

**IBM solidDB
IBM solidDB Universal Cache
V7.0**

SQL 指南



声明

在使用本资料及其支持的产品之前，请阅读第 397 页的『声明』中的信息。

第一修订版

此修订版适用于 IBM solidDB V7R0（产品编号 5724-V17）和 IBM solidDB Universal Cache（产品编号 5724-W91）以及所有后续发行版和修订版，直到在新版本中另有声明为止。

© Oy International Business Machines Ab Ltd. 1993, 2011

目录

图	ix
表	xi
关于本手册	xv
印刷约定	xv
语法表示法约定	xvi
1 数据库概念	1
1.1 关系数据库	1
1.1.1 表、行和列	1
1.1.2 关联不同表中的数据	2
1.2 客户机/服务器体系结构	4
1.3 多用户能力	4
1.4 事务	4
1.5 事务日志记录与恢复	5
1.5.1 背景	5
1.6 摘要	6
2 SQL 入门	7
2.1 表、行和列	7
2.2 SQL	7
2.3 SQL 的数学起源	9
2.4 创建具有相关数据的表	9
2.4.1 表别名	11
2.4.2 子查询	11
2.5 对于每种数据类型使用哪些格式	12
2.5.1 BLOB (或二进制数据类型)	13
2.5.2 NULL IS NOT NULL (即, 如何在 SQL 中 表达“并非上述任何一项”)	14
2.5.3 NOT NULL	16
2.5.4 表达式与强制类型转换	16
2.5.5 行值构造器	18
2.6 有关事务的更多说明	19
2.7 摘要	20
2.8 在何处查找有关 SQL 的其他信息	20
3 SQL 扩展	21
3.1 存储过程	21
3.1.1 存储过程 - SQL	22
3.1.2 存储过程 - 外部	56
3.2 函数	57
3.3 触发器	57
3.3.1 触发器 - 操作规则	58
3.3.2 创建和修改触发器	58
3.3.3 触发器与过程	59
3.3.4 触发器与事务	61
3.3.5 触发器特权和安全性	67
3.3.6 从触发器中引发错误	67
3.3.7 获取触发器信息	68
3.3.8 触发器参数设置	69
3.3.9 触发器示例	69
3.4 序列	72
3.5 事件	73
3.5.1 使用事件 - 例 1	75
3.5.2 使用事件 - 例 2	77
3.5.3 使用事件 - 例 3	79
4 使用 solidDB SQL 进行数据库管理	83
4.1 使用 solidDB SQL 语法	83
4.1.1 solidDB SQL 数据类型	83
4.1.2 solidDB ADMIN COMMANDs	83
4.1.3 标量函数	83
4.2 管理用户特权和角色	84
4.2.1 用户特权	84
4.2.2 用户角色	84
4.2.3 SQL 语句示例	85
4.3 管理表	88
4.3.1 访问系统表	88
4.3.2 SQL 语句示例	89
4.4 管理索引	90
4.4.1 SQL 语句示例	91
4.4.2 主键索引	91
4.4.3 辅键索引	92
4.4.4 避免索引重复	92
4.5 引用完整性	93
4.5.1 主键与候选键	93
4.5.2 外键	94
4.5.3 引用动作	96
4.5.4 动态约束管理	97
4.6 管理数据库对象	98
4.6.1 简介	98
4.6.2 目录	99
4.6.3 模式	99
4.6.4 唯一地标识目录和模式中的对象	99
4.6.5 SQL 语句示例	100
5 管理事务	103
5.1 定义只读事务和读/写事务	103
5.2 并行控制和锁定	103
5.2.1 PESSIMISTIC 并行控制与 OPTIMISTIC 并 行控制	104
5.2.2 锁定和锁定方式	106
5.2.3 设置并行控制	109
5.3 选择事务耐久性级别	112
5.3.1 设置事务耐久性级别	113
6 SQL 诊断与故障诊断	115
6.1 观察性能	115
6.1.1 “SQL 信息”工具	115
6.1.2 EXPLAIN PLAN FOR 语句	116
6.2 用于存储过程和触发器的跟踪工具	121

6.2.1	用户可定义的过程代码的跟踪输出	121
6.2.2	执行跟踪过程	121
6.3	测量和提高 START AFTER COMMIT 语句的性能	122
6.3.1	调整 START AFTER COMMIT 语句的性能	122
6.3.2	分析 START AFTER COMMIT 语句的故障	123
7	使用 SQL 微调性能	125
7.1	调整 SQL 语句和应用程序	125
7.1.1	评估应用程序性能	125
7.1.2	使用存储过程语言	126
7.2	优化单表 SQL 查询	126
7.3	使用索引来提高查询性能	126
7.3.1	全表扫描	127
7.3.2	并置型索引	128
7.4	等待事件	128
7.5	优化批处理插入和更新	129
7.5.1	提高批处理插入和更新速度	129
7.6	使用优化器提示	130
7.7	对性能不佳问题进行诊断	131
附录 A	. 语句	133
A.1	ADMIN COMMAND	133
A.2	ADMIN EVENT	144
A.3	ALTER REMOTE SERVER	145
A.4	ALTER TABLE	145
A.4.1	ALTER TABLE ... SET HISTORY COLUMNS	146
A.4.2	ALTER TABLE ... SET SYNCHISTORY	147
A.5	ALTER TRIGGER	149
A.6	ALTER USER	149
A.6.1	ALTER USER (副本)	150
A.7	CALL	151
A.7.1	执行远程存储过程调用时的访问权	152
A.8	COMMIT WORK	154
A.9	CREATE CATALOG	154
A.10	CREATE EVENT	156
A.11	CREATE FUNCTION	157
A.12	CREATE FUNCTION (外部)	159
A.13	CREATE INDEX	162
A.14	CREATE PROCEDURE	163
A.14.1	用法	164
A.14.2	parameter_modes	166
A.14.3	prepare_statement	166
A.14.4	execute_statement	167
A.14.5	fetch_statement	168
A.14.6	post_statement	168
A.14.7	wait_register_statement	168
A.14.8	wait_event_statement	169
A.14.9	control_statement	169
A.14.10	writetrace_statement	170
A.14.11	exec_direct_statement	171
A.14.12	过程堆栈函数	171
A.14.13	动态游标名	171
A.14.14	示例	172
A.15	CREATE PROCEDURE (外部)	175

A.16	CREATE [OR REPLACE] PUBLICATION	180
A.17	CREATE [OR REPLACE] REMOTE SERVER	183
A.18	CREATE ROLE	184
A.19	CREATE SCHEMA	184
A.20	CREATE SEQUENCE	186
A.21	CREATE SYNC BOOKMARK	187
A.22	CREATE TABLE	188
A.23	CREATE TRIGGER	191
A.23.1	trigger_name	192
A.23.2	BEFORE AFTER 子句	193
A.23.3	INSERT UPDATE DELETE 子句	194
A.23.4	table_name	195
A.23.5	trigger_body	195
A.23.6	REFERENCING 子句	195
A.23.7	触发器用途注释和限制	196
A.24	CREATE USER	197
A.25	CREATE VIEW	197
A.26	DELETE	197
A.27	DESCRIBE	198
A.28	DROP CATALOG	199
A.29	DROP EVENT	200
A.30	DROP FUNCTION	200
A.31	DROP INDEX	200
A.32	DROP MASTER	201
A.33	DROP PROCEDURE	201
A.34	DROP PUBLICATION	202
A.35	DROP PUBLICATION REGISTRATION	202
A.36	DROP REMOTE SERVER	203
A.37	DROP REPLICA	203
A.38	DROP ROLE	204
A.39	DROP SCHEMA	204
A.40	DROP SEQUENCE	205
A.41	DROP SUBSCRIPTION	205
A.42	DROP SYNC BOOKMARK	206
A.43	DROP TABLE	207
A.44	DROP TRIGGER	208
A.45	DROP USER	209
A.46	DROP VIEW	209
A.47	EXPLAIN PLAN FOR	209
A.48	EXPORT SUBSCRIPTION	210
A.49	EXPORT SUBSCRIPTION TO REPLICA	212
A.50	GRANT	214
A.51	GRANT PASSTHROUGH	214
A.52	GRANT REFRESH	215
A.53	HINT	215
A.53.1	solidDB-支持提示	219
A.54	IMPORT	221
A.55	INSERT	223
A.56	LIST	224
A.57	LOCK TABLE	225
A.58	MESSAGE APPEND	228
A.59	MESSAGE BEGIN	230
A.60	MESSAGE DELETE	231
A.60.1	支持环境	231
A.60.2	用法	231
A.60.3	用于主数据库	232

A.60.4 用于副本数据库	232	A.73 REVOKE PASSTHROUGH	250
A.60.5 示例	232	A.74 REVOKE REFRESH	251
A.61 MESSAGE DELETE CURRENT		A.74.1 支持环境	251
TRANSACTION.	233	A.74.2 用法	251
A.61.1 支持环境	233	A.74.3 用于主数据库	251
A.61.2 用法	233	A.74.4 用于副本数据库	251
A.61.3 用于主数据库	233	A.74.5 示例	251
A.61.4 用于副本数据库	233	A.74.6 返回值	251
A.61.5 示例	233	A.75 ROLLBACK WORK	252
A.61.6 返回值	234	A.76 SAVE	252
A.62 MESSAGE END.	234	A.76.1 支持环境	252
A.62.1 支持环境	234	A.76.2 用法	252
A.62.2 用法	234	A.76.3 用于主数据库	253
A.62.3 用于主数据库	234	A.76.4 用于副本数据库	253
A.62.4 用于副本数据库	234	A.76.5 示例	253
A.62.5 来自副本数据库的返回值	235	A.76.6 返回值	253
A.62.6 来自主数据库的返回值	235	A.77 SAVE PROPERTY	253
A.63 MESSAGE EXECUTE.	235	A.77.1 支持环境	254
A.63.1 支持环境	235	A.77.2 用法	254
A.63.2 用法	236	A.77.3 用于主数据库	254
A.63.3 用于主数据库	236	A.77.4 用于副本数据库	254
A.63.4 用于副本数据库	236	A.77.5 “PUT_PARAM()”与“SAVE PROPERTY	
A.63.5 结果集	236	property_name VALUE property_value;”之间的区	
A.63.6 示例	236	别	254
A.63.7 返回值	236	A.77.6 示例	254
A.64 MESSAGE FORWARD	237	A.77.7 返回值	255
A.64.1 支持环境	237	A.77.8 结果集	255
A.64.2 用法	237	A.78 SELECT	255
A.64.3 示例	238	A.79 SET.	257
A.64.4 来自副本数据库的返回值	239	A.79.1 SET 与 SET TRANSACTION 之间的差别	258
A.64.5 来自主数据库的返回值	240	A.79.2 SET (读/写级别)	259
A.65 MESSAGE FROM REPLICA DELETE	241	A.79.3 SET CATALOG	259
A.66 MESSAGE FROM REPLICA EXECUTE.	241	A.79.4 SET DELETE CAPTURE	259
A.66.1 支持环境	241	A.79.5 SET DURABILITY	259
A.66.2 用法	241	A.79.6 SET ISOLATION LEVEL	260
A.66.3 用于主数据库	241	A.79.7 SET PASSTHROUGH	260
A.66.4 用于副本数据库	241	A.79.8 SET SAFENESS	261
A.66.5 示例	241	A.79.9 SET SCHEMA.	261
A.66.6 返回值	241	A.79.10 SET SEQUENCE	262
A.67 MESSAGE FROM REPLICA RESTART.	242	A.79.11 SET SQL	262
A.68 MESSAGE GET REPLY	242	A.79.12 SET STATEMENT MAXTIME	263
A.68.1 支持环境	242	A.79.13 SET SYNC	263
A.68.2 用法	242	A.79.14 SET TIMEOUT	270
A.68.3 用于主数据库	243	A.80 SET TRANSACTION	271
A.68.4 用于副本数据库	243	A.80.1 SET 与 SET TRANSACTION 之间的差别	271
A.68.5 示例	243	A.80.2 SET TRANSACTION (读/写级别)	272
A.68.6 来自副本数据库的返回值	243	A.80.3 SET TRANSACTION DELETE CAPTURE	272
A.68.7 来自主数据库的返回值	244	A.80.4 SET TRANSACTION DURABILITY	272
A.68.8 结果集	245	A.80.5 SET TRANSACTION ISOLATION	
A.69 POST EVENT	246	LEVEL	273
A.70 REFRESH.	246	A.80.6 SET TRANSACTION PASSTHROUGH	273
A.70.1 用法	246	A.80.7 SET TRANSACTION SAFENESS	274
A.70.2 示例	247	A.81 START AFTER COMMIT	274
A.70.3 返回值	247	A.82 TRUNCATE TABLE	276
A.71 REGISTER EVENT.	249	A.83 UNLOCK TABLE	276
A.72 REVOKE	250	A.84 UNREGISTER EVENT	278

A.85 UPDATE	278
A.86 WAIT EVENT	278
A.87 常用子句	279
A.87.1 check_condition	279
A.87.2 data_type	280
A.87.3 表达式	280
A.87.4 query_specification	282
A.87.5 搜索条件	282
A.87.6 table_reference	283
A.87.7 SELECT 语句中的伪列	284
A.88 日期和时间字面值	284
A.89 通配符	284
A.89.1 使用 SQL 通配符	285
A.89.2 作为字面值的通配符	285

附录 B. 函数 287

B.1 字符串函数	287
B.2 数字函数	288
B.3 日期时间函数	290
B.4 系统函数	292
B.5 其他函数	293
B.6 高级复制函数	293
B.6.1 GET_PARAM() 函数	293
B.6.2 PUT_PARAM() 函数	294
B.7 触发器函数	295

附录 C. 数据类型 297

C.1 字符数据类型	297
C.2 数字数据类型	299
C.3 二进制数据类型	300
C.4 日期数据类型	301
C.5 时间数据类型	301
C.6 时间戳记数据类型	302
C.7 最小的可能非零数值	302
C.8 BLOB 和 CLOB	302

附录 D. 保留字 305

附录 E. 数据库系统表和系统视图 319

E.1 系统表	319
E.1.1 SQL_LANGUAGES	319
E.1.2 SYS_ATTAUTH	319
E.1.3 SYS_AUDIT_TRAIL	319
E.1.4 SYS_BACKGROUNDJOB_INFO	320
E.1.5 SYS_BLOBS	321
E.1.6 SYS_CARDINAL	322
E.1.7 SYS_CATALOGS	322
E.1.8 SYS_CHECKSTRINGS	322
E.1.9 SYS_COLUMNS	323
E.1.10 SYS_COLUMNS_AUX	324
E.1.11 SYS_DL_REPLICA_CONFIG	324
E.1.12 SYS_DL_REPLICA_DEFAULT	324
E.1.13 SYS_EVENTS	325
E.1.14 SYS_FEDT_DB_PARTITION	325
E.1.15 SYS_FEDT_TABLE_PARTITION	326
E.1.16 SYS_FORKEYPARTS	326

E.1.17 SYS_FORKEYS	327
E.1.18 SYS_HOTSTANDBY	327
E.1.19 SYS_INFO	327
E.1.20 SYS_KEYPARTS	327
E.1.21 SYS_KEYS	328
E.1.22 SYS_LOGPOS	328
E.1.23 SYS_PROCEDURES	329
E.1.24 SYS_PROCEDURE_COLUMNS	330
E.1.25 SYS_PROPERTIES	330
E.1.26 SYS_RELAUTH	331
E.1.27 SYS_SCHEMAS	331
E.1.28 SYS_SEQUENCES	331
E.1.29 SYS_SERVER	332
E.1.30 SYS_SYNC_REPLICA_PROPERTIES	332
E.1.31 SYS_SYNONYM	333
E.1.32 SYS_TABLEMODES	333
E.1.33 SYS_TABLES	333
E.1.34 SYS_TRIGGERS	334
E.1.35 SYS_TYPES	335
E.1.36 SYS_URole	335
E.1.37 SYS_USERS	336
E.1.38 SYS_VIEWS	336
E.2 用于执行数据同步的系统表	336
E.2.1 SYS_BULLETIN_BOARD	336
E.2.2 SYS_PUBLICATION_ARGS	337
E.2.3 SYS_PUBLICATION_REPLICA_ARGS	337
E.2.4 SYS_PUBLICATION_REPLICA_STMTARGS	338
E.2.5 SYS_PUBLICATION_REPLICA_STMTS	338
E.2.6 SYS_PUBLICATION_STMTARGS	338
E.2.7 SYS_PUBLICATION_STMTS	339
E.2.8 SYS_PUBLICATIONS	339
E.2.9 SYS_PUBLICATIONS_REPLICA	340
E.2.10 SYS_SYNC_BOOKMARKS	340
E.2.11 SYS_SYNC_HISTORY_COLUMNS	341
E.2.12 SYS_SYNC_INFO	341
E.2.13 SYS_SYNC_MASTER_MSGINFO	341
E.2.14 SYS_SYNC_MASTER_RECEIVED_BLOB_REFS	342
E.2.15 SYS_SYNC_MASTER_RECEIVED_MSGPARTS	343
E.2.16 SYS_SYNC_MASTER_RECEIVED_MSGS	343
E.2.17 SYS_SYNC_MASTER_STORED_BLOB_REFS	343
E.2.18 SYS_SYNC_MASTER_STORED_MSGPARTS	344
E.2.19 SYS_SYNC_MASTER_STORED_MSGS	344
E.2.20 SYS_SYNC_MASTER_SUBSC_REQ	345
E.2.21 SYS_SYNC_MASTER_VERSIONS	345
E.2.22 SYS_SYNC_MASTERS	346
E.2.23 SYS_SYNC_RECEIVED_BLOB_ARGS	346
E.2.24 SYS_SYNC_RECEIVED_STMTS	347
E.2.25 SYS_SYNC_REPLICA_MSGINFO	347
E.2.26 SYS_SYNC_REPLICA_RECEIVED_BLOB_REFS	348

E.2.27	
SYS_SYNC_REPLICA_RECEIVED_MSGPARTS	. . . 349
E.2.28 SYS_SYNC_REPLICA_RECEIVED_MSGS	349
E.2.29	
SYS_SYNC_REPLICA_STORED_BLOB_REFS	. . . 349
E.2.30 SYS_SYNC_REPLICA_STORED_MSGS	350
E.2.31	
SYS_SYNC_REPLICA_STORED_MSGPARTS	. . . 350
E.2.32 SYS_SYNC_REPLICA_VERSIONS	. . . 350
E.2.33 SYS_SYNC_REPLICAS 351
E.2.34 SYS_SYNC_SAVED_BLOB_ARGS	. . . 351
E.2.35 SYS_SYNC_SAVED_STMTS 352
E.2.36 SYS_SYNC_TRX_PROPERTIES 352
E.2.37 SYS_SYNC_USERMAPS 353
E.2.38 SYS_SYNC_USERS 353
E.3 系统视图 353
E.3.1 COLUMNS 354
E.3.2 SERVER_INFO 354
E.3.3 TABLES 355
E.3.4 USERS 355
E.4 与同步相关的视图 355
E.4.1 SYNC_FAILED_MESSAGES 355
E.4.2 SYNC_FAILED_MASTER_MESSAGES	. . . 356
E.4.3 SYNC_ACTIVE_MESSAGES 356
E.4.4 SYNC_ACTIVE_MASTER_MESSAGES	. . . 357

附录 F. 数据库虚拟表 359

F.1 SYS_LOG 359
-------------	---------------

附录 G. 系统存储过程 363

G.1 与同步相关的存储过程 363
G.1.1 SYNC_SETUP_CATALOG 363
G.1.2 SYNC_REGISTER_REPLICA 364
G.1.3 SYNC_UNREGISTER_REPLICA 365
G.1.4 SYNC_REGISTER_PUBLICATION 366
G.1.5 SYNC_UNREGISTER_PUBLICATION 367
G.1.6 SYNC_SHOW_SUBSCRIPTIONS 368
G.1.7 SYNC_SHOW_REPLICA_SUBSCRIPTIONS	369
G.1.8 SYNC_DELETE_MESSAGES 370
G.1.9 SYNC_DELETE_REPLICA_MESSAGES	371
G.2 其他存储过程 372
G.2.1 SYS_GETBACKGROUNDJOB_INFO	. . . 372

附录 H. 系统事件 373

H.1 其他事件 373
H.2 导致 SYS_EVENT_ERROR 的错误 379
H.3 导致 SYS_EVENT_MESSAGES 的条件或警告	380

索引 383

声明 397



1. 同步拉取通知	54	4. 执行图 1	119
2. 示例: 带有引用约束的表	94	5. 执行图 2	121
3. 自引用约束.	95		

表

1. 印刷约定	xv	48. IMPORT 返回值.	222
2. 语法表示法约定	xvi	49. MESSAGE APPEND 返回值.	229
3. 数据库表示例	1	50. 来自副本数据库的 MESSAGE BEGIN 返回值	231
4. 数据库表示例	7	51. 来自主数据库的 MESSAGE BEGIN 返回值	231
5. 数据库表示例	7	52. 来自副本数据库的 MESSAGE DELETE 返回	
6. 比较运算符.	27	值	232
7. 逻辑运算符: NOT	28	53. 来自主数据库的 MESSAGE DELETE 返回值	232
8. 逻辑运算符: AND	28	54. MESSAGE DELETE CURRENT	
9. 逻辑运算符: OR	28	TRANSACTION 返回值	234
10. 确定参数的数据类型.	37	55. 来自副本数据库的 MESSAGE END 返回值	235
11. BEFORE/AFTER 触发器的		56. 来自主数据库的 MESSAGE END 返回值	235
INSERT/UPDATE/DELETE 操作.	63	57. MESSAGE EXECUTE 返回值	236
12. 示例条目 1.	65	58. 来自副本数据库的 MESSAGE FORWARD 返	
13. 示例条目 2.	66	回值	239
14. 存储过程中的序列	73	59. 来自主数据库的 MESSAGE FORWARD 返回	
15. 系统角色	85	值	240
16. 查看表并授予访问权.	88	60. MESSAGE FROM REPLICA EXECUTE 返回	
17. 表达式和运算符	97	值	241
18. “SQL 信息”级别.	115	61. 来自副本数据库的 MESSAGE GET REPLY 返	
19. EXPLAIN PLAN FOR 单元.	116	回值	243
20. EXPLAIN PLAN FOR 表格列	117	62. 来自主数据库的 MESSAGE GET REPLY 返回	
21. 单元 INFO 列中的文本	117	值	245
22. EXPLAIN PLAN FOR, 示例 1.	119	63. MESSAGE GET REPLY 结果集表.	245
23. EXPLAIN PLAN FOR, 示例 2.	120	64. REFRESH 返回值	247
24. 对性能不佳问题进行诊断.	131	65. REVOKE REFRESH 返回值.	251
25. ADMIN COMMAND 语法和选项	134	66. SAVE 返回值	253
26. ALTER TABLE SET HISTORY COLUMNS		67. SAVE PROPERTY 返回值	255
返回值	147	68. SET SYNC 返回值.	264
27. ALTER TABLE SET SYNCHISTORY 返回值	149	69. SET SYNC CONNECT 返回值.	265
28. ALTER USER 返回值.	151	70. 不同的操作如何应用于同步历史记录表	266
29. 参数方式的比较.	166	71. SET SYNC MODE 返回值	266
30. 控制语句	170	72. SET SYNC NODE 返回值	268
31. CREATE PUBLICATION 返回值	182	73. SET SYNC PARAMETER 返回值.	269
32. 系统角色	184	74. UNLOCK TABLE 返回值	277
33. 存储过程中的序列	187	75. check_condition	279
34. CREATE SYNC BOOKMARK 返回值	188	76. data_type	280
35. 触发器中的语句原子性	196	77. 表达式.	280
36. DROP MASTER 返回值	201	78. query_specification	282
37. DROP PUBLICATION 返回值	202	79. 搜索条件	282
38. DROP PUBLICATION REGISTRATION 返回		80. table_reference	283
值	203	81. 伪列	284
39. DROP REPLICA 返回值	204	82. 日期和时间字面值	284
40. DROP SUBSCRIPTION 返回值.	206	83. 通配符.	284
41. DROP SYNC BOOKMARK 返回值	207	84. 字符串函数	287
42. EXPLAIN PLAN FOR 单元.	209	85. 数字函数	288
43. EXPLAIN PLAN FOR 表格列	210	86. 日期时间函数	290
44. EXPORT SUBSCRIPTION 返回值.	212	87. 系统函数	292
45. EXPORT SUBSCRIPTION TO REPLICA 返回		88. 其他函数	293
值	213	89. GET_PARAM 返回值	294
46. GRANT REFRESH 返回值	215	90. PUT_PARAM() 返回值	295
47. solidDB-支持提示	219	91. 数据类型表中的缩写	297

92. 字符数据类型	297	148. SYS_SYNC_INFO	341
93. 数字数据类型	299	149. SYS_SYNC_MASTER_MSGINFO	342
94. 二进制数据类型	300	150. SYS_SYNC_MASTER_	
95. 日期数据类型	301	RECEIVED_BLOB_REFS	343
96. 时间数据类型	301	151. SYS_SYNC_MASTER_RECEIVED	
97. 时间戳记数据类型	302	_MSGPARTS	343
98. 最小的可能非零数值	302	152. SYS_SYNC_MASTER_RECEIVED_MSGS	343
99. 保留字列表	305	153. SYS_SYNC_MASTER_STORED_BLOB_REFS	344
100. SQL_LANGUAGES 系统表	319	154. SYS_SYNC_MASTER_STORED_MSGPARTS	344
101. SYS Attauth	319	155. SYS_SYNC_MASTER_STORED_MSGS	344
102. SYS_AUDIT_TRAIL 表定义	320	156. SYS_SYNC_MASTER_SUBSC_REQ	345
103. SYS_BACKGROUNDJOB_INFO	321	157. SYS_SYNC_MASTER_VERSIONS	345
104. SYS_BLOBS	321	158. SYS_SYNC_MASTERS	346
105. SYS_CARDINAL	322	159. SYS_SYNC_RECEIVED_BLOB_ARGS	346
106. SYS_CATALOGS	322	160. SYS_SYNC_RECEIVED_STMTS	347
107. SYS_CHECKSTRINGS	322	161. SYS_SYNC_REPLICA_MSGINFO	348
108. SYS_COLUMNS	323	162. SYS_SYNC_REPLICA_RECEIVED_	
109. SYS_COLUMNS_AUX	324	BLOB_REFS	348
110. SYS_DL_REPLICA_CONFIG	324	163. SYS_SYNC_REPLICA_RECEIVED_	
111. SYS_DL_REPLICA_DEFAULT	324	MSGPARTS	349
112. SYS_EVENTS	325	164. SYS_SYNC_REPLICA_RECEIVED_MSGS	349
113. SYS_FEDT_DB_PARTITION 表定义	325	165. SYS_SYNC_REPLICA_STORED_BLOB_REFS	350
114. SYS_FEDT_TABLE_PARTITION 表定义	326	166. SYS_SYNC_REPLICA_STORED_MSGS	350
115. SYS_FORKEYPARTS	326	167. SYS_SYNC_REPLICA_STORED_MSGPARTS	350
116. SYS_FORKEYS	327	168. SYS_SYNC_REPLICA_VERSIONS	350
117. SYS_INFO	327	169. SYS_SYNC_REPLICAS	351
118. SYS_KEYPARTS	327	170. SYS_SYNC_SAVED_BLOB_ARGS	351
119. SYS_KEYS	328	171. SYS_SYNC_SAVED_STMTS	352
120. SYS_LOGPOS 表定义	329	172. SYS_SYNC_TRX_PROPERTIES	352
121. SYS_PROCEDURES	329	173. SYS_SYNC_USERMAPS	353
122. SYS_PROCEDURE_COLUMNS	330	174. SYS_SYNC_USERS	353
123. SYS_PROPERTIES	330	175. COLUMNS	354
124. SYS_RELAUTH	331	176. SERVER_INFO	354
125. SYS_SCHEMAS	331	177. TABLES	355
126. SYS_SEQUENCES	331	178. USERS	355
127. SYS_SERVER 表定义	332	179. SYNC_FAILED_MESSAGES	356
128. SYS_SYNC_REPLICA_PROPERTIES	332	180. SYNC_FAILED_MASTER_MESSAGES	356
129. SYS_SYNONYM	333	181. SYNC_ACTIVE_MESSAGES	356
130. SYS_TABLEMODES	333	182. SYNC_ACTIVE_MASTER_MESSAGES	357
131. SYS_TABLES	334	183. SYS_LOG 表定义	359
132. SYS_TRIGGERS	334	184. 记录标识描述	360
133. SYS_TYPES	335	185. 行数据的 DATA 类型描述	361
134. SYS_URole	335	186. DDL 数据的 DATA 类型描述	362
135. SYS_USERS	336	187. SYNC_SETUP_CATALOG 错误代码	363
136. SYS_VIEWS	336	188. SYNC_REGISTER_REPLICA 错误代码	364
137. SYS_BULLETIN_BOARD	336	189. SYNC_UNREGISTER_REPLICA 错误代码	365
138. SYS_PUBLICATION_ARGS	337	190. SYNC_REGISTER_PUBLICATION 错误代码	366
139. SYS_PUBLICATION_REPLICA_ARGS	337	191. SYNC_UNREGISTER_PUBLICATION 错误代	
140. SYS_PUBLICATION_REPLICA_STMTARGS	338	码	367
141. SYS_PUBLICATION_REPLICA_STMTS	338	192. CREATE PROCEDURE	
142. SYS_PUBLICATION_STMTARGS	338	SYNC_SHOW_SUBSCRIPTIONS 结果集	368
143. SYS_PUBLICATION_STMTS	339	193. SYNC_SHOW_SUBSCRIPTIONS 错误代码	368
144. SYS_PUBLICATIONS	339	194. SYNC_SHOW_REPLICA_SUBSCRIPTIONS 结	
145. SYS_PUBLICATIONS_REPLICA	340	果集	369
146. SYS_SYNC_BOOKMARKS	340	195. SYNC_SHOW_REPLICA_SUBSCRIPTIONS 错	
147. SYS_SYNC_HISTORY_COLUMNS	341	误代码	369

