回各部门内的 SALARY 列的最小移动值：
```
SELECT deptnum, empnum, MIN(salary)
OVER (PARTITION BY deptnum ORDER BY empnum ROWS 3
PRECEDING)
FROM persnl.employee;
```

7.14 RANK 窗口函数

RANK 是一种窗口函数，它返回 inline-window-specification 指定的当前分区里的每一行的排名。该排名与 inline-window-specification 指定的排序有关。窗口第一行的 RANK 函数的返回值从 1 开始。数值相同的给定表达式具有相同的排名；当数值发生变化时，RANK 函数值将向前推进至该行在窗口里的相对位置。

```
RANK() OVER (inline-window-specification)
```

inline-window-specification 是：

```
     [PARTITION BY expression [, expression]...]
     [ORDER BY expression [ASC[ENDING] | DESC[ENDING]]
          [,expression [ASC[ENDING] | DESC[ENDING]]]...]
```

inline-window-specification

指定一个进行 RANK 函数计算的窗口。inline-window-specification 包含可选的 PARTITION BY 子句和可选的 ORDER BY 子句。PARTITION BY 子句指定中间结果表的分区方案，而 ORDER BY 子句则指定每个分区内部的行排序方式。

7.14.1 RANK 窗口函数例子

- 基于雇员编号返回每位雇员的排名：

```
SELECT RANK () OVER (ORDER BY empnum), *
FROM persnl.employee;
```

- 基于工资返回各部门内的每位雇员的排名：

```
SELECT RANK() OVER (PARTITION BY deptnum ORDER BY
salary), *
FROM persnl.employee;
```

7.15 ROW_NUMBER 窗口函数

ROW_NUMBER 是一种窗口函数，它返回 inline-window-specification 指定的的当前窗口的每一行的行号。

```
ROW_NUMBER () OVER (inline-window-specification)

inline-window-specification 是:
     [PARTITION BY expression [, expression]...]
     [ORDER BY expression [ASC[ENDING] | DESC[ENDING]]
          [,expression [ASC[ENDING] | DESC[ENDING]]]...]
```

inline-window-specification

指定一个进行 ROW_NUMBER 函数计算的窗口。*inline-window-specification* 包含可选的 PARTITION BY 子句和可选的 ORDER BY 子句。PARTITION BY 子句指定中间结果表的分区方案，而 ORDER BY 子句指定每个分区内部的行排序方式。

window-frame-clause

指定分区内进行 ROW_NUMBER 函数计算的窗口。

7.15.1 ROW_NUMBER 窗口函数例子

- 返回雇员表每一行的行号:

  ```
  SELECT ROW_NUMBER () OVER (ORDER BY empnum), *
  FROM persnl.employee;
  ```

- 返回各部门内的每一行的行号:

  ```
  SELECT ROW_NUMBER () OVER (PARTITION BY deptnum ORDER
  BY empnum), *
  FROM persnl.employee;
  ```

7.16 STDDEV 窗口函数

STDDEV 是一种窗口函数，它为 inline-window-specification 指定的当前窗口返回给定表达式的所有非空值的标准差。

```
STDDEV ([ALL] expression) OVER (inline-window-
specification)

inline-window-specification 是:
     [PARTITION BY expression [, expression]...]
     [ORDER BY expression [ASC[ENDING] | DESC[ENDING]]
          [,expression [ASC[ENDING] | DESC[ENDING]]]...]
     [ window-frame-clause  ]

window-frame-clause 是:
     ROWS CURRENT ROW
     | ROWS preceding-row
     | ROWS BETWEEN preceding-row AND preceding-row
     | ROWS BETWEEN preceding-row AND CURRENT ROW
     | ROWS BETWEEN preceding-row AND following-row
     | ROWS BETWEEN CURRENT ROW AND CURRENT ROW
     | ROWS BETWEEN CURRENT ROW AND following-row
     | ROWS BETWEEN following-row AND following-row

preceding-row 是:
     UNBOUNDED PRECEDING
     | unsigned-integer PRECEDING

following-row 是:
     UNBOUNDED FOLLOWING
     | unsigned-integer FOLLOWING
```

ALL

指定是否将重复值包含在该表达式的 STDDEV 函数的计算中。

默认选项为 ALL，这样会使所有重复值都被包含在内。

expression

指定一个 NUMERIC 型数值或 INTERVAL 型数值表达式，用于确定计算 STDDEV 的值。

inline-window-specification
指定一个计算STDDEV的窗口。*inline-window-specification*包含可选的PARTITION BY子句、可选的ORDER BY子句和可选的window frame子句。PARTITION BY子句指定中间结果表的分区方案，而ORDER BY子句指定各分区内的行排序方式。

window-frame-clause
指定分区内进行STDDEV函数计算的窗口。

7.16.1 STDDEV 函数例子

- 返回雇员表每一行的工资的标准差:

```
SELECT STDDEV(salary) OVER(ORDER BY empnum
                           ROWS UNBOUNDED
                           PRECEDING),*
 FROM persnl.employee;
```

- 返回各部门内的每一行的标准差:

```
SELECT STDDEV() OVER(PARTITION BY deptnum ORDER BY
empnum
ROWS UNBOUNDED PRECEDING), *
FROM persnl.employee;
```

7.17 STDDEV_POP 窗口函数

STDDEV_POP 是一种窗口函数，它为 inline-window-specification 指定的当前窗口返回给定表达式所有非空值的总体标准差。

```
STDDEV_POP ([ALL] expression) OVER (inline-window-
specification)

inline-window-specification is:
     [PARTITION BY expression [, expression]...]
     [ORDER BY expression [ASC[ENDING] | DESC[ENDING]]
     [,expression [ASC[ENDING] | DESC[ENDING]]]...]
     [ window-frame-clause ]

window-frame-clause is:
     ROWS CURRENT ROW
     | ROWS preceding-row
     | ROWS BETWEEN preceding-row AND preceding-row
     | ROWS BETWEEN preceding-row AND CURRENT ROW
     | ROWS BETWEEN preceding-row AND following-row
     | ROWS BETWEEN CURRENT ROW AND CURRENT ROW
     | ROWS BETWEEN CURRENT ROW AND following-row
     | ROWS BETWEEN following-row AND following-row

preceding-row is:
     UNBOUNDED PRECEDING
     | unsigned-integer PRECEDING

following-row is:
     UNBOUNDED FOLLOWING
     | unsigned-integer FOLLOWING
```

ALL

指定计算该表达式的 STDDEV_POP 函数是否将重复值包含在内。

默认选项为 ALL，这样会使所有重复值都被包含在内。

expression
指定一个 NUMERIC 型数值或 INTERVAL 型数值表达式，它确定参与 STDDEV_POP 函数计算的数值。

inline-window-specification
指定一个进行 STDDEV 函数计算的窗口。*inline-window-specification* 包含可选的 PARTITION BY 子句、可选的 ORDER BY 子句和可选的 window frame 子句。 PARTITION BY 子句指定中间结果表的分区方案，而 ORDER BY 子句指定各分区内的行排序方式。

window-frame-clause
指定分区内计算 STDDEV 的窗口。

7.17.1 STDDEV_POP 窗口函数例子

- 返回雇员表每一行的工资的总体标准差:
```
SELECT STDDEV_POP(salary) OVER(ORDER BY empnum
ROWS UNBOUNDED PRECEDING),*  FROM persnl.employee;
```

- 返回各部门内的每一行的总体标准差:
```
SELECT STDDEV_POP() OVER(PARTITION BY deptnum ORDER BY
empnum ROWS UNBOUNDED PRECEDING), * FROM
persnl.employee;
```

7.18 SUM 窗口函数

SUM 是一种窗口函数，它为 inline-window-specification 指定的当前窗口返回给定表达式所有非空值的总和。

```
SUM ([ALL] expression) OVER (inline-window-specification)

inline-window-specification 是:
[PARTITION BY expression [, expression]...]
[ORDER BY expression [ASC[ENDING] | DESC[ENDING]]
[,expression [ASC[ENDING] | DESC[ENDING]]]...]
[ window-frame-clause  ]

window-frame-clause 是:
ROWS CURRENT ROW
| ROWS preceding-row
| ROWS BETWEEN preceding-row AND preceding-row
| ROWS BETWEEN preceding-row AND CURRENT ROW
| ROWS BETWEEN preceding-row AND following-row
| ROWS BETWEEN CURRENT ROW AND CURRENT ROW
| ROWS BETWEEN CURRENT ROW AND following-row
| ROWS BETWEEN following-row AND following-row

preceding-row 是:
UNBOUNDED PRECEDING
| unsigned-integer PRECEDING

following-row 是:
UNBOUNDED FOLLOWING
| unsigned-integer FOLLOWING
```

ALL

指定是否将重复值包含在该表达式的 SUM 函数的计算中。默认选项为 ALL，这样会使所有重复值都被包含在内。

expression
指定一个NUMERIC型数值或INTERVAL型数值表达式，用于确定计算SUM的值。更多信息，请参阅表达式。

inline-window-specification
指定一个进行 SUM 函数计算的窗口。inline-window-specification 包含可选的 PARTITION BY 子句、可选的 ORDER BY 子句和可选的 window frame 子句。PARTITION BY 子句指定中间结果表的分区方案，而 ORDER BY 子句则指定各分区内的行排序方式。

window-frame-clause
指定分区内进行SUM函数计算的窗口。

7.18.1 SUM 窗口函数例子

- 返回SALARY列的动态求和数值:

```
SELECT empnum, SUM (salary)
OVER (ORDER BY empnum ROWS UNBOUNDED PRECEDING)
FROM persnl.employee;
```

- 返回各部门内的SALARY列的动态求和:

```
SELECT deptnum, empnum, SUM (salary)
OVER (PARTITION BY deptnum ORDER BY empnum ROWS
UNBOUNDED PRECEDING)
FROM persnl.employee;
```

- 在最后4行的窗口中返回各部门内的SALARY列的移动求和:

```
SELECT deptnum, empnum, SUM (salary)
OVER (PARTITION BY deptnum ORDER BY empnum ROWS 3
PRECEDING)
FROM persnl.employee;
```

7.19 SYS_CONNECT_BY_PATH 窗口函数

SYS_CONNECT_BY_PATH 仅在分层查询中有效。 它返回从根到节点的列值的路径，其中列值由 CONNECT BY 条件返回的每一行的 char 分隔。

SYS_CONNECT_BY_PATH(column, char)

column 和 char 的类型可以是 CHAR，VARCHAR2，NCHAR 或 NVARCHAR2 中的任何一种。返回的字符串是 VARCHAR2 数据类型，与列的字符集相同。