196. SYNC_DELETE_MESSAGES 错误代码	370	198. 其他事件	373
197. SYNC_DELETE_REPLICA_MESSAGES 错误代 码	371	199. 导致 SYS_EVENT_ERROR 的错误	379
		200. 导致 SYS_EVENT_MESSAGES 的警告	380

关于本手册

本指南介绍了关系数据库服务器原理和 SQL 编程语言。同时附加显示了 IBM® solidDB® 支持的所有 SQL 语句的语法，并描述了可以用于表和 SQL 语句的数据类型。

本指南面向想学习有关基本 SQL 和 solidDB 特殊 SQL 的用户。

印刷约定

solidDB 文档使用下列印刷约定：

表 1. 印刷约定

格式	适用于
数据库表	此字体用于所有普通文本。
NOT NULL	采用此字体的大写字母指示 SQL 关键字和宏名称。
solid.ini	这些字体指示文件名和路径表达式。
SET SYNC MASTER YES; COMMIT WORK;	此字体用于程序代码和程序输出。示例 SQL 语句也使用此字体。
run.sh	此字体用于样本命令行。
TRIG_COUNT()	此字体用于函数名。
java.sql.Connection	此字体用于接口名称。
LockHashSize	此字体用于参数名、函数自变量和 Windows 注册表条目。
<i>argument</i>	此类强调词指示用户或应用程序必须提供的信息。
管理员指南	这种样式用于引用其他文档或者同一文档中的章节。新术语和强调的问题也按此样式书写。
文件路径表示	除非另有声明，否则文件路径按 UNIX 格式表示。斜杠 (/) 字符表示安装根目录。
操作系统	如果文档包含有关操作系统之间的差别的内容，那么首先提到的是 UNIX 格式。Microsoft Windows 格式位于 UNIX 格式之后并括在括号中。其他操作系统将单独列出。对于不同的操作系统还可能有不同的章节进行描述。

语法表示法约定

solidDB 文档使用下列语法表示法约定:

表 2. 语法表示法约定

格式	适用于
INSERT INTO <i>table_name</i>	语法描述采用此字体。可替换部分采用此字体。
solid.ini	此字体指示文件名和路径表达式。
[]	方括号指示可选项；如果是粗体文本，那么必须将方括号包含在语法中。
	竖线，用于将语法行中的两个互斥选项分隔开。
{ }	大括号用于对语法行中的一组互斥选项进行定界；如果是粗体文本，那么必须将大括号包括在语法中。
...	省略号指示可以多次重复使用自变量。
· · ·	由三个点组成的一列表示这是先前代码行的延续。

1 数据库概念

如果您不熟悉关系数据库服务器（例如 solidDB），那么可以阅读本章。

本章解释下列概念：

- 关系数据库
 - 表、行和列
 - 关联不同表中的数据
- 多用户功能/并行控制和锁定
- 客户机/服务器体系结构
- 事务
- 事务日志记录与恢复

1.1 关系数据库

1.1.1 表、行和列

大多数关系数据库服务器（其中包括 solidDB 系列）使用称为结构化查询语言（SQL）的编程语言。SQL 是面向集合的编程语言，旨在允许人们查询和更新包含信息的表。本章讨论表以及数据在表中的表示方式。本手册的后续内容将更详细地讨论 SQL 语言的语法。

所有信息都存储在表中。表由行和列构成。（SQL 理论家将列称为“属性”，将行称为“元组”，但我们使用更为人所熟知的术语“列”和“行”。我们还将交替使用术语“记录”和“行”。）每个数据库都包含 0 个或更多个表。大部分数据库包含多个表。下面是一个表的示例。

表 3. 数据库表示例

ID	NAME	ADDRESS
1	Beethoven	23 Ludwig Lane
2	Dylan	46 Robert Road
3	Nelson	79 Willie Way

这个表包含 3 行数据。（这里显示包含标签“ID”、“NAME”和“ADDRESS”的顶“行”仅仅是为了便于读者阅读。数据库中实际的表未包含这样的行。）此表包含 3 个列（ID、NAME 和 ADDRESS）。

SQL 提供了用于创建表、将行插入到表中、更新表中的数据、从表中删除行以及查询表中的行的命令。

与 C 语言之类的编程语言不同，SQL 中的表不是同构表。在 SQL 中，一个列可以具有一种数据类型（例如 INTEGER），而相邻列可以具有截然不同的数据类型（例如 CHAR(20)，即 20 个字符的数组）。

表中的行数可以变化。您随时可以插入和删除行；不需要为最大数目的行预先分配空间。（所有数据库服务器对能够处理的最大行数都有所限制。例如，大部分运行于 32 位操作系统上的数据库服务器的限制都约为 20 亿行。在大部分应用程序中，此最大值已远远超出您的可能需求。）

每一行（“记录”）都必须至少具有一个唯一的值或值组合。如果表中有两个名为 David Jones 的作曲家，并且我们只需要更新其中一位作曲家的地址，那么就需要通过某种方式对他们进行区分。在某些情况下，您可以找到唯一的列组合，即使找不到任何包含唯一值的单一列亦如此。例如，如果姓名列不足以确保唯一型，那么姓名与地址的组合可能唯一。但是，在事先不知道所有数据的情况下，难以绝对保证每个值都唯一。大部分数据库设计者会添加一个“额外”的列，而其用途仅仅是唯一并且方便地标识每个记录。例如，在上表中，ID 号是唯一的。您可能已注意到，当我们实际尝试更新或删除记录时，我们通过记录的唯一标识来标识该记录（例如，“... WHERE id = 1”），而不是使用另一个可能不唯一的值（例如名称）。

1.1.2 关联不同表中的数据

如果 SQL 每次只能处理一个表，那么确实很方便，但功能却有所欠缺。SQL 和关系数据库的实力依赖于一个事实，即，各个表可以通过有用的方式相互相关，并且 SQL 查询能够从多个表中收集数据并以合乎逻辑的方式显示该数据。

我们将使用银行作为示例来说明多个表的实用性。

银行的每个客户可以有多个帐户。对于一个客户可以拥有的帐户数，没有实际的限制。一个客户可以拥有支票帐户、储蓄帐户、存单、抵押和信用卡等等。此外，客户还可以拥有多个相同类型的帐户。例如，客户可能有一个退休金储蓄帐户，并可以开设另一个相同类型的储蓄帐户来储蓄其女儿的大学学费。我们将客户与其帐户之间的“关系”描述为“一对多”关系，即，一个客户可以有多个帐户。

由于对客户可以拥有的帐户数没有限制，因此无法事先设计一种能够处理各个帐户之间的所有可能组合的记录结构。并且，如果创建一个记录结构来存放任何客户实际拥有的最大数目个帐户，那么必然会浪费大量空间。假定我们尝试构建一个表来存放所有关于一个银行客户及其帐户的信息。我们的第一份草稿可能类似于：

```
Customer ID Number
Customer Name
Customer Address
Checking Account #1 ID
Checking Account #1 Balance
CD #1 ID
CD #1 Balance
CD #2 ID
CD #2 Balance
...
```

正如您所见，由于对每个客户可以拥有的帐户数没有明显的限制，所以我们不知道何时停止。

另一种解决方案是创建多个记录（即，为每个帐户创建一个记录）并为每个帐户复制客户信息。因此，表如下所示：

Customer Name
Customer Address
Account ID
Account Balance

即使客户有多个帐户，我们也只需要为每个帐户创建一个完整的记录。这种方法的效果比较好，但也意味着每个帐户记录都包含所有关于该客户的信息。这将浪费存储空间，并且还导致难以在客户搬迁时更新客户的地址（可能必须在多个位置更新该地址）。

您可以通过关系数据库（例如 solidDB 的关系数据库）解决此问题。我们将为客户创建一个表，并为帐户创建另一个表。（在现实的银行中，我们可以将帐户分到多个表中，例如一个表用于支票帐户，另一个表用于储蓄帐户等等。）然后我们在客户和她的每个帐户之间建立“链接”。这样，浪费的空间就非常少，但可供我们使用的信息仍相当全面。

正如我们前面提到的那样，在作曲家示例中，每个记录都应该有唯一键，以使我们能够标识该记录。通常，唯一值仅仅是一个整数。我们将使用该唯一整数来帮助我们将客户与其帐户“相关”。第 7 页的 2，《SQL 入门》将对此进行更详细的讨论。

为客户创建帐户时，我们存储该客户的标识号作为帐户信息的组成部分。具体而言，帐户表中的每一行都包含一个 `customer_id`，该 `customer_id` 值与拥有该帐户的客户的标识匹配。Smith 的客户标识为 1，因此 Smith 的每个帐户的 `customer_id` 字段都包含 1。这意味着，我们可以通过执行以下操作找到 Smith 的所有帐户记录：

1. 在 `customers` 表中查找 Smith 的记录。
2. 找到 Smith 的记录后，查看该记录中的 `id` 号。（对于 Smith 而言，`id` 为 1。）
3. 现在，在 `accounts` 表中查找所有其 `customer_id` 字段值为 1 的帐户。

这就像是子女上学前，您在每个子女的额头贴上家庭电话号码。当您有急事并需要请一位出租车司机去学校接子女回家时，您只需将电话号码告知该司机，他将检查学校中的每个学生，以确定该学生的电话号码是否与您的电话号码相同。（这样做的效率并不高，但却有效。）知道了父母的标识号，就可以标识所有子女。相反，知道了每个子女，就可以标识父母。例如，如果您的其中一个子女在校外野营时迷路，那么任何善意人士都可以看到该子女额头上的电话号码并致电给您。

正如您所见，父母与子女相互链接，尽管不存在任何物理形式的联系。仅仅有标识号（电话号码）便足以确定父母的子女以及每个子女的父母。无论您有多少子女，这种技术都有效。

关系数据库使用同一技术。注意，连接操作并不限于在两个表之间进行。您可以对几乎任意数目的表创建连接。作为对我们的银行示例的现实扩展，我们可以创建另一个表“checks”来存放关于所签署的每张支票的信息。因此，不仅存在从每个客户到其帐户的一对多关系，还存在从每个支票帐户到针对该帐户签署的所有支票的一对多关系。当然，您可以编写一个查询来列示客户已签署的所有支票，即使该客户有多个支票帐户亦如此。

1.2 客户机/服务器体系结构

solidDB 使用客户机/服务器模型。在客户机/服务器模型中，单一“服务器”可以处理来自一个或多个“客户机”的请求。这与饭店的工作方式相当类似 - 一位服务员和厨师可以处理来自许多客户的请求。

在客户机/服务器数据库模型中，服务器是一个专用的计算机程序，它知道如何高效地存储和检索数据。服务器通常接受四种基本类型的请求：

- 插入新信息
- 更新现有的信息
- 检索现有的信息
- 删除现有的信息

服务器能够存储几乎任何类型的数据，但通常不知道数据的“含义”。服务器通常对记帐和库存之类的“业务问题”了解甚少或完全不了解。它不知道特定信息是库存记录、银行存款描述还是歌曲“American Pie”的数字化副本数据库。

“客户机”负责了解有关特定业务问题以及数据“含义”的信息。例如，我们可以编写一个了解记帐信息的客户机程序。例如，此客户机程序可能知道如何计算逾期付款的利息。或者，此客户机可能能够识别特定数据是歌曲，并能够将数字数据转换为模拟音频输出。

您可以编写既执行“客户机”工作也执行“服务器”工作的单一程序。读取并播放数字化音乐的程序还可以将该数据存储到磁盘并根据请求执行查找。但是，每间公司都编写自己的数据存储和检索例程并不是高效的做法。通常，更为高效的做法是，购买通用性足以满足需求但性能相对较高的现成数据存储解决方案。

1.3 多用户能力

客户机/服务器体系结构的一项重要优点是，它通常使您能够方便地使用不止一个客户机。与大多数关系数据库服务器相同，solidDB 允许多个用户访问表中的数据。

两个用户尝试更新同一数据时，有可能会发生危险。如果更新不相同，那么一个用户的更新可能会覆盖另一用户的更新。solidDB 使用并行控制机制来防止这种情况发生。有关更多信息，请参阅第 103 页的 5.2，『并行控制和锁定』。

1.4 事务

SQL 允许将多个语句分组到称为事务的单一“原子”（不可分）工作中。例如，如果您在杂货店签署一张支票，那么从您的帐户扣除款项时，杂货店的银行帐户将立即接收到该款项。如果您支付款项但杂货店未接收到该款项，或者杂货店接收到款项但您的帐户未支付该款项，那么没有意义。如果任何一项操作（将款项加入杂货店的帐户或者从您的帐户中扣除款项）失败，那么另一项操作也应该失败。如果两个语句在同一事务中，其中一个语句失败，您可以用 `ROLLBACK` 命令将其还原为事务开始前的状态 - 这避免了事务只完成一半情况的发生。如果我们金融事务的这两部分成功，我们希望数据库事务也成功。成功的事务由命令 `COMMIT WORK` 保留。以下是一个简单的示例。

```
COMMIT WORK; -- Finish the previous transaction.
UPDATE stores SET balance = balance + 199.95
  WHERE store_name = 'Big Tyke Bikes';
UPDATE checking_accounts SET balance = balance - 199.95
  WHERE name = 'Jay Smith';
COMMIT WORK;
```

1.5 事务日志记录与恢复

购置商用数据库服务器的其中一项主要优点是，大部分此类服务器的设计都能够在数据库服务器由于任何原因（例如电源故障、硬件故障或数据库软件本身的故障）而意外关闭时保护数据。

可以通过许多不同的方法来帮助保护数据。我们着重阐述一种称为“事务日志记录”的方法。

1.5.1 背景

假定您正在将数据写入磁盘驱动器或者其他永久存储介质，但电源突然发生故障。您所写的数据库可能未能完全写入到存储器。例如，您可能尝试写帐户余额“122.73”，但由于电源故障而只写了“12”。帐户丢失金钱的人员肯定会相当不满。我们如何确保始终写完整的数据？部分解决方案是使用“事务日志”。

注：

在计算机世界，存在各种各样的“日志”。例如，solidDB 写多个日志文件，其中包括事务日志文件和错误消息日志文件。现在，我们只讨论事务日志文件。

正如我们前面提到的那样，工作通常在“事务”中完成。程序将落实或回滚整个事务。不允许不完整的事务。对于这里描述的情况，即，我们开始写某个人的新帐户余额但在完成操作前发生断电，我们希望回滚此事务。任何已完成并已正确写入磁盘的事务都应该保留下来。

为了帮助我们跟踪已成功写入哪些数据以及尚未成功写入哪些数据，我们实际上将数据分别写入“事务日志”以及数据库表。事务日志实际上是已执行的操作（即，已落实的事务）的线性序列。此文件包含用于指示每个事务结束的标记。如果文件中的最后一个事务没有“事务结束”标记，那么我们知道该片段事务未完成，因此应该回滚而非落实该事务。

当服务器在发生故障后重新启动时，它将读取事务日志并逐个应用已完成的事务。换言之，它将使用事务日志文件中的信息来更新数据库中的表。此过程称为“恢复”。正确完成的恢复过程甚至能够保护恢复过程本身不受电源故障影响。

这里并不是有关事务日志记录功能如何防止数据损坏的完整描述。我们已说明服务器如何确保不丢失事务。但是，我们未真正说明服务器如何防止数据库文件由于服务器将记录写入磁盘驱动器中的表期间发生写故障而损坏。该主题更为高级，在这里不作讨论。

1.6 摘要

这个关系数据库简介解释了您开始使用关系数据库前需要了解的概念。现在，您应该能够回答下列问题：

什么是表、行和列？

能否同时处理多个表中的数据？

事务如何帮助确保数据一致？

为何将事务数据写（“记录”）到磁盘驱动器？

2 SQL 入门

本章帮助您快速了解或重温 SQL 概述。

2.1 表、行和列

SQL 是面向集合的编程语言，旨在允许人们查询和更新包含信息的表。

所有信息都存储在表中。表由行和列构成。（SQL 理论家将列称为“属性”，将行称为“元组”，但我们使用更为人所熟知的术语“列”和“行”。我们还将交替使用术语“记录”和“行”。）每个数据库都包含 0 个或多个表。大部分数据库包含多个表。下面是一个表的示例。

表 4. 数据库表示例

ID	NAME	ADDRESS
1	Beethoven	23 Ludwig Lane
2	Dylan	46 Robert Road
3	Nelson	79 Willie Way

此表包含 3 行数据。（这里显示包含标签“ID”、“NAME”和“ADDRESS”的顶“行”仅仅是为了便于读者阅读。数据库中实际的表未包含这样的行。）此表包含 3 个列（ID、NAME 和 ADDRESS）。SQL 提供了用于创建表、将行插入到表中、更新表中的数据、从表中删除行以及查询表中的行的命令。

2.2 SQL

以下 SQL“程序”创建以下实例中所示的表:

```
CREATE TABLE composers (id INTEGER PRIMARY KEY, name CHAR(20),  
address CHAR(50));  
INSERT INTO composers (id, name, address) VALUES (1, 'Beethoven',  
'23 Ludwig Lane');  
INSERT INTO composers (id, name, address) VALUES (2, 'Dylan',  
'46 Robert Road');  
INSERT INTO composers (id, name, address) VALUES (3, 'Nelson',  
'79 Willie Way');
```

表 5. 数据库表示例

ID	NAME	ADDRESS
1	Beethoven	23 Ludwig Lane
2	Dylan	46 Robert Road
3	Nelson	79 Willie Way

列“id”指定为该表的“主键”。这意味着，通过使用此列，可以唯一的标识每一行。从现在起，系统将保证“id”的值唯一并且始终存在（即它具有 NOT NULL 属性）。

如果 Dylan 先生搬迁到 61 Bob Street，那么您可以使用以下命令来更新他的数据：

```
UPDATE composers SET ADDRESS = '61 Bob Street' WHERE ID = 2;
```

由于每个作曲家的 ID 字段都是唯一的，并且由于此命令中的 WHERE 子句仅指定了一个 ID，所以将仅对一位作曲家执行此更新操作。

如果 Beethoven 先生辞世，并且您需要删除他的记录，那么可以使用以下命令执行此操作：

```
DELETE FROM composers WHERE ID = 1;
```

最后，如果要列示表中的所有作曲家，那么可以使用以下命令：

```
SELECT id, name, address FROM composers;
```

注意，与上面列示的 UPDATE 和 DELETE 语句不同，SELECT 语句未包含 WHERE 子句。所以，此命令将应用于所指定表中的所有记录。因此，此 SQL 语句的结果是选择（并列示）该表中列示的所有作曲家。

ID	NAME	ADDRESS
1	Beethoven	23 Ludwig Lane
2	Dylan	46 Robert Road
3	Nelson	79 Willie Way

注意，虽然输入的字符串带有引号，但它们在显示时将不带引号。

以上简单命令帮助显示有关 SQL 的一些重要事项。

- SQL 是相对“高级”的语言。单一命令可以创建包含任意数目的列的表。同样，单一命令可以执行几乎任意复杂的更新。虽然在这里未提供示例，但您确实可以同时更新多个列，甚至可以同时更新多行。可以在单一 SQL 命令中执行需要数十或数百行代码（例如 C 或 Java 语言代码）才能完成的操作。
- 与某些其他计算机语言不同，SQL 使用单引号对字符串进行定界。例如，'Beethoven' 是一个字符串。“Beethoven”是别的内容。（在技术上，这是定界标识，本章对此不作讨论。）如果您熟悉 C 之类使用双引号对字符串进行定界（字符数组）并使用单引号对单个字符进行定界的编程语言，那么必须适应 SQL 的工作方式。

虽然以上示例未作阐明，但确实存在您需要了解的其他一些有关基本 SQL 的事项：

- 虽然 SQL 是功能非常高级的语言，但也有很多局限性。SQL 用于执行面向表和记录的操作。它只能执行非常少的低级操作。例如，无法直接打开文件或者将位向左或向右移。SQL 还独立于硬件，这既是优点也是缺点。您无权控制 SQL 查询的输出格式；您可以选择列序，并通过使用 ORDER BY 子句可以控制行序，但是不可以进行一些操作，如，控制屏幕上的字体大小、或者在输出的每个已输出页面下方的输出页面数。SQL 并不是 C、Java 和 Pascal 之类的综合性编程语言。
- 每种 SQL 实现都有一组固定的数据类型。solidDB 以及大多数其他 SQL 实现中的数据类型包括整型（INTEGER）、字符数组（CHAR）、浮点型（FLOAT）、日期型（DATE）和时间型（TIME）。
- SQL 通常是“解释型”语言，而不是“编译型”语言。要执行一个或多个 SQL 语句，您通常执行一个独立的程序，该程序读取并执行脚本。既不会生成“经过编译的程序”或“可执行文件”，也不会将其存储下来以供将来使用。每次运行程序时，都将再次

解释此程序。（存储过程可以复用，而不必进行重新解释。第 133 页的附录 A，《语句》对存储过程作了简要讨论，第 21 页的 3，《SQL 扩展》对其进行详尽阐述。）

- 在 SQL 中，表名和列名不区分大小写。在我们的示例中，关键字（例如 CREATE、INSERT 和 SELECT）是大写的，而表名和列名是小写的。但是，这仅仅是约定，而不是必要条件。
- SQL 对于命令是写成一行还是分为多行也不挑剔。在本章的随后内容中，有多行语句的示例。
- SQL 命令可以极为复杂，即，在查询中嵌套多“层”查询。确定如何编写复杂查询可能相当困难 - 确定如何理解别人编写的查询也同样困难。与任何其他编程语言相同，最好提供代码的文档！
- 为了帮助您提供代码的文档，SQL 允许添加“注释”。注释仅供人员阅读；SQL 解释器将跳过这些注释。要创建注释，您有两个选项：
 - 行注释：在行的始端放两个破折号 (--) 并使用换行符结束该注释。该注释不可以扩展到一个新行。
 - 块（多线）注释：用一个斜杠和星号 (/*) 开始注释并用一个星号和斜杠 (*/) 结束注释。

直到该行末尾为止的所有后续字符都将被忽略。（“优化器提示”例外，这是本章所未讨论的另一个高级主题。）

2.3 SQL 的数学起源

最初，关系数据库和 SQL 部分基于集合论的数学概念。如果您熟悉集合论，那么有助于您理解关系数据库的工作方式。即使您不熟悉集合论，也不必担心；这仅仅是一种看待关系数据库和 SQL 的方式。

您可以将表想像成数学集合，而集合中的每个元素都是一行。（在以上示例中，每个人/作曲家都是集合的元素。表包含“作曲家”集合的所有元素。）在数学中，集合是无序的。同样，在 SQL 中，很大程度上可把表视为无序，甚至如果您查看磁盘上的位和字节，会发现在任何给定时间，记录都按特定顺序存储。

不进行排序十分重要，原因是这表示您每次运行查询时，查询结果可以按不同的顺序显示。在单一磁盘驱动器上存储小型数据集时，您通常会发现相同的行每次都具有同一顺序，但将数据分布到多个文件或磁盘驱动器时，情况不一定如此。

由于 SQL 是面向集合的语言，因此您可以使用此语言来执行某些面向集合的操作，例如 UNION（即，将两个输入集合组合成一个输出集合）。但是，UNION 之类的操作要求集合相互匹配 - 即，它们包含相同数目的列，并且对应列的数据类型相同或兼容。例如，如果集合 1 中的第一列的类型为 DATETIME，集合 2 中的第一列的类型为 INTEGER，那么无法执行 UNION 操作。

同样，即使您不习惯使用集合论，也不必担心。这仅仅是另一种看待关系数据库的方式。

2.4 创建具有相关数据的表

正如上一章所述，银行的每个客户都有多个帐户。我们将客户与其帐户之间的“关系”描述为“一对多”关系，即，一个客户可以有多个帐户。

由于对客户可以拥有的帐户数没有限制，因此无法事先设计一种能够处理各个帐户之间的所有可能组合的记录结构。

关系数据库旨在解决此问题。我们将为客户创建一个表，并为帐户创建另一个表。（在现实的银行中，我们可以将帐户分到多个表中，例如一个表用于支票帐户，另一个表用于储蓄帐户等等。）然后，我们在客户与其每个帐户之间创建“链接”。这样，浪费的空间就非常少，但可供我们使用的信息仍相当全面。

正如我们前面提到的那样，在作曲家示例中，每个记录都应该有主键，以使我们能够标识该记录。主键通常只是一个整数。现在，我们将使用该唯一整数来帮助我们使客户与其帐户“相关”。以下是用于创建和填充客户表的命令：

```
CREATE TABLE customers (id INTEGER PRIMARY KEY, name CHAR(20),
address CHAR(40));
INSERT INTO customers (id, name, address) VALUES (1, 'Smith',
'123 Main Street');
INSERT INTO customers (id, name, address) VALUES (2, 'Jones',
'456 Fifth Avenue');
```

我们已插入了两个分别名为 Smith 和 Jones 的客户。现在，让我们创建帐户表：

```
CREATE TABLE accounts (id INTEGER PRIMARY KEY, balance FLOAT,
customer_id INT REFERENCES customers);
```

这里，我们指定列 *customer_id* 作为指向客户表的“外键”（由 REFERENCES 关键字指示）。此列的值应该与“customers”表中相应客户行中的“id”值（主键）完全相同。这样，我们将使帐户行与客户行相关联。数据库的功能允许以一种称为“引用完整性”的可靠方式来维护此类关系，用于定义此类关系的相应 SQL 语法元素称为“引用完整性约束”。有关引用完整性的更多信息，请参阅第 93 页的 4.5，『引用完整性』。

客户 Smith 有两个帐户，客户 Jones 有一个帐户。

```
INSERT INTO accounts (id, balance, customer_id)
VALUES (1001, 200.00, 1);
INSERT INTO accounts (id, balance, customer_id)
VALUES (1002, 5000.00, 1);
INSERT INTO accounts (id, balance, customer_id)
VALUES (1003, 222.00, 2);
```

由于 Smith 有两个帐户，因此 Smith 的每个帐户的 *customer_id* 字段值都是 1。这意味着，用户可以通过执行以下操作找到 Smith 的所有帐户记录：

1. 在 customers 表中查找 Smith 的记录。
2. 找到 Smith 的记录后，查看该记录中的 id 号。（对于 Smith 而言，id 为 1。）
3. 现在，在 accounts 表中查找所有其 *customer_id* 字段值为 1 的帐户。

这就像是子女上学前，您在每个子女的额头贴上家庭电话号码。当您有急事并需要请一位出租车司机去学校接子女回家时，您只需将电话号码告知该司机，他将检查学校中的每个学生，以确定该学生的电话号码是否与您的电话号码相同。（这样做的效率并不高，但却有效。）知道了父母的标识号，就可以标识所有子女。相反，知道了每个子女，就可以标识父母。例如，如果您的其中一个子女在校外野营时迷路，那么任何善意人士都可以看到该子女额头上的电话号码并致电给您。

正如您所见，父母与子女相互链接，尽管不存在任何物理形式的联系。仅仅有标识号（电话号码）便足以确定父母的子女以及每个子女的父母。无论您有多少子女，这种技术都有效。

关系数据库使用同一技术。由于我们已创建了客户表和帐户表，因此可以显示每个客户以及该客户的每个帐户。为此，我们使用 SQL 程序员称之为“连接”的操作。SELECT 语句中的 WHERE 子句将对帐户的 *customer_id* 号与客户 *id* 号匹配的那些记录对进行“连接”。

```
SELECT name, balance
  FROM customers, accounts
 WHERE accounts.customer_id = customers.id;
```

此查询的输出类似于：

```
NAME  BALANCE
Smith  200.0
Smith  5000.0
Jones  222.0
```

如果一个客户有多个帐户，那么她可能想知道所有帐户的总金额。计算机可以通过使用以下查询来提供此信息：

```
SELECT customers.id, SUM(balance)
  FROM customers, accounts
 WHERE accounts.customer_id = customers.id
 GROUP BY customers.id;
```

此查询的输出类似于：

```
NAME  BALANCE
Smith  5200.0
Jones  222.0
```

注意，这次 *Smith* 只出现了一次，并且显示了她的所有帐户的余额总计。

此查询使用了 **GROUP BY** 子句以及名为 **SUM()** 的聚集函数。**GROUP BY** 子句的主题提供了比这个 SQL 简介更为详尽的信息。此查询仅仅是向您简单介绍 SQL 在单一语句中所能完成的实用工作类型。在其他语言（例如 C）中获得同一结果需要许多语句。

注意，连接操作并不限于在两个表之间进行。您可以对几乎任意数目的表创建连接。作为对我们的银行示例的现实扩展，我们可以创建另一个表“checks”来存放关于所签署的每张支票的信息。因此，不仅存在从每个客户到其帐户的一对多关系，还存在从每个支票帐户到针对该帐户签署的所有支票的一对多关系。当然，您可以编写一个查询来列示客户已签署的所有支票，即使该客户有多个支票帐户亦如此。

2.4.1 表别名

SQL 允许您在某些查询中使用别名来代替表名。在以下的查询中，别名“a”代表 *accounts* 表，“c”代表 *customers* 表。

```
SELECT name, balance
  FROM customers c, accounts a
 WHERE a.customer_id = c.id;
```

别名在 FROM 子句中定义，在 WHERE 子句中使用。

2.4.2 子查询

SQL 允许一个查询包含另一个查询（称为“子查询”）。

回到我们的银行示例：随着时间的推移，某些客户添加帐户，也有一些客户销户。在某些情况下，客户可能会逐步销户，直到没有任何帐户为止。例如，我们的银行可能想标识所有没有任何帐户的客户，以便可以删除那些客户的记录。一种标识没有任何帐户的客户的方法是，使用子查询和 EXISTS 子句。

为了进行试验，我们需要创建一个没有任何帐户的客户：

```
INSERT INTO customers (id, name, address) VALUES (3, 'Zu', 'B St');
```

在列示所有没有帐户的客户之前，让我们先列示所有那些有帐户的客户。

```
SELECT id, name
FROM customers c
WHERE EXISTS (SELECT * FROM accounts a WHERE a.customer_id = c.id);
```

子查询（也称为“内层查询”）是括在括号中的查询。对于外层查询所选择的每个记录，内层查询都将执行一次。（此函数与另一个编程语言中嵌套循环的工作方式很相似，但对于 SQL 而言，我们可以在单一语句中执行嵌套循环。）如果外层循环所处理的特定客户有任何帐户，那么将把那些帐户记录返回给外层查询。

实际上，外层查询中的“EXISTS”子句指出“我们不关心那些记录中的值；我们只关心是否存在任何记录。”因此，如果客户有任何帐户，那么 EXISTS 将返回 true。如果客户没有任何帐户，那么 EXISTS 将返回 false。EXISTS 子句不关心是否存在多个帐户还是单一帐户。它不关心帐户所包含的值。所有 EXISTS 想要知道的都是“是否至少存在一个记录？”。

因此，整条语句列示那些至少有一个帐户的客户。无论客户有多少个帐户（只要至少有 1 个），都只列示该客户一次。

现在，让我们列示所有那些没有任何帐户的客户：

```
SELECT id, name
FROM customers c
WHERE NOT EXISTS (SELECT * FROM accounts a WHERE a.customer_id = c.id);
```

您只需添加关键字 NOT 即可掉转此查询的含义。

子查询本身可以包含子查询。实际上，子查询的嵌套深度几乎不受限制。

2.5 对于每种数据类型使用哪些格式

如上所示，SQL 要求以特定的方式表达值。例如，字符串必须由单引号定界。

此外，还必须正确地对其他值进行格式化。所需的确切格式取决于数据类型。除 CHAR 数据类型以外，还有多种数据类型要求使用单引号对输入的值进行定界。

以下示例说明如何格式化 solidDB 所支持的大部分数据类型的输入数据。我们将以简单 SQL 脚本的形式提供此示例，如果您愿意的话，可以执行此脚本。注意，在此脚本中，许多命令分为多行。这在 SQL 中是合法的。这是大部分 SQL 解释器期望使用分号来分隔每个 SQL 语句的其中一个原因，尽管 ANSI 的 SQL 标准实际上并未要求在每个语句的末尾指定分号。

```
CREATE TABLE one_of_almost_everything (
  int_col INTEGER,
  float_col FLOAT,
  string_col CHAR(20),
  wide_string_col WCHAR(20), -- "wide" 表示宽字符，例如 unicode。
```

```

varchar_col VARCHAR, -- Note that we did not have to specify width.
date_col DATE,
time_col TIME,
timestamp_col TIMESTAMP
);

INSERT INTO one_of_almost_everything (
    int_col,
    float_col,
    string_col,
    wide_string_col,
    varchar_col,
    date_col,
    time_col,
    timestamp_col
)
VALUES (
    1,
    2.0,
    'three',
    'four',
    'five point zero zero zero zero zero zero zero zero zero ...',
    '2002-12-31',
    '11:59:00',
    '1999-12-31 23:59:59.000000'
);

```

正如您所见，时间戳记值按“最高有效位”到“最低有效位”的顺序输入。同样，日期和时间值也按最高有效位到最低有效位的顺序输入。所有这三种数据类型（时间戳记、日期和时间）都使用标点符号来分隔各个字段。

要求特定格式的原因是，某些其他可能的格式具有二义性。例如，对于美国用户而言，“07-04-1776”是 1776 年 7 月 4 日，原因是美国人通常以“mm-dd-yyyy”格式或“mm/dd/yyyy”格式书写日期。但对于欧洲用户而言，此日期显然是 4 月 7 日，而不是 7 月 4 日，原因是大部分欧洲人以“dd-mm-yyyy”格式书写日期。虽然格式过多这一问题似乎无法通过另外添加一种格式来很好地解决，但 SQL 采用的这种以最高有效位开头并稳定地移至最低有效位的方法有一些优点。首先，这意味着全部三种数据类型（日期、时间和时间戳记）遵循同一规则。其次，日期格式和时间格式都是时间戳记格式的完美子集。第三，尽管需要另外记忆一种格式，但规则却相当简单并与“西方”语言书写数字的方式一致（最高有效位在最左边）。最后，通过做到与现有格式明显不兼容，不存在用户意外地写一个日期（例如“07-04-1776”）但让机器解释成另一个日期的机会。

2.5.1 BLOB（或二进制数据类型）

迄今为止，我们已讨论了用于存储由人读取的数据的数据类型。某些类型的数据并非由人直接读取，但仍可以存储在数据库中。例如，来自数码像机的图片或者来自 CD 的歌曲作为一系列数字存储。这些数字对于人而言几乎毫无意义。但是，数字化的图片和声音也可以作为 BINARY 数据存储。solidDB 支持三种二进制数据类型：BINARY、VARBINARY 和 LONG VARBINARY（即 BLOB）。

在大多数情况下，您将从 C 程序中使用 ODBC（开放式数据库连接）API 或者从 Java 程序中使用 JDBC API 来读写二进制数据。但是，也可以使用执行 SQL 语句的实用程序将数据插入到二进制字段中。要将值插入到二进制字段中，必须将该值表示为括在单引号中的一系列十六进制数字。例如，如果要将值分别为 1、9、11 和 255 的一系列字节插入到二进制字段中，那么可以执行以下 SQL 语句：

```
INSERT INTO table1 (binary_col) VALUES (CAST('01090BFF' AS VARBINARY));
```

由于此命令指示服务器将值强制转换 (CAST) 为 VARBINARY 类型, 因此服务器会自动将该字符串解释为一系列十六进制数字, 而不是解释为字符串字面值。

您也可以直接插入字符串字面值, 例如:

```
INSERT INTO table1 (binary_col) VALUES ('Thank you');
```

通过 solsql (用于执行 SQL 语句的 solidDB 实用程序) 检索数据时, 二进制列中的返回值以十六进制表示, 而无论您最初是否以十六进制格式输入该值。因此, 插入值 “Thank you” 之后, 如果从表中选择此值, 那么您将看到:

```
5468616E6B20796F75
```

其中, 54 表示大写 “T”, 68 表示小写 “h”, 61 表示小写 “a”, 6E 表示小写 “n”, 等等。

另请注意, 对于长值, 将仅显示前几个数字。

2.5.2 NULL IS NOT NULL (即, 如何在 SQL 中表达“并非上述任何一项”)

有时, 您没有足够的信息来完整地填写表单。SQL 使用关键字 NULL 来表示“未知”或“没有值”。(这与 NULL 在 C 之类的编程语言中的含义有所不同。) 例如, 如果您正在将 Joni Mitchell 的记录插入到作曲家表中, 并且您不知道 Joni Mitchell 的地址, 那么可以执行以下语句:

```
INSERT INTO composers (id, name, address) VALUES (5, 'Mitchell', NULL);
```

如果未指定地址字段, 那么它在缺省情况下将包含 NULL。

```
INSERT INTO composers (id, name) VALUES (5, 'Mitchell');
```

以下示例是显示值 NULL 的某些异常特征的样本程序。

```
-- Data of any data type may contain NULL.
-- For example, a column of type INTEGER may contain not
-- only valid integer values, but also NULL.

-- Set up for experiments...
CREATE TABLE table1 (x INTEGER, name CHAR(30));

-- The value NULL means "there is no value".
-- NULL is not the same as zero, or an empty string.
-- (It is also not a pointer value, as it is in
-- programming languages such as C.)
-- To help show this, insert 3 rows, one of which has
-- "normal" values, one of which has a 0 and an empty string,
-- and one of which has two NULL values.
INSERT INTO table1 (x, name) VALUES (2, 'Ludwig Von Beethoven');
INSERT INTO table1 (x, name) VALUES (0, '');
INSERT INTO table1 (x, name) VALUES (NULL, NULL);
-- This returns only the row containing 0,
-- not the row containing NULL.
SELECT * FROM table1 WHERE x = 0;
-- This returns only the row containing the empty string,
-- not the row containing NULL.
SELECT * FROM table1 WHERE name = '';

-- As expected, NULL does not match other values.
-- However NULL does not match even itself.
-- (A mathematician would say that NULL violates the
-- reflexive property "a = a".)
SELECT * FROM table1 WHERE x = x;
```



```

-- Since NULL does not equal NULL, what will the following query return?
SELECT * FROM table1 WHERE x <> x;

-- Similarly, although you might think that the
-- expression below is always true, it is actually
-- always false.
SELECT * FROM table1 WHERE NULL IN (NULL, 2);

-- The result set will contain 2 (since 2 is in
-- the set (NULL, 2)), but the result set will
-- not contain NULL.
SELECT * FROM table1 WHERE x IN (NULL, 2);

-- But suppose that you *want* to find all the records that
-- have NULL values. How to do that if it is not possible to say ... = NULL?
SELECT * FROM table1 WHERE x IS NULL;
-- And the opposite query is ...
SELECT * FROM table1 WHERE x IS NOT NULL;

-- Set up for more experiments...
CREATE TABLE parent (id INTEGER, name CHAR(20));
CREATE TABLE children (id INTEGER, name CHAR(12), parent_id INT);
INSERT INTO parent (id, name) VALUES (1, 'Smith');
INSERT INTO children (id, name, parent_id) VALUES (11, 'Smith child', 1);
INSERT INTO children (id, name, parent_id) VALUES (131, 'orphan', NULL);
INSERT INTO parent (id, name) VALUES (NULL, 'Has Null');

-- Since NULL <> NULL, if a "parent" record has NULL and a "child"
-- record has NULL, the child's value will not match the parent's value.
-- This result set will contain 'Smith', but not 'Has Null'.
SELECT p.name FROM parent p, children c
WHERE c.parent_id = p.id;

-- Note that a row that contains nothing but a
-- single NULL is still a row.
-- In the following query, an EXISTS clause is used.
-- It evaluates to TRUE if the subquery returns
-- any rows. Even a row that contains nothing but a
-- single NULL value is still a row, and so if the
-- subquery returns a single NULL the EXISTS clause
-- still evaluates to TRUE.
-- Even though the subquery below returns NULL rather than a name
-- or ID, the EXISTS expression evaluates to TRUE, and Smith is printed.
SELECT name FROM parent p
WHERE EXISTS(SELECT NULL FROM children c WHERE c.parent_id = p.id);

-- Now that you recognize that NULL <> NULL,
-- the following may confuse you since it breaks the pattern.
-- Contrary to what you might expect, the UNIQUE keyword
-- DOES filter out multiple NULL values.
INSERT INTO table1 (x, name) VALUES (NULL, 'any name');
-- Now the table has more than one row in which x is NULL,
-- but a query with UNIQUE nonetheless returns only a
-- single NULL value.
SELECT DISTINCT x FROM table1;
-- You may be interested to know that a UNIQUE index
-- will allow only a single NULL value. (Note that a primary key
-- will not allow any NULL values.)

```

```
-- Clean up.  
DROP TABLE parent;  
DROP TABLE children;  
DROP TABLE table1;
```

2.5.3 NOT NULL

与 NULL 相反，NOT NULL 是 SQL 的其中一个数据约束。NOT NULL 表明，在该表的任何一行中，都不允许指定的列包含 NULL 值。有关更多信息以及示例，请参阅第 133 页的附录 A，『语句』。

2.5.4 表达式与强制类型转换

SQL 允许在 SQL 语句的某些部分中使用表达式。例如，以下语句将一个列中的值乘以 12:

```
SELECT monthly_average * 12 FROM table1;
```

作为另一个示例，以下语句使用内置函数 SQRT 来计算列“variance”中每个值的平方根。

```
SELECT SQRT(variance) FROM table1;
```

下一个示例使用“REPLACE”函数将数字由美国格式转换为欧洲格式。在美国格式中，数字使用句点字符 (.) 作为小数点，但在欧洲格式中，使用逗号 (,)。例如，在美国，PI 可以适当地写作“3.14”，但在欧洲写作“3,14”。我们可以使用 REPLACE 函数将“.”字符替换为“,”字符。以下语句序列提供了示例。

```
CREATE TABLE number_strings (n VARCHAR);  
INSERT INTO number_strings (n) VALUES ('3.14'); -- 用美式输入。  
SELECT REPLACE(n, '.', ',') FROM number_strings; -- 用欧式输出。
```

输出将类似于

```
n  
-----  
3,14
```

注意，一个函数可以调用另一个函数。以下表达式先计算一个数字的平方根，然后计算该平方根的自然对数:

```
SELECT LOG(SQRT(x)) FROM table1;
```

solidDB SQL 并非在所有子句中都接受完全通用的表达式。例如，在 SELECT 子句中，可以使用预定义的函数，但不能调用已创建的存储过程。即使已创建名为“foo”的存储过程，以下语句也无效:

```
SELECT foo(column1) FROM table1;
```

使用表达式时，您可能想对列指定新名称。例如，在使用以下表达式时:

```
SELECT monthly_average * 12 FROM table1;
```

您可能不希望输出列名为“monthly_average”。solidDB 服务器实际上使用表达式本身作为列名。在本例中，列名将将是“monthly_average * 12”。此名称的描述性确实不错，但对于比较长的表达式而言，却过于累赘。您可以使用“AS”关键字对输出列指定特定的名称。在以下示例中，输出的列标题将是“yearly_average”。

```
SELECT monthly_average * 12 AS yearly_average FROM table1;
```


注意，AS 子句适用于任何输出列，而不仅仅适用于表达式。如果您愿意的话，可以执行类似于以下的操作：

```
SELECT ssn AS SocialSecurityNumber FROM table2;
```

CASE 子句允许您根据输入来控制输出。以下是一个简单的示例，它将数字（1-12）转换为月份的名称：

```
CREATE TABLE dates (m INT);
INSERT INTO dates (m) VALUES (1);
-- ...
INSERT INTO dates (m) VALUES (12);
INSERT INTO dates (m) VALUES (13);

SELECT
    CASE m
        WHEN 1 THEN 'January'
        --
        WHEN 12 THEN 'December'
        ELSE 'Invalid value for month'
    END
    AS month_name
FROM dates;
```

注意，这不仅允许您转换有效的值，还允许在发生错误时生成适当的输出。“ELSE”子句允许您指定出现意外的输入值时要使用的替代值。

在某些情况下，您可能想将值强制转换到另一种数据类型。例如，在插入 BLOB 数据时，比较方便的做法是创建包含数据的字符串，然后将该字符串插入到 BINARY 列。您可以执行强制类型转换，如下所示：

```
CREATE TABLE table1 (b BINARY(4));
INSERT INTO table1 VALUES ( CAST('FF00AA55' AS BINARY));
```

此强制类型转换允许您接收十六进制数字系列形式的输入并将其输入，就像该数据是字符串一样。在引号所括住的字符串中，每一对十六进制数字都代表一个字节的输入。由于共有 8 个十六进制数字，因此共有 4 个字节的输入。

可以通过强制类型转换对输入和输入进行更改。在以下较为复杂的代码样本中，CASE 子句中的表达式将输出格式由“2003-01-20 15:33:40”转换为“2003-Jan-20 15:33:40”。

```
CREATE TABLE sample1(dt TIMESTAMP);
COMMIT WORK;

INSERT INTO sample1 VALUES ('2003-01-20 15:33:40');
COMMIT WORK;

SELECT
    CASE MONTH(dt)
        WHEN 1 THEN REPLACE(CAST(dt AS varchar), '-01-', '-Jan-')
        WHEN 2 THEN REPLACE(CAST(dt AS varchar), '-02-', '-Feb-')
        WHEN 3 THEN REPLACE(CAST(dt AS varchar), '-03-', '-Mar-')
        WHEN 4 THEN REPLACE(CAST(dt AS varchar), '-04-', '-Apr-')
        WHEN 5 THEN REPLACE(CAST(dt AS varchar), '-05-', '-May-')
        WHEN 6 THEN REPLACE(CAST(dt AS varchar), '-06-', '-Jun-')
        WHEN 7 THEN REPLACE(CAST(dt AS varchar), '-07-', '-Jul-')
        WHEN 8 THEN REPLACE(CAST(dt AS varchar), '-08-', '-Aug-')
        WHEN 9 THEN REPLACE(CAST(dt AS varchar), '-09-', '-Sep-')
        WHEN 10 THEN REPLACE(CAST(dt AS varchar), '-10-', '-Oct-')
        WHEN 11 THEN REPLACE(CAST(dt AS varchar), '-11-', '-Nov-')
```

```

        WHEN 12 THEN REPLACE(CAST(dt AS varchar), '-12-', '-Dec-')
    END
    AS formatted_date
FROM sample1;

```

此示例从名为 `dt` 的列中获取一个值，将该值由时间戳记转换为 `VARCHAR`，然后将月份号替换为月份缩写（例如，将“-01-”替换为“-Jan-”）。通过使用 `CASE/WHEN/END` 语法，可以指定每个可能输入的对应输出。注意，由于此表达式比较复杂，因此几乎必须使用 `AS` 子句来指定输出中的列头。

2.5.5 行值构造器

本节说明其中一种并不为人熟知的表达式类型，*行值构造器 (RVC)*，及其如何与关系运算符一起运用，例如大于、小于等等。

行值构造器是由圆括号定界的有序值序列，例如：

```

(1, 4, 9)
('Smith', 'Lisa')

```

您可以将其想像成根据一系列元素/值来构造行，就像表中的行由一系列字段组成一样。

与单个的值相同，行值构造器可以用于比较。例如，就像可以使用以下表达式一样：

```

WHERE x > y;
WHERE 2 > 1;

```

也可以使用以下表达式：

```

WHERE (2, 3, 4) > (1, 2, 3);
WHERE (t1.last_name, t1.first_name) = (t2.last_name, t2.first_name);

```

使用行值构造器进行比较时，务必谨慎。下面，我们通过提供示例并进行类比来帮助了解模式，而不是提供有关比较的技术定义（您可以在 `SQL-92` 标准的 8.2 节（比较谓词）中找到该定义）。

下列表达式为 `true`：

```

(9, 9, 9) > (1, 1, 1)
('Baker', 'Barbara') > ('Alpert', 'Andy')
(1, 1) = (1, 1)
(3, 2, 1) <> (4, 3, 2)

```

以上示例比较简单，原因是该表达式对于每一对相应的元素都正确，因此对于 `RVC` 也正确。例如：

```

'Baker' > 'Alpert' 并且 'Barbara' > 'Andy',
因此 ('Baker', 'Barbara') > ('Alpert', 'Andy')

```

但是，在比较行值构造器时，表达式不必对于每个相应的元素都为 `true`。在行值构造器中，越左边的元素越重要。因此，以下表达式也为 `true`：

```

(9, 1, 1) > (1, 9, 9)
('Zoomer', 'Andy') > ('Alpert', 'Zelda')

```

在这些示例中，由于第一个 `RVC` 的最重要元素大于第二个 `RVC` 的相应元素，因此表达式为 `true`，而与其他元素的值无关。同样，在以下示例中，虽然第一个元素相同，但表达式整体为 `true`：

```
(1, 1, 2) > (1, 1, 1)
(1, 2, 1) > (1, 1, 1)
('Baker', 'Zelda') > ('Baker', 'Allison')
```

同样，在行值构造器中，越左边的元素越重要。这与我们比较多位数字的方式类似。在 3 位的数字（例如 911）中，百位数比十位数重要，十位数比个位数重要。因此，数字 911 大于数字 199，尽管并非 911 的各个位都大于 199 的相应位。

在比较多个相关的列时，此功能非常有用。此功能的一项实际应用是比较人员的姓名。例如，假定有 2 个表并且它们都包含 *lname*（姓氏）和 *fname*（名字）列。假定我们要查找所有姓名小于 Michael Morley 的人员。在这种情况下，我们希望姓氏比名字重要。下列姓名已按姓氏的正确字母顺序显示：

Adams, Zelda

Morley, Michael

Young, Anna

如果要列示名字小于 Michael Morley 的所有人员，那么我们将不想使用以下表达式：

```
table1.lname < 'Morley' and table1.fname < 'Michael'
```

如果使用此表达式，那么我们将拒绝 Zelda Adams，原因是她的名字按字母顺序排在 Michael Morley 的名字之后。使用行值构造器方法是一种正确的解决方案：

```
(table1.lname, table1.fname) < ('Morley', 'Michael')
```

注意，进行等于比较时，表达式必须对于 RCV 的所有元素都为 true。例如：

```
(1, 2, 3) = (1, 2, 3)
```

对于不等运算符，表达式只能对于一个元素为 true：

```
(1, 2, 1) <> (1, 1, 1)
```

2.6 有关事务的更多说明

如上一章所述，SQL 允许将多个语句分组到称为事务的单一“原子”（不可分）工作中。成功的事务由命令 `COMMIT WORK` 保留。以下是一个简单的示例。

```
COMMIT WORK; -- Finish the previous transaction.
UPDATE stores SET balance = balance + 199.95
  WHERE store_name = 'Big Tyke Bikes';
UPDATE checking_accounts SET balance = balance - 199.95
  WHERE name = 'Jay Smith';
COMMIT WORK;
```

如果您不想保留特定事务，那么可以使用以下命令将其回滚：

```
ROLLBACK WORK;
```

如果您未显式地落实或回滚工作，那么服务器会自动地将其回滚。换言之，除非您通过执行落实来确认要保留数据，否则将废弃该数据。

2.7 摘要

这个有关 SQL 和关系数据库的简介说明了您开始使用 SQL 前需要了解的概念。现在，您应该能够回答下列问题：

什么是表、行和列？

如何创建表？

如何将数据放入表？

如何更新表中的数据？

如何删除表中的数据？

如何列示表中的数据？

如何列示两个不同的表中的相关数据？

如何确保共同执行多个语句（以使它们作为一个组共同失败或共同成功）？

2.8 在何处查找有关 SQL 的其他信息

本手册的其他各章提供了更多有关 SQL 和特定于 solidDB 的功能部件的信息。但是，本手册不是完整的教程，也不是有关 SQL 的详尽参考资料。您可能希望获取有关 SQL 的其他文档。

市面上有许多关于 SQL 的书籍。这些书籍并非专门阐述 solidDB 的 SQL 实现；大多数资料都是通用资料并适用于任何符合 ANSI 标准的数据库服务器，例如 solidDB 的数据库服务器。通用的 SQL 书籍包括：

- *Introduction to SQL: Mastering the Relational Database Language*（作者：Rick van der Lans，出版商：Addison-Wesley）。

有关 SQL 的 ANSI 标准包括：

- 数据库语言 - 具有完整性增强功能的 SQL，ANSI，1989 ANSI X3.135-1989。
- 数据库语言 - SQL：ANSI X3H2 和 ISO/IEC JTC1/SC21/WG3 9075:1992（SQL-92）。

有关 ANSI 标准的更多信息，请参阅 <http://www.ansi.org/>。

ISO（国际标准组织）也制订了 SQL 标准。有关更多信息，请参阅 <http://www.iso.org>。

3 SQL 扩展

solidDB 中的 SQL 支持能和任何以 SQL 为基础的高级系统媲美；solidDB 提供了绝大部分常规预期功能和一系列应用 solidDB（一种特定的非标准 SQL 语法）的有用扩展。程序类 SQL 扩展（例如，存储过程、存储函数和触发器）可允许将应用程序逻辑的部分移到数据库中。这些扩展可以减少网络流量，从而提高性能。

solidDB 还支持用户定义的内部和外部存储函数及过程。外部函数和过程可使用 C 编程语言进行编写并用作迁移工具；丢失功能可通过编写 C 程序来添加。

注：外部函数和过程在共享内存访问 (SMA) 和链接库访问 (LLA) 设置中受支持。

3.1 存储过程

存储过程是简单的程序或过程，它们在 solidDB 数据库中执行。solidDB 支持存储在服务器上、用 SQL 写成的过程，也支持存储为动态库、用 C 编程语言写成的过程。

共有两种类型的存储过程：

- 用 solidDB 专有 SQL 过程语言编写的存储过程。
- 用 C 编程语言编写外部存储过程。

外部存储过程在运行时使用操作系统提供的标准动态库界面加载。如果要允许使用外部存储过程，您必须使用共享内存访问 (SMA) 或链接库访问 (LLA) 将您的应用程序链接至 solidDB。

由于存储过程在服务器内直接储存并执行，使用存储过程可以减少网络流量，从而提高性能。例如，可以在服务器本身上运行与数据绑定的复杂事务。外部存储过程还可用作迁移工具；丢失的功能可通过编写 C 程序来添加。

您可以创建包含多条 SQL 语句或完整事务的过程，并可以通过单一调用语句来执行该过程。除可以使用 SQL 语句以外，还可以使用 3GL 类型控制结构，从而启用过程式控制。当您在一个过程中执行另一个过程时，也可以创建嵌套存储过程。

存储过程也可用于控制访问权和数据库操作。授予对存储过程的执行权限将自动撤销对该过程中使用的所有数据库对象的必需访问权。因此，允许通过过程来访问关键数据可以大大简化数据库访问权管理工作。

使用 SQL 语句注册和调用存储过程。

共有三种方法调用该存储过程：

- 本地过程是在本地数据库服务器上执行的过程。
- 远程过程是存储在一个服务器上由另一个服务器调用的过程。远程存储过程仅适用于高级复制安装。
- 延期过程是落实已完成后才调用的过程。

3.1.1 存储过程 - SQL

本节详细说明如何使用 SQL 存储过程。在本节开头，说明了有关使用过程的一般概念。后续各节更深入地描述过程中不同语句的实际语法。本节的末尾将讨论事务管理、序列和其他高级存储过程功能。

基本过程结构

存储过程是您可以使用标准 DDL 语句 CREATE 和 DROP 进行处理的标准 solidDB 数据库对象。

最简单的存储过程定义类似于：

```
"CREATE PROCEDURE procedure_name
parameter_section
BEGIN
declare_section_local_variables
procedure_body
END";
```

以下示例创建名为 TEST 的过程：

```
"CREATE PROCEDURE test
BEGIN
END"
```

要运行过程，请发出 CALL 语句并接着指定所要调用的过程的名称：

```
CALL test
```

对过程命名

过程名必须在数据库模式中唯一。

所有适用于数据库对象的标准命名限制（例如使用保留字、标识长度等）也适用于存储过程名。要获取保留字的概述和完整列表，请参阅第 305 页的附录 D，『保留字』。

参数部分

存储过程通过参数与调用程序进行通信。solidDB 支持通过两种方法将值返回给调用程序。第一种方法是标准的 SQL-99 方法，即使用参数；另一种方法是 solidDB 专有方法 RETURNS，即使用结果集。

使用参数： 使用参数是标准的 SQL-99 数据返回方法。存储过程接受三类参数：

- 输入参数，这些参数用作过程的输入。缺省情况下，参数是输入参数。因此，关键字 IN 是可选的。
- 输出参数，这些参数是过程的返回值。
- 输入/输出参数，这些参数将值传递到过程并将一个值返回给调用过程。

通过在过程头中声明输入参数，就可以在该过程中通过引用参数名来访问这些参数的值。此外，还必须声明参数数据类型。要了解受支持的数据类型，请参阅第 297 页的附录 C，『数据类型』。

参数声明中使用的语法是（要了解完整的语法，请参阅第 133 页的附录 A，『语句』）：

```
parameter_definition ::= [parameter_mode] parameter_name data_type
parameter_mode ::= IN | OUT | INOUT
```

可以存在任意数目的参数。调用过程时，必须按照输入参数的定义顺序来提供这些参数。

创建过程时，可以对参数指定缺省值。声明参数时，只需在参数数据类型后面加上等号 (=) 和缺省值。例如：

```
"CREATE PROCEDURE participants( adults integer = 1,
children integer = '0',
pets integer = '0')
BEGIN
END"
```

在调用所定义的参数具有缺省值的过程时，不必对所有参数指定值。要对所有参数使用缺省值，只需使用以下命令：

```
call participants()
```

要对某个参数指定值，请在 CALL 语句中使用该参数名，并使用等号字符来指定参数值，如以下示例所示：

```
call participants(children = 2)
```