7.19.1SYS_CONNECT_BY_PATH 函数例子

```
SQL>select  sys_connect_by_path(subid,'>')  from  a_test
start with subid='7' connect by subid = prior parentid;

(EXPR)
------------------------------------------------------------
7
7>10
7>11
7>10>12
7>10>13

--- 5 row(s) selected.
```

7.20 VARIANCE 窗口函数

VARIANCE 是一种窗口函数，它为 inline-window-specification 指定的当前窗口返回给定表达式所有非空值的方差。

```
VARIANCE ([ALL] expression) OVER (inline-window-
specification)

inline-window-specification 是:
[PARTITION BY expression [, expression]...]
[ORDER BY expression [ASC[ENDING] | DESC[ENDING]]
[,expression [ASC[ENDING] | DESC[ENDING]]]...]
[ window-frame-clause  ]

window-frame-clause 是:
ROWS CURRENT ROW
| ROWS preceding-row
| ROWS BETWEEN preceding-row AND preceding-row
| ROWS BETWEEN preceding-row AND CURRENT ROW
| ROWS BETWEEN preceding-row AND following-row
| ROWS BETWEEN CURRENT ROW AND CURRENT ROW
| ROWS BETWEEN CURRENT ROW AND following-row
| ROWS BETWEEN following-row AND following-row

preceding-row 是:
UNBOUNDED PRECEDING
| unsigned-integer PRECEDING

following-row 是:
UNBOUNDED FOLLOWING
| unsigned-integer FOLLOWING
```

ALL

指定计算 VARIANCE 函数时是否将重复值包含在内。默认选项为 ALL，这样会使所有重复值都被包含在内。

expression

指定一个 NUMERIC 型数值或 INTERVAL 型数值表达式，它确定需要进行求方差的数值。更多信息，请参阅表达式。

inline-window-specification
指定一个进行 VARIANCE 函数计算的窗口。*inline-window-specification* 包含可选的 PARTITION BY 子句、可选的 ORDER BY 子句和可选的 window frame 子句。PARTITION BY 子句指定中间结果表的分区方案，而 ORDER BY 子句则指定各分区内的行排序方式。

window-frame-clause
指定分区内进行 VARIANCE 函数计算的窗口。

7.20.1 VARIANCE 窗口函数例子

- 返回 SALARY 列的方差：

```
SELECT empnum, VARIANCE (salary)
OVER (ORDER BY empnum ROWS UNBOUNDED PRECEDING)
FROM persnl.employee;
```

- 返回各部门内的 SALARY 列的方差：

```
SELECT deptnum, empnum, VARIANCE (salary)
OVER (PARTITION BY deptnum ORDER BY empnum ROWS
UNBOUNDED PRECEDING)
FROM persnl.employee;
```

8. SQL 运行时统计信息

运行时管理系统（RMS）显示正在运行中的查询命令的状态。RMS 能处理来自 EsgynDB 命令行界面（TrafCI）的按需请求，以获取一个给定查询 ID 的数据或一个给定进程里的运行查询语句的数据。此外，RMS 还能提供自身的相关信息，以确定 RMS 基础架构的健康状况。

RMS 提供每个碎片实例的汇总数据和一个给定查询语句的每个运算符（TDB_ID）的详细数据。一条查询语句如果正在编译或正在执行将被视为活跃的查询语句。variable_input 列的输出将以多个名/值对（token=value）的形式返回。

更多信息，请参阅获取一个查询语句的每个碎片实例的统计信息时的注意事项。

RMS 已被启用、并且随时可用。

本章目录:

8.1 PERTABLE 和 OPERATOR 统计
8.2 自适应统计集合
8.3 检索 SQL 运行时统计信息
8.4 显示 SQL 运行时统计信息
8.5 使用 QUERYID_EXTRACT 函数
8.6 收集 RMS 的相关统计信息
8.7 一个查询语句的每个碎片实例的统计信息的考虑事项

8.1 PERTABLE 和 OPERATOR 统计

SQL 数据库引擎确定适合一条查询语句的数据集合类型。RMS 基础架构提供一条正在执行中的查询的相关运行时指标。您可以识别那些使用过多资源的查询、暂停一条查询语句，以确定它对资源造成的影响，并在必要时取消一条查询语句。PERTABLE 统计计算行数、并报告磁盘进程里的运算符的估算行数和花费在 ESP 进程里的时间。尽管从磁盘里读出所有行后 PERTABLE 统计也进行推断，但无法正确地评估一条查询语句的当前状态。

复杂的查询诸如联接、排序和分组结果集经常因为容量过大导致无法载入内存，因此中间结果必须溢出到临时文件中，这些运算符被称为大内存运算符（BMO）。由于 BMO 的存在，RMS 提供 OPERATOR 统计，其中包含一组更丰富的统计信息，由此能随时确定一条查询语句的当前状态。

通过使用 OPERATOR 统计，将检测所有 SQL 运算符并收集以下统计信息：

- 运算符里消耗的节点时间
- 流向父运算符的实际行数
- 流向父运算符的（优化器）估算行数
- BMO 里使用的虚拟内存
- 溢出到临时文件并回读到查询中的数据量。更多信息，请参阅显示 SQL 运行时统计信息。

8.2 适应性统计集合

SQL 数据库引擎基于查询类型选择适当的统计集合类型。默认 SQL 数据库引擎统计集合为 OPERATOR 统计。

您查看不同格式的统计信息：PERTABLE、ACCUMULATED、PROGRESS 和 DEFAULT。这种自适应统计集合保证提供充足的统计信息，同时又不会对查询语句的执行性能造成任何影响。对于一些查询而言，既不收集它们的统计信息、也不收集 PERTABLE 统计信息。

查询类型	统计集合类型
OLT 优化查询	PERTABLE
Unique queries	PERTABLE
CQD	没有统计
SET 命令	没有统计
EXPLAIN	没有统计
GET STATISTICS	没有统计

所有其他查询	DEFAULT

8.3 检索 SQL 运行时统计信息

8.3.1 使用 GET STATISTICS 命令

GET STATISTICS 命令显示统计信息:

- 单条查询的 ID（QID）
- 针对进程 ID（PID）的查询语句
- RMS 自身

一条查询语句如果正在编译或正在执行，那就是运行的查询语句。一条 SELECT 查询语句将被执行直到语句或结果集被关闭为止。从逻辑上讲，当编译结束时间为-1、且编译开始时间不为-1 时，或执行结束时间为-1、且执行开始时间不为-1 时，将一条查询视为查询语句。

8.3.2 GET STATISTICS 命令语法

```
GET STATISTICS FOR QID { query-id|CURRENT}[stats-view-
type]}
     |PID { process-name|[nodeid, pid]}[ACTIVE n][stats-
     view-type]
     |RMS node-num|ALL[RESET]
```

stats-view-type 是:

```
     ACCUMULATED|PERTABLE|PROGRESS|DEFAULT
```

QID

请求一条特定查询的统计信息必需的关键字

query-id

这是查询 ID；如果该查询 ID 里的用户名包含小写字母或句号，则必须用双引号将 *query-id* 括起来。

注意： *query-id* 是编译（预处理）一条查询语句时生成的SQL语句的唯一标识符。对于某些TrafCI命令所执行的查询而言，*query-id* 是可见的。

CURRENT

提供最近预处理或执行的语句（与您运行的 GET STATISTICS FOR QID CURRENT 命令在同一个会话里）的统计信息。您必须在 PREPARE 或 EXECUTE 语句后立即发出 GET STATISTICS FOR QID CURRENT 命令。

PID
请求一个给定进程中的查询语句统计信息时必需的关键字。

process-name
这是一个进程 ID 的对应名称，其格式为$Znnn。该进程名既用于主服务器进程（MXOSRVR）、也用于执行服务器进程（ESP）。如果该进程名对应于 ESP，则 ACTIVE n query 将是 ESP 里的第 n 个查询，并且可能并非 ESP 里当前的查询语句。

ACTIVE *n*
描述 RMS 将为之返回统计信息的查询语句。默认值为 ACTIVE 1。
ACTIVE 1 将返回第一个查询语句的统计信息，ACTIVE 2 将返回第二个查询语句的统计信息。

stats-view-type
将统计视图类型设置为不同的格式。默认在运算符层面收集统计信息。
更多信息，请参阅自适应统计集合。

ACCUMULATED
使统计信息都被显示在一条查询中的所有表的汇总摘要里。

PERTABLE
显示查询里每个表的统计信息。这就是默认的 stats-view-type（统计视图类型），尽管所收集的统计信息都来自运算符层面，但如果自适应统计的原因导致收集工作在更低的层面进行，则默认值为更低的（统计信息）收集层面。更多信息，请参阅自适应统计集合。

PROGRESS
除了 Pertable 统计视图类型，还显示一条查询所涉及的每个大内存运算符（BMO）的各行信息。更多信息，请参阅 PERTABLE 和 OPERATOR 统计。

DEFAULT
采用（与收集统计信息）相同的方式显示统计信息。

RMS
请求 RMS 自身的统计信息时必需的关键字

node-num
返回一个给定节点的 RMS 基础架构的相关统计信息。

ALL
返回集群里每个节点的 RMS 基础架构的相关统计信息。

RESET
重置累计 RMS 统计计数器。

8.3.3 GET STATISTICS 命令例子

以下例子显示了各种 GET STATISTICS 命令所返回的运行时统计信息。
有关运行时统计信息和 RMS 计数器的更多信息，请参阅显示 SQL 运行时统计信息。