此命令对参数“children”指定值 2 并对参数“adults”和“pets”指定缺省值。

如果在 CALL 语句中未指定参数名，那么 solidDB 将假定参数按照 CREATE 语句中给定的顺序指定。

示例：

```
call participants(1)
```

此命令对参数“adults”使用值 1 并对参数“children”和“pets”使用缺省值。

```
call participants(1,2)
```

此命令对参数“adults”使用值 1 并对参数“children”使用值 2。对于参数“pets”，使用缺省值。

如果对某个参数指定了名称，那么还必须对它后面的所有参数指定名称。这就是以下命令：

```
call participants(adults = 1,2)
```

返回错误。

```
call participants(1,children = 2)
```

此命令对参数“adults”使用值 1 并对参数“children”使用值 2。对于参数“pets”，使用缺省值。

使用 RETURNS: 您可以使用存储过程来返回一个结果集表，该表可以包含多行由不同的列组成的数据。这是 solidDB 所专有的数据返回方法，此方法通过 RETURNS 结构完成。

使用 RETURNS 结构时，必须为输出数据行单独声明结果集列名。可以存在任意数目的结果集列名。结果集列名在过程定义的 RETURNS 部分中声明：

```
"CREATE PROCEDURE procedure_name
[ (IN input_param1 datatype[,
input_param2 datatype, ... ]) ]
[ RETURNS
```

```
(output_column_definition1 datatype[,
output_column_definition2 datatype, ... ]) ]
BEGIN
END";
```

缺省情况下，此过程仅返回一行数据，该行数据包含此存储过程运行或被强制退出时的值。但是，也可以使用以下语法从过程中返回结果集：

```
return row;
```

每个 RETURN ROW 调用都对返回的结果集添加一个新行，其中，列值是结果集列名的当前值。

以下语句将创建一个过程，此过程有两个输入参数，并且输出行包含两个结果集列名：

```
"CREATE PROCEDURE PHONEBOOK_SEARCH
(IN FIRST_NAME VARCHAR, LAST_NAME VARCHAR)
RETURNS (PHONE_NR NUMERIC, CITY VARCHAR)
BEGIN
-- procedure_body
END";
```

调用此过程时，应该指定两个数据类型为 VARCHAR 的输入参数。此过程返回一个输出表，该表由两个列组成，即类型为 NUMERIC 的 PHONE_NR 以及类型为 VARCHAR 的 CITY。

例如：

```
call phonebook_search ('JOHN','DOE');
```

对过程主体进行编程后，结果表将类似于：

PHONE_NR	CITY
3433555	NEW YORK
2345226	LOS ANGELES

以下语句将创建一个计算器过程：

```
"create procedure calc(i1 float, op char(1),
i2 float)
returns (calresult float)
begin
declare i integer;

if op = '+' then
calresult := i1 + i2;
elseif op = '-' then
calresult := i1 - i2;
elseif op = '*' then
calresult := i1 * i2;
elseif op = '/' then
calresult := i1 / i2;
else
calresult := 'Error: illegal op';
end if
end";
```

您可以使用以下命令来测试此计算器：

```
call calc(1,'/',3);
```

借助 RETURNS，还可以将 SELECT 语句合并到数据库过程中。以下语句创建一个过程，该过程使用 SELECT 语句来返回根据数据库创建的备份：


```

"create procedure show_backups
  returns (backup_number varchar, date_created varchar)
begin
-- First set action for failing statements.
  exec sql whenever sqlerror rollback, abort;

-- Prepare and execute the select statement
  exec sql prepare sel_cursor select
    replace(property, 'backup ', ''),
    substring(value_str, 1, 19) from sys_info
  where property like 'backup %';
  exec sql execute sel_cursor into (backup_number, date_created);

-- Fetch first row;
  exec sql fetch sel_cursor;
-- Loop until end of table
  while sqlsuccess loop
-- Return the fetched row
    return row;
-- Fetch next
    exec sql fetch sel_cursor;
  end loop;
end";

```

声明部分

过程中用于临时存储列和控制值的局部变量是在存储过程中紧跟 **BEGIN** 关键字的一个独立部分中定义的。

声明变量的语法如下:

```
DECLARE variable_name datatype;
```

注意, 每个声明语句都应该以分号 (;) 结尾。

变量名是用于标识变量的字母数字字符串。变量的数据类型可以是任何受支持的有效 SQL 数据类型。要了解受支持的数据类型, 请参阅第 297 页的附录 C, 『数据类型』。

例如:

```

"CREATE PROCEDURE PHONEBOOK_SEARCH
  (FIRST_NAME VARCHAR, LAST_NAME VARCHAR)
RETURNS (PHONE_NR NUMERIC, CITY VARCHAR)
BEGIN
DECLARE i INTEGER;

DECLARE dat DATE;

END";

```

注意, 在过程中, 可以像处理局部变量一样处理输入和输出参数, 但输入参数具有预置的值, 而输出参数值将被返回或者追加到所返回的结果集。

过程主体

过程主体包含基于赋值、表达式和 SQL 语句的实际存储过程程序。

在过程主体中, 可以使用任何类型的表达式, 其中包括标量函数。要了解有效的表达式, 请参阅第 280 页的 A.87.3, 『表达式』。

赋值: 要对变量进行赋值, 可以使用下列其中一种语法:

```
SET variable_name = expression;
```

或者

```
variable_name := expression;
```

示例:

```
SET i = i + 20 ;
```

```
i := 100;
```

包含赋值的标量函数: 标量函数是由函数名指示的操作，在函数名后面跟着一对括号括住零个或多个指定的自变量。每个标量函数都返回一个值。注意，可以将标量函数与赋值配合使用，例如:

```
"CREATE PROCEDURE scalar_sample
RETURNS (string_var VARCHAR(20))
BEGIN
-- CHAR(39) is the single quote/apostrophe
string_var := 'Joe' + {fn CHAR (39)} + 's Garage';
END";
```

此存储过程的结果是以下输出:

```
Joe's Garage
```

要获取 solidDB 支持的标量函数 (SQL-92) 的列表，请参阅第 133 页的附录 A，『语句』。注意，*solidDB Programmer Guide* 包含一个附录，该附录描述了与 SQL-92 有所区别的 ODBC 标量函数。

赋值中的变量、常量和参数: 每次执行过程时，都将初始化变量和常量。缺省情况下，变量将初始化为 NULL。除非已显式地初始化变量，否则它的值为 NULL，如以下示例所示:

```
BEGIN
DECLARE total INTEGER;
...
total := total + 1; -- assigns a null to total
...
```

因此，在对变量进行赋值前，不应引用该变量。

赋值运算符后面的表达式可以任意复杂，但它生成的数据类型必须与该变量的数据类型相同或者能够转换为后者。

有可能时，solidDB 过程语言可以隐式地进行数据类型转换。这样，就可以在期望一种类型的位置使用另一种类型的字面值、变量和参数。

在下列情况下，不可能进行隐式转换:

- 信息可能会在转换过程中丢失
- 要转换为整数的字符串包含非数字数据

示例:

```
DECLARE integer_var INTEGER;
integer_var := 'NR:123';
```

返回错误。

```
DECLARE string_var CHAR(3);
string_var := 123.45;
```

返回变量 `string_var` 中的值“123”。

```
DECLARE string_var VARCHAR(2);  
string_var := 123.45;
```

返回错误。

字符串赋值中的单引号和撇号： 字符串由单引号定界。如果要想字符串包含单引号，那么可以并排指定两个单引号（''）以便在输出中生成一个引号。这通常被称为“转义序列”。 以下是使用此技术的存储过程：

```
"CREATE PROCEDURE q  
RETURNS (string_var VARCHAR(20))  
BEGIN  
string_var := 'Joe''s Garage';  
END";  
CALL q;
```

结果为：

```
Joe's Garage
```

以下是其他一些示例：

```
'I'm writing.'
```

将变为：

```
I'm writing.
```

而

```
'Here are two single quotes:'''''
```

将变为：

```
Here are two single quotes:''
```

注意，在最后一个示例中，字符串末尾共有 5 个单引号。其中的最后一个单引号是定界符（右引号）；前面的 4 个是数据的组成部分。这 4 个引号被视为两对引号，每一对引号都被视为代表一个单引号的转义序列。

表达式：

比较运算符：

比较运算符对两个表达式进行比较。结果始终为 `TRUE`、`FALSE` 或 `NULL`。通常，比较运算与条件控制语句配合使用，并允许对任意复杂的表达式进行比较。

表 6. 比较运算符

运算符	含义
=	等于
<>	不等于
<	小于
>	大于

表 6. 比较运算符 (续)

运算符	含义
<=	小于或等于
>=	大于或等于
!=	不等于 注: != 符号表示法不能用在存储过程内, 请用 ANSI-SQL 兼容的 <>。

逻辑运算符: 逻辑运算符可用于构造较为复杂的查询。逻辑运算符 AND、OR 和 NOT 根据以下真值表所阐述的三态逻辑进行运算。AND 和 OR 是二元运算符, NOT 是一元运算符。

表 7. 逻辑运算符: NOT

NOT	true	false	null
	false	true	null

表 8. 逻辑运算符: AND

AND	true	false	null
true	true	false	null
false	false	false	false
null	null	false	null

表 9. 逻辑运算符: OR

OR	true	false	null
true	true	true	true
false	true	false	null
null	true	null	null

如真值表所示, AND 仅当它的两个操作数都为 true 时, 才返回 TRUE 值。另一方面, OR 在它的任何一个操作数为 true 时都返回 TRUE 值。NOT 返回它的操作数的相反值 (逻辑求反)。例如, NOT TRUE 返回 FALSE。

由于 NULL 值是不确定的, 因此 NOT NULL 返回 NULL。

如果未使用圆括号来指定求值顺序, 那么顺序由运算符优先级确定。

注意, “true”和“false”不是 SQL 解析器所接受的字面值, 而是值。可以把逻辑表达式值解释为数字变量:

false = 0 或 NULL

true = 1 或任何其他数字值

示例:

```
IF expression = TRUE THEN
```

可以简化为

```
IF expression THEN
```

IS NULL 运算符: 如果 IS NULL 运算符的操作数为 NULL, 那么此运算符将返回布尔值 TRUE, 否则将返回布尔值 FALSE。涉及 NULL 值的比较运算的结果始终为 NULL。要测试某个值是否为 NULL, 请不要使用以下表达式:

```
IF variable = NULL THEN...
```

这是因为, 此表达式的求值结果永远不会是 TRUE。

而是, 请改为使用以下语句:

```
IF variable IS NULL THEN...
```

注意, 在 solidDB 存储过程中使用多个逻辑运算符时, 应该将各个逻辑表达式括在圆括号中, 如下所示:

```
((A >= B) AND (C = 2)) OR (A = 3)
```

控制结构: 下列各节描述可以在过程主体中使用的语句, 其中包括分支语句和循环语句。

IF 语句: 您经常需要根据情况来执行不同的操作。IF 语句有条件地执行一系列语句。IF 语句共有三种形式, 即 IF-THEN、IF-THEN-ELSE 和 IF-THEN-ELSEIF。

IF-THEN: 最简单的 IF 语句形式使一个条件与括在关键字 THEN 和 END IF (不是 ENDIF) 中的语句列表相关联, 如下所示:

```
IF condition THEN
  statement_list;
END IF
```

仅当条件求值为 TRUE 时, 才会执行该语句序列。如果该条件求值为 FALSE 或 NULL, 那么该 IF 语句不执行任何操作。无论在哪一种情况下, 控制都将传递到下一个语句。下面是一个示例:

```
IF sales > quota THEN
  SET pay = pay + bonus;
END IF
```

IF-THEN-ELSE: IF 语句的第二种形式添加了关键字 ELSE, 后跟备用语句列表, 如下所示:

```
IF condition THEN
  statement_list1;
ELSE
  statement_list2;
END IF
```

仅当条件求值为 FALSE 或 NULL 时, 才会执行 ELSE 子句中的语句列表。因此, ELSE 子句确保执行某个语句列表。在以下示例中, 当条件为 true 或 false 时, 将分别执行第一个或第二个赋值语句:

```

IF trans_type = 'CR' THEN
    SET balance = balance + credit;
ELSE
    SET balance = balance - debit;
END IF

```

THEN 和 ELSE 子句都可以包含 IF 语句。即，IF 语句可以嵌套，如以下示例所示：

```

IF trans_type = 'CR' THEN
    SET balance = balance + credit ;
ELSE
    IF balance >= minimum_balance THEN
        SET balance = balance - debit ;
    ELSE
        SET balance = minimum_balance;
    END IF
END IF

```

IF-THEN-ELSEIF: 在某些时候，有必要从多个互斥的备用操作中进行选择。IF 语句的第三种形式使用关键字 ELSEIF 来引入其他条件，如下所示：

```

IF condition1 THEN
    statement_list1;
ELSEIF condition2 THEN
    statement_list2;
ELSE
    statement_list3;
END IF

```

如果第一个条件求值为 FALSE 或 NULL，那么 ELSEIF 子句将测试另一个条件。一个 IF 语句可以包含任意数目的 ELSEIF 子句；最后一个 ELSE 子句可选。条件将按从上到下顺序逐个进行求值。如果任何条件求值为 TRUE，那么将执行它的相关联语句列表并跳过 IF-THEN-ELSEIF 中的其余语句。如果所有条件都求值为 FALSE 或 NULL，那么将执行 ELSE 子句中的序列。请考虑以下示例：

```

IF sales > 50000 THEN
    bonus := 1500;
ELSEIF sales > 35000 THEN
    bonus := 500;
ELSE
    bonus := 100;
END IF

```

如果“sales”的值大于 50000，那么第一个和第二个条件将为 true。然而，“bonus”将被赋予正确的值 1500，这是因为从未对第二个条件进行测试。当第一个条件求值为 TRUE 时，将执行它的相关联语句，并且控制将传递到 IF-THEN-ELSEIF 后的下一个语句。

有可能时，请使用 ELSEIF 子句来代替嵌套的 IF 语句。这样，代码更易于阅读和理解。请比较下列 IF 语句：

<pre> IF condition1 THEN statement_list1; ELSE IF condition2 THEN statement_list2; ELSE IF condition3 THEN statement_list3; END IF END IF END IF END IF </pre>	<pre> IF condition1 THEN statement_list1; ELSEIF condition2 THEN statement_list2; ELSEIF condition3 THEN statement_list3; END IF </pre>
--	---

这些语句在逻辑上是等同的，但第一个语句的逻辑流晦涩难懂，而第二个语句却清晰明了。

在 *IF-THEN* 语句中使用圆括号： 在 *IF-THEN* 语句中使用括号遵循以下规则：

- *IF* 条件下的单一逻辑表达式可以使用括号，但是不是必要项。

示例：使用了括号

```
IF (x > 0) THEN
x := x - 1;
END IF;
```

示例：未使用括号

```
IF x > 0 THEN
x := x - 1;
END IF;
```

- 如果逻辑条件中存在多种表达式，每个子表达式周围要求括号。

```
IF (x > 0) AND (y > 0) THEN
x := x - 1;
END IF;
```

示例 1

该示例显示在 *IF* 语句中使用有效逻辑条件。

```
"CREATE PROCEDURE sample_if_conditions
BEGIN
DECLARE x INT;
DECLARE y INT;
x := 2;
y := 2;
```

示例 2

该示例显示在整个表达式周围使用附加括号。

```
IF ((x > 0) AND (y > 0)) THEN
x := x - 1;
END IF;
```

WHILE-LOOP: *WHILE-LOOP* 语句使一个条件与一系列由关键字 *LOOP* 和 *END LOOP* 括起来的语句相关联，如下所示：

```
WHILE condition LOOP
statement_list;
END LOOP
```

在此循环的每次迭代之前，都将对该条件进行求值。如果该条件求值为 *TRUE*，那么将执行语句列表，然后控制在循环顶部继续。如果该条件求值为 *FALSE* 或 *NULL*，那么将绕过该循环，并且控制将传递至下一个语句。下面是一个示例：

```
WHILE total <= 25000 LOOP
...
total := total + salary;
END LOOP
```

迭代次数视条件而定，并且直到该循环完成后才确定。由于在循环顶部测试条件，因此该序列可能执行零次。在后一个示例中，如果“total”的初始值大于 25000，那么条件将求值为 *FALSE*，这将完全绕过该循环。

循环可以嵌套。当内部循环完成时，控制将返回到下一个循环。过程将从 END LOOP 后的下一个语句继续执行。

离开循环： 可能有必要强制过程提早离开循环。您可以使用 LEAVE 关键字来实现此目标：

```
WHILE total < 25000 LOOP
  total := total + salary;
  IF exit_condition THEN
    LEAVE;
  END IF
END LOOP
statement_list2
```

exit_condition 求值成功后，将离开该循环，并且过程将从 *statement_list2* 继续。

注：

虽然 solidDB 数据库支持 ANSI-SQL CASE 语法，但无法在存储过程中将 CASE 构造用作控制结构。

在 WHILE 循环中使用圆括号： 以下代码演示在 WHILE 循环中使用圆括号时需遵循的规则。另请参阅发行说明，以便了解有关在 WHILE 循环中使用圆括号的其他信息。

--- 代码的本部分显示了有效逻辑条件的示例。--- 该示例用于 WHILE 循环。

```
"CREATE PROCEDURE sample_while_conditions
BEGIN
DECLARE x INT;
DECLARE y INT;
x := 2;
y := 2;
```

--- 如下所示，在 WHILE 条件下单个逻辑表达式
--- 可能使用插入成分。WHILE (x > 0) LOOP
x := x - 1;
END LOOP;

--- 如下所示，在 WHILE 条件下
--- 尽管单一逻辑表达式可能使用插入成分，但可能不会请求该插入成分。WHILE x > 0 LOOP
x := x - 1;
END LOOP;

--- 如下所示，如果在逻辑条件内有多个表达式
--- 则您需要在 EACH 单独表达式周围
--- 使用插入成分。WHILE (x > 0) AND (y > 0) LOOP
x := x - 1;
y := y - 1;
END LOOP;

--- 下例The example below is the same as the preceding example,
--- except that this has additional parentheses around the
--- entire expression.
WHILE ((x > 0) AND (y > 0)) LOOP
x := x - 1;
y := y - 1;
END LOOP;

NOT 运算符： 对 NULL 值应用逻辑运算符 NOT 的结果为 NULL。因此，下面这两个语句并非始终等同：


```

IF x > y THEN          IF NOT (x > y) THEN
  high := x;          high := y;
ELSE                  ELSE
  high := y;          high := x;
END IF                END IF

```

当 IF 条件求值为 FALSE 或 NULL 时，将执行 ELSE 子句中的语句序列。如果“x”和“y”中的任何一个为 NULL 或者两者都为 NULL，那么第一个 IF 语句将“y”的值赋予“high”，但第二个 IF 语句将“x”的值赋予“high”。如果“x”和“y”都不为 NULL，那么这两个 IF 语句都会将相应的值赋予“high”。

处理 NULL 值： NULL 值可能会导致令人混淆的行为。为了避免一些常见错误，遵循下列规则：

- 涉及 NULL 值的比较运算的结果始终为 NULL
- 对 NULL 值应用逻辑运算符 NOT 的结果为 NULL
- 在条件控制语句中，如果条件求值为 NULL，那么将不会执行它的相关联语句序列

在以下示例中，您可能期望将执行语句列表，这是因为“x”与“y”似乎不相等。记住，虽然 NULL 值是不确定的，但“x”是否等于“y”却是未知的。因此，IF 条件将求值为 NULL，并且会绕过语句列表。

```

x := 5;
y := NULL;
...
IF x <> y THEN -- evaluates to NULL, not TRUE
  statement_list; -- not executed
END IF

```

在下一个示例中，您可能会认为将执行语句列表，这是因为“a”与“b”似乎相等。但是，这同样是未知的，因此 IF 条件将求值为 NULL，并且会绕过语句列表。

```

a := NULL;
b := NULL;
...
IF a = b THEN -- evaluates to NULL, not TRUE
  statement_list; -- not executed
END IF

```

零长度字符串： solidDB 服务器将零长度字符串作为长度为零的字符串处理，而不是将其作为 NULL 进行处理。要赋予 NULL 值，应该明确地按如下方式进行赋值：

```
SET a = NULL;
```

这还意味着，对零长度字符串检查 NULL 值将返回 FALSE。

退出过程： 您可以通过在任何位置发出

```
RETURN;
```

关键字来提早退出过程。在此关键字之后，控制将直接传递到调用该过程的程序，并且将返回与过程定义的 RETURNS 部分中指定的结果集列名绑定的值。

返回数据： 可以使用 OUT 参数方式来返回数据，这是标准的 SQL-99 数据返回方法。此方法允许您将数据从过程传递回到程序。有关语法信息，请参阅第 133 页的附录 A，『语句』。

OUT 参数方式具有下列特征：

- OUT 参数方式允许您将数据从过程传递回到调用程序。在调用程序中，OUT 参数的行为与变量相同。这意味着，可以像使用局部变量那样使用 OUT 参数。您可以更改它的值，也可以通过任何方法来引用该值。
- 与 OUT 参数相对应的实参必须是变量；它不能是常量或表达式。
- 与变量相同，OUT 参数将被初始化为 NULL。

在退出过程之前，必须明确地对所有 OUT 参数赋值。否则，相应的实参将为 NULL。如果成功退出，那么 solidDB 将对实参赋值。但是，如果在发生未处理的异常的情况下退出，那么 solidDB 不会对实参赋值。

要了解 solidDB 所专有的数据返回方法，请参阅第 23 页的『使用 RETURNS』。

存储过程示例： 以下是一个简单过程的示例，此过程根据生日输入参数来确定某个人是否成人。

请留意对标量函数使用的 {fn ...} 以及用于结束赋值的分号。

```
"CREATE PROCEDURE grown_up
(birth_date DATE)
RETURNS (description VARCHAR)
BEGIN
DECLARE age INTEGER;
-- determine the number of years since the day of birth
age := {fn TIMESTAMPDIFF(SQL_TSI_YEAR, birth_date, now())};
IF age >= 18 THEN
-- If age is at least 18, then it's an adult
description := 'ADULT';
ELSE
-- otherwise it's still a minor
description := 'MINOR';
END IF
END";
```

存储过程中的 SQL： 在存储过程中使用 SQL 语句与直接从 solidDB 编辑器 (solsql) (需要一个特殊的语法) 这样的工具中发布 SQL 略有不同。

有两种方法可在过程中执行 SQL 语句：

- 使用 EXECDIRECT 语法执行语句。
- 将 SQL 语句视为“游标”。

EXECDIRECT： EXECDIRECT 语法特别适合于没有结果集的语句以及不必使用任何变量来指定参数值语句。例如，以下语句将插入一行数据：

```
EXEC SQL EXECDIRECT insert into table1 (id, name) values (1, 'Smith');
```

有关更多信息，请参阅第 171 页的 A.14.11，『exec_direct_statement』。

使用游标： 当存在结果集，或您想重复单一基本语句但却采用来自局部变量的不同值作为参数（例如，在一个循环中）时，游标适用于这些语句。

游标是在服务器进程内存中分配的特定部分，用于跟踪正在处理的语句。分配的内存空间用于存放底层语句的一行，以及有关当前行（对于 SELECTS）或语句所影响行数（对于 UPDATES、INSERTS 和 DELETES）的某些状态信息。

这样，就可以按每次一行的方式来处理查询结果。存储过程逻辑负责完成对各个行的实际处理以及将游标定位于所需的行。

处理游标的过程分为 5 个基本步骤:

1. 准备游标 - 定义
2. 执行游标 - 执行语句
3. 对游标执行访问 (对于 SELECT 过程调用) - 逐行获取结果
4. 在使用后关闭游标 - 使其仍处于启用状态以便再次执行
5. 从内存中删除游标 - 将其除去

1. 准备游标: 要定义 (准备) 游标, 请使用以下语法:

```
EXEC SQL PREPARE cursor_name SQL_statement;
```

通过准备游标, 将分配内存空间以便存放语句的结果集中的一行, 并且将解析并优化该语句。

对语句指定的游标名称在连接中必须唯一。这意味着, 不能以递归方式调用包含游标的过程 (至少不能从位于 PREPARE CURSOR 之后但在相应 DROP CURSOR 之前的语句中调用该过程)。准备游标时, solidDB 服务器将检查当前是否未打开其他具有此名称的游标。如果已存在这样的游标, 那么将返回错误号 14504。

注意, 也可以使用 ODBC API 来打开语句游标。这些游标名称不能与过程中打开的游标名称相同。

示例:

```
EXEC SQL PREPARE sel_tables
  SELECT table_name
  FROM sys_tables
  WHERE table_name LIKE 'SYS%';
```

此语句将准备名为 *sel_tables* 的游标, 但不执行它所包含的语句。

2. 执行游标: 成功地准备语句之后, 就可以执行该语句。EXECUTE 将可能的输入和输出变量与该语句绑定并运行实际语句。

EXECUTE 语句的语法如下所示:

```
EXEC SQL EXECUTE cursor_name
  [ INTO ( var1 [, var2...] ) ];
```

可选的 INTO 部分将语句的结果数据与变量绑定。

在运行 SELECT 或 CALL 语句时, 将使用 INTO 关键字后面的圆括号中列示的变量。SELECT 或 CALL 语句所生成的列将在该语句执行时与这些变量绑定。这些变量将从该语句中最左边列示的列开始进行绑定。绑定变量这一过程将在下一列继续, 直到变量列表中的所有变量都绑定完成为止。例如, 要扩展先前准备的游标 *sel_tables* 的序列, 我们需要运行下列语句:

```
EXEC SQL PREPARE sel_tables
  SELECT table_name
  FROM sys_tables
  WHERE table_name LIKE 'SYS%'
```

```
EXEC SQL EXECUTE sel_tables INTO (tab);
```

该语句现已执行完毕, 生成的表名将在后续 FETCH 语句中返回到变量 *tab* 中。

3. 对游标执行访存: 准备并执行 SELECT 或 CALL 语句之后, 就可以访存该语句所返回的数据。其他语句 (UPDATE、INSERT、DELETE 和 DDL) 没有结果集, 因此不要求执行访存。要访存结果, 请使用 FETCH 语法:

```
EXEC SQL FETCH cursor_name;
```

此命令访存游标中的单一行并将其提取到执行语句时通过 INTO 关键字绑定的变量。

要完成上一个示例以便实际获取结果行, 各个语句将如下所示:

```
EXEC SQL PREPARE sel_tables
  SELECT table_name
  FROM sys_tables
  WHERE table_name LIKE 'SYS%'
EXEC SQL EXECUTE sel_tables INTO (tab);
EXEC SQL FETCH sel_tables;
```

执行这些语句后, 变量 tab 将包含所找到的第一个符合 WHERE 子句的表的表名。

对游标 sel_tables 执行的后续 FETCH 调用将获取后续的行 (如果 SELECT 找到多行的话)。

要访存所有表名, 可以使用循环构造:

```
WHILE expression LOOP
  EXEC SQL FETCH sel_tables;
END LOOP
```

注意, 完成此循环后, 变量 tab 将包含所访存的最后一个表名。

4. 关闭游标: 可以通过发出以下命令来关闭游标:

```
EXEC SQL CLOSE cursor_name;
```

这并不会从内存中除去实际的游标定义; 有需要时, 可以重新执行该游标。

5. 删除游标: 可以从内存中删除游标, 从而释放该语句占用的所有资源:

```
EXEC SQL DROP cursor_name;
```

游标中的参数标记: 为了提高游标的动态性, SQL 语句可以包含参数标记, 后者指示在执行时与实际参数值绑定的值。“?”符号用作参数标记。

语法示例:

```
EXEC SQL PREPARE sel_tabs
  SELECT table_name
  FROM sys_tables
  WHERE table_name LIKE ?
  AND table_schema LIKE ?;
```

对 EXEC 语句进行修改, 即, 包括 USING 关键字以容纳变量与参数标记的绑定。

```
EXEC SQL EXECUTE sel_tabs USING ( var1, var2 ) INTO (tabs);
```

这样, 就可以多次使用单一游标, 而不必重新准备该游标。由于准备游标还涉及解析和优化语句, 因此通过使用可重复使用的游标, 可以显著提高性能。

注意, USING 列表只接受变量; 不能以此方式直接传递数据。例如, 如果应该对一个表执行插入, 并且该表的其中一列的值应该始终相同 (status = 'NEW'), 那么以下语法将是错误的:

```
EXEC SQL EXECUTE ins_tab USING (nr, desc, dat, 'NEW');
```

正确的方法是，在准备部分中定义常量值：

```
EXEC SQL PREPARE ins_tab  
  INSERT INTO my_tab (id, descript, in_date, status)  
  VALUES (?, ?, ?, 'NEW');  
EXEC SQL EXECUTE ins_tab USING (nr, desc, dat);
```

注意，变量在 USING 列表中可以使用多次。

SQL 语句中的参数不具有内在数据类型或显式声明。因此，仅当可以根据语句中的另一操作数来推断参数标记的数据类型时，才能在 SQL 语句中包括那些参数标记。

例如，在 ? + COLUMN1 之类的算术表达式中，可以根据由 COLUMN1 表示的所指定列的数据类型来推断该参数的数据类型。过程无法使用无法确定数据类型的参数标记。