- 该 GET STATISTICS 命令为同一个会话里的最近执行语句返回 PERTABLE 统计信息：

```
SQL>get statistics for qid current;
Qid
MXID11008018370212168211672476672000000000030000_59_SQL
_CUR_6
Compile Start Time 2011/03/30 07:29:15.332216
Compile End Time 011/03/30 07:29:15.339467
Compile Elapsed Time 0:00:00.007251
Execute Start Time 2011/03/30 07:29:15.383077
Execute End Time 2011/03/30 07:29:15.470222
Execute Elapsed Time 0:00:00.087145
State CLOSE
Rows Affected 0

SQL Error Code 100
Stats Error Code 0
```

```
Query Type SQL_SELECT_NON_UNIQUE
Estimated Accessed Rows 0
Estimated Used Rows 0
Parent Qid NONE
Child Qid NONE
Number of SQL Processes 1
Number of Cpus 1
Execution Priority -1
Transaction Id -1
Source String SELECT
CUR_SERVICE,PLAN,TEXT,CUR_SCHEMA,RULE_NAME,APPL_NAME,S
ESSION_NAME,DSN_NAME,ROLE_NAME,DEFAULT_SCHEMA_ACCESS_O
NLY
FROM(VALUES(CAST('HP_DEFAULT_SERVICE' as
VARCHAR(50)),CAST(0 AS INT),CAST(0 AS
INT),CAST('NEO.USR' as VARCHAR(260)),CAST('' as
VARCHAR(
SQL Source Length 548
Rows Returned 1
First Row Returned Time 2011/03/30 07:29:15.469778
Last Error before AQR 0
Number of AQR retries 0
Delay before AQR 0
No. of times reclaimed 0
Stats Collection Type OPERATOR_STATS
SQL Process Busy Time 0
UDR Process Busy Time 0
SQL Space Allocated 32 KB
SQL Space Used 3 KB
SQL Heap Allocated 7 KB
SQL Heap Used 1 KB
EID Space Allocated 0 KB
EID Space Used 0 KB
EID Heap Allocated 0 KB
EID Heap Used 0 KB
Processes Created 0
Process Create Time 0
Request Message Count 0
Request Message Bytes 0
Reply Message Count 0
Reply Message Bytes 0
Scr. Overflow Mode DISK
Scr File Count 0
Scr. Buffer Blk Size 0
```

```
Scr. Buffer Blks Read 0
Scr. Buffer Blks Written 0
Scr. Read Count 0
Scr. Write Count 0

--- SQL operation complete.
```

- 该 GET STATISTICS 命令返回特定查询 ID 的 PERTABLE 统计信息 （应在同一个会话里运行该命令）:

```
SQL>get statistics for qid
"MXID11008005179212168187528072672000000000030000_48_SQ
L_CUR_2";

Qid
MXID11008005179212168187528072672000000000030000_48_SQL
_CUR_2
Compile Start Time 2011/03/30 00:53:21.382211
Compile End Time 2011/03/30 00:53:22.980201
Compile Elapsed Time 0:00:01.597990
Execute Start Time 2011/03/30 00:53:23.079979
Execute End Time -1
Execute Elapsed Time 7:16:13.494563
State OPEN
Rows Affected -1
SQL Error Code 0
Stats Error Code 0
Query Type SQL_SELECT_NON_UNIQUE
Estimated Accessed Rows 2,487,984
Estimated Used Rows 2,487,984
Parent Qid NONE
Child Qid NONE
Number of SQL Processes 129
Number of Cpus 9
Execution Priority -1
Transaction Id 34359956800
Source String select count(*) from
MANAGEABILITY.INSTANCE_REPOSITORY.EVENTS_TEXT K,
MANAGEABILITY.INSTANCE_REPOSITORY.EVENTS_TEXT J,
MANAGEABILITY.INSTANCE_REPOSITORY.EVENTS_TEXT H,
MANAGEABILITY.INSTANCE_REPOSITORY.EVENTS_TEXT G
SQL Source Length 220
Rows Returned 0
First Row Returned Time -1
```

```
Last Error before AQR 0
Number of AQR retries 0
Delay before AQR 0
No. of times reclaimed 0
Stats Collection Type OPERATOR_STATS
SQL Process Busy Time 830,910,830,000
UDR Process Busy Time 0
SQL Space Allocated 179,049 KB
SQL Space Used 171,746 KB
SQL Heap Allocated 1,140,503 KB
SQL Heap Used 1,138,033 KB

EID Space Allocated 46,080 KB
EID Space Used 42,816 KB
EID Heap Allocated 18,624 KB
EID Heap Used 192 KB
Processes Created 32
Process Create Time 799,702
Request Message Count 202,214
Request Message Bytes 27,091,104
Reply Message Count 197,563
Reply Message Bytes 1,008,451,688
Scr. Overflow Mode DISK
Scr File Count 0
Scr. Buffer Blk Size 0
Scr. Buffer Blks Read 0
Scr. Buffer Blks Written 0
Scr. Read Count 0
Scr. Write Count 0
Table Name
Records Accessed Records Used Disk Process Open Open
Estimated/Actual Estimated/Actual Busy Time Count Time
Disk I/Os
Message Count
Message Bytes
Lock Escl
Lock wait
MANAGEABILITY.INSTANCE_REPOSITORY.EVENTS_TEXT(H)
621,996 621,996
621,998 621,998 0 441 10,666,384 0 0
303,955 32 15,967
MANAGEABILITY.INSTANCE_REPOSITORY.EVENTS_TEXT(J)
621,996 621,996
621,998 621,998 0 439 10,666,384 0 0
```

```
289,949 32 19,680
MANAGEABILITY.INSTANCE_REPOSITORY.EVENTS_TEXT(K)
621,996 621,996
621,998 621,998 0 439 10,666,384 0 0
301,956 32 14,419
MANAGEABILITY.INSTANCE_REPOSITORY.EVENTS_TEXT(G)
0 621,996
0 0 0 192 4,548,048 0 0
0 32 40,019

--- SQL operation complete.
```

- 该 GET STATISTICS 命令返回同一个会话里的最近执行语句的 ACCUMULATED（累计）统计信息:

```
SQL>get statistics for qid current accumulated;

Qid
MXID11008025173212168212775343040000000000340000_957_SQ
L_CUR_6
Compile Start Time 2011/03/30 08:05:07.646667
Compile End Time 2011/03/30 08:05:07.647622
Compile Elapsed Time 0:00:00.000955
Execute Start Time 2011/03/30 08:05:07.652710
Execute End Time 2011/03/30 08:05:07.740461
Execute Elapsed Time 0:00:00.087751
State CLOSE
Rows Affected 0
SQL Error Code 100
Stats Error Code 0
Query Type SQL_SELECT_NON_UNIQUE
Estimated Accessed Rows 0
Estimated Used Rows 0
Parent Qid NONE
Child Qid NONE
Number of SQL Processes 0
Number of Cpus 0
Execution Priority -1
Transaction Id -1
Source String SELECT
CUR_SERVICE,PLAN,TEXT,CUR_SCHEMA,RULE_NAME,APPL_NAME,S
ESSION_NAME,DSN_NAME,ROLE_NAME,DEFAULT_SCHEMA_ACCESS_O
NLY
FROM(VALUES(CAST('HP_DEFAULT_SERVICE' as
```

```
VARCHAR(50)),CAST(0 AS INT),CAST(0 AS
INT),CAST('NEO.SCH' as VARCHAR(260)),CAST('' as
VARCHAR(
SQL Source Length 548
Rows Returned 1
First Row Returned Time 2011/03/30 08:05:07.739827
Last Error before AQR 0
Number of AQR retries 0
Delay before AQR 0
No. of times reclaimed 0
Stats Collection Type OPERATOR_STATS
Accessed Rows 0
Used Rows 0
Message Count 0
Message Bytes 0
Stats Bytes 0
Disk IOs 0
Lock Waits 0
Lock Escalations 0
Disk Process Busy Time 0
SQL Process Busy Time 0
UDR Process Busy Time 0
SQL Space Allocated 32 KB
SQL Space Used 3 KB

SQL Heap Allocated 7 KB
SQL Heap Used 1 KB
EID Space Allocated 0 KB
EID Space Used 0 KB
EID Heap Allocated 0 KB
EID Heap Used 0 KB
Opens 0
Open Time 0
Processes Created 0
Process Create Time 0
Request Message Count 0
Request Message Bytes 0
Reply Message Count 0
Reply Message Bytes 0
Scr. Overflow Mode UNKNOWN
Scr. File Count 0
Scr. Buffer Blk Size 0
Scr. Buffer Blks Read 0
Scr. Buffer Blks Written 0
```

```
Scr. Read Count 0
Scr. Write Count 0

--- SQL operation complete.
```

- 这些 GET STATISTICS 命令返回指定的进程 ID 里的第一个查询语句的 PERTABLE 统计信息：

```
SQL>get statistics for pid 0,27195;
SQL>get statistics for pid $Z000F3R;
```

8.4 显示 SQL 运行时统计信息

GET STATISTICS 语句默认逐表显示统计信息（PERTABLE）。如果需要查看不同格式的统计信息，请使用 GET STATISTICS 命令中的适当视图选项。

RMS 既为预处理语句提供缩略的统计信息，也为执行语句提供完整的运行时统计信息。

该表显示 GET STATISTICS 语句所返回的 RMS 计数器、与 RMS 计数器相关的 STATISTICS 表值函数里的令牌、以及计数器和令牌的描述。

SQL 运行时统计的 RMS 计数器信息

计数器名称	STATISTICS 表值函数里的令牌	描述
Qid	Qid	为每个查询生成唯一的 ID。每次预处理一条SQL语句时都会生成一个新的查询ID。
编译开始时间	CompStartTime	查询编译开始时间或查询 PREPARE 开始时间。
编译结束时间	CompEndTime	查询编译结束时间或查询 PREPARE 结束时间。
编译需要时间	CompElapsedTime	查询预处理所需的实际时间量。
执行开始时间	ExeStartTime	查询执行开始时间。
执行结束时间	ExeEndTime	查询执行结束时间。对于一个正在执行的

		查询而言，执行结束时间为－1。
执行需要时间	ExeElapsedTime	SQL 执行器执行一条查询所用的实际时间量。
State	State	内部使用。
影响行数	RowsAffected	表示 INSERT、UPDATE 或 DELETE（IUD） SQL 语句的影响行数。对于 SELECT 语句和非 IUD SQL 语句而言，该值为–1。
SQL错误代码	SQLErrorCode	该查询返回的顶级错误代码；指示一个查询是否伴有警告完成、伴有错误完成或成功完成。正数表示警告，负数则表示错误。返回值可能不会精确到 GETSTATISTICS 命令的执行点。
统计收集器的错误代码	StatsErrorCode	从 RMS 获取统计信息时返回给统计收集器的错误代码。 如果出现了错误代码，则说明计数器数值可能有误。这时请重新发出 GET STATISTICS 命令。
查询类型	QueryType	• DML 语句的类型和枚举值: • SQL_SELECT_UNIQUE=1 • SQL_SELECT_NON_UNIQUE=2 • SQL_INSERT_UNIQUE=3 • SQL_INSERT_NON_UNIQUE=4 • SQL_UPDATE_UNIQUE=5 • SQL_UPDATE_NON_UNIQUE=6 • SQL_DELETE_UNIQUE=7 • SQL_DELETE_NON_UNIQUE=8 • SQL_CONTROL=9 • SQL_SET_TRANSACTION=10 • SQL_SET_ CAT ALOG=11 • SQL_SET_SCHEMA=12 • SQL_CALL_NO_RESULT_SETS=13 • SQL_CALL_WITH_RESULT_SETS=14 • SQL_SP_RESULT_SET=15 • SQL_INSERT_ROWSET_SIDETREE=16 • SQL_CAT_UTIL=17

		• SQL_EXE_UTIL=18 • SQL_OTHER=1 • SQL_UNKNOWN=0
估算访问行数	EstRowsAccessed	编译器估算的运算符里的执行器的访问行数。
估算应用行数	Estimated Used Rows	编译器估算应用谓词后运算符里的执行器返回的行数。
父查询ID	parent Qid	父查询的唯一 ID。如果不存在与该查询相关联的父查询 ID，则 RMS 将返回 NONE。更多信息，请参阅使用父查询 ID。
子查询ID	child Qid	子查询的唯一 ID。如果不存在子查询，则 RMS 将返回 NONE。更多信息，请参阅子查询。
SQL 进程数量	numSqlProcs	表示执行查询时所涉及的 SQL 进程数量。
CPU数量	numCpus	表示 SQL 处理查询时所用的节点数量。
事务ID	transId	表示执行查询时所涉及的事务 ID。如果不存在事务，则事务 ID 将为-1。
源字符串	sqlSrc	包含源字符串的前 254 个字节。
SQL源字符长度	sqlSrcLen	SQL 源字符串的实际长度。
返回行数	rowsReturned	表示主执行进程中根运算符返回的行数。
返回第一行时的实际时间	firstRowReturnedTime	表示主要的根运算符返回第一行时的实际时间。
触发自动查询重试（AQR）的错误代码	LastErrorBeforeAQR	一个触发自动查询重试（AQR）的错误代码。如果该值不为 0，则为触发最近 AQR 的错误代码。
重试次数	AQRNumRetries	当前查询到现在为止的重试次数。
发起 AQR 前的等待延时	Delay before AQR	SQL 发起 AQR 前的等待延时。

查询空间的回收次数。	reclaimSpaceCnt	如果一个进程正面临虚拟内存的压力，很早之前执行的查询所占用的执行空间将被回收，以便为即将到来的查询释放空间。该计数器表示特定的查询空间的回收次数。
	statsRowType	• statsRowType 是以下类型之一: • SQLSTATS_DESC_OPER_STATS=0 • SQLSTATS_DESC_ROOT_OPER_STATS=1 • SQLSTATS_DESC_PERTABLE_STATS=11 • SQLSTATS_DESC_UDR_STATS=13 • SQLSTATS_DESC_MASTER_STATS=15 • SQLSTATS_DESC_RMS_STATS=16 • SQLSTATS_DESC_BMO_STATS=17
集合类型	StatsType	集合类型默认为 OPERATOR_STATS by StatsType 是以下类型之一: • SQLCLI_NO_STATS=0 • SQLCLI_ACCUMULATED_STATS=2 • SQLCLI_PERTABLE_STATS=3 • SQLCLI_OPERATOR_STATS=5
访问的实际行数	AccessedRows	运算符里的执行器所访问的实际行数。
SQL 处理时间	CpuTime	花费在该查询所涉及的主站和 ESP 里的总节点时间的近似值（以微秒为单位）。
UDR服务器进程时间（等同于 UDR CPU Time）	udrCPUTime	花费在 UDR 服务器进程里的总节点时间的近似值（以微秒为单位）。
UDR 服务器ID	UDRServerId	MXUDR 进程 ID。
请求的时间戳		最近发送给 MXUDR 的请求的实际时间戳。

最新回复时间戳		MXUDR 最近收到的请求的实际时间戳。
分配 SQL“空间”型内存	SpaceTotal1	该查询所涉及的主站和 ESP 里所分配的空间”型内存，以 KB 为单位。
所用的 SQL”空间”型内存量	SpaceUsed1	该查询所涉及的主站和 ESP 里所用的“空间”型内存量，以 KB 为单位。
分配的 SQL“堆”型内存量	HeapTotal2	该查询所涉及的主站和 ESP 里所分配的“堆”型内存量，以 KB 为单位。
使用 SQL“堆”型内存量	HeapUsed2	该查询所涉及的主站和 ESP 里所用的“堆”型内存量，以 KB 为单位。
分配 EID“空间”类型的内存量	Dp2SpaceTotal	查询中涉及的在 TSEs 中的执行程序中分配的以 KB 为单位的“空间”类型的内存量。
使用 EID“空间”类型的内存量	Dp2SpaceUsed	查询中涉及的在 TSEs 中的执行程序中分配的以 KB 为单位的“空间”类型的内存量。
分配 EID“堆”类型的内存量	Dp2HeapTotal	查询中涉及的在 TSEs 中的执行程序中分配的以 KB 为单位的“堆”类型的内存量。
Open 调用次数	Opens	SQL 执行器代表该语句执行的 OPEN 调用次数。
Open 操作花费时间	OpenTime	该进程代表该语句进行 OPEN 操作所花费的时间（以微秒为单位）。
创建进程数量	Newprocess	主执行器为该语句创建的进程数量（ESP 和 MXCMP）。
进程创建时间	NewprocessTime	创建这些进程所耗用的时间。
表名	AnsiName	查询里的一个表名。
发出消息数量	reqMsgCnt	从主站向 ESP 发出的消息的数量或从 ESP 向 ESP 发出的消息数量。
发送消息字节的数量	regMsgBytes	从主站向 ESP 发送的消息字节的数量或从 ESP 向 ESP 发送的消息数量，作为请求消息的一部分。
回复消息的数量	replyMsgCnt	ESP 针对请求而回复的消息的数量。

回复消息字节数量	replyMsgBytes	作为回复消息的一部分的发送字节的数量。
溢出模式	scrOverFlowMode	表示 Scratch 溢出模式。模式为 DISK_TYPE 或 SSD_TYPE。
用于执行查询的临时文件数量	scrFileCount	创建用于执行查询的临时文件数量。默认文件大小为 2GB。
读取的临时缓冲区块的数量	scrBufferBlockSize	用于读/写临时文件的缓冲区块的大小。
读取的临时缓冲区块的数量	scrBufferRead	从临时文件中读取的临时缓冲区块的数量。
写入临时文件中的临时缓冲区块的数量	scrBufferWritten	写入临时文件中的临时缓冲区块的数量。通过将 Scr. Buffer Blk Size 与该计数器相乘，得到临时文件的准确大小。
读取文件系统调用次数	scrReadCount	从临时文件里读取缓存块时所涉及的文件系统调用次数。每次调用同时读取多个缓存块。
写入文件系统调用次数	scrWrite Count	将缓存块写入临时文件时所涉及的文件系统调用次数。一次调用同时写入多个缓存块。
BMO 所用的“堆”型内存量	bmoHeapUsed	BMO 运算符里所用的“堆”型内存量。BMO 运算符包括 HASH_JOIN（所有种类的 HASH_JOIN)、HASH_GROUPBY（所有种类的 HASH_GROUPBY）和 SORT（所有种类的 SORT)。
BMO“堆”型内存量	bmoHeapTotal	BMO 运算符里分配的“堆”型内存量。
BMO最大内存量	bmoHeapWM	BMO 运算符里所用的最大内存量。
BMO 空间缓冲区的大小	bmoSpaceBufferSize	针对内存类型分配的空间缓冲区的大小，以 KB 为单位。
BMO 空间缓冲	bmoSpaceBufferCount	针对内存类型分配的空间缓冲区计数。

区计数		
执行器所访问的近似实际行数		运算符里的执行器所访问的实际行数。
ID		执行查询时的运算符的TDB ID。
LCID		左子运算符 ID。
RCID		右子运算符 ID。
PaID		父运算符 ID（TDB-ID）。
ExID		解释计划运算符 ID。
Frag		该运算符所属的碎片 ID。
Dispatches	OperCpuTime	SQL 执行器里安排的运算次数。
Oper CPU Time		运算符执行查询所花费的节点时间。
元组数量的近似值		流向父运算符的元组数量的近似值。
实际元组数量		流向父运算符的实际元组数量。
	ProcessId	进程 ID 的对应名称；格式为$Znnn； 进程名既用于主服务器进程（MXOSRVR）、也用于执行服务器进程（ESP）。

空间是执行器拥有的内存池所分配的内存。释放语句所用内存空间之前，请求内存的执行器运算符预计不会返回内存。
堆内存用于临时分配。这些运算符在释放语句所用内存空间之前返回堆内存。由此在必要时重复使用该内存。

8.4.1 显示 SQL 运行时统计信息的例子

注意：为了可读性，部分输出结果格式被重新调整。

一条预处理语句的统计信息

该例子显示当前预处理语句的输出：

```
SQL>get statistics for qid current;
```

```
Qid
MXID11000006497212158373059979520000000001930000_4200_Q1
Compile Start Time 2010/12/06 10:55:40.931000 Compile End
Time 2010/12/06 10:55:42.131845 Compile Elapsed Time
0:00:01.200845 Execute Start Time -1
Execute End Time -1
Execute Elapsed Time 0:00:00.000000 State CLOSE
Rows Affected -1
SQL Error Code 0
Stats Error Code 0
Query Type SQL_SELECT_NON_UNIQUE Estimated Accessed Rows
100,010
Estimated Used Rows 100,010 Parent Qid NONE
Child Qid NONE
Number of SQL Processes 0
Number of Cpus 0
Execution Priority -1
Transaction Id -1
Source String select * from t100k where b in (select b
from t10) SQL Source Length 50
Rows Returned 0
First Row Returned Time -1 Last Error before AQR 0
Number of AQR retries 0
Delay before AQR 0
No. of times reclaimed 0
Stats Collection Type OPERATOR_STATS

   --- SQL operation complete.
```