下表描述如何确定各种类型的参数的数据类型。

表 10. 确定参数的数据类型

参数的位置	假定的数据类型
二元算术运算符或比较运算符的一个操作数	与另一个操作数相同
BETWEEN 子句中的第一个操作数	与另一个操作数相同
BETWEEN 子句中的第二个或第三个操作数	与第一个操作数相同
与 IN 配合使用的表达式	与第一个值或者子查询的结果列相同
与 IN 配合使用的值	与表达式相同
与 LIKE 配合使用的模式值	VARCHAR
与 UPDATE 配合使用的更新值	与更新列相同

应用程序不能在下列位置指定参数标记：

- 作为 SQL 标识（表名和列名等）。
- 在 SELECT 列表中。
- 同时作为比较谓词中的两个表达式。
- 同时作为二元运算符的两个操作数。
- 同时作为 BETWEEN 运算的第一个和第二个操作数。
- 同时作为 BETWEEN 运算的第一个和第三个操作数。
- 同时作为 IN 运算的表达式和第一个值。
- 作为一元 + 或 - 运算的操作数。
- 作为集合函数引用的自变量。

有关更多信息，请参阅 ANSI SQL-92 规范。

在以下示例中，存储过程将使用多个游标从一个表中读取行并将部分行插入到另一个表中：

```

"CREATE PROCEDURE tabs_in_schema (schema_nm VARCHAR)
RETURNS (nr_of_rows INTEGER)
BEGIN
DECLARE tab_nm VARCHAR;
EXEC SQL PREPARE sel_tab
SELECT table_name
FROM sys_tables
WHERE table_schema = ?;
EXEC SQL PREPARE ins_tab
INSERT INTO my_table (table_name, schema) VALUES (?,?);

nr_of_rows := 0;

EXEC SQL EXECUTE sel_tab USING (schema_nm) INTO (tab_nm);
EXEC SQL FETCH sel_tab;
WHILE SQLSUCCESS LOOP
nr_of_rows := nr_of_rows + 1;
EXEC SQL EXECUTE ins_tab USING(tab_nm, schema_nm);
IF SQLROWCOUNT <> 1 THEN
RETURN SQLERROR OF ins_tab;
END IF
EXEC SQL FETCH sel_tab;
END LOOP
END";

```

错误处理:

SQLSUCCESS: 在过程主体中执行的最近一个 EXEC SQL 语句的返回值将存储到变量 SQLSUCCESS 中。将为每个过程自动生成此变量。如果上一个 SQL 语句成功, 那么 SQLSUCCESS 中存储的值为 1。如果该 SQL 语句失败, 那么 SQLSUCCESS 中存储的值为 0。

例如, 可以使用 SQLSUCCESS 的值来确定游标何时到达结果集末尾, 如以下示例所示:

```

EXEC SQL FETCH sel_tab;
-- loop as long as last statement in loop is successful
WHILE SQLSUCCESS LOOP
-- do something with the results, for example, return a row
EXEC SQL FETCH sel_tab;

END LOOP

```

SQLERRNUM: 此变量包含最近执行的 SQL 语句的错误代码。将为每个过程自动生成此变量。如果执行成功, 那么 SQLERRNUM 将包含零 (0)。

SQLERRSTR: 此变量包含上一个失败的 SQL 语句所返回的错误字符串。

SQLROWCOUNT: 执行 UPDATE、INSERT 和 DELETE 语句后, 将有一个附加的变量可用于检查该语句的结果。变量 SQLROWCOUNT 包含上一个语句所影响的行数。

SQLERROR: 要从过程中生成用户错误, 可以使用 SQLERROR 变量将导致语句失败的实际错误字符串返回到调用应用程序。语法为:

```

RETURN SQLERROR 'error string'
RETURN SQLERROR char_variable

```

此错误将按以下格式返回:

用户错误: *error_string*

SQLERROR OF cursorname: 要对 EXEC SQL 语句进行错误检查, 可以按本节开头有关 SQLSUCCESS 的内容中描述的方式来使用 SQLSUCCESS 变量。要将导致语句失败的实际错误返回到调用应用程序, 可以使用以下语法:

```
EXEC SQL PREPARE cursorname sql_statement;
EXEC SQL EXECUTE cursorname;
IF NOT SQLSUCCESS THEN
    RETURN SQLERROR OF cursorname;
END IF

END IF
```

执行此语句时, 处理将立即停止, 并且过程返回码是 SQLERROR。可以使用 SQLERROR 函数来返回实际的数据库错误:

Sqlid 数据库错误 10033: 主键唯一约束违例

可以使用以下语句来声明过程的一般错误处理方法:

```
EXEC SQL WHENEVER SQLERROR [ROLLBACK [WORK],] ABORT;
```

如果在存储过程中包括此语句, 那么将对所执行的 SQL 语句的所有返回值检查错误。如果所执行的语句返回错误, 那么该过程将自动中止, 并且将返回最后一个游标的 SQLERROR。您还可以选择将事务回滚。

应该将此语句包括在任何 EXEC SQL 语句之前并且紧跟在变量的 DECLARE 部分之后。

以下是一个完整过程的示例, 此过程返回 SYS_TABLES 中所有以“SYS”开头的表名:

```
"CREATE PROCEDURE sys_tabs
RETURNS (tab VARCHAR)
BEGIN
-- abort on errors
EXEC SQL WHENEVER SQLERROR ROLLBACK, ABORT;
-- prepare the cursor
EXEC SQL PREPARE sel_tables
SELECT table_name
FROM sys_tables
WHERE table_name LIKE 'SYS%';
-- execute the cursor
EXEC SQL EXECUTE sel_tables INTO (tab);
-- loop through rows
EXEC SQL FETCH sel_tables;
WHILE sqlsuccess LOOP
    RETURN ROW;
    EXEC SQL FETCH sel_tables;
END LOOP
-- close and drop the used cursors
EXEC SQL CLOSE sel_tables;
EXEC SQL DROP sel_tables;
END";
```

示例: 带有 EXECDIRECT 和 游标的存储过程: 以下存储过程在一个地方使用 EXECDIRECT, 在另外一个地方使用游标。

```
"CREATE PROCEDURE p2
BEGIN

-- This variable holds an ID that we insert into the table.
DECLARE id INT;

-- Here are simple examples of EXECDIRECT.
EXEC SQL EXECDIRECT create table table1 (id_col INT);
```



```

EXEC SQL EXECEDIRECT insert into table1 (id_col) values (1);

-- 此处是游标的一个例子。
EXEC SQL PREPARE cursor1 INSERT INTO table1 (id_col) values (?);
id := 2;
WHILE id <= 10 LOOP
    EXEC SQL EXECUTE cursor1 USING (id);
    id := id + 1;
END LOOP;
EXEC SQL CLOSE cursor1;
EXEC SQL DROP cursor1;

END";

```

调用其他过程

由于调用过程是受支持 SQL 语法的组成部分，因此，您可以从一个存储过程中调用另一个存储过程。嵌套过程层数的缺省限制是 16。达到最大层数后，事务将失败。最大嵌套层数由 `solid.ini` 配置文件中的 `MaxNestedProcedures` 参数设置。有关详细信息，请参阅《solidDB 管理指南》中的附录『配置参数』。

与所有 SQL 语句相同，应该准备并执行游标，如下所示：

```

EXEC SQL PREPARE cp CALL myproc(?, ?);
EXEC SQL EXECUTE cp USING (var1, var2);

```

如果过程 `myproc` 返回一个或多个值，那么以后应该对游标 `cp` 执行访存以检索那些值：

```

EXEC SQL PREPARE cp call myproc(?,?);
EXEC SQL EXECUTE cp USING (var1, var2) INTO
(ret_var1, ret_var2);
EXEC SQL FETCH cp;

```

注意，如果被调用过程使用 `RETURN ROW` 语句，那么调用过程应该使用 `WHILE LOOP` 构造来访问所有结果。

可以进行递归调用，但建议您不要这样做，这是因为游标名称在连接级是唯一的。

定位型更新和删除： 在 `solidDB` 过程中，可以执行定位型更新和删除操作。这意味着，将对给定游标当前所在的行执行更新或删除。通过使用存储过程中使用的游标名，还可以在存储过程中执行定位型更新和删除操作。

要执行定位型更新操作，请使用以下语法：

```

UPDATE table_name
SET column = value
WHERE CURRENT OF cursor_name

```

要执行定位型删除操作，请使用以下语法：

```

DELETE FROM table_name
WHERE CURRENT OF cursor_name

```

在这两种情况下，`cursor_name` 都引用一个语句，该语句对要在其中执行更新/删除的表执行 `SELECT`。

在 SQL 标准中，定位型游标更新是语义有疑问的概念，对于 `solidDB` 服务器，它还可能会引起一些奇怪的现象。

以下示例中包含的伪代码将导致 `solidDB` 服务器执行无限循环（为了缩短篇幅以及确保清晰明了，省略了错误处理、绑定变量和其他重要任务）：


```

"CREATE PROCEDURE ENDLESS_LOOP
BEGIN
EXEC SQL PREPARE MYCURSOR SELECT * FROM TABLE1;
EXEC SQL PREPARE MYCURSOR_UPDATE
  UPDATE TABLE1 SET COLUMN2 = 'new data';
  WHERE CURRENT OF MYCURSOR;"
EXEC SQL EXECUTE MYCURSOR;
EXEC SQL FETCH MYCURSOR;
WHILE SQLSUCCESS LOOP
  EXEC SQL EXECUTE MYCURSOR_UPDATE;
  EXEC SQL COMMIT WORK;
  EXEC SQL FETCH MYCURSOR;
END LOOP
END";

```

此无限循环由以下事实引起：落实更新操作时，在游标中将出现该行的新版本，并且下一个 `FETCH` 语句将访问该行。发生这种情况的原因是，递增的行版本号包括在键值中，游标将找到已更改的行作为当前位置后的下一个更大键值。当再次更新该行时，键值将更改，并且该行将再次成为所找到的下一行。

在以上示例中，不会假定更新后的 `COLUMN2` 是表的主键的组成部分，并且行版本号是已更改的索引条目的唯一组成部分。但是，如果更改游标用于搜索数据的索引所包含的列值，那么更改后的行可能会在结果集中前后跳动。

因此，通常不建议使用定位型更新操作，您应该尽可能改为使用搜索型更新操作。但是，更新逻辑有时太复杂以致于无法在 `SQL WHERE` 子句中表达，在此类情况下，可以使用定位型更新操作，如下所示：

在 `solidDB` 中，如果 `WHERE` 子句能够确保更新后的行不会与条件匹配，因此不会再次出现在访存循环中，那么定位型游标更新操作就能够以确定无误的方式工作。构造此类搜索条件可能需要使用其他仅用于此用途的列。

注意，除非在同一个数据库会话中落实用户更改，否则在打开的游标中，那些更改不会变为可视。

事务： 就像数据库的任何其他接口使用事务一样，存储过程也使用事务。您可以在过程内或过程外落实或回滚事务。在过程内，落实或回滚通过以下语法完成：

```

EXEC SQL COMMIT WORK;
EXEC SQL ROLLBACK WORK;

```

这些语句将结束先前事务并启动新事务。

如果未在过程内落实事务，那么可以通过下列各项以外方式结束该事务：

- `solidDB SA`
- 另一个存储过程
- 自动落实，即，将连接的 `AUTOCOMMIT` 开关设置为 `ON`

对连接激活自动落实功能后，就不会在过程内强制执行自动落实。而是，将在该过程退出时执行落实。

缺省游标管理： 缺省情况下，过程退出时，将关闭该过程中打开的所有游标。关闭游标意味着游标保持处于已准备状态并可以重复执行。

过程退出后，会将它放入过程高速缓存。从高速缓存中删除该过程时，将最终删除所有游标。

保留在高速缓存中的过程数由 `solid.ini` 文件设置确定:

```
[SQL]
ProcedureCache = nbr_of_procedures
```

这意味着，只要该过程在过程高速缓存中，就可以重复使用所有未删除的游标。`solidDB` 服务器本身通过跟踪已声明的游标来管理过程高速缓存，并且将注意游标所包含的语句是否已准备完毕。

由于游标管理有可能耗用相当大量的服务器资源，在高负载多用户环境中尤其如此，因此，最好始终立即关闭游标，并且最好删除所有不再使用的游标。只有最常用的游标才应该保持处于未删除状态，以便降低游标准备工作量。

注意，事务与过程或其他语句无关。因此，落实或回滚不会释放在过程中的任何资源。

有关 **SQL** 的注意事项:

- 对使用的 **SQL** 语句没有约束。您可以在存储过程中使用任何有效的 **SQL** 语句，其中包括 **DDL** 和 **DML** 语句。
- 可以在存储过程中的任意位置声明游标。对于肯定要使用的游标，最好在声明部分后面立即进行准备。
- 对于在控制结构内使用并因此并非始终必需的游标而言，最好在它们的激活位置进行声明，从而限制打开的游标数并因此限制内存使用量。
- 游标名是未声明的标识，而不是变量；它仅用于引用查询。您不能对游标名赋值或者在表达式中使用游标名。
- 可以反复地重新执行游标，而不必进行重新准备。注意，这可能会对性能产生严重影响；与重新执行已准备的游标相比，在类似的语句中反复准备游标可能会导致性能下降近 40%！
- 任何 **SQL** 语句都必须使用关键字 `EXEC SQL` 进行准备。

用于查看过程堆栈的函数: 可以在存储过程中使用下列函数来分析过程堆栈的当前内容:

- `PROC_COUNT ()`

此函数返回过程堆栈中的过程数目，其中包括当前过程。

- `PROC_NAME (N)`

此函数返回堆栈中的第 `N` 个过程名。第一个过程的位置为零。

- `PROC_SCHEMA (N)`

此函数返回过程堆栈中第 `N` 个过程的模式名。

这些函数支持根据从应用程序中调用还是从过程中调用的不同而具有不同行为的存储过程。

过程特权

存储过程由创建者拥有，并且是创建者的模式的组成部分。用户如果需要运行其他模式中的存储过程，那么必须获得对该过程的执行特权:

```
GRANT EXECUTE ON procedure_name TO { USER | ROLE };
```

此函数返回过程堆栈中第 `N` 个过程的模式名。

对于授权执行的过程以及后续调用的过程中访问的所有数据库对象，将根据该过程的所有者的权限来访问那些对象。不必进行特殊的授权。

由于采用创建者的特权来运行过程，因此该过程不仅拥有创建者的权限来访问表之类的对象，还将使用创建者的模式和目录。例如，假定用户“Sally”运行由用户“Jasmine”创建的过程“Proc1”。并且，假定 Sally 和 Jasmine 都有名为“table1”的表。缺省情况下，存储过程 Proc1 将使用 Jasmine 的模式中的 table1，即使 Proc1 由用户 Sally 调用亦如此。

有关特权和远程存储过程调用的更多信息，另请参阅第 44 页的『访问权』。

高级复制配置中的远程存储过程

在高级复制配置中，可通过本地方式或远程方式调用存储过程。远程过程是一个数据库服务器调用另一个数据库服务器的存储过程。远程存储过程调用使用以下语法：

```
CALL procedure_name AT node-ref;
```

其中：

node-ref 指示远程存储过程所在的数据库服务器。

只能在两个具有主数据库/副本数据库关系的 solidDB 服务器之间进行远程存储过程调用。可在两个方向执行调用操作；即主数据库可以调用副本数据库上的存储过程，副本数据库可以调用主数据库上的存储过程。

可在任何允许本地过程调用的上下文中调用远程存储过程。因此，例如，您可以使用 CALL 语句来直接调用远程存储过程，也可以从触发器、另一个存储过程或者 START AFTER COMMIT 语句中调用该远程过程。

以远程方式调用的存储过程可以包含其他任何存储过程能够包含的任何命令。所有存储过程都使用相同的语法规则进行创建。您可以在不同的时候以本地方式和远程方式调用同一个存储过程。

以远程方式调用存储过程时，该存储过程接受来自调用者的参数（即把这些调用当做本地调用）。但是，远程存储过程无法返回结果集；它只能返回错误代码。

本地存储过程调用和远程存储过程调用都是同步调用；无论以本地方式还是远程方式来调用过程，调用者需一直等待，直到值返回为止；在后台执行该存储过程期间，调用者不会继续执行。如果是从 START AFTER COMMIT 中调用存储过程，则该存储过程调用本身是同步的，但 START AFTER COMMIT 不是同步的，因此该存储过程将作为异步后台进程执行。

远程存储过程的事务处理方式与本地存储过程的事务处理方式不同。以远程方式调用存储过程时，该存储过程的执行不属于含有该调用的事务。因此，您无法通过回滚调用该存储过程的事务来回滚该过程。

远程存储过程的 CALL 语法

调用远程存储过程的命令的完整语法如下所示：

```
CALL <proc-name>[(param [, param...])] AT node-def;  
node-def ::= DEFAULT | 'replica name' | 'master name'
```

DEFAULT 仅与 START AFTER COMMIT 语句配合使用。

例如:

```
CALL MyProc('Smith', 750) AT replica1;  
CALL MyProcWithoutParameters AT replica2;
```

注: 对于每个 CALL，只能列示一个节点定义。例如，如果要通知多个副本数据库，那么必须单独地调用每个副本数据库。但是，您可以创建包含多条 CALL 语句的存储过程，然后对该过程进行单一调用。

创建远程存储过程

远程存储过程始终在执行该过程的服务器上创建，而不是在调用该过程的服务器上创建。例如，如果主数据库将要调用过程 foo() 以便在 replica1 上执行该过程，则必须已在 replica1 上创建过程 foo()。主数据库不知道它以远程方式调用的存储过程的内容。实际上，除了 CALL 语句本身指定的信息外，主数据库不知道存储过程的任何信息。

例如，下列 CALL 语句包括该过程的名称、一些参数值以及要执行该过程的副本数据库的名称:

```
CALL foo(param1, param2) AT replica1
```

该存储过程不会向调用者注册。这意味着，调用者盲目地调用过程，甚至不知道该过程是否存在。如果调用者尝试调用不存在的过程，该调用者会收到一条错误信息。

支持动态参数绑定。例如，以下调用合法:

```
CALL MYPROC(?, ?) AT MYREPLICA1;
```

存储过程调用不会进行缓存或排队。如果您调用了不存在的一个存储过程，则该调用不会为了等待存储过程的出现一直存在。同样，如果该过程存在，但是该过程所在的服务器已关闭、与网络断开连接或由于其他原因无法访问该过程，则该调用不会一直处于打开状态，不会等待服务器再次变为可访问时进行重试。使用“同步拉取通知”（推送同步）功能时，记住这一点十分重要。

访问权: 要调用存储过程，调用者对该过程必须具有 EXECUTE 特权。（对于任何存储过程均如此，而无论是以本地方式还是远程方式调用该过程。）

以本地方式调用过程时，将使用调用者的特权来执行该过程。以远程方式调用过程时，可以使用所指定用户在远程服务器上具有的特权或者与本地调用者相对应的远程用户的特权来执行该过程。（在调用该存储过程之前，副本服务器用户与主服务器用户必须已相互映射。有关将副本服务器用户映射到主服务器用户的更多信息，请参阅 *IBM solidDB Advanced Replication User Guide*。）

如果从副本服务器调用远程存储过程并将在主服务器上执行该过程，那么您可以选择指定要使用哪个主服务器用户的特权来执行该过程。

如果从主服务器调用远程存储过程并将在副本服务器上执行该过程，或者您未指定要使用哪个用户的特权，那么调用服务器将根据调用该存储过程的用户以及副本服务器用户与主服务器用户之间的映射来确定应该使用哪个用户的特权。

下面对这些可能性进行更详细的说明。

1. 如果从副本服务器调用远程存储过程并将在主服务器上执行该过程，那么您可以执行 SET SYNC USER 语句以指定要使用哪个主服务器用户的特权。在调用远程存储过程之前，必须在本地服务器上执行 SET SYNC USER。在调用服务器上指定同步

用户之后，每次调用远程存储过程时，调用服务器都将向远程服务器（主服务器）发送用户名和密码。远程服务器将尝试使用随过程调用一起发送的用户标识和密码来执行该过程。该用户标识和密码在远程服务器中必须存在，并且指定的用户必须对数据库具有适当的访问权并对调用的过程具有 EXECUTE 特权。

SET SYNC USER 语句仅在副本服务器上有效，因此仅当副本服务器调用主服务器上的存储过程时，您才能指定同步用户。

2. 如果调用者是主服务器，或者从副本服务器执行调用并且您执行调用前未指定同步用户，那么该服务器将尝试确定远程服务器上的哪个用户与本地服务器上的用户相对应。

如果调用服务器是副本服务器 (R → M)

调用服务器在调用远程过程时，将向远程服务器发送下列信息：

主服务器的名称 (SYS_SYNC_MASTERS.NAME)。

副本服务器标识 (SYS_SYNC_MASTERS.REPLICA_ID)。

主服务器用户标识（这个主服务器用户标识就是与调用该过程的本地用户的用户标识相对应的主服务器用户标识。显然，这个本地用户必须已映射到相应的主服务器用户。）

注意，这种选择主服务器用户标识的方法与副本服务器刷新数据时使用的方法相同 - 副本服务器在 SYS_SYNC_USERS 表中查找映射到当前本地副本服务器用户的主服务器用户。

如果调用服务器是主服务器 (M → R)

调用服务器在调用远程过程时，将向远程服务器发送下列信息：

主服务器的名称 (SYS_SYNC_REPLICAS.MASTER_NAME)。

副本服务器标识 (SYS_SYNC_REPLICAS.ID)。

调用者的用户名。

调用者的用户标识。

当副本服务器接收到主服务器用户标识时，副本服务器将查找映射到该主服务器标识的本地用户。由于可能有多个副本服务器用户映射到单一主服务器用户，因此服务器将使用找到的第一个映射到所指定主服务器用户并具有执行此存储过程所需特权的本地用户。

在主服务器可以调用副本服务器上的存储过程之前，主服务器必须知道副本服务器的连接字符串。如果副本服务器允许从主服务器执行调用，那么副本服务器应该在 solid.ini 文件中定义它自己的连接字符串信息。此信息将提供给主服务器（副本服务器在将任何消息转发到主服务器时，都将包括此信息的一个副本）。当主服务器从副本服务器接收到连接字符串时，如果新值与先前值不同，那么主服务器将替换先前值。

示例：


```
[Synchronizer]
ConnectStrForMaster=tcp replicahost 1316
```

也可以使用以下语句向主服务器提供副本服务器的连接字符串:

```
SET SYNC CONNECT <connect-info> TO REPLICA <replica-name>
```

如果主服务器需要调用副本服务器, 但副本服务器还没有向主服务器提供其连接字符串(即, 尚未将任何消息转发到主服务器), 那么此方法很有用。

高级复制配置中的延迟型过程

延迟型过程是已处理落实后调用的存储过程。

在高级复制配置中, 当已落实的事务结束时, 您可能需要执行特定的操作。例如, 如果该事务更新了“主数据库”发布中的某些数据, 那么您可能想将“主数据库数据已更新”这一情况通知副本数据库。solidDB 允许 `START AFTER COMMIT` 语句指定当前事务落实时要执行的 SQL 语句。可将指定的 SQL 语句称为 `START AFTER COMMIT` 的“主体”。该主体将以异步方式在一个独立的连接中执行。

例如, 如果您希望在事务落实时调用存储过程 `my_proc()`, 那么可以编写以下语句:

```
START AFTER COMMIT NONUNIQUE CALL
    my_proc;
```

此语句可以出现在事务中的任何位置; 它可以是第一个语句、最后一个语句或其间的任何一个语句。无论 `START AFTER COMMIT` 语句本身出现在事务中的什么位置, 主体(对 `my_proc` 的调用)都只在该事务落实时执行。在以上示例中, 主体在单独的一行中, 但这并不是语法方面的规定。

由于语句的主体与 `START AFTER COMMIT` 语句本身并非同时执行, 因此, `START AFTER COMMIT` 命令分为两个不同的阶段, 即定义阶段和执行阶段。

- 在 `START AFTER COMMIT` 的定义阶段, 指定但不执行主体。定义阶段可以在事务中的任何位置发生; 换言之, 语句“`START AFTER COMMIT...`”相对于同一事务中的其他 SQL 语句可以处于任何顺序。
- 在执行阶段, 实际执行 `START AFTER COMMIT` 语句的主体。执行阶段在事务的 `COMMIT WORK` 语句执行时发生。也可以采用自动落实方式来执行 `START AFTER COMMIT`, 但很少有理由这样做。

以下示例说明如何在事务中使用 `START AFTER COMMIT` 语句。

```
-- Any valid SQL statement(s)...
...
-- Creation phase. The function my_proc() is not actually called here.
START AFTER COMMIT NONUNIQUE CALL my_proc(x, y);
...
-- Any valid SQL statement(s)...

-- Execution phase: This ends the transaction and starts execution
-- of the call to my_proc().
COMMIT WORK;
```

`START AFTER COMMIT` 直到事务成功落实完毕后才执行。如果回滚包含 `START AFTER COMMIT` 的事务, 那么将不会执行 `START AFTER COMMIT` 的主体。如果要将更新后的数据从副本数据库传播到主数据库, 那么这是优点, 原因是您只想传播落实后的数据。如果使用触发器来启动传播, 那么将在落实数据前传播该数据。

START AFTER COMMIT 命令仅应用于当前事务，即从中发出 START AFTER COMMIT 命令的事务。它不会应用于后续事务，也不会应用于当前在其他连接中打开的任何其他事务。

START AFTER COMMIT 命令允许您只指定一个要在 COMMIT 发生时执行的 SQL 语句。但是，该 SQL 语句可以调用存储过程，该存储过程可以包含多个语句，其中包括对其他存储过程的调用。并且，每个事务可以有多个 START AFTER COMMIT 命令。每一个 START AFTER COMMIT 语句的主体都将在该事务落实时执行。但是，这些主体将以异步方式相互独立地执行；它们不必按照相应 START AFTER COMMIT 语句的顺序执行，并且，它们的执行有可能重叠（不保证一个主体执行完成后才启动下一个主体）。

高级复制设置中 START AFTER COMMIT 的一种常见用法是帮助实现“同步拉取通知”（推送同步）功能。

如果 START AFTER COMMIT 的主体是对存储过程的调用，那么该过程可以是本地过程，也可以是远程副本数据库（或主数据库）中的远程过程。

如果正在使用“同步拉取通知”功能，那么您可能想对多个副本数据库调用同一个过程。要完成此任务，必须使用略微间接的方法。最简单的方法是，编写一个对各个副本数据库调用多个过程的本地过程。例如，如果 START AFTER COMMIT 语句的主体是“CALL my_proc”，那么可以将 my_proc 编写成类似于以下内容：

```
CREATE PROCEDURE my_proc
BEGIN
CALL update_inventory(x) AT replica1;
CALL update_inventory(x) AT replica2;
CALL update_inventory(x) AT replica3;
END;
```

如果副本数据库列表是静态的，那么此方法的效果不错。但是，如果您预期将来要添加新的副本数据库，那么您会发现，根据副本数据库属性成组地更新副本数据库更为方便。这使您能够添加具有特定属性的新副本数据库，然后对那些新的副本数据库执行现有的存储过程。这是通过使用两项功能实现的：START AFTER COMMIT 中的 FOR EACH REPLICA 子句以及远程存储过程调用中的 DEFAULT 子句。

如果在 START AFTER COMMIT 中使用 FOR EACH REPLICA 子句，那么将为每个符合 WHERE 子句中的条件的副本数据库执行一次该语句。将为每个副本数据库执行一次该语句，而不是针对每个副本数据库执行一次该语句。如果 CALL 语句未包含“AT node-ref”子句，那么将以本地方式调用该存储过程，即在执行 START AFTER COMMIT 的服务器上调用该过程。要确保对每个副本数据库调用一次存储过程，必须使用 DEFAULT 子句。典型的做法是，创建一个包含使用 DEFAULT 子句的远程过程调用的本地存储过程。

例如，假定 my_local_proc 包含以下内容：

```
CALL update_sales_statistics AT DEFAULT;
```

START AFTER COMMIT 语句如下：

```
START AFTER COMMIT FOR EACH REPLICA
WHERE region = 'north'
UNIQUE
CALL my_local_proc;
```

WHERE 子句如下:

```
WHERE region = 'north'
```

因此, 针对具有 `region = 'north'` 属性的每个副本, 您将调用名为 `my_local_proc` 的存储过程。

接着, 该本地过程将执行以下语句:

```
CALL update_sales_statistics() AT DEFAULT
```

关键字 `DEFAULT` 可解析为副本数据库的名称。每次从 `START AFTER COMMIT` 的主体中调用 `my_local_proc` 时, `DEFAULT` 关键字都是另一个具有“`region = 'north'`”属性的副本数据库的名称。

注: 在此示例中:

- 有可能并非所有副本数据库都包含名为 `update_sales_statistics()` 的过程。在这种情况下, 将只对那些包含该过程的副本数据库执行该过程。主数据库将不会向每个副本数据库发送该过程的副本; 主数据库仅仅是调用现有的过程。
- 有可能即使所有副本数据库都包含名为 `update_sales_statistics()` 的过程, 但是其包含的过程内容不同。每个副本数据库都可以有自己的定制过程版本。

在对每个副本数据库执行每条语句之前, 将与该副本数据库建立连接。

使用 `START AFTER COMMIT` 命令来调用多个副本数据库时, 可以在 `CALL` 命令的语法中使用可选关键字“`DEFAULT`”。例如, 假定使用以下语句:

```
START AFTER COMMIT
  FOR EACH REPLICA
  WHERE location = 'India'
  UNIQUE CALL push;
```

然后, 在本地过程“推送”中, 可以使用关键字“`DEFAULT`”, 此关键字用作包含相应副本数据库的名称的变量。

```
CREATE PROCEDURE push
BEGIN
EXEC SQL EXECDIRECT CALL remoteproc AT DEFAULT;
END
```

对于每个“`location`”属性的值为“`India`”的副本数据库, 都将调用一次过程“`push`”。每次调用该过程时, 都会将“`DEFAULT`”设置为该副本数据库的名称。因此, 以下语句将对特定副本数据库调用该过程。

```
CALL remoteproc AT DEFAULT;
```

可以使用以下语句在主数据库中设置副本数据库属性:

```
SET SYNC PROPERTY propname = 'value' FOR REPLICA replica_name;
```

例如:

```
SET SYNC PROPERTY location = 'India' FOR REPLICA asia_hq;
```

`START AFTER COMMIT` 中指定的语句作为独立事务执行。它不是包含该 `START AFTER COMMIT` 命令的事务的组成部分。这个独立事务就像是自动落实方式处于打开状态那样运行; 换言之, 不需要显式的 `COMMIT WORK` 来落实此语句中执行的工作。

但是，在其他方面，此语句的执行与事务并不相似。首先，不保证该语句将执行到完成。该语句将作为独立的后台任务启动。如果服务器崩溃，或者该语句由于其他某种原因而无法执行，那么该语句将消失，而不会执行到完成。

其次，由于该语句作为后台任务执行，因此不存在用于返回错误的机制。第三，无法回滚该语句；如果该语句执行完成，那么“事务”语句将自动落实，而无论是否检测到任何错误。如果该语句是过程调用，那么该过程本身可能包含 COMMIT 和 ROLLBACK 命令。

您可以使用“RETRY”子句，以便在该语句失败时多次尝试执行该语句。RETRY 子句允许您指定服务器应该重试失败语句的次数。您必须指定两次重试之间要等待的秒数。

如果未使用 RETRY 子句，那么服务器将仅尝试执行该语句一次，然后将废弃该语句。例如，如果该语句尝试调用一个远程过程，但远程服务器已关闭或者由于网络问题而不可访问，那么将不会执行该语句，并且您不会接收到任何错误消息。

任何语句（其中包括 START AFTER COMMIT 中指定的语句）都在特定“上下文”中执行。上下文包含诸如缺省目录、缺省模式等因素。对于从 START AFTER COMMIT 中执行的语句而言，语句的上下文基于执行 START AFTER COMMIT 时的上下文，而不是基于实际导致运行 START AFTER COMMIT 中的语句的 COMMIT WORK 的上下文。在以下示例中，将在目录 foo_cat 和模式 foo_schema 中执行“CALL FOO_PROC”，而不是在 bar_cat 和 bar_schema 中执行该语句。

```
SET CATALOG FOO_CAT;  
SET SCHEMA FOO_SCHEMA;  
START AFTER COMMIT UNIQUE CALL FOO_PROC;  
...  
SET CATALOG BAR_CAT;  
SET SCHEMA BAR_SCHEMA;  
COMMIT WORK;
```

UNIQUE/NONUNIQUE 关键字确定服务器是否尝试避免多次发出同一个命令。

<stmt> 前的 UNIQUE 关键字指定，仅当没有完全相同的语句正在执行或者处于“暂挂”状态等待执行时，才执行该语句。语句以简单字符串比较方式进行比较。例如，“call foo (1)”与“call foo(2)”不同。在比较时，还将考虑副本数据库；换言之，UNIQUE 并不会阻止服务器对不同的副本数据库执行同一个触发器调用。注意，“UNIQUE”仅阻塞语句的重叠执行，而不会阻止在当前调用运行完成后通过再次调用同一个语句来再次执行该语句。

NONUNIQUE 表示可以在后台同时执行重复的语句。

示例：程序认为下列语句全都是不同的语句，因此，尽管它们都包含 UNIQUE 关键字，也要执行这些语句。（Name 是副本数据库的唯一属性。）

```
START AFTER COMMIT UNIQUE call myproc;  
START AFTER COMMIT FOR EACH REPLICA WHERE name='R1' UNIQUE call myproc;  
START AFTER COMMIT FOR EACH REPLICA WHERE name='R2' UNIQUE call myproc;  
START AFTER COMMIT FOR EACH REPLICA WHERE name='R3' UNIQUE call myproc;
```

但是，如果在先前语句的事务中执行以下语句，并且副本数据库 R1、R2 和 R3 的其中某个具有属性“color='blue'”，那么将不再对那些副本数据库执行该调用。

```
START AFTER COMMIT FOR EACH REPLICA WHERE color='blue'  
UNIQUE call myproc;
```

唯一性不阻止“自动”执行覆盖“手动”执行。例如，如果以手动方式执行命令以便根据特定发布来执行刷新，并且主数据库也调用一个远程存储过程以便根据该发布来执行刷新，那么主数据库不会由于手动刷新已在运行中而“跳过”该调用。唯一性仅应用于 START AFTER COMMIT 启动的语句。

可以在存储过程中使用 START AFTER COMMIT 语句。例如，假定您希望在事务成功完成时并且仅当成功完成时才发出事件。您可以编写一个执行 START AFTER COMMIT 语句的存储过程，并且，该语句只有在事务落实完成后才发出事件（如果回滚该事务，那么不发出事件）。代码如下所示：

此样本还包含“接收”并使用事件参数的示例。请参阅脚本 1 中名为“wait_on_event_e”的存储过程。

```
-- To run this demo properly, you will need two users/connections.
-- This demo contains 5 separate "scripts", which must be executed
-- in the order shown below:
--     User1 executes the first script.
--     User2 executes the second script.
--     User1 executes the third script.
--     User2 executes the fourth script.
--     User1 executes the fifth script.
-- You may notice that there are some COMMIT WORK statements
-- in surprising places. These are to ensure that each user sees the
-- most recent changes of the other user. Without the COMMIT WORK
-- statements, in some cases one user would see an out-of-date
-- "snapshot" of the database.
--
-- Set autocommit off for both users/connections!
```

```
----- SCRIPT 1 (USER 1) -----
CREATE EVENT e (i int);
CREATE TABLE table1 (a int);

-- This inserts a row into table1. The value inserted into the is copied
-- from the parameter to the procedure.
"CREATE PROCEDURE inserter(i integer)
BEGIN
EXEC SQL PREPARE c_inserter INSERT INTO table1 (a) VALUES (?);
EXEC SQL EXECUTE c_inserter USING (i);
EXEC SQL CLOSE c_inserter;
EXEC SQL DROP c_inserter;
END";

-- This posts the event named "e".
"CREATE PROCEDURE post_event(i integer)
BEGIN
POST EVENT e(i);
END";

-- This demonstrates the use of START AFTER COMMIT inside a
-- stored procedure. After you call this procedure and
-- call COMMIT WORK, the server will post the event.
"CREATE PROCEDURE sac_demo
BEGIN
DECLARE MyVar INT;
MyVar := 97;
EXEC SQL PREPARE c_sacdemo START AFTER COMMIT NONUNIQUE CALL
    post_event(?);
EXEC SQL EXECUTE c_sacdemo USING (MyVar);
EXEC SQL CLOSE c_sacdemo;
EXEC SQL DROP c_sacdemo;
```

```

END";

-- When user2 calls this procedure, the procedure will wait until
-- the event named "e" is posted, and then it will call the
-- stored procedure that inserts a record into table1.
"CREATE PROCEDURE wait_on_event_e
BEGIN
-- Declare the variable that will be used to hold the event parameter.
-- Although the parameter was declared when the event was created, you
-- still need to declare it as a variable in the procedure that receives
-- that event.
DECLARE i INT;
WAIT EVENT
  WHEN e (i) BEGIN
    -- After we receive the event, insert a row into the table.
    EXEC SQL PREPARE c_call_inserter CALL inserter(?);
    EXEC SQL EXECUTE c_call_inserter USING (i);
    EXEC SQL CLOSE c_call_inserter;
    EXEC SQL DROP c_call_inserter;
  END EVENT
END WAIT
END";

COMMIT WORK;

----- SCRIPT 2 (USER 2) -----
-- Make sure that user2 sees the changes that user1 made.
COMMIT WORK;

-- Wait until user1 posts the event.
CALL wait_on_event_e;
-- Do not commit work again (yet).

----- SCRIPT 3 (USER 1) -----
COMMIT WORK;

-- User2 should be waiting on event e, and should see the event after
-- we execute the stored procedure named sac_demo and then commit work.
-- Note that since START AFTER COMMIT statements are executed
-- asynchronously, there may be a slight delay between the COMMIT WORK
-- and the associated POST EVENT.
CALL sac_demo;
COMMIT WORK;

----- SCRIPT 4 (USER 2) -----
-- Commit the INSERT that we did earlier when we called inserter()
-- after receiving the event.
COMMIT WORK;

-----SCRIPT 5 (USER 1) -----
-- Ensure that we see the data that user2 inserted.
COMMIT WORK;

-- Show the record that user2 inserted.
SELECT * FROM table1;

COMMIT WORK;

```

Important considerations on START AFTER COMMIT

- 当延迟型过程调用（START AFTER COMMIT）的主体执行时，它将以异步方式在后台运行。这允许服务器立即开始执行程序中的下一个 SQL 命令，而不必等待延迟型过程调用语句完成。这还意味着，与服务器断开连接前，您不必等待该语句完成。在大多数情况下，这是一个优点。但是，在某些情况下，这可能是缺点。例如，如

果延迟型过程调用的主体锁定了程序中后续 SQL 命令所需的记录，那么让延迟型过程调用的主体在后台运行并让下一个 SQL 命令在前台运行但必须等待访问那些记录可能并不合适。

- 仅当事务通过 COMMIT 完成而不是通过 ROLLBACK 完成时，要执行的语句才会执行。如果显式地回滚整个事务，或者该事务中止并因此进行隐式回滚（例如，由于连接发生故障），那么将不会执行 START AFTER COMMIT 的主体。
- 虽然可以将包含延迟型过程调用的事务回滚（从而阻止该延迟型过程调用的主体运行），但该延迟型过程调用的主体本身在执行后将无法回滚。由于它以异步方式在后台运行，因此不存在用于在该主体开始执行后将其取消或回滚的机制。
- 包含延迟型过程调用的语句不保证能够运行到完成或者作为“原子”事务运行。例如，如果服务器崩溃，那么该语句将不会在服务器启动后继续执行，并且任何在服务器崩溃前已完成的操作都将保留下来。为了防止此类情况下发生数据不一致问题，您必须谨慎地进行编程并正确地使用引用约束之类的功能来确保数据完整性。
- 如果以自动落实方式来执行 START AFTER COMMIT 语句，那么 START AFTER COMMIT 的主体将“立即”执行（即，在 START AFTER COMMIT 执行并自动落实后立即执行）。乍一看，这似乎毫无用处 - 为何不直接执行 START AFTER COMMIT 的主体？但是，存在几项细微的差别。首先，对 my_proc 进行的直接调用是同步调用；服务器在该存储过程执行完成前将不会返回控制权。但是，如果作为 START AFTER COMMIT 的主体来调用 my_proc，那么该调用是异步调用；服务器不等待 my_proc 结束即允许您执行下一条 SQL 语句。并且，由于 START AFTER COMMIT 语句实际上并非“立即”执行（即，在事务落实时执行），而是有可能在服务器繁忙时短暂延迟执行，因此可能会也可能不会在 my_proc 开始执行前实际地开始运行下一条 SQL 语句。这不大可能是您所期望的行为。但是，如果您确实想启动当您继续运行程序时将在后台运行的异步存储过程，那么可以采用自动落实方式来执行 START AFTER COMMIT。
- 如果在同一个事务中执行不止一个延迟型过程调用，那么所有 START AFTER COMMIT 语句的主体都将以异步方式运行。这意味着它们不一定按照您在该事务中执行 START AFTER COMMIT 语句的顺序运行。
- START AFTER COMMIT 的主体只能包含一条 SQL 语句。但是，该语句可以是过程调用，并且该过程可以包含多条 SQL 语句，其中包括其他过程调用。
- START AFTER COMMIT 语句将仅应用于在其中定义该语句的事务。如果在当前事务中执行 START AFTER COMMIT，那么延迟型过程调用的主体将只在当前事务落实时执行；它不会在后续事务中执行，也不会通过任何其他连接执行的事务中执行。START AFTER COMMIT 语句不会创建“持久”行为。如果您希望在多个事务结束时调用同一个主体，那么必须在每个事务中执行“START AFTER COMMIT ... CALL my_proc”语句。
- 执行延迟型过程调用（START AFTER COMMIT）语句的主体的“结果”不会以任何方式返回到运行该延迟型过程调用的连接。例如，如果延迟型过程调用的主体返回一个值以指示是否发生错误，那么将废弃该值。
- 几乎任何 SQL 语句都可以用作 START AFTER COMMIT 语句的主体。虽然典型情况是调用存储过程，但也可以使用 UPDATE、CREATE TABLE 或几乎任何其他语句。（但是，我们建议您不要在 START AFTER COMMIT 中放置另一条 START AFTER COMMIT 语句。）注意，由于不返回结果，因此 SELECT 之类的语句在延迟型过程调用中通常没有用处。

- 由于在事务中执行 `START AFTER COMMIT` 语句时并不执行主体，因此，除非延迟型过程自己调用自己、主体包含语法错误或某些其他不必实际执行主体就能检测到的错误，否则 `START AFTER COMMIT` 语句很少会失败。

如果您不希望程序中的下一条 SQL 语句直到延迟型过程调用语句运行完成后才运行，应如何处理？变通方法如下所示：

1. 在延迟型过程调用语句结束时（例如，在延迟型过程调用语句调用的存储过程结束时），发出一个事件。（有关事件的描述，请参阅 *IBM solidDB Programmer Guide*。）
2. 在落实指定了延迟型过程调用的事务之后，立即调用等待该事件存储过程。
3. 在等待事件的存储过程调用后面，放置程序要执行的下一条 SQL 语句。

例如，程序可能类似于：

```
...  
  START AFTER COMMIT ... CALL myproc;  
  ...  
  COMMIT WORK;  
  CALL wait_for_sac_completion;  
  UPDATE ...;
```

存储过程 `wait_for_sac_completion` 将等待 `myproc` 发出的事件。因此，`UPDATE` 语句直到延迟型过程调用语句完成后才会运行。

然而此变通方法略有风险。由于延迟型过程调用语句不保证执行到完成，因此存储过程 `wait_for_sac_completion` 有可能永远无法获得它所等待的事件。

为何设计可能会也可能不会运行到完成的命令？这是因为，`START AFTER COMMIT` 功能的主要用途是支持“同步拉取通知”。“同步拉取通知”功能允许主控服务器将“数据已更新”以及“副本服务器可以请求执行刷新以获取新数据”这一情况通知它的副本服务器。即使此通知过程由于某种原因而失败，也不会导致数据损坏；这仅仅意味着副本数据库刷新数据前的延迟时间较长。由于副本数据库将始终获得上次成功的刷新操作后的所有数据，因此延迟接收数据并不会导致副本数据库永久丢失任何数据。有关更多详细信息，请参阅 *IBM solidDB Advanced Replication User Guide* 中的 *Introduction to Sync Pull Notify* 一节。

注：`START AFTER COMMIT` 的主体中的语句可以是任何语句，其中包括 `SELECT`。但请记住，`START AFTER COMMIT` 的主体不会以任何方式返回它的结果，因此 `SELECT` 语句在 `START AFTER COMMIT` 中通常没有用处。

注：如果您处于自动落实方式并执行 `START AFTER COMMIT...`，那么给定的语句将在后台立即启动。这里，“立即”实际上表示“尽快”，这是因为它仍然在服务器有时间执行它时以异步方式执行。

示例：Sync Pull Notify (“拉取同步”)： 该示例展示如何使用 `START` 和 `CALL` 语句实现同步拉取通知（主数据库可通知所有的相关副本数据库，已有新数据可提供副本数据库请求刷新）。该示例也使用触发器。

想象一个存在主数据库 `M1` 及副本数据库 `R1` 和 `R2` 的场景。

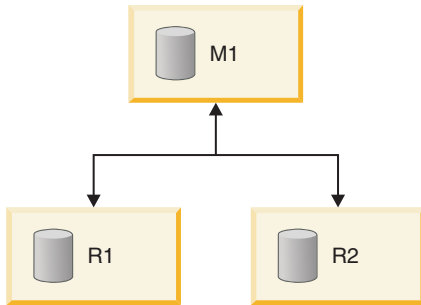


图 1. 同步拉取通知

要执行“同步拉取通知”，请执行下列步骤：

1. 在主数据库 M1 中定义过程 Pm1。在过程 Pm1 中，包括下列语句：

```
EXECDIRECT CALL Pr1 AT R1;
EXECDIRECT CALL Pr1 AT R2;
```

对于所涉及的每个副本数据库，都要执行一次调用。即使副本数据库的名称有变化，每个副本数据库中过程名称通常相同。

2. 在副本数据库 R1 中定义过程 Pr1。如果主数据库将要在多个副本数据库中调用 Pr1，那么应该为所涉及的每个副本数据库定义 Pr1。请参阅以下示例节中的副本数据库过程示例。

3. 为所有相关的 DML 操作定义触发器，例如，这些操作包括：

- INSERT
- UPDATE
- DELETE

4. 在每个触发器主体中，嵌入以下语句：

```
EXECDIRECT START [UNIQUE] CALL Pm1;
```

5. 将 EXECUTE 权限授予每个副本数据库中的适当用户。（副本数据库中的用户 Ur1 应该已映射到主数据库中的相应用户 Um1。用户 Um1 必须执行以下语句：

```
EXECDIRECT START [UNIQUE] CALL Pm1;
```

当 Um1 以远程方式调用该过程时，在副本数据库中执行该调用期间，该调用实际上使用 Ur1 的特权执行。

示例：切片副本数据库

在该示例中，销售程序有一个名为 CUSTOMER 的列表，它包括一个名为 SALESMAN 的列。主数据库包含所有销售员的信息。每位销售人员都有她自己的副本数据库，该副本数据库只含有主数据库数据切片；每个销售人员的副本数据库具有该销售人员的数据切片。例如，销售员 Smith 的副本数据库只包含销售员 Smith 的数据。如果对特定客户分配的销售员发生变动，那么应该通知正确的副本数据库。如果对 XYZ 公司分配的销售员由 Smith 更改为 Jones，那么销售员 Jones 的副本数据库应该添加与 XYZ 公司相关的数据，而销售员 Smith 的副本数据库应该删除与 XYZ 公司相关的数据。以下是用于同时更新这两个副本数据库的代码：

```
-- 如果把客户重新分配给一位不同的销售人员，我们应
-- must notify both the old and new salesmen.
-- NOTE: This sample shows only the "UPDATE" trigger, but in
-- reality, you'd also need to define INSERT and DELETE triggers.
```



```

CREATE TRIGGER T_CUST_AFTERUPDATE ON CUSTOMER
AFTER UPDATE
REFERENCING NEW SALESMAN AS NEW_SALESMAN,
REFERENCING OLD SALESMAN AS OLD_SALESMAN
BEGIN
IF NEW_SALESMAN <> OLD_SALESMAN THEN
EXEC SQL EXECDIRECT
  START AFTER COMMIT
  FOR EACH REPLICa WHERE NAME=OLD_SALESMAN
  UNIQUE CALL CUST(OLD_SALESMAN);
EXEC SQL EXECDIRECT
  START AFTER COMMIT
  FOR EACH REPLICa WHERE NAME=NEW_SALESMAN
  UNIQUE CALL CUST(NEW_SALESMAN);
ENDIF
END;

```

假定在应用程序中，用户将销售区域“CA”中的所有客户分配给销售员 Mike。

```

UPDATE CUSTOMER SET SALESMAN='Mike' WHERE SALES_AREA='CA';
COMMIT WORK;

```

主控服务器包含以下过程:

```

CREATE PROCEDURE CUST(salesman VARCHAR)
BEGIN
EXEC SQL EXECDIRECT CALL CUST(salesman) AT salesman;
COMMIT WORK;
END

```

每个副本服务器包含以下过程:

```

CREATE PROCEDURE CUST(salesman VARCHAR)
BEGIN
MESSAGE s BEGIN;
MESSAGE s APPEND REFRESH CUSTS(salesman);
MESSAGE s END;
COMMIT WORK;
MESSAGE s FORWARD TIMEOUT FOREVER;
COMMIT WORK;
END

```

在过程 CUST() 中，我们强制销售员的副本数据库根据主数据库中的数据执行刷新。在所有副本数据库中都定义了过程 CUST()。如果对重新分配了客户的副本数据库以及先前对已分配客户的副本数据库都调用该过程，那么该过程将同时更新这两个副本数据库。实际上，这将从不再包含此客户的副本数据库中删除过时的数据，并且会将该数据插入到现在对此客户负责的副本数据库。如果已正确地定义发布及其参数，那么我们不需要编写其他详细的逻辑来处理每项可能的操作，例如将客户从一位销售员重新分配给另一位销售员；而是，我们只需指示每个副本数据库根据最新数据执行刷新。

注:

- 可以在不使用触发器的情况下实现“同步拉取通知”。应用程序可以调用适当的过程以实现同步拉取。除语句 START AFTER COMMIT 和远程过程调用以外，触发器也是一种实现“同步拉取通知”的方法。
- 有时，在“同步拉取通知”过程中，副本数据库有可能必须毫无必要地额外交换一轮消息。发生这种情况的原因可能是，主数据库调用的过程尝试将消息发送到刚刚将更改发到主数据库并导致更改主数据库中“热数据”的副本数据库。但是，您可以通过谨慎地使用 START AFTER COMMIT 语句来避免这种情况。务必确保不要创建“无穷循环”，即，对主数据库执行的每次更新操作都将导致立即更新副本数据库，而

这又导致立即更新主数据库... 避免这种情况的最佳方法是，在可能将经过更新的数据“立即”发送至主数据库并且主数据库又将“立即”通知副本数据库再次执行刷新的副本数据库中创建触发器时，务必谨慎。

跟踪后台作业的执行： `START AFTER COMMIT` 语句返回包含一个 `INTEGER` 列的结果集。此整数是唯一的“作业”标识，并可用于查询由于 `SQL` 语句无效、无访问权以及副本数据库不可用等原因而未能启动的语句的状态。

达到未落实的延迟型过程调用语句的最大数目之后，如果继续发出延迟型过程调用，那么将返回错误。您可以在 `solid.ini` 中配置最大数目。请参阅《*IBM solidDB 管理员指南*》。

如果某个语句无法启动，那么故障原因将记录到系统表 `SYS_BACKGROUNDJOB_INFO` 中。

```
SYS_BACKGROUNDJOB_INFO
(
  ID INTEGER NOT NULL,
  STMT WVARCHAR NOT NULL,
  USER_ID INTEGER NOT NULL,
  ERROR_CODE INTEGER NOT NULL,
  ERROR_TEXT WVARCHAR NOT NULL,
  PRIMARY KEY(ID)
);
```

只有失败的 `START AFTER COMMIT` 语句才会记录到此表中。如果语句（例如，过程调用）成功启动，那么不会将任何信息存储到此系统表中。

用户可以通过使用 `SQL SELECT` 查询或者通过调用系统过程 `SYS_GETBACKGROUNDJOB_INFO` 从 `SYS_BACKGROUNDJOB_INFO` 表中检索信息。输入参数是作业标识。返回的值是：`ID INTEGER`、`STMT WVARCHAR`、`USER_ID INTEGER`、`ERROR_CODE INTEGER` 和 `ERROR_TEXT INTEGER`。

并且，如果语句无法启动，那么将发出事件 `SYS_EVENT_SACFAILED`。

```
CREATE EVENT SYS_EVENT_SACFAILED (ENAME WVARCHAR,
POSTSRVTIME TIMESTAMP,
UID INTEGER,
NUMDATAINFO INTEGER,
TEXTDATA WVARCHAR);
```

`NUMDATAINFO` 字段包含作业标识。应用程序可以等待此事件并使用作业标识从系统表 `SYS_BACKGROUNDJOB_INFO` 中检索原因。

您可以使用管理命令 `cleanbgjobinfo` 来清空系统表 `SYS_BACKGROUNDJOB_INFO`。必须具有 `DBA` 特权才能执行此命令，这意味着只有 `DBA` 才能从该表中删除行。

控制后台任务： 您可以通过 `SSC API` 和管理命令来控制后台任务（有关 `SSC API` 的详细信息，请参阅《*IBM solidDB 共享内存访问和链接库访问用户指南*》）。对于执行通过 `START AFTER COMMIT` 启动的语句的任务，服务器使用任务类型 `SSC_TASK_BACKGROUND`。注意，可能存在多个这样的任务，但您无法逐个地对其进行控制。

3.1.2 存储过程 - 外部

`solidDB` 支持用 `C` 编程语言编写的外部存储过程。通过编写 `C` 程序，外部存储过程可令您扩展服务器功能。

使用 CREATE PROCEDURE (外部) 语句注册外部存储过程。

使用操作系统提供的标准动态库界面 (Linux 和 UNIX 中的共享库及 Windows 中的 DLL) 访问外部过程。如果要允许使用外部存储过程, 您必须使用共享内存访问 (SMA) 或链接库访问 (LLA) 将您的应用程序链接至 solidDB。

用与调用 SQL 存储过程同样的 CALL 语句语法调用外部过程; 系统会把线程控制传递给外部库例程, 直至其返回。此外, 外部过程的数据库界面与 ODBC 应用程序的相似, 唯一不同的是, 它的连接句柄 (hdbc) 会传递给外部例程, 并且, 它能立即开始使用数据库界面而不用建立连接。

使用 DROP PROCEDURE 语句除去外部外部过程。

相关参考

第 175 页的 A.15, 『CREATE PROCEDURE (外部)』

第 151 页的 A.7, 『CALL』

第 201 页的 A.33, 『DROP PROCEDURE』

3.2 函数

函数是由函数名称指示的一项操作, 在函数后面跟着一对括号括住的自变量规范(可能没有自变量)。除了内置函数外, solidDB 支持用户定义的内部和外部存储函数。

内置函数在数据库引擎内提供。

使用 CREATE FUNCTION 语句将用户定义的函数注册到数据库中。用户定义的函数有两类:

- 用户定义的存储函数可用 solidDB 专有 SQL 过程语言编写。
- 用户定义的外部存储函数可用 C 编程语言编写。外部存储函数在运行时使用操作系统提供的标准动态库界面装入。通过编写 C 程序, 外部存储过程可令您扩展服务器功能。

相关参考

第 287 页的附录 B, 『函数』

函数是由函数名称指示的一项操作, 在函数后面跟着一对括号括住的自变量规范(可能没有自变量)。除了内置函数外, solidDB 支持用户定义的内部和外部存储函数。

第 157 页的 A.11, 『CREATE FUNCTION』

第 159 页的 A.12, 『CREATE FUNCTION (外部)』

3.3 触发器

触发器是用于在特定操作 (INSERT、UPDATE 或 DELETE) 发生时执行一系列 SQL 语句的机制。该触发器包括一个调用时需要执行的 SQL 语句。使用 solidDB 专有存储过程语法创建触发器。

您可以在表上创建一个或更多的触发器, 每个触发器都定义为用于激活特定的 INSET、UPDATE 或 DELETE 命令。当用户修改表中的数据时, 将激活与该命令相对应的触发器。

只能将直接插入 SQL 或存储过程与触发器配合使用。如果您在该触发器中使用存储过程，则必须用 CREATE PROCEDURE 创建该过程。从触发器主体中调用的过程可以调用其他触发器。

触发器使您能够执行下列操作：

- 执行如“检查某些条件是否处于维护状态”这样的特殊完整性约束，例如，防止用户作出不正确或不一致的数据更改。
- 根据修改之前或之后的行值执行操作。
- 将大量逻辑处理转移到服务器，从而减少应用程序需要执行的工作量以及降低网络流量。

3.3.1 触发器 – 操作规则

在 solidDB 的 DML 执行模型中，solidDB 服务器先执行多项验证检查，然后再执行数据处理语句（INSERT、UPDATE 或 DELETE）。对于单一 DML 语句，数据验证、触发器执行和完整性约束检查的执行顺序如下所示。

1. 如果值是语句的组成部分（即，未绑定），那么对其进行验证。这包括 NULL 值检查以及数据类型检查（例如，检查是否为数字）等。
2. 执行表级安全性检查。
3. 对于该 SQL 语句所影响的每一行进行循环。对于每一行，按以下顺序执行下列操作：
 - a. 执行列级安全性检查。
 - b. 触发 BEFORE 行触发器。
 - c. 如果值已绑定，那么对其进行验证。这包括 NULL 值检查、数据类型检查和大小检查（例如，检查字符串是否过长）。

注：即使对于未绑定的值，也将执行大小检查。

 - d. 执行 INSERT/UPDATE/DELETE
 - e. 触发 AFTER 行触发器
4. 落实语句
 - a. 执行并行冲突检查。
 - b. 执行重复值检查。
 - c. 对调用 DML 执行引用完整性检查。