8.4.2 一条正在执行中的语句的 PERTABLE 统计信息

该例子显示一条正在执行的语句的 PERTABLE 统计信息:

```
SQL>get statistics for qid current;


Qid
MXID11000006497212158373059979520000000001930000_4200_Q1
Compile Start Time 2010/12/06 10:55:40.931000 Compile End
Time 2010/12/06 10:55:42.131845 Compile Elapsed Time
0:00:01.200845 Execute Start Time 2010/12/06
10:56:16.254686 Execute End Time 2010/12/06
10:56:18.434873 Execute Elapsed Time 0:00:02.180187 State
CLOSE
Rows Affected 0
```

```
SQL Error Code 100
Stats Error Code 0
Query Type SQL_SELECT_NON_UNIQUE Estimated Accessed Rows
100,010
Estimated Used Rows 100,010 Parent Qid NONE
Child Qid NONE
Number of SQL Processes 7
Number of Cpus 1
Execution Priority -1
Transaction Id 18121
Source String select * from t100k where b in (select b from
t10) SQL Source Length 50
Rows Returned 100
First Row Returned Time 2010/12/06 10:56:18.150977 Last
Error before AQR 0
Number of AQR retries 0
Delay before AQR 0
No. of times reclaimed 0
Stats Collection Type OPERATOR_STATS SQL Process Busy Time
600,000
UDR Process Busy Time 0
SQL Space Allocated 1,576 KB
SQL Space Used 1,450 KB
SQL Heap Allocated 199 KB
SQL Heap Used 30 KB
EID Space Allocated 704 KB
EID Space Used 549 KB
EID Heap Allocated 582 KB
EID Heap Used 6 KB
Processes Created 4

Process Create Time 750,762 Request Message Count 701
Request Message Bytes 135,088 Reply Message Count 667
Reply Message Bytes 3,427,664 Scr. Overflow Mode DISK

Scr File Count 0
Scr. Buffer Blk Size 0
Scr. Buffer Blks Read 0
Scr. Buffer Blks Written 0 Table Name

Table Name
  Records Accessed   Records Used    Disk   Message
Message    Lock     Lock
Disk Process Open  Open
  Estimated/Actual Estimated/Actual  I/Os   Count
```

```
Bytes      Escl     wait
Busy Time    Count Time
NEO.SCTEST.T10
               10             10
               10             10       0         2
5,280       0       0
2,000 32 15,967
NEO.SCTEST.T100K
           100,000       100,000
           100,000       100,000       0       110
3,235,720      0       0
351,941        4 48,747
--- SQL operation complete.
```

8.4.3 一条正在执行中的语句的 ACCUMULATED 统计信息

该例子显示一条正在执行的语句的ACCUMULATED（累计）统计信息:

```
SQL>get statistics for qid current accumulated;

Qid
MXID11000006497212158373059979520000000001930000_4200_Q1
Compile Start Time 2010/12/06 10:55:40.931000 Compile End
Time 2010/12/06 10:55:42.131845 Compile Elapsed Time
0:00:01.200845 Execute Start Time 2010/12/06
10:56:16.254686 Execute End Time 2010/12/06
10:56:18.434873 Execute Elapsed Time 0:00:02.180187 State
CLOSE
Rows Affected 0
SQL Error Code 100
Stats Error Code 0
Query Type SQL_SELECT_NON_UNIQUE Estimated Accessed Rows
100,010
Estimated Used Rows 100,010 Parent Qid NONE
Child Qid NONE
Number of SQL Processes 7
Number of Cpus 1
Execution Priority -1
Transaction Id 18121
Source String select * from t100k where b in (select b
from t10) SQL Source Length 50
Rows Returned 100
First Row Returned Time 2010/12/06 10:56:18.150977 Last
Error before AQR 0
Number of AQR retries 0
Delay before AQR 0
No. of times reclaimed 0
```

```
Stats Collection Type OPERATOR_STATS Accessed Rows
100,010
Used Rows 100,010
Message Count 112
Message Bytes 3,241,000
Stats Bytes 2,904
Disk IOs 0
Lock Waits 0
Lock Escalations 0
Disk Process Busy Time 353,941 SQL Process Busy Time
600,000 UDR Process Busy Time 0
SQL Space Allocated 1,576 KB
SQL Space Used 1,450 KB
SQL Heap Allocated 199 KB
SQL Heap Used 30 KB
EID Space Allocated 704 KB
EID Space Used 549 KB
EID Heap Allocated 582 KB
EID Heap Used 6 KB
Opens 4
Open Time 48,747
Processes Created 4

Process Create Time 750,762 Request Message Count 701
Request Message Bytes 135,088 Reply Message Count 667
Reply Message Bytes 3,427,664 Scr. Overflow Mode DISK
Scr. File Count 0
Scr. Buffer Blk Size 0
Scr. Buffer Blks Read 0
Scr. Buffer Blks Written 0

--- SQL operation complete.
```

8.4.4 一条正在执行中的语句的 PROGRESS 统计信息

该例子显示一条正在执行的语句的 PROGRESS 统计信息:

```
SQL>get statistics for qid current PROGRESS;
Qid
MXID11000006497212158373059979520000000001930000_4200_Q1
Compile Start Time 2010/12/06 10:55:40.931000
Compile End Time   2010/12/06 10:55:42.131845
Compile Elapsed Time          0:00:01.200845
Execute Start Time 2010/12/06 10:56:16.254686
Execute End Time   2010/12/06 10:56:18.434873
Execute Elapsed Time          0:00:02.180187
State     CLOSE
```

```
Rows Affected  0
SQL Error Code 100
Stats Error Code         0
Query Type     SQL_SELECT_NON_UNIQUE
Estimated Accessed Rows  100,010
Estimated Used Rows 100,010
Parent Qid     NONE
Child Qid NONE
Number of SQL Processes  7
Number of Cpus 1
Execution Priority       -1
Transaction Id 18121
Source String  select * from t100k where b in (select b
from t10)
SQL Source Length   50
Rows Returned  100
First Row Returned Time 2010/12/06 10:56:18.150977
Last Error before AQR    0
Number of AQR retries    0
Delay before AQR         0
No. of times reclaimed   0
Stats Collection Type    OPERATOR_STATS
SQL Process Busy Time    600,000
SQL Space Allocated      1,576     KB
SQL Space Used 1,450     KB
SQL Heap Allocated       199  KB
SQL Heap Used  30   KB
EID Space Allocated      704  KB
EID Space Used 549  KB
EID Heap Allocated       582  KB
EID Heap Used  6    KB
Processes Created        4
Process Create Time 750,762
Request Message Count    701
Request Message Bytes    135,088
Reply Message Count 667
Reply Message Bytes      3,427,664

Table Name
   Records Accessed Records Used   Disk Message Message
Lock Lock
Disk Process   Open   Open
   Estimated/Actual   Estimated/Actual  I/Os Count
     Bytes      Escl      wait Busy Time       Count
Time
NEO.SCTEST.T10
                10                 10
                10                 10      0    2 5,280
         0         0
2,000     0    0  0
NEO.SCTEST.T100K
```

```
          100,000             100,000
          100,000               100,000    0   110
     3,235,720 0    0
351,941     4  48,747

Id TDB Mode Phase        Phase  BMO   BMO  BMO   BMO   BMO
File  Scratch
Buffer Cpu
   Name  Phase         Start  Heap  Heap Heap Space Spacez
Count
Size/Read/Written Time
                       Time   Used  Total WM  BufSz  BufCnt
16 EX_HASHJ DISK        0      0     56    0   0       -1
0     0
60,000
```

8.4.5 一条正在执行中的语句的 DEFAULT 统计信息

该例子显示一条正在执行的语句的 DEFAULT（默认）统计信息:

```
SQL>get statistics for qid current DEFAULT;

Qid
MXID11000006497212158373059979520000000001930000_4200_Q1
Compile Start Time 2010/12/06 10:55:40.931000 Compile End
Time  2010/12/06  10:55:42.131845  Compile  Elapsed  Time
0:00:01.200845    Execute    Start    Time    2010/12/06
10:56:16.254686    Execute    End    Time    2010/12/06
10:56:18.434873 Execute Elapsed Time 0:00:02.180187 State
CLOSE
Rows Affected 0
SQL Error Code 100
Stats Error Code 0
Query Type SQL_SELECT_NON_UNIQUE Estimated Accessed Rows
100,010
Estimated Used Rows 100,010 Parent Qid NONE
Child Qid NONE
Number of SQL Processes 7
Number of Cpus 1
Execution Priority -1
Transaction Id 18121
Source String select * from t100k where b in (select b from
t10) SQL Source Length 50
Rows Returned 100
First Row Returned Time 2010/12/06 10:56:18.150977 Last
Error before AQR 0
Number of AQR retries 0
Delay before AQR 0
No. of times reclaimed 0
```

```
Stats Collection Type OPERATOR_STATS

Id LCId RCId PaId ExId Frag TDB Name Dispatches Oper CPU   Records
Records
                                                 Time   Est.   Used
Act.
Used Details
21 20    .    .   10   0    EX_ROOT          15        0          0
100
20 19    .    21   9   0    EX_SPLIT_TOP     13        0        100
100
19 18    .    20   9   0    EX_SEND_TOP      20        0        100
100
18 17    .    19   9   2    EX_SEND_BOTTOM   72        0        100
100
17 16    .    18   9   2    EX_SPLIT_BOTTOM  88        0        100
100
16 15    .    17   8   2    EX_HASHJ      1,314   60,000        100
15 14    .    16   7   2    EX_SPLIT_TOP  1,343   20,000    100,000
100,000
14 13    .    15   7   2    EX_SEND_TOP   1,342  120,000    100,000
100,000
13 12    .    14   7   5    EX_SEND_BOTTOM 1,534 200,000    100,000
100,000
12 11    .    13   7   5    EX_SPLIT_BOTTOM 493   70,000    100,000
100,000
11 10    .    12   6   5    EX_SPLIT_TOP    486   70,000    100,000
100,000
10  9    .    11   5   5    EX_PARTN_ACCESS 1,634 60,000    100,000
0
9   8    .    10   5   6    EX_EID_ROOT      12        0    100,000
100,000
8   7    .     9   4   6    EX_DP2_SUBS_OPER 160 170,000    100,000
10
7   6    .     8   3   2    EX_SPLIT_TOP      16       0         10
10
6   5    .     7   3   2    EX_SEND_TOP       17       0         10
10
5   4    .     6   3   3    EX_SEND_BOTTOM    17       0         10
10
4   3    .     5   3   3    EX_SPLIT_BOTTOM    9       0         10
10
3 2 . 4 2 3 EX_PARTN_ACCESS 6 0 10
10
2 1 . 3 2 4 EX_EID_ROOT 3 0 10
0
1 . . 1 1 4 EX_DP2_SUBS_OPER 3 100,000 10
10

--- SQL operation complete.
```

8.4.6 使用父查询 ID

执行某些 SQL 语句时，它们会附带执行其他 SQL 语句，由此产生父子关系。例如：执行 UPDATE STATISTICS、MAINTAIN 和 CALL 语句时，它们会执行其

他被称作子查询的 SQL 语句。这些子查询甚至可能执行更多的子查询，因此引入一种具有父子关系的 SQL 语句层次结构。“父查询 ID”将子查询映射到直系父级 SQL 语句，帮助您将一条子级 SQL 语句追溯到用户发出的 SQL 语句。

在运行时统计输出里，父查询 ID 可被用作一种指示符 Parent Qid。用户直接发出的查询不具有父查询 ID，计数器将显示“None.”。

8.4.7 子查询 ID

在很多情况下，一个子查询将与其父查询在同一个节点下执行。此时，父查询 ID 的 GET STATISTICS 语句报告将包含最近执行的子查询的 ID 值。相反，如果不存在任何子查询或子查询在不同的节点下执行，则将不会报告子查询 ID。

以下例子显示了父查询和一个子查询的 GET STATISTICS 语句输出，两个查询都在用户发出 CREATE TABLE AS 命令后被执行:

```
SQL> -- get statistics for the parent query

SQL>get statistics for qid
+>MXID01001091200212164828759544076000000000217DEFAULT_MX
CI_USER00_34 SQLCI_DML_LAST
+>;
Qid
MXID11001091200212164828759544076000000000217DEFAULT_MXCI
_USER00_34 SQLCI_DML_LAST
Compile Start Time 2011/02/18 14:49:04.606513 Compile End
Time 2011/02/18 14:49:04.631802 Compile Elapsed Time
0:00:00.025289 Execute Start Time 2011/02/18
14:49:04.632142 Execute End Time -1
Execute Elapsed Time 0:03:29.473604 State CLOSE
Rows Affected -1
SQL Error Code 0
Stats Error Code 0
Query Type SQL_INSERT_NON_UNIQUE Estimated Accessed Rows
0
Estimated Used Rows 0
Parent Qid NONE
Child Qid
MXID11001091200212164828759544076000000000217DEFAULT_MXCI
_USER00_37_86
```

```
Number of SQL Processes 1
Number of Cpus 1
Execution Priority 148
Transaction Id -1
Source String create table odetail hash partition by
(ordernum, partnum) as select * from SALES.ODETAIL;
SQL Source Length 91
Rows Returned 0
First Row Returned Time -1 Last Error before AQR 0
Number of AQR retries 0
Delay before AQR 0
No. of times reclaimed 0
Stats Collection Type OPERATOR_STATS
Id LCId RCId PaId ExId Frag TDB Name Dispatches Oper CPU
Time Est. Records Used Act. Records Used Details
2 1 . . 2 0 EX_ROOT 0 0 0
0
1 . . 2 1 0 CREATE_TABLE_AS 0 0 0
0
--- SQL operation complete.

SQL> -- get statistics for the child query SQL>get
statistics for qid
+>MXID11001091200212164828759544076000000000217DEFAULT_MX
CI_USER00_37_86
+>;
Qid
MXID01001091200212164828759544076000000000217DEFAULT_MXCI
_USER00_37_86
Compile Start Time 2011/02/18 14:49:07.632898 Compile End
Time 2011/02/18 14:49:07.987334 Compile Elapsed Time
0:00:00.354436 Execute Start Time 2011/02/18
14:49:07.987539 Execute End Time -1
Execute Elapsed Time 0:02:33.173486 State OPEN
Rows Affected -1
SQL Error Code 0
Stats Error Code 0
Query Type SQL_INSERT_NON_UNIQUE Estimated Accessed Rows
101
Estimated Used Rows 101
Parent Qid
MXID10100109120021216482875954407600000000217DEFAULT_MXC
I_USER00_34 SQLCI_DML_LAST
Child Qid NONE
```

```
Number of SQL Processes 1
Number of Cpus 1
Execution Priority 148
Transaction Id \ARC0101(2).9.9114503
Source String insert using sideinserts into CAT.SCH.ODETAIL
select * from SALES.ODETAIL;

SQL Source Length 75
Rows Returned 0
First Row Returned Time -1
Last Error before AQR 0
Number of AQR retries 0
Delay before AQR 0
No. of times reclaimed 0
Stats Collection Type OPERATOR_STATS
Id LCId RCId PaId ExId Frag TDB Name Dispatches Oper CPU
Time Est. Records Used Act. Records Used Details
4 3 . 9 3 0 EX_SPLIT_TOP 1 10,062 100
0
3 2 . 4 2 0 EX_PARTN_ACCESS 66 9,649 100
0
--- SQL operation complete.
```

8.5 收集 RMS 的相关统计信息

使用 GET STATISTICS FOR RMS 命令获取 RMS 自身的相关信息。GET STATISTICS FOR RMS 语句可用于检索一个节点或所有节点的相关信息。为每个节点都提供单个报告。

计数器	描述
CPU	EsgynDB 集群的节点号
RMS Version	RMS 的内部版本
SSCP PID	SQL 统计控制进程 ID.
SSCP Creation Timestamp	创建 SQL 统计控制进程时的实际时间戳
SSMP PID	SQL 统计合并进程 ID
SSMP Creation Timestamp	创建 SQL 统计合并时的时间戳
Source String Store Len	源字符串的存储长度
Stats Heap Allocated	执行在此时的 RMS 共享内存段里的给定节点中的所有查询所分配的内存量
Stats Heap Used	执行在此时的 RMS 共享内存段里的给

	定节点中执行的所有查询所用的内存量
Stats Heap High WM	到现在为止 RMS 共享内存段的给定节点里执行的所有查询所用的最高的内存量
No. of Process Regd.	注册在共享内存段里的进程数量
No. of Query Fragments Regd.	注册在共享内存段里的查询碎片数量
RMS Semaphore Owner	此时锁定信号的进程 ID
No. of SSCPs Opened	已打开的统计控制进程数量。正常情况下应等于 EsgynDB 集群的节点数量
No. of SSCPs Opene Deleted	通信中断的统计控制进程。通常情况下，该值为 0
Last GC Time	共享内存段进行垃圾回收最近时间戳
Queries GCed in Last Run	最近运行 GC（垃圾收集器）时被作为垃圾回收的查询的数量
Total Queries GCed	自统计重置时间戳生效以来被作为垃圾回收的查询的总数
SSMP Request Message Count	自统计重置时间戳生效以来 SSMP 进程发送消息的计数
SSMP Request Message Bytes	自统计重置时间戳生效以来发送的消息字节（作为 SSMP 进程请求的一部分）的数量
SSMP Reply Message Count	自统计重置时间戳生效以来 SSMP 进程接收到的回复消息的计数
SSMP Reply Message Bytes	自统计重置时间戳生效以来 SSMP 进程接收到的消息字节（作为回复消息的一部分发送）的计数
SSCP Request Message Count	自统计重置时间戳生效以来 SSCP 进程发送消息的计数
SSCP Request Message Bytes	自统计重置时间戳生效以来发送的消息字节（作为 SSCP 进程请求的一部分）
SSCP Reply Message Count	自统计重置时间戳生效以来 SSCP 进程接收到的回复消息的计数

SSCP Reply Message Bytes	自统计重置时间戳生效以来 SSCP 进程接收到的消息字节（作为回复消息的一部分发送）的数量
RMS Stats Reset Timestamp	RMS 统计重置所用的时间戳

```
SQL>get statistics for rms all;
Node name
CPU 0
RMS Version 2511
SSCP PID 19521
SSCP Priority 0
SSCP Creation Timestamp 2010/12/05 02:32:33.642752 SSMP PID
19527
SSMP Priority 0
SSMP Creation Timestamp 2010/12/05 02:32:33.893440 Source
String Store Len 254
Stats Heap Allocated 0
Stats Heap Used 3,002,416
Stats Heap High WM 3,298,976
No.of Process Regd. 157
No.of Query Fragments Regd. 296 RMS Semaphore Owner -1
No.of SSCPs Opened 1
No.of SSCPs Open Deleted 0
Last GC Time 2010/12/06 10:53:46.777432
Queries GCed in Last Run 55
Total Queries GCed 167
SSMP Request Message Count 58,071 SSMP Request Message
Bytes 14,161,144 SSMP Reply Message Count 33,466 SSMP Reply
Message Bytes 15,400,424 SSCP Request Message Count 3,737
SSCP Request Message Bytes 837,744 SSCP Reply Message Count
3,736 SSCP Reply Message Bytes 5,015,176
RMS Stats Reset Timestamp 2010/12/05 14:32:33.891083

--- SQL operation complete.
```

8.6 使用 QUERYID_EXTRACT 函数

在一条 SQL 语句中使用 QUERYID_EXTRACT 函数抽取一个查询 ID 的分量用于一个 SQL 查询。查询 ID 或 QID 是集群范围内的一个查询的唯一标识符，每当预处理一个 SQL 字符串时，都会生成这种标识符用于动态 SQL 语句。

8.6.1 QUERYID_EXTRACT 函数语法

QUERYID_EXTRACT 函数的语法是:

QUERYID_EXTRACT (*'query-id', 'attribute'*)

query-id

字符串格式的查询 ID

atttribute

需要抽取的属性。属性值为查询 ID 的这些部分之一:

属性值	描述
SEGMENTNUM	EsgynDB 集群里的逻辑节点 ID
CPUNUM or CPU	EsgynDB 集群里的逻辑节点 ID
PIN	Linux 进程 ID 号
EXESTARTTIME	执行器启动时间
SESSIONNUM	会话号
USERNAME	用户名
SESSIONNAME	会话名
SESSIONID	会话 ID
QUERYNUM	查询号
STMTNAME	语句 ID 或句柄

注意: SEGMENTNUM和CPUNUM 具有相同属性。

QUERYID_EXTRACT 函数结果的数据类型为 VARCHAR，长度必须足以容纳结果。所有返回值都是字符串格式。这里是一条 SELECT 语句里的 QUERYID_EXTRACT 函数:

```
select queryid_extract('query-id', 'attribute-value')
from (values(1)) as t1;
```

8.6.2 QUERYID_EXTRACT 函数例子

- 该命令返回查询 ID 的节点号:

  ```
  SQL>select
  ```

```
substr(queryid_extract('MXID110000226752121705545487622
40000000000206U6553500_21_S1', 'CPU'),1,
20) from (values(1)) as t1;

(EXPR)
-------------------------------------------------------
0

--- 1 row(s) selected.
```

- 该命令返回查询 ID 的 PIN:

```
SQL>select
substr(queryid_extract('MXID110000226752121705545487622
40000000000206U6553500_21_S1', 'PIN'),1,
20) from (values(1)) as t1;

(EXPR)
-------------------------------------------------------
22675

--- 1 row(s) selected.
```

8.7 一个查询语句的每个碎片实例的统计信息

通过使用STATISTICS 表值函数，您检索到一个正在执行的查询统计信息。根据使用的语法，您可以获取该查询的每个并行碎片实例的汇总统计信息或每个碎片实例中的任何运算符的统计信息。

8.7.1 STATISTICS 表值函数语法

Table(statistics (NULL, '*qid-str*'))

qid-str 是:
QID=*query-id*[,{TDBID_DETAIL=tdb-id|DETAIL=1}]

query-id
这是系统生成的查询 ID。例如:
QID=MXID110000226752121705545487622400000000000206U6553500