注：触发器本身可以导致 DML 执行，这适用于以上模型所示的步骤。

3.3.2 创建和修改触发器

使用 CREATE TRIGGER 语句创建触发器。

通过使用 ALTER TRIGGER 语句，可以禁用现有触发器或者对表定义的所有触发器。ALTER TRIGGER 语句将导致 solidDB 服务器在发出激活的 DML 语句时忽略触发器。借助此语句，还可以启用当前处于不活动状态的触发器。

要从系统目录中删除触发器，请使用 DROP TRIGGER 语句。

相关参考

第 149 页的 A.5, 『ALTER TRIGGER』

第 191 页的 A.23, 『CREATE TRIGGER』

第 208 页的 A.44, 『DROP TRIGGER』

3.3.3 触发器与过程

触发器可以调用存储过程并致使 solidDB 服务器执行其他触发器。您可以在触发器主体中调用过程。您也可以定义只包含过程调用的触发器主体。从触发器主体中调用的过程可以调用其他触发器。

在触发器主体中使用存储过程时, 首先必须使用 CREATE PROCEDURE 语句来存储该过程。

在过程定义中, 可以使用 COMMIT 和 ROLLBACK 语句。但是, 在触发器主体中, 不能使用 COMMIT (其中包括 AUTOCOMMIT 和 COMMIT WORK) 以及 ROLLBACK 语句。只能使用 WHENEVER SQLERROR ABORT 语句。

缺省情况下, 触发器最多可以嵌套 16 层。使用 SQL.MaxNestedTriggers 参数可更改该限定。如果触发器进入无穷循环, 则 solidDB 将在嵌套层数达到最大值时检测到这种递归操作, 并向用户返回错误。例如, 可以通过尝试对表 T1 执行插入来激活触发器, 该触发器可以调用一个存储过程, 后者也尝试对 T1 执行插入, 从而以递归方式激活该触发器。

如果一组嵌套触发器在任何时候失败, solidDB 会回滚最初激活该触发器的语句。

设置缺省值或派生的列

您可以创建触发器, 以便在 INSERT 和 UPDATE 操作中设置缺省值或派生的列值。使用 CREATE TRIGGER 命令来创建用于此用途的触发器时, 该触发器必须遵循下列规则:

- 该触发器必须在 INSERT 或 UPDATE 操作之前执行。只能使用 BEFORE 触发器来修改列值。由于必须在 INSERT 或 UPDATE 操作之前设置列值, 因此使用 AFTER 触发器来设置列值没有意义。另请注意, DELETE 操作不适用于修改列值。
- 对于 INSERT 和 UPDATE 操作而言, REFERENCING 子句必须包含用于执行修改的新列值。注意, 修改旧列值没有意义。
- 可以通过简单地更改引用部分中定义的变量值来设置新列值。

对触发器使用参数和变量

当您更新记录并且该更新调用了触发器时, 触发器本身可以更改该记录中某些列的值。在某些情况下, 您可能想在触发器中同时引用“旧”值和“新”值。

REFERENCING 子句允许为旧值和新值创建“别名”, 以便在同一个触发器中引用其中任何一个值。例如, 假定有两个表, 一个表保存客户信息, 另一个表保存发票信息。该表除了存储每张发票的金额以外, 对于每个客户还包含一个“total_bought”字段; 这个“total_bought”字段包含发送给此客户的所有发票的累计总金额。(此字段可以用来标识大客户。)

每当发票的 total_amount 更新时, 客户表中该客户的记录的“total_bought”值也将更新。执行此更新时, 将减去发票中存储的旧值的金额, 并加上发票中新值的金额。例如,

如果客户的发票以前金额为 \$100，现在更改为 \$150，那么将在“total_bought”字段中减去 \$100，然后加上 \$150。通过正确地使用 REFERENCING 子句，触发器可以同时“看到”旧值和价格列，从而能够更新 total_bought 列。

由 REFERENCING 子句创建的列别名只在触发器中有效。

示例: 通过引用子句使用触发器

```
-- 该 SQL 简要的展示了如何使用
-- "REFERENCING OLD AS old_col, REFERENCING NEW AS new_col"
-- to have simultaneous access to both the "OLD" and "NEW"
-- column values of the field while inside a trigger.
-- In this scenario, we have customers and invoices.
-- For each customer, we keep track of the cumulative total of
-- all purchases by that customer.
-- Each invoice stores the total amount of all purchases on
-- that invoice. 如果必须要
-- adjusted, then the cumulative value of that customer's
-- purchases must also be adjusted.
-- Therefore, we update the cumulative total by subtracting
-- the "old" price on the invoice and adding the "new" price.
-- For example, if the amount on a customer's invoice was
-- changed from $100 to $150 (an increase of $50), then we
-- would update the customer's cumulative total by
-- subtracting $100 and adding $150 (a net increase of $50).
-- Drop the sample tables if they already exist.
DROP TABLE customers;
DROP TABLE invoices;
CREATE TABLE customers (
    customer_id INTEGER,    -- ID for each customer.
    total_bought FLOAT     -- The cumulative total price of
                          -- all this customer's purchases.
);
-- Each customer may have 0 or more invoices.
CREATE TABLE invoices (
    customer_id INTEGER,
    invoice_id INTEGER,    -- unique ID for each invoice
    invoice_total FLOAT    -- total price for this invoice
);
-- If the total_price on an invoice changes, then
-- update customers.total_bought to take into account
-- the change. Subtract the old invoice price and add the
-- new invoice price.
"CREATE TRIGGER old_and_new ON invoices
AFTER UPDATE
    REFERENCING OLD invoice_total AS old_invoice_total,
    REFERENCING NEW invoice_total AS new_invoice_total,
    -- If the customer_id doesn't change, we could use
    -- either the NEW or OLD customer_id.
    REFERENCING NEW customer_id AS new_customer_id
BEGIN
    EXEC SQL PREPARE upd_curs
        UPDATE customers
        SET total_bought = total_bought - ? + ?
        WHERE customers.customer_id = ?;
    EXEC SQL EXECUTE upd_curs
        USING (old_invoice_total, new_invoice_total,
              new_customer_id);
    EXEC SQL CLOSE upd_curs;
    EXEC SQL DROP upd_curs;
END";
-- When a new invoice is created, we update the total_bought
-- in the customers table.
"CREATE TRIGGER update_total_bought ON invoices
AFTER INSERT
    REFERENCING NEW invoice_total AS new_invoice_total,
```

```

REFERENCING NEW customer_id AS new_customer_id
BEGIN
    EXEC SQL PREPARE ins_curs
        UPDATE customers
            SET total_bought = total_bought + ?
            WHERE customers.customer_id = ?;
    EXEC SQL EXECUTE ins_curs
        USING (new_invoice_total, new_customer_id);
    EXEC SQL CLOSE ins_curs;
    EXEC SQL DROP ins_curs;
END";
-- Insert a sample customer.
INSERT INTO customers (customer_id, total_bought)
VALUES (1000, 0.0);
-- Insert invoices for a customer; the INSERT trigger will
-- update the total_bought in the customers table.
INSERT INTO invoices (customer_id, invoice_id, invoice_total)
VALUES (1000, 5555, 234.00);
INSERT INTO invoices (customer_id, invoice_id, invoice_total)
VALUES (1000, 5789, 199.0);
-- Make sure that the INSERT trigger worked.
SELECT * FROM customers;
-- Now update an invoice; the total_bought in the customers
-- table will also be updated and the trigger that does
-- this will use the REFERENCING clauses
--     REFERENCING NEW invoice_total AS new_invoice_total,
--     REFERENCING OLD invoice_total AS old_invoice_total
UPDATE invoices SET invoice_total = 235.00
WHERE invoice_id = 5555;
-- Make sure that the UPDATE trigger worked.
SELECT * FROM customers;
COMMIT WORK;

```

3.3.4 触发器与事务

触发器不要求从调用事务中执行落实即可触发；DML 语句本身即可导致触发器触发。在触发器主体中，也不允许使用 COMMIT WORK。

在过程定义中，可以使用 COMMIT 和 ROLLBACK 语句。但在触发器主体中，不能使用 COMMIT 和 ROLLBACK 语句。只能使用 WHENEVER SQLERROR ABORT 语句。注意，如果自动落实功能处于打开状态，那么触发器中的每条语句都不会被视为独立的语句，并且不会在执行时落实；而是，整个触发器主体作为导致该触发器被触发的 INSERT、UPDATE 或 DELETE 语句的组成部分执行。整个触发器（以及导致该触发器被触发的语句）都将落实，否则整个触发器（以及导致该触发器被触发的语句）都将回滚。

递归和并行冲突错误

如果 DML 语句更新/删除某一行并导致触发器被触发，那么您将无法在该触发器中再次更新/删除同一行。在这种情况下，AFTER 触发器事件可能会引起递归错误，而 BEFORE 触发器事件可能会引起并行冲突错误。

下列各节说明这些术语、提供引起这些问题的触发器的一些示例并提供一个表（如第 63 页的『触发器情况摘要』所示）以指示那些将会以及不会引起递归错误或并行冲突错误的触发器情况。

触发器与递归： 如果代码将导致自身再次执行，那么该代码被称为“递归”代码。例如，调用自身的存储过程被称为递归存储过程。在存储过程中，递归功能有时很有用。另一方面，触发器可能会创建类型更为隐蔽的递归，这种递归无效，并且将被 solidDB 服务器禁止。如果触发器中包含的语句将导致对同一记录再次执行同一触发器，那么该

触发器被称为递归触发器。例如，对于删除触发器而言，如果它尝试删除一个记录，而删除该记录将触发同一触发器，那么该触发器被称为递归触发器。

如果数据库服务器允许触发器中存在递归情况，那么该服务器将进入“无穷循环”，并且永远不会完成执行导致该触发器被触发的语句。如果触发器执行的操作尝试在同一个 SQL 语句中执行同一类型的操作（例如删除操作），从而与导致该触发器被触发的语句发生“竞争”关系，那么将发生并行冲突错误。例如，如果创建将在一个记录被删除时触发的触发器，并且该触发器尝试删除同一记录，而该记录被删除时将导致该触发器被触发，那么实际上将有两个不同的“并发”删除语句“竞争”删除该记录；这将导致发生并行冲突。下一节提供了一个有缺陷的删除触发器的示例。

导致递归的有缺陷触发器的示例

本节中的示例仅仅说明触发器所涉及到的众多约束和规则中的一小部分。

在此场景中，一位职员辞职，因此需要将医疗保险退保。此外，该职员的家眷也需要将医疗保险退保。这种情况的业务规则通过创建触发器实现；该触发器将在职员的记录被删除时执行，然后，该触发器中的语句将删除该职员的家眷。（此示例假定职员及其家眷存储在同一个表中；在现实世界中，家眷通常存储在另一个表中。此示例还假定每个家族的名字都唯一。）

```
CREATE TRIGGER do_not_try_this ON employees_and_dependents
AFTER DELETE
REFERENCING OLD last_name AS old_last_name
BEGIN
  EXEC SQL PREPARE del_cursor
  DELETE FROM employees_and_dependents
  WHERE last_name = ?;
  EXEC SQL EXECUTE del_cursor USING (old_last_name);
  -- ... close and drop the cursor.
END;
```

假定职员“John Smith”辞职，因此您需要删除他的医疗保险。您删除“John Smith”时，删除 John Smith 后将立即调用该触发器，该触发器将尝试删除所有名为“John Smith”的记录，其中不仅包括该职员的家眷，还包括该职员本身，原因是他的名字符合 WHERE 子句中的条件。

每次尝试删除该职员的记录时，此操作都将再次导致该触发器被触发。于是，代码将再次导致该触发器被触发，然后再次尝试执行删除，从而以递归方式不断尝试删除该职员。如果数据库服务器不禁止此类递归或者检测这种情况，那么服务器可能会进入无穷循环。如果服务器检测这种情况，那么它将返回相应的错误，例如“嵌套的触发器过多”。

对于 UPDATE 而言，可能会发生类似的情况。假定一个触发器在记录每次被更新时累计销售税。下面是一个将会导致递归错误的示例：

```
CREATE TRIGGER do_not_do_this_either ON invoice
AFTER UPDATE
REFERENCING NEW total_price AS new_total_price
BEGIN
  -- Add 8% sales tax.
  EXEC SQL PREPARE upd_curs1
  UPDATE invoice SET total_price = 1.08 * total_price
  WHERE ...;
  -- ... execute, close, and drop the cursor...
END;
```


在此场景中，客户 Ann Jones 打电话要求更改订单；新价格（含销售税）通过将新的小计乘以 1.08 计算而得。将对记录进行更新以使其包含新的总价；该记录每次被更新时，该触发器都将被触发，因此更新一次记录将导致该触发器再次更新该记录，而这些更新操作将在无穷循环中反复执行。

既然 AFTER 触发器会导致递归或循环，那么 BEFORE 触发器的情况如何？回答是，在某些情况下，BEFORE 触发器会导致并行问题。让我们返回到第一个触发器示例（该触发器用于删除职员及其家眷的医疗保险）。如果该触发器是 BEFORE 触发器（而不是 AFTER 触发器），那么就在该职员被删除之前，我们将执行该触发器，在本例中，该触发器将删除每个名为 John Smith 的人员。该触发器执行完成后，引擎将继续执行它的原始任务（即，删除职员 John Smith 本身），但服务器会发现该职员已不存在，或者由于该职员的记录已被标记为即将删除而无法将其删除 - 换言之，由于尝试删除同一个记录两次而引起并行冲突。

触发器情况摘要：除了上一节中描述的示例以外，下表概述了其他一些情况，包括那些涉及 INSERT 以及 UPDATE 和 DELETE 的情况。

表分为以下五列：

- 触发器方式（即，BEFORE 或 AFTER）
- 操作（INSERT、DELETE 或 UPDATE）
- 触发器操作（触发器本身尝试执行的操作，例如更新刚刚插入的记录）
- 锁定类型（“乐观”或“悲观”）
- 您将看到的结果（例如，触发器操作成功，或者触发器由于上一节中讨论的递归错误之类的原因而失败）。

有关此表中的触发器条目的解释的详细信息，请参阅本节中后面的示例条目 1。

表 11. BEFORE/AFTER 触发器的 INSERT/UPDATE/DELETE 操作

触发器方式	操作	触发器操作	锁定类型	结果
AFTER	INSERT	通过对值添加一个数字来更新同一行	乐观	记录被更新。
AFTER	INSERT	通过对值添加一个数字来更新同一行	悲观	记录被更新。
BEFORE	INSERT	通过对值添加一个数字来更新同一行	乐观	因为触发器主体中 UPDATE 的 WHERE 条件返回了 NULL 结果集（例如期望的行仍未插入到表中），所以无法更新记录。
BEFORE	INSERT	通过对值添加一个数字来更新同一行	悲观	因为触发器主体中 UPDATE 的 WHERE 条件返回了 NULL 结果集（例如期望的行仍未插入到表中），所以无法更新记录。
AFTER	INSERT	删除正在插入的行	乐观	记录被删除。

表 11. BEFORE/AFTER 触发器的 INSERT/UPDATE/DELETE 操作 (续)

触发器方式	操作	触发器操作	锁定类型	结果
AFTER	INSERT	删除正在插入的行	悲观	记录被删除。
BEFORE	INSERT	删除正在插入的行	乐观	由于触发器主体中 DELETE 的 WHERE 条件返回 NULL 结果集（这是因为，尚未在表中插入期望的行），因此不删除记录。
BEFORE	INSERT	删除正在插入的行	悲观	因为触发器主体中 UPDATE 的 WHERE 条件返回了 NULL 结果集（例如期望的行仍未插入到表中），所以无法更新记录。
AFTER	INSERT	插入行	乐观	嵌套的触发器过多。
AFTER	INSERT	插入行	悲观	嵌套的触发器过多。
BEFORE	INSERT	插入行	乐观	嵌套的触发器过多。
BEFORE	INSERT	插入行	悲观	嵌套的触发器过多。
AFTER	UPDATE	通过对值添加一个数字来更新同一行	乐观	生成 Solid® 表错误：嵌套的触发器过多。
AFTER	UPDATE	通过对值添加一个数字来更新同一行	悲观	生成 Solid 表错误：嵌套的触发器过多。
BEFORE	UPDATE	通过对值添加一个数字来更新同一行。	乐观	记录被更新，但是未进入嵌套循环，这是因为触发器主体中的 WHERE 条件返回了 NULL 结果集并且未更新任何行，因此未以递归方式触发触发器。
BEFORE	UPDATE	通过对值添加一个数字来更新同一行。	悲观	记录被更新，但是未进入嵌套循环，这是因为触发器主体中的 WHERE 条件返回了 NULL 结果集并且未更新任何行，因此未以递归方式触发触发器。
AFTER	UPDATE	删除正在更新的行	乐观	记录被删除。
AFTER	UPDATE	删除正在更新的行	悲观	记录被删除。
BEFORE	UPDATE	删除正在更新的行	乐观	发生并行冲突错误。
BEFORE	UPDATE	删除正在更新的行	悲观	发生并行冲突错误。

表 11. BEFORE/AFTER 触发器的 INSERT/UPDATE/DELETE 操作 (续)

触发器方式	操作	触发器操作	锁定类型	结果
AFTER	DELETE	插入值相同的行。	乐观	在删除之后插入同一个记录。
AFTER	DELETE	插入值相同的行。	悲观	触发触发器时挂起。
BEFORE	DELETE	插入值相同的行。	乐观	在删除之后插入同一个记录。
BEFORE	DELETE	插入值相同的行。	悲观	触发触发器时挂起。
AFTER	DELETE	插入值相同的行。	乐观	记录被删除。
AFTER	DELETE	通过对值添加一个数字来更新同一行。	悲观	记录被删除。
BEFORE	DELETE	通过对值添加一个数字来更新同一行。	乐观	记录被删除。
BEFORE	DELETE	通过对值添加一个数字来更新同一行	悲观	记录被删除。
AFTER	DELETE	删除同一行	乐观	嵌套的触发器过多。
AFTER	DELETE	删除同一记录	悲观	嵌套的触发器过多
BEFORE	DELETE	删除同一记录	乐观	发生并行冲突错误。
BEFORE	DELETE	删除同一记录	悲观	发生并行冲突错误。

以下是表中的示例条目及该条目的说明:

表 12. 示例条目 1

触发器	操作	触发器操作	锁定类型	结果
AFTER	INSERT	通过对值添加一个数字来更新同一行	乐观	记录被更新。

在这种情况下，我们有一个在执行 INSERT 操作之后触发的触发器。触发器的主体包含在插入之后更新同一行（即，与触发触发器的那一行相同）的语句。如果锁定类型是“乐观”，那么结果将是该记录被更新。（因为未发生冲突，所以是否进行锁定（乐观锁定或悲观锁定）没有区别）。

注意，在这种情况下，不会有递归问题，即使我们更新刚刚插入的行也一样。“触发”触发器的操作与触发器中执行的操作不同，因此不会引起递归/循环情况。

以下是表中的另一个示例:

表 13. 示例条目 2

触发器	操作	触发器操作	锁定类型	结果
BEFORE	INSERT	通过对值添加一个数字来更新同一行	乐观	因为触发器主体中 UPDATE 的 WHERE 条件返回了 NULL 结果集（例如期望的行仍未插入到表中），所以无法更新记录。

在本例中，我们尝试插入记录，但在执行插入之前，触发器运行。在本例中，触发器将尝试更新该记录（例如添加销售税）。但是，因为尚未插入该记录，所以触发器中的 UPDATE 命令找不到该记录，导致无法添加销售税。因此，结果相当于从未触发该触发器。没有错误消息，因此您不会立即认识到触发器未执行预期操作的原因。

有缺陷的触发器: 在以下示例中，触发器逻辑有缺陷，即，在 BEFORE UPDATE 触发器中删除了同一行；这将导致 solidDB 生成并行冲突错误。

有缺陷的触发器

```
DROP EMP;
COMMIT WORK;

CREATE TABLE EMP(C1 INTEGER);
INSERT INTO EMP VALUES (1);
COMMIT WORK;

"CREATE TRIGGER TRIG1 ON EMP
  BEFORE UPDATE
  REFERENCING OLD C1 AS OLD_C1
BEGIN
  EXEC SQL WHENEVER SQLERROR ABORT;
  EXEC SQL PREPARE CUR1 DELETE FROM EMP WHERE C1 = ?;
  EXEC SQL EXECUTE CUR1 USING (OLD_C1);
END";

UPDATE EMP SET C1=200 WHERE C1 = 1;
SELECT * FROM EMP;

ROLLBACK WORK;
```

注:

如果所更新/删除的行基于唯一键，而不是基于普通的列（如以上示例的情况），那么 solidDB 将生成以下错误消息：*1001: 找不到键值。*

为了避免递归和并行冲突错误，请确保检查应用程序逻辑并采取预防措施，从而确保应用程序不会导致两个事务更新或删除同一行。

错误处理: 如果一个过程返回错误给触发器，那么该触发器将使它的调用 DML 命令失败并返回错误。要在执行 DML 语句期间自动返回错误，必须在触发器主体中使用 WHENEVER SQLERROR ABORT 语句。否则，必须在触发器主体中的每个过程调用或 SQL 语句后显式地检查错误。

对于用户编写的作为触发器主体组成部分的业务逻辑中的错误，用户必须使用 RETURN SQLERROR 语句。有关详细信息，请参阅第 67 页的 3.3.6，『从触发器中引发错误』。

如果未指定 RETURN SQLERROR, 那么 SQL 语句执行失败时, 系统将返回缺省错误消息。当前 DML 语句对数据库所作的任何更改都将被撤销, 并且事务仍处于活动状态。实际上, 即使触发器执行失败, 事务也不会回滚, 但当前执行语句将回滚。

注:

触发 SQL 语句是调用事务的组成部分。如果调用 DML 语句由于触发器或者该触发器外部生成的另一错误而失败, 那么该触发器中的所有 SQL 语句都将随失败的调用 DML 命令一起回滚。

调用事务负责落实或回滚触发器的过程中执行的任何 DML 语句。但是, 如果相关联的触发器导致调用触发器的 DML 命令失败, 那么此规则不适用。在这种情况下, 该触发器的过程中执行的任何 DML 语句都将自动回滚。

COMMIT 和 ROLLBACK 语句必须在触发器主体外部执行, 而不能在触发器主体内执行。如果在触发器主体内或者在触发器主体或另一个触发器中调用的过程内执行 COMMIT 或 ROLLBACK, 那么该用户将接收到运行时错误。

嵌套的触发器与递归触发器: 如果触发器进入无穷循环, 那么 solidDB 服务器将在嵌套层数达到 16 (或者达到 MaxNestedTriggers 系统参数最大值) 时检测到这种递归操作。例如, 对表 T1 执行的插入尝试将激活一个触发器, 该触发器可能调用一个存储过程, 后者也尝试对表 T1 执行插入, 从而以递归方式激活该触发器。对于用户的插入尝试, solidDB 服务器将返回错误。

如果一组嵌套的触发器在任何时候失败, 那么 solidDB 服务器将回滚最初激活触发器的命令。

3.3.5 触发器特权和安全性

由于触发器可能会在用户尝试插入、更新或删除数据时被激活, 因此, 无需任何特权即可执行触发器。

用户调用触发器时, 该用户将采用该触发器所基于的表的所有者的特权。操作语句将以表所有者的身份执行, 而不是以激活触发器的用户的身份执行。但是, 要创建使用存储过程的触发器, 触发器的创建者必须满足下列其中一个条件:

- 具有 DBA 特权。
- 是正在定义的触发器所基于的表的所有者。
- 被授予表的所有特权。

如果创建者具有 DBA 权限并为另一个用户创建表, 那么 solidDB 服务器将假定 TRIGGER 命令中指定的未限定名称属于该用户。例如, 以下命令将采用 DBA 权限执行:

```
CREATE TRIGGER A.TRIG ON EMP BEFORE UPDATE
```

由于未对 EMP 表进行限定, 因此 solidDB 服务器假定限定的表名为 A.EMP, 而不是 DBA.EMP。

3.3.6 从触发器中引发错误

有时, 您执行触发器时可能会接收到错误。此错误可能是由于执行 SQL 语句或业务逻辑所致。如果触发器返回错误, 那么将导致调用 DML 命令失败。要在执行 DML 语句

期间自动返回错误，必须在触发器主体中使用 `WHENEVER SQLERROR ABORT` 语句。否则，必须在触发器主体中的每个过程调用或 `SQL` 语句后显式地检查错误。

对于用户编写的作为触发器主体组成部分的业务逻辑错误，用户可以在过程变量中使用 `RETURN SQLERROR error_string` 或 `RETURN SQLERROR char_variable` 语句接收错误。

此错误将按以下格式返回：

用户错误: `error_string`

如果用户在触发器主体中未指定 `RETURN SQLERROR` 语句，则捕获的所有 `SQL` 错误都将使用由系统确定的缺省 `error_string` 引发。

注：触发 `SQL` 语句是调用事务的组成部分。如果调用 `DML` 语句由于触发器或者该触发器外部生成的另一错误而失败，那么该触发器中的所有 `SQL` 语句都将随失败的调用 `DML` 命令一起回滚。

以下示例使用 `WHENEVER SQLERROR ABORT` 来确保触发器捕获它所调用的存储过程中的错误。

```
-- 如果您从存储过程中返回 SQLERROR，则错误会
-- 显示。然而，如果存储过程是从
-- 触发器内部调用，则该错误不会显示，除非您使用
-- SQL 语句 WHENEVER SQLERROR ABO。
CREATE TABLE table1 (x INT);
CREATE TABLE table2 (x INT);

"CREATE PROCEDURE stproc1
BEGIN
    RETURN SQLERROR 'Here is an error!';
END";
COMMIT WORK;

"CREATE TRIGGER displays_error ON table1 BEFORE INSERT
BEGIN
    EXEC SQL WHENEVER SQLERROR ABORT;
    EXEC SQL EXECDIRECT CALL stproc1();
END";
COMMIT WORK;

"CREATE TRIGGER does_not_display_error ON table2 BEFORE INSERT
BEGIN
    EXEC SQL EXECDIRECT CALL stproc1();
END";
COMMIT WORK;

-- 如果您执行了存储过程，这会显示错误返回。CALL stproc1();

-- 显示错误是因为触发器有 WHENEVER SQL ERROR ABORT。INSERT INTO 表 1 (x) 值 (1);
-- 不显示错误。INSERT INTO 表 2 (x) 值 (1);
```

3.3.7 获取触发器信息

有以下几种方式可以获得触发器的信息：

- 使用 第 295 页的 B.7，『触发器函数』。
- 在触发器系统表 第 334 页的 E.1.34，『SYS_TRIGGERS』中执行查询操作。

3.3.8 触发器参数设置

设置嵌套触发器的最大数目

触发器可以调用其他触发器，此外也可以调用它自己（递归触发器）。嵌套或递归触发器的最大数目可通过 `SQL.MaxNestedTriggers` 参数设置。

```
[SQL]
MaxNestedTriggers = n;
```

其中， n 是嵌套触发器的最大数目。

嵌套触发器的缺省数目是 16。

设置触发器高速缓存

在独立的高速缓存中对触发器进行高速缓存。每个用户都有一个用于触发器的独立高速缓存。在执行触发器操作时，触发器的过程逻辑会高速缓存在触发器高速缓存中，并在触发器再次执行时重复使用。