_21_S1

tdb-id

这是一个给定运算符的 TDB ID。TDB 值从 GET STATISTICS 命令返回的报告中获取。

8.7.2 获取查询语句的每个碎片实例的统计信息时的注意事项

如果在查询未执行时使用 DETAIL=1 或 TDBID_DETAIL=tdb_id 选项，则 STATISTICS 表值函数将不会返回任何结果。

STATISTICS 表值函数与一条 SELECT 语句结合使用，以返回多个列。variable_info 列里存在很多不同的计数器。此列里的计数器将被格式化为 token-值对；所报告的计数器则取决于所用的具体选项：DETAIL=1 或 TDBID_DETAIL=tdb_id。如果使用 TDBID_DETAIL 选项，则所报告的计数器还将取决于 tdb_id 指定的运算符类型。另外，所报告的计数器还可由 statsRowType 计数器确定。

这些计数器的令牌被列在“Tokens in STATISTICS Table-Valued Function”（STATISTICS 表值函数里的令牌）列里。

- 该查询列出了通过给定 QID 标识的一个正在执行中的查询的所有 ESP 的进程名：

```
SQL>select
+>substr(variable_info,
+> position('ProcessId:' in variable_info), 20) as
processes
+>from
+>table(statistics(NULL,
+>'QID=MXID11000032684212170811581160672000000000206U65
53500_19_S1,DETAIL=1' ))
+>group by 1;
```

```
PROCESSES
--------------------

ProcessId: $Z0000GS ProcessId: $Z0000GT

ProcessId:  $Z0000GU  ProcessId:  $Z0000GV  ProcessId:
$Z0102IQ  ProcessId:  $Z000RNU  ProcessId:  $Z0102IR
ProcessId: $Z0102IS ProcessId: $Z0102IT
--- 9 row(s) selected.
```

- 该查询给出了 BMO 堆，它用于一个正在执行中的查询（通过给定的 QID 标识）的标识为 TDB #15 的哈希联接:

```
SQL>select cast (
+> substr(variable_info,
+> position('bmoHeapUsed:' in variable_info),
+> position('bmoHeapUsed:' in variable_info) +
+> 13 + (position(' ' in
+> substr(variable_info,
+> 13 + position('bmoHeapUsed:' in variable_info))) -
+> position('bmoHeapUsed:' in variable_info)))
+> as char(25))
+> from table(statistics(NULL,
+>'QID=MXID11000021706212170733911504160000000000206U65
53500_25_S1,TDBID_DETAIL=15'));

(EXPR)
-------------------------

bmoHeapUsed: 3147
bmoHeapUsed: 3147
bmoHeapUsed: 3147
bmoHeapUsed: 3147
bmoHeapUsed: 3147
bmoHeapUsed: 3147
bmoHeapUsed: 3147
bmoHeapUsed: 3147

--- 8 row(s) selected.
```

附录 A　保留字

本附录里所列的字都将被保留供 EsgynDB SQL 使用。为了防止出现语法错误，请避免在 EsgynDB SQL 里使用这些文字作为标识符。在 EsgynDB SQL 里，如果一个操作系统名称中含有一个保留字，您必须用双引号（"）将它括起来后才能访问该列或对象。

注意事项： 在EsgynDB SQL，ABSOLUTE，DATA，EVERY，INITIALIZE，OPERATION，PATH，SPACE，STATE，STATEMENT，STATIC和START不是保留字。

保留的 EsgynDB SQL 标识符

如果它们是 EsgynDB SQL 存储文本的一部分，EsgynDB SQL 将这些文字视为保留字。它们不得被用作标识符，除非您用双引号将其括起来。

保留的SQL标识符-A				
ACTION	ADD	ADMIN	AFTER	AGGREGATE
ALIAS	ALL	ALLOCATE	ALTER	AND
ANY	ARE	ARRAY	AS	ASC
ASSERTION	ASYNC	AT	AUTHORIZATION	AVG

保留的SQL标识符-B				
BEFORE	BEGIN	BETWEEN	BINARY	BIT
BIT_LENGTH	BLOB	BOOLEAN	BOTH	BREADTH
BY				

保留的SQL标识符-C				
CALL	CASCADE	CASCADED	CASE	CAST
CATALOG	CHAR	CHARACTER	CHARACTER_LENGTH	CHAR_LENGTH
CHECK	CLASS	CLOB	CLOSE	COALESCE
COLLATE	COLLATION	COLUMN	COMMIT	COMPLETION

CONNECT	CONNECTION	CONSTRAINT	CONSTRAINTS	CONSTRUCTOR
CONTINUE	CONVERT	CORRESPONDING	COUNT	CREATE
CROSS	CUBE	CURRENT	CURRENT_DATE	CURRENT_PATH
CURRENT_ROLE	CURRENT_TIME	CURRENT_TIMESTAMP	CURRENT_USER	CURRNT_USR_INTN
CURSOR	CYCLE			

保留的SQL标识符-D				
DATE	DATETIME	DAY	DEALLOCATE	DEC
DECIMAL	DECLARE	DEFAULT	DEFERRABLE	DEFERRED
DELETE	DEPTH	DEREF	DESC	DESCRIBE
DESCRIPTOR	DESTROY	DESTRUCTOR	DETERMINISTIC	DIAGNOSTICS
DICTIONARY	DISCONNECT	DISTINCT	DOMAIN	DOUBLE
DROP	DYNAMIC			

保留的SQL标识符-E				
EACH	ELSE	ELSEIF	END	END-EXEC
EQUAL	ESCAPE	EXCEPT	EXCEPTION	EXEC
EXECUTE	EXISTS	EXTERNAL	EXTRACT	

保留的SQL标识符-F				
FALSE	FETCH	FIRST	FLOAT	FOR
FOREIGN	FOUND	FRACTION	FREE	FROM
FULL	FUNCTION			

保留的SQL标识符-G				
GENERAL	GET	GLOBAL	GO	GOTO
GRANT	GROUP	GROUPING		

保留的SQL标识符-H				
HAVING	HOST	HOUR		

保留的SQL标识符-I				
IDENTITY	IF	IGNORE	IMMEDIATE	IN
INDICATOR	INITIALLY	INNER	INOUT	INPUT
INSENSITIVE	INSERT	INT	INTEGER	INTERSECT
INTERVAL	INTO	IS	ISOLATION	ITERATE

保留的SQL标识符-J				
JOIN				

保留的SQL标识符-K				
KEY				

保留的SQL标识符-L				
LANGUAGE	LARGE	LAST	LATERAL	LEADING
LEAVE	LEFT	LESS	LEVEL	LIKE
LIMIT	LOCAL	LOCALTIME	LOCALTIMESTAMP	LOCATOR
LOOP	LOWER			

保留的SQL标识符-M				
MAINTAIN	MAP	MATCH	MATCHED	MAX
MERGE	MIN	MINUTE	MODIFIES	MODIFY
MODULE	MONTH			

保留的SQL标识符-N				
NAMES	NATIONAL	NATURAL	NCHAR	NCLOB
NEW	NEXT	NO	NONE	NOT
NULL	NULLIF	NUMERIC		

保留的SQL标识符-O				
OCTET_LENGTH	OF	OFF	OID	OLD
ON	ONLY	OPEN	OPERATORS	OPTION
OPTIONS	OR	ORDER	ORDINALITY	OTHERS
OUT	OUTER	OUTPUT	OVERLAPS	

保留的SQL标识符-P				
PAD	PARAMETER	PARAMETERS	PARTIAL	PENDANT
POSITION	POSTFIX	PRECISION	PREFIX	PREORDER
PREPARE	PRESERVE	PRIMARY	PRIOR	PRIVATE
PRIVILEGES	PROCEDURE	PROTECTED	PROTOTYPE	PUBLIC

保留的SQL标识符-Q				
QUALIFY				

保留的SQL标识符-R				
READ	READS	REAL	RECURSIVE	REF
REFERENCES	REFERENCING	RELATIVE	REORG	REORGANIZE
REPLACE	RESIGNAL	RESTRICT	RESULT	RETURN
RETURNS	REVOKE	RIGHT	ROLLBACK	ROLLUP
ROUTINE	ROW	ROWS	PROTOTYPE	

保留的SQL标识符-S				
SAVEPOINT	SCHEMA	SCOPE	SCROLL	SEARCH
SECOND	SECTION	SELECT	SENSITIVE	SESSION
SESSION_USER	SESSN_USR_INTN	SET	SETS	SIGNAL
SIMILAR	SIZE	SMALLINT	SOME	SPECIFIC
SPECIFICTYPE	SQL	SQL_CHAR	SQL_DATE	SQL_DECIMAL
SQL_DOUBLE	SQL_FLOAT	SQL_INT	SQL_INTEGER	SQL_REAL
SQL_SMALLINT	SQL_TIME	SQL_TIMESTAMP	SQL_VARCHAR	SQLCODE
SQLERROR	SQLEXCEPTION	SQLSTATE	SQLWARNING	STRUCTURE

SUBSTRING	SUM	SYNONYM	SYSTEM_USER	

保留的SQL标识符-T				
TABLE	TEMPORARY	TERMINATE	TEST	THAN
THEN	THERE	TIME	TIMESTAMP	TIMEZONE_HOUR
TIMEZONE_MINUTE	TO	TRAILING	TRANSACTION	TRANSLATE
TRANSLATION	TRANSPOSE	TREAT	TRIGGER	TRIM
TRUE				

保留的SQL标识符-U				
UNDER	UNION	UNIQUE	UNKNOWN	UNNEST
UPDATE	UPPER	UPSHIFT	USAGE	USER
USING				

保留的SQL标识符-V				
VALUE	VALUES	VARCHAR	VARIABLE	VARYING
VIEW	VIRTUAL	VISIBLE		

保留的SQL标识符-W				
WAIT	WHEN	WHENEVER	WHERE	WHILE
WITH	WITHOUT	WORK	WRITE	

保留的SQL标识符-Y				
YEAR				

保留的SQL标识符-Z				
ZONE				

附录 B　控制查询默认（CQD）属性

本附录描述用于覆盖系统级默认设置的 CQD。

1. HBase 环境 CQD

本小节描述 CQD“HBASE_INTERFACE”，它用来定义 HBase 接口。

1) HBASE_INTERFACE

类别	HBase
描述	用来访问 HBase 的接口
数值	指定以下值之一： • JNI 使用 JNI 接口 • JNI_TRX 通过 JNI 使用 HBase-trx 和事务接口 默认值为 JNI_TRX

2. Hive 环境 CQD

本小节描述的 CQD “HIVE_MAX_STRING_LENGTH”定义了 Hive 里字符串数据类型的最大字符串长度。

1) HIVE_MAX_STRING_LENGTH

类别	Hive
描述	Hive 里字符串数据类型支持的最大字符串长度。Hive 表里的所有字符串列都转换为 VARCHAR（n 字节）CHARACTER SET UTF8，n 为该 CQD 的值。
数值	默认值为 32000。

3. 管理直方图

本小节描述用于管理直方图的 CQD:

1) CACHE_HISTOGRAMS_REFRESH_INTERVAL

类别	Histograms
描述	定义一个间隔期，这段时间过后，将核对是否需要更新缓存直方图的时间戳。
数值	无符号整数; 数值单位为秒; 默认值为‘3600’(1 小时)。
使用	直方图统计信息会被缓存，这样编译器能避免访问元数据表，缩短了编译时间。在 CQD 指定的间隔期，这些表的时间戳与缓存直方图的时间戳会互相核对，以查看缓存直方图是否需要更新。 如果您无需更频繁地获取最新的统计信息（以提高查询性能），您能延长间隔期，以减轻对编译时间的影响。也有可能因为默认的时间间隔太长，您希望选择更频繁地（而非在默认的 1 小时后）刷新统计信息，以增加编译时间为代价改进查询性能。 该设置取决于表统计信息的更新频率。如果统计信息在一段时间内没有被更新，则不必频繁地刷新它们。另一方面，如果经常在刚装载的表上更新统计信息或在刚装载的表上首次生成统计信息，并且您希望编译器能够立即获得这些数据，且发现它们对于计划的质量产生巨大影响，您能够更频繁地进行刷新。
生产使用	不适用。
影响	较长的直方图刷新间隔期改进编译时间。然而，刷新间隔期越长，则直方图越过时。这可能导致利用最新统计信息的查询表现较差的性能。
级别	系统或服务。
解决真正问题	不适用。

2) HIST_NO_STATS_REFRESH_INTERVAL

类别	Histograms
描述	定义一个间隔期，该期限过后应无条件刷新缓存里的“假”直方图。

数值	整数； 数值单位为秒； 默认值为‘3600’(1 小时)。
使用	如果尚未运行 UPDATE STATISTICS，则直方图统计信息是“假”的；相反，如果一个客户正在直接使用统计信息更新直方图表（以指导优化器进行工作），则直方图统计信息是“真实”的。如果一个表（例如临时表）的数据非常多变，则无法更新统计信息，因为表里的数据需要不断刷新和填充；这时就需要手动设置统计信息，以指导优化器生成一个良好的计划。 如果这些“假”统计信息被不断更新，以反映数据流失情况，则可将默认值设置为 0。由此保证包含“假”统计信息的直方图将不会被缓存、并将被始终刷新。如果这些“假”统计信息已经被设置完毕、以后也不会再改动，则应设置很高的间隔期。
生产使用	不适用。
影响	设置一个较高的间隔期改进编译时间。然而，如果统计信息正在更新，则编译器可能还在使用过时的直方图统计信息，这样可能导致生成较差的计划。
级别	服务。
冲突/协作	不适用。
解决真正问题	不适用。

3) HIST_PREFETCH

类别	Histograms
描述	促使编译器预取直方图，并将它们保存在缓存中。
数值	‘ON’：预取直方图 ‘OFF’：不预取直方图 默认值为‘ON’。
使用	如果不希望预取大量直方图（其中很多可能都用不上），则关闭该属性。
生产使用	不适用。
影响	虽然这会加快编译速度，但也可能导致直方图缓存里填充一

	些永远不会使用的直方图。
级别	系统或服务。
冲突/协作	该 CQD 与 CACHE_HISTOGRAMS 结合使用。如果 CACHE_HISTOGRAMS 关闭，则该 CQD 将没有任何效果。
解决真正问题	不适用。

4) HIST_ROWCOUNT_REQUIRING_STATS

类别	Histograms
描述	指定优化器生成直方图所需的最小行数，以计算更佳的基数估值。优化器不会为尺寸小于该 CQD 值的表发出任何统计信息缺失警告。
数值	整数； 默认值为‘50000’。
使用	使用该 CQD 减少统计信息警告数量。
生产使用	不适用。
影响	对于较小的表则不会显示统计信息缺失警告；大多数情况下，它们不会影响计划的质量。然而，也可能存在异常情况，某些较小的表缺失统计信息会导致生成的计划不太理想。
级别	系统。
冲突/协作	同时使用该 CQD 与 HIST_MISSING_STATS_WARNING_LEVEL。如果 CQD 的警告级别为 0，则该 CQD 将没有任何效果。而且，如果表里的行数少于该 CQD 里的设定值，则不会显示任何警告，无论何种警告级别都是如此。
解决真正问题	不适用。

4. 优化器

本小节描述优化器使用的 CQD。

1) JOIN_ORDER_BY_USER

类别	影响查询计划
描述	启用或禁用一种联接顺序，优化器使用该顺序将表联接成这个查询的 FROM 子句里的表序列。
数值	‘ON’：强制使用联接顺序 ‘OFF’：优化器决定是否使用联接顺序 默认值为‘OFF’。
使用	如果数值设置为 ON，则优化器将仅考虑那联接顺序匹配 FROM 子句的表序列的执行计划。
生产使用	该设置仅用于强制实施一种并非优化器默认生成的期望的联接顺序。它可被用作具有低效联接顺序的查询计划的变通方案。
影响	由于您正在强制优化器使用一种计划，以按照 FROM 子句里指定的顺序联接表，所以生成的计划可能并非最优计划。
级别	查询。
冲突/协作	不适用。
解决真正问题	不适用。

2) MDAM_SCAN_METHOD

类别	影响查询计划
描述	启用或禁用多维访问方法。
数值	‘ON’：启用 MDAM ‘OFF’：禁用 MDAM 默认值为‘ON’。
使用	在某些情况下，优化器可能不当地选择了 MDAM，导致性能不佳。在这种情况下，您可能希望为受影响的查询关闭 MDAM。
生产使用	不适用。
影响	通过非前导的 Clustering Key 列上的谓词进行的表扫描可能受益于 MDAM 访问方式，如果前导列带有少量的不同值。关闭

	MDAM 将会延长此类查询的扫描时间。
级别	如果 MDAM 对于特定的查询无法有效运行，则在查询层次设置该 CQD。然而，可能存在这种情况（通常是一个缺陷），其中一个更大的查询集合会受到 MADAM 的负面影响。 在这种情况下，您可能需要将它设置在服务层级或系统层级。
冲突/协作	不适用。
解决真正问题	不适用。

3) SUBQUERY_UNNESTING

类别	影响查询计划
描述	控制优化器将嵌套的子查询转换为常规联接树的能力。
数值	‘ON’：启用子查询解套 ‘OFF’：禁用子查询解套 默认值为‘ON’。
使用	如果子查询解套会导致产生低效的查询执行计划时，应使用该控件禁用子查询解套。
生产使用	不适用。
影响	一般而言，子查询解套为带有嵌套子查询的查询产生更有效的执行计划。仅可用作所观察到的子查询解套问题的变通方案。
级别	查询。
冲突/协作	不适用。
解决真正问题	不适用。

5. 管理 Schema

本小节描述用于管理 Schema 的 CQD:

1) Schema

类别	Schema 控制
描述	为会话设置默认 Schema。
数值	SQL 标识符; 默认值为 SEABASE。
使用	一条 SET SCHEMA 语句或 CONTROL QUERY DEFAULT SCHEMA 语句可被用来覆盖默认 Schema 名称。
生产使用	它为您提供便利，因此不必键入两部分名称。
影响	不适用。
级别	任何。
冲突/协作	此外，也可使用 SET SCHEMA 语句。
解决真正问题	不适用。

6. 事务控制和锁定

本小节描述用于事务控制和锁定的 CQD。

1) BLOCK_TO_PREVENT_HALLOWEEN

类别	运行时控制
描述	自引用插入是将数据插入到目标表；作为查询的一部分，对同一个目标表进行扫描，以产生需要插入的行。如果一条插入语句扫描相同语句已经插入的行，则会产生不一致的结果。有时这被称为“Halloween 问题”，EsgynDB 使用这种方法防范 Halloween 问题： 阻塞方法使用 SORT 运算保证所有的行都在插入之前得到扫描；或 如果执行计划的静态分析表明磁盘进程锁定方法不可用，则编译器将选择阻塞方法。 然而，编译器不会针对一个可能阻止使用磁盘进程锁定方法的条件进行求值： 所执行的语句里有 AUTOCOMMIT 设置。相反，编译器会假设所执行的语句带有 AUTOCOMMIT 的默认设置‘ON’。如果 AUTOCOMMIT 被设置为‘OFF’，并且使用磁盘进程锁定方法的一条自引用插入语句已被执行，则会引发一个运行时错误（SQLCODE 8107）。 该 CQD 强制编译器使用阻塞方法，以防出现错误 8107。
数值	‘OFF’：编译器自由选择使用何种方法防范 Halloween 问题。 ‘ON’：强制编译器使用阻塞方法 默认值为‘ON’。
使用	在会话里执行一条关闭 AUTOCOMMIT 选项的自引用插入语句将引发错误 8107，将默认值改为‘ON’。
生产使用	不适用。
影响	如果关闭 AUTOCOMMIT 选项时执行的自引用插入语句受限于一个服务级别，则应将默认值设置为仅对该服务级别‘ON’（开启），否则应针对整个系统进行该设置。
级别	查询。
冲突/协作	不适用。
解决真正问题	不适用。

2) UPD_ORDERED

类别	影响查询计划
描述	控制是否应该按照 Clustering Key 的顺序插入、更新或删除行。
数值	‘OFF’：优化器生成并启用计划，各行都将按照 Clustering Key 的顺序插入、更新或删除。 ‘ON’：优化器不会生成计划，各行都必须按照 Clustering Key 的顺序插入、更新或删除。 默认值为‘ON’。
使用	强烈推荐按照 Clustering Key 的顺序插入、更新或删除行，该方法是是最有效的。关闭该 CQD 属性可能会节省数据排序成本，但同时也会带来低效的随机 I/O 和插入/更新/删除操作。 如果了解该数据已按照或基本按照 Clustering Key 的顺序进行排序，不会产生随机 I/O，则应将该 CQD 设置为 OFF。
生产使用	不适用。
影响	如果该选项被关闭，则系统在执行插入/更新/删除操作时可能会执行大量无效的随机 I/O。
级别	查询。
冲突/协作	不适用。
解决真正问题	不适用。

附录 C　使用限制

本附录列出了 EsgynDB SQL 各部分的使用限制。

列名	最长 128 个字符或 256 个 UTF8 文本字节，取两者中的较小者。
Schema 名	最长 128 个字符或 256 个 UTF8 文本字节，取两者中的较小者。
表名	ANSI 名称的形式为 schema.object，其中每部分最长 128 个字符或 256 个 UTF8 文本字节，取两者中的较小者。