可使用 `SQL.TriggerCache` 参数设置触发器高速缓存的大小。

```
[SQL]
TriggerCache = n;
```

其中， n 是为高速缓存保留的触发器的数目。

3.3.9 触发器示例

触发器示例

本示例显示简单触发器的工作方式。本示例包含一些能够正确工作的触发器以及一些包含错误的触发器。对于本示例中的成功触发器，将创建一个表（名为 `trigger_test`）并在该表中创建 6 个触发器。每个触发器在触发后，都会在另一个表（名为 `trigger_output`）中插入一个记录。在执行导致触发器触发的 DML 语句（INSERT、UPDATE 和 DELETE）之后，通过从 `trigger_output` 表中选择所有记录来显示触发器的结果。

```
DROP TABLE TRIGGER_TEST;
DROP TABLE TRIGGER_ERR_TEST;
DROP TABLE TRIGGER_ERR_B_TEST;
DROP TABLE TRIGGER_ERR_A_TEST;
DROP TABLE TRIGGER_OUTPUT;
COMMIT WORK;
-- Create a table that has a column for each of the possible trigger
-- types (for example, BI = a trigger that is on Insert
-- operations and that executes as a "Before" trigger).
CREATE TABLE TRIGGER_TEST(
    XX VARCHAR,
    BI VARCHAR, -- BI = Before Insert
    AI VARCHAR, -- AI = After Insert
    BU VARCHAR, -- BU = Before Update
    AU VARCHAR, -- AU = After Update
    BD VARCHAR, -- BD = Before Delete
    AD VARCHAR -- AD = After Delete
);
COMMIT WORK;

-- Table for 'before' trigger errors
CREATE TABLE TRIGGER_ERR_B_TEST(
    XX VARCHAR,
```

```

        BI VARCHAR,
        AI VARCHAR,
        BU VARCHAR,
        AU VARCHAR,
        BD VARCHAR,
        AD VARCHAR
    );

INSERT INTO TRIGGER_ERR_B_TEST VALUES('x','x','x','x','x',
    'x','x');
COMMIT WORK;

-- Table for 'after X' trigger errors
CREATE TABLE TRIGGER_ERR_A_TEST(
    XX VARCHAR,
    BI VARCHAR, -- Before Insert
    AI VARCHAR, -- After Insert
    BU VARCHAR, -- Before Update
    AU VARCHAR, -- After Update
    BD VARCHAR, -- Before Delete
    AD VARCHAR  -- After Delete
);

INSERT INTO TRIGGER_ERR_A_TEST VALUES('x','x','x','x','x',
    'x','x');
COMMIT WORK;

CREATE TABLE TRIGGER_OUTPUT(
    TEXT VARCHAR,
    NAME VARCHAR,
    SCHEMA VARCHAR
);
COMMIT WORK;

-----
-- Successful triggers
-----

-- Create a "Before" trigger on insert operations. When a record is
-- inserted into the table named trigger_test, then this trigger is
-- fired. When this trigger is fired, it inserts a record into the
-- "trigger_output" table to show that the trigger actually executed.

"CREATE TRIGGER TRIGGER_BI ON TRIGGER_TEST
    BEFORE INSERT
    REFERENCING NEW BI AS NEW_BI
BEGIN
    EXEC SQL PREPARE BI INSERT INTO TRIGGER_OUTPUT VALUES(
        'BI', TRIG_NAME(0), TRIG_SCHEMA(0));
    EXEC SQL EXECUTE BI;
    SET NEW_BI = 'TRIGGER_BI';
END";
COMMIT WORK;

"CREATE TRIGGER TRIGGER_AI ON TRIGGER_TEST
    AFTER INSERT
    REFERENCING NEW AI AS NEW_AI
BEGIN
    EXEC SQL PREPARE AI INSERT INTO TRIGGER_OUTPUT VALUES(
        'AI', TRIG_NAME(0), TRIG_SCHEMA(0));
    EXEC SQL EXECUTE AI;
    SET NEW_AI = 'TRIGGER_AI';
END";
COMMIT WORK;

"CREATE TRIGGER TRIGGER_BU ON TRIGGER_TEST
    BEFORE UPDATE
    REFERENCING NEW BU AS NEW_BU

```

```

BEGIN
    EXEC SQL PREPARE BU INSERT INTO TRIGGER_OUTPUT VALUES(
        'BU', TRIG_NAME(0), TRIG_SCHEMA(0));
    EXEC SQL EXECUTE BU;
    SET NEW_BU = 'TRIGGER_BU';
END";
COMMIT WORK;

"CREATE TRIGGER TRIGGER_AU ON TRIGGER_TEST
    AFTER UPDATE
    REFERENCING NEW AU AS NEW_AU
BEGIN
    EXEC SQL PREPARE AU INSERT INTO TRIGGER_OUTPUT VALUES(
        'AU', TRIG_NAME(0), TRIG_SCHEMA(0));
    EXEC SQL EXECUTE AU;
    SET NEW_AU = 'TRIGGER_AU';
END";
COMMIT WORK;

"CREATE TRIGGER TRIGGER_BD ON TRIGGER_TEST
    BEFORE DELETE
    REFERENCING OLD BD AS OLD_BD
BEGIN
    EXEC SQL PREPARE BD INSERT INTO TRIGGER_OUTPUT VALUES(
        'BD', TRIG_NAME(0), TRIG_SCHEMA(0));
    EXEC SQL EXECUTE BD;
    SET OLD_BD = 'TRIGGER_BD';
END";
COMMIT WORK;

"CREATE TRIGGER TRIGGER_AD ON TRIGGER_TEST
    AFTER DELETE
    REFERENCING OLD AD AS OLD_AD
BEGIN
    EXEC SQL PREPARE AD INSERT INTO TRIGGER_OUTPUT VALUES(
        'AD', TRIG_NAME(0), TRIG_SCHEMA(0));
    EXEC SQL EXECUTE AD;
    SET OLD_AD = 'TRIGGER_AD';
END";
COMMIT WORK;

-----
-- This attempt to create a trigger will fail. The statement
-- specifies the wrong data type for the error variable named
-- ERRSTR.
-----

"CREATE TRIGGER TRIGGER_ERR_AU ON TRIGGER_ERR_A_TEST
    AFTER UPDATE
    REFERENCING NEW AU AS NEW_AU
BEGIN
    -- The following line is incorrect; ERRSTR must be declared
    -- as VARCHAR, not INTEGER;
    DECLARE ERRSTR INTEGER;
    -- ...
    RETURN SQLERROR ERRSTR;
END";
COMMIT WORK;

-----
-- Trigger that returns an error message.
-----

"CREATE TRIGGER TRIGGER_ERR_BI ON TRIGGER_ERR_B_TEST
    BEFORE INSERT
    REFERENCING NEW BI AS NEW_BI
BEGIN
    -- ...

```

```

        RETURN SQLERROR 'Error in TRIGGER_ERR_BI';
END";
COMMIT WORK;

-----
-- Success trigger tests. These Insert, Update, and Delete
-- statements will force the triggers to fire. The SELECT
-- statements will show you the records in the trigger_test and
-- trigger_output tables.
-----

INSERT INTO TRIGGER_TEST(XX) VALUES ('XX');
COMMIT WORK;

-- Show the records that were inserted into the trigger_test
-- table. (The records for trigger_output are shown later.)

SELECT * FROM TRIGGER_TEST;
COMMIT WORK;

UPDATE TRIGGER_TEST SET XX = 'XX updated';
COMMIT WORK;

-- Show the records that were inserted into the trigger_test
-- table. (The records for trigger_output are shown later.)

SELECT * FROM TRIGGER_TEST;
COMMIT WORK;

DELETE FROM TRIGGER_TEST;
COMMIT WORK;

SELECT * FROM TRIGGER_TEST;

-- Show that the triggers did run and did add values to the
-- trigger_output table. You should see 6 records one for
-- each of the triggers that executed. The 6 triggers are:
--   BI, AI, BU, AU, BD, AD.

SELECT * FROM TRIGGER_OUTPUT;
COMMIT WORK;

-----
-- Error trigger test
-----

INSERT INTO TRIGGER_ERR_B_TEST(XX) VALUES ('XX');
COMMIT WORK;

```

3.4 序列

序列器对象是用于获取序号的对象。序列对象用来有效地获取序号。

语法为:

```
CREATE [DENSE] SEQUENCE sequence_name
```

根据序列创建方式的不同，序列可能包含也可能不包含间隔（该序列可能是稀疏序列或紧密序列）。紧密序列将确保各个序号之间不存在间隔。序号分配与当前事务绑定。如果事务回滚，那么序号分配也将回滚。紧密序列的缺点是，该序列在当前事务结束前不可供其他事务使用。

如果不需要紧密序列，那么可以使用稀疏序列。稀疏序列将确保所返回的值唯一，但它不与当前事务绑定。如果一个事务分配稀疏序号并接着回滚，那么该序号仅仅会丢失。

例如，您可以使用序列对象来生成主键编号。使用序列对象代替独立的表的优点是，序列对象已专门针对高速执行进行微调，并且所需的开销低于常规 UPDATE 语句。

紧密序列和稀疏序列都从 1 开始编号。

在使用 CREATE SEQUENCE 语句创建序列之后，您可以通过在 SQL 语句中使用下列构造来访问序列对象值：

- *sequence_name*.CURRVAL，返回序列的当前值
- *sequence_name*.NEXTVAL，将序列递增 1 并返回下一个值。

以下示例自动创建表的唯一标识：

```
INSERT INTO ORDERS (id, ...) VALUES (order_seq.NEXTVAL, ...);
```

可以使用 SET SEQUENCE 语句来动态地设置序列值：

```
SET SEQUENCE <sequence_name> VALUE <BIGINT_value>
```

存储过程中的序列

此外，还可以在存储过程中使用序列。

表 14. 存储过程中的序列

存储过程语句	用法
EXEC SEQUENCE <i>sequence_name</i> .CURRENT INTO <i>variable</i>	检索新序列值
EXEC SEQUENCE <i>sequence_name</i> .NEXT INTO <i>variable</i>	检索当前序列值
EXEC SEQUENCE <i>sequence_name</i> SET VALUE USING <i>variable</i>	设置序列值 提示：另外，您可以选择使用 SET SEQUENCE <name> VALUE <bigint value> 语句。

以下示例是检索新序号的存储过程：

```
"CREATE PROCEDURE get_my_seq  
RETURNS (val INTEGER)  
BEGIN  
EXEC SEQUENCE my_sequence.NEXT INTO (val);  
END";
```

3.5 事件

事件警报是指 solidDB 数据库中作为信号事件的数据库对象。和存储过程一起，这些事件可用于自动化管理。您可以令您的应用程序使用事件警报来代替轮询，后者使用的资源较多。

事件机制基于一个连接等待一个事件，直到另一连接发布该事件为止。不止一个连接可以等待同一事件。如果多个连接等待同一事件，当发出该事件时将通知所有等待的连接。一个连接还可以等待多个事件，当发生那些事件中的任何一个时将通知它。

共有两种类型的事件：

- 用户定义的事件

用户定义的事件只能在存储过程内使用。

用户定义的事件对象使用 CREATE EVENT 语句创建，使用 DROP EVENT 事件除去。

CREATE EVENT 定义事件名称及一组参数。

```
CREATE EVENT event_name [(parameter_name datatype)]
```

名称可以是用户指定的任何字母数字字符串。

事件参数必须是触发事件的存储过程中的本地变量或参数。

- 系统事件

系统事件可在存储过程内使用；此外，您还可以使用 ADMIN EVENT 命令在存储过程外等待系统事件。

在 第 373 页的附录 H，『系统事件』中描述系统事件名称和参数。

当应用程序调用等待发生特定事件的存储过程时，应用程序将处于锁定状态直到发出并接收到事件为止。在多线程的环境下，在事件等待期间，独立的线程和连接可以用来访问数据库。正在等待发出的事件的所有客户机将接收到该事件。

事件经常通过 POST EVENT 存储过程语句在存储过程中发布（发出）。存储过程中也能接收用户事件。

```
post_statement ::= POST EVENT event_name [( parameters)]
```

每个连接都有它自己的事件队列。在事件队列中，收集到的事件由 REGISTER EVENT 存储过程语句指定（注册），由 UNREGISTER EVENT 语句从该事件队列中将其除去。

```
wait_register-statement ::= REGISTER EVENT | UNREGISTER EVENT event_name
```

用 WAIT EVENT 存储过程语句使过程等待事件发生：

```
wait_event_statement ::=
  WAIT EVENT
  [event_specification...]
END WAIT
event_specification ::=
  WHEN event_name [(parameters)] BEGIN
  statements
END EVENT
```

您可以等待系统定义的事件或用户定义的事件。

提示：对于系统事件，您还可以使用 ADMIN EVENT 语句等待事件，而不必使用存储过程。然而，您不能使用 ADMIN EVENT 发出事件。

```
ADMIN EVENT 'register sys_event_hsbstateswitch';
ADMIN EVENT 'wait';
```

如果要想让存储过程停止等待事件，可以使用从客户机应用程序中的独立线程中调用的 ODBC 函数 SQLCancel()。此函数将取消执行语句。此外，您也可以创建并发送特定的用户事件。要等待这个附加的事件，必须修改等待存储过程。客户机应用程序将识别此事件并退出等待循环。

访问权和事件

事件的创建者或数据库管理员可以授予和撤销对该事件的访问权。可以将访问权授予用户和角色。如果用户对事件具有“SELECT”访问权，那么该用户有权等待该事件。如果用户对事件具有“INSERT”访问权，那么该用户可以发出该事件。

相关参考

第 144 页的 A.2, 『ADMIN EVENT』

第 156 页的 A.10, 『CREATE EVENT』

第 200 页的 A.29, 『DROP EVENT』

第 163 页的 A.14, 『CREATE PROCEDURE』

3.5.1 使用事件 - 例 1

该示例使用一对 SQL 脚本显示如何使用事件。

该示例包含两个脚本。脚本 1 等待事件，脚本 2 发出事件。一旦发出事件之后，正在等待的事件将完成等待并移至下一个命令。

要执行此示例代码，您需要两个控制台，这样，您才能启动 WaitOnEvent.sql 脚本，然后在 WaitOnEvent.sql 等待时运行 PostEvent.sql 脚本。

在该示例中，在发出事件之后，等待的存储过程实际上不会执行任何操作；该脚本仅仅完成等待并返回至调用者。然后，调用者可以继续执行它所要的任何操作，在本例中，将选择我们在等待时插入的记录。

此示例只等待一个事件，即“record_was_inserted”。稍后，在本章中，有另一个使用“WAIT”等待多个事件的脚本。

```
===== SCRIPT 1=====
-- SCRIPT NAME: WaitOnEvent.sql
-- PURPOSE:
-- This is one of a set of scripts that demonstrates posting events
-- and waiting on events. The sequence of steps is shown below:
--
-- THIS SCRIPT (WaitOnEvent.sql) PostEvent.sql script
-----
-- CREATE EVENT.
-- CREATE TABLE.
-- WAIT ON EVENT.
--   Insert a record into table.
--   Post event.
-- SELECT * FROM TABLE.
--
-- To perform these steps in the proper order, start running this
-- script FIRST, but remember that this script does not finish running
-- until after the post_event script runs and posts the event.
-- Therefore, you will need two open consoles so that you can leave
-- this running/waiting in one window while you run the other script
-- (post_event) in the other window.
-- Create a simple event that has no parameters.
-- Note that this event (like any event) does not have any
-- commands or data; the event is just a label that allows both the
-- posting process and the waiting process to identify which event has
-- been posted (more than one event may be registered at a time).
-- As part of our demonstration of events, this particular event
-- will be posted by the other user after he or she inserted a record.
CREATE EVENT record_was_inserted;
-- Create a table that the other script will insert into.
```

```

CREATE TABLE table1 (int_col INTEGER);
-- Create a procedure that will wait on an event
-- named "record_was_inserted".
-- The other script (PostEvent.sql) will post this event.
"CREATE PROCEDURE wait_for_event
BEGIN
-- If possible, avoid holding open a transaction. Note that in most
-- cases it's better to do the COMMIT WORK before the procedure,
-- not inside it. See "Waiting on Events" at the end of this example.
EXEC SQL COMMIT WORK;
-- Now wait for the event to be posted.
WAIT EVENT
  WHEN record_was_inserted BEGIN
  -- In this demo, we simply fall through and return from the
  -- procedure call, and then we continue on to the next
  -- statement after the procedure call.
  END EVENT
END WAIT;
END";
-- Call the procedure to wait. Note that this script will not
-- continue on to the next step (the SELECT) until after the
-- event is posted.
CALL wait_for_event();
COMMIT WORK;
-- Display the record inserted by the other script.
SELECT * FROM table1;

```

落实脚本 1 中的事务的指南 (WaitOnEvent.sql)

每当有可能的时候，在等待事件之前完成任何当前事务。如果您在事务中执行 WAIT，那么事务将保持打开状态，直到事件发生并执行下一个 COMMIT 或 ROLLBACK 为止。这意味着在等待期间，服务器将保持锁定，这可能导致过多的 Bonsai 树增大。有关 Bonsai 树以及防止其增大的详细信息，请参阅《solidDB 管理指南》中『通过落实事务减少 Bonsai 树大小』一节。

在此示例中，我们将 COMMIT WORK 放在 WAIT 之前的过程中。但是，这通常不是一个好的解决方案；将 COMMIT 或 ROLLBACK 放在“wait”过程中意味着，如果将过程作为另一事务的一部分进行调用，那么 COMMIT 或 ROLLBACK 将终止封装的事务并开始新的事务，这可能不是您所要的。例如，如果您正使用引用约束将数据输入“子”表，且正在等待引用数据输入“父”表，那么将事务分成两个事务会导致“子”表记录插入失败，因为“父”表尚未插入。

最佳策略是，将程序设计成事务中不需要 WAIT；而是，有可能的话，应该在事务之间调用“等待”过程。通过使用事件/进行等待，您可以对操作的执行顺序进行一定程度的控制，并且可以使用此功能来帮助确保符合依赖关系，而不必实际地将所有操作都放入单一事务。例如，在“异步”情况下，您可以等待插入子女和父母记录，如果数据库服务器不具有“事件”功能，那么您可以要求在同一事务中插入这两个记录，从而能够确保引用完整性。

通过使用事件/进行等待，您可以确保首先插入父母；然后，可以将子女记录的插入操作放入另一个事务，从而保证插入子女时，父母始终存在。（更正确地说，您几乎可以保证插入子女时父母存在。如果将插入操作分到两个不同的事务中，那么即使您确保在插入子女前插入父母，也有很小的机会导致在程序尝试插入子女记录前删除父母。）

```

===== SCRIPT 2=====
-- SCRIPT NAME: PostEvent.sql
-- PURPOSE:

```

```

-- This script is one of a set of scripts that demonstrates posting
-- events and waiting on events. The sequence of steps is shown below:
--
-- WaitOnEvent.sql THIS SCRIPT (PostEvent.sql)
-----
-- Create event.
-- Create table.
-- Wait on event.
--   INSERT A RECORD INTO TABLE.
--   POST THE EVENT.
-- Select * from table.
-- Insert a record into the table.
INSERT INTO table1 (int_col) VALUES (99);
COMMIT WORK;
-- Create a stored procedure to post the event.
"CREATE PROCEDURE post_event
BEGIN
  -- Post the event.
  POST EVENT record_was_inserted;
END";
-- Call the procedure that posts the event.
CALL post_event();
DROP PROCEDURE post_event;
COMMIT WORK;

```

3.5.2 使用事件 - 例 2

该示例说明如何编写等待多个事件的存储过程，该过程将在发出这些事件中的任何一个时完成等待。

该示例包含两个脚本。脚本 1 等待不止一个事件。脚本 2 是个存储过程。

```

===== SCRIPT 1=====
-- SCRIPT NAME: MultiWaitExamplePart1.sql
-- PURPOSE:
-- This code shows how to wait on more than one event.
-- If you run this demonstration, you will see that a "wait" lasts only
-- until one of the events is received. Thus a wait on multiple events
-- is like an "OR" (rather than an "AND"); you wait until event1 OR
-- event2 OR ... occurs.
--
-- This demo uses 2 scripts, one of which waits for an event(s) and one
-- of which posts an event.
-- To run this example, you will need 2 consoles.
-- 1) Run this script (MultiWaitExamplePart1.sql) in one window. After
-- this script reaches the point where it is waiting for the event, then
-- start Step 2.
-- 2) Run the script MultiWaitExamplePart2.sql in the other window.
-- This will post one of the events.
-- After the event is posted, the first script will finish.
-- Create the 3 different events on which we will wait.
CREATE EVENT event1;
CREATE EVENT event2(i INTEGER);
CREATE EVENT event3(i INTEGER, c CHAR(4));
-- When an event is received, the process that is waiting on the event
-- will insert a record into this table. That lets us see which events
-- were received.
CREATE TABLE event_records(event_name CHAR(10));
-- This procedure inserts a record into the event_records table.
-- This procedure is called when an event is received.
"CREATE PROCEDURE insert_a_record(event_name_param CHAR(10))
BEGIN
  EXEC SQL PREPARE insert_cursor
  INSERT INTO event_records (event_name) VALUES (?);
  EXEC SQL EXECUTE insert_cursor USING (event_name_param);
  EXEC SQL CLOSE insert_cursor;

```

```

EXEC SQL DROP insert_cursor;
END";
-- This procedure has a single "WAIT" command that has 3 subsections;
-- each subsection waits on a different event.
-- The "WAIT" is finished when ANY of the events occur, and so the
-- event_records table will hold only one of the following:
-- "event1",
-- "event2", or
-- "event3".
"CREATE PROCEDURE event_wait(i1 INTEGER)
RETURNS (eventresult CHAR(10))
BEGIN
  DECLARE i INTEGER;
  DECLARE c CHAR(4);
  -- The specific values of i and c are irrelevant in this example.
  i := i1;
  c := 'mark';
  -- Set eventresult to an empty string.
  eventresult := '';
  -- Will we exit after any of these 3 events are posted, or must
  -- we wait until all of them are posted? The answer is that
  -- we will exit after any one event is posted and received.
  WAIT EVENT
    -- When the event named "event1" is received...
  WHEN event1 BEGIN
    eventresult := 'event1';
    -- Insert a record into the event_records table showing that
    -- this event was posted and received.
    EXEC SQL PREPARE call_cursor
      CALL insert_a_record(?);
    EXEC SQL EXECUTE call_cursor USING (eventresult);
    EXEC SQL CLOSE call_cursor;
    EXEC SQL DROP call_cursor;
    RETURN;
  END EVENT
  WHEN event2(i) BEGIN
    eventresult := 'event2';
    EXEC SQL PREPARE call_cursor2
      CALL insert_a_record(?);
    EXEC SQL EXECUTE call_cursor2 USING (eventresult);
    EXEC SQL CLOSE call_cursor2;
    EXEC SQL DROP call_cursor2;
    RETURN;
  END EVENT
  WHEN event3(i, c) BEGIN
    eventresult := 'event3';
    EXEC SQL PREPARE call_cursor3
      CALL insert_a_record(?);
    EXEC SQL EXECUTE call_cursor3 USING (eventresult);
    EXEC SQL CLOSE call_cursor3;
    EXEC SQL DROP call_cursor3;
    RETURN;
  END EVENT
  END WAIT
END";
COMMIT WORK;
-- Call the procedure that waits until one of the events is posted.
CALL event_wait(1);
-- See which event was posted.
SELECT * FROM event_records;

===== SCRIPT 2 =====
-- SCRIPT NAME: MultiWaitExamplePart2.sql
-- PURPOSE:
-- This is script 2 of 2 scripts that show how to wait for multiple
-- events. See the instructions at the top of MultiWaitExamplePart1.sql.
-- Create a stored procedure to post an event.

```

```

"CREATE PROCEDURE post_event1
BEGIN
  -- Post the event.
  POST EVENT event1;
END";
--Create a stored procedure to post the event.
"CREATE PROCEDURE post_event2(param INTEGER)
BEGIN
  -- Post the event.
  POST EVENT event2(param);
END";
--Create a stored procedure to post the event.
"CREATE PROCEDURE post_event3(param INTEGER, s CHAR(4))
BEGIN
  -- Post the event.
  POST EVENT event3(param, s);
END";
COMMIT WORK;
-- Notice that to finish the "wait", only one event needs to be posted.
-- You may execute any one of the following 3 CALL commands to post an
-- event.
-- We've commented out 2 of them; you may change which one is de
-- commented.
CALL post_event1();
--CALL post_event2(2);
--CALL post_event3(3, 'mark');

```

3.5.3 使用事件 - 例 3

此示例展示 REGISTER EVENT 和 UNREGISTER EVENT 命令的简单用法。

示例 1 和示例 2 未使用 REGISTER EVENT，但是它们的 WAIT 命令也会成功。其原因在于，等待一个事件时，如果尚未以显式方式对该事件注册，那么将以隐式方式对该事件注册。因此，仅当您希望立即开始对事件进行排队，但在完成排队前不想开始等待这些事件时，才需要以显式方式注册事件。

```

CREATE EVENT e0;
CREATE EVENT e1 (param1 int);
COMMIT WORK;

-- Create a procedure to register the events to that when they occur
-- they are put in this connection's event queue.
"CREATE PROCEDURE eeregister
BEGIN
  REGISTER event e0;
  REGISTER EVENT e1;
END";

CALL eeregister;
COMMIT WORK;

-- Create a procedure to post the events.
"CREATE PROCEDURE eepost
BEGIN
  DECLARE x int;
  x := 1;
  POST EVENT e0;
  POST EVENT e1(x);
END";

COMMIT WORK;

-- Post the events. Even though we haven't yet waited on the events,
-- they will be stored in our queue because we registered for them.
CALL eepost;

```



```

COMMIT WORK;

-- Now create a procedure to wait for the events.
"CREATE PROCEDURE eewait
  RETURNS (whichEvent VARCHAR(100))
  BEGIN
    DECLARE i INT;

    WAIT EVENT
      WHEN e0 BEGIN
        whichEvent := 'event0';
      END EVENT

      WHEN e1(i) BEGIN
        whichEvent := 'event1';
      END EVENT

    END WAIT

  END";

COMMIT WORK;

-- Since we already registered for the 2 events and we already
-- posted the 2 events, when we call the eewait procedure twice
-- it should return immediately, rather than waiting.
CALL eewait;
CALL eewait;
COMMIT WORK;

-- Unregister for the events.
"CREATE PROCEDURE eeunregister
  BEGIN
    UNREGISTER event e0;
    UNREGISTER EVENT e1;
  END";

CALL eeunregister;
COMMIT WORK;
CREATE EVENT e0;
CREATE EVENT e1 (param1 int);
COMMIT WORK;

-- Create a procedure to register the events to that when they occur
-- they are put in this connection's event queue.
"CREATE PROCEDURE eeregister
  BEGIN
    REGISTER event e0;
    REGISTER EVENT e1;
  END";

CALL eeregister;
COMMIT WORK;

-- Create a procedure to post the events.
"CREATE PROCEDURE eepost
  BEGIN
    DECLARE x int;
    x := 1;
    POST EVENT e0;
    POST EVENT e1(x);
  END";

COMMIT WORK;

-- Post the events. Even though we haven't yet waited on the events,

```

```
-- they will be stored in our queue because we registered for them.  
CALL eepost;  
COMMIT WORK;
```

```
-- Now create a procedure to wait for the events.
```

```
"CREATE PROCEDURE eewait  
  RETURNS (whichEvent VARCHAR(100))  
  BEGIN  
    DECLARE i INT;  
  
    WAIT EVENT  
      WHEN e0 BEGIN  
        whichEvent := 'event0';  
      END EVENT  
  
      WHEN e1(i) BEGIN  
        whichEvent := 'event1';  
      END EVENT  
  
    END WAIT  
  
  END";  
COMMIT WORK;
```

```
-- Since we already registered for the 2 events and we already  
-- posted the 2 events, when we call the eewait procedure twice  
-- it should return immediately, rather than waiting.  
CALL eewait;  
CALL eewait;  
COMMIT WORK;
```

```
-- Unregister for the events.
```

```
"CREATE PROCEDURE eeunregister  
  BEGIN  
    UNREGISTER event e0;  
    UNREGISTER EVENT e1;  
  END";
```

```
CALL eeunregister;  
COMMIT WORK;
```

4 使用 solidDB SQL 进行数据库管理

您可以使用 solidDB SQL 语句来管理 solidDB 数据库及其用户和模式。本章描述使用 solidDB SQL 执行的管理任务。这些任务包括管理角色和特权、表、索引、事务、目录和模式。

4.1 使用 solidDB SQL 语法

solidDB SQL 语法基于 ANSI X3H2-1989 (SQL-89) 第 2 级标准，其中包括重要的 SQL-92 和 SQL-99 扩展。

要执行 SQL 语句，您可以使用 solidDB SQL Editor (**solsql**) 或 solidDB SQL Remote Control (**solcon**)，或不同的 ODBC 或 JDBC 兼容工具。此外，您可以从文件中执行 SQL 语句，例如，自动执行任务。以后，您也可以使用此类文件来重新运行 SQL 语句，或者将其用作用户、表或索引的文档。

切记：当使用 **solsql** 时，SQL 语句必须以 (;) 结尾。否则，以分号终止 SQL 语句将导致语法错误。

相关参考

第 133 页的附录 A，『语句』

本部分提供了 solidDB SQL 语句的描述及示例。

4.1.1 solidDB SQL 数据类型

solidDB SQL 支持“SQL-92 标准入门级”规范中指定的数据类型以及重要的中间级别增强功能。有关受支持的数据类型的完整描述，请参阅第 297 页的附录 C，『数据类型』。

您还可以定义一些具有可选长度、标度和精度参数的数据类型。在这种情况下，将不会使用相应数据类型的缺省属性。

4.1.2 solidDB ADMIN COMMANDs

solidDB SQL 包括了扩展 **ADMIN COMMAND 'command [command_args]'** 来执行基本的管理任务，例如备份、执行监视和关闭。

要了解可用的 ADMIN COMMAND 的简短描述，请执行 **ADMIN COMMAND 'help'**。

相关信息

第 133 页的 A.1，『ADMIN COMMAND』

4.1.3 标量函数

所有 solidDB 专有标量函数可按常规方式使用。例如：

```
SELECT substring(line, 1,4) FROM test;
```

另一方面，名称与保留字匹配的函数必须与转义字符配合使用。例如：

```
SELECT "left"(line,4) FROM test;
```

或者：

```
SELECT {fn left(line,4)} FROM test;
```

后者与独立于 ODBC 实现的语法相对应。它可以用于所有 API 和 GUI 接口。

4.2 管理用户特权和角色

您可以使用 solidDB 电传打字工具和许多 ODBC 兼容 SQL 工具来修改用户特权。可以使用 SQL 语句或命令来创建和删除用户及角色。包含多个 SQL 语句的文件称为 SQL 脚本。

在 `samples/sql` 目录中，您将找到 SQL 脚本 `sample.sql`，该脚本会给定创建用户和角色的示例。您可以使用 `sosql` 来运行此脚本。要创建您自己的用户和角色，您可以建立自己的脚本来描述用户环境。

4.2.1 用户特权

在多用户环境中使用 solidDB 数据库时，您可能想应用用户特权，以便对某些用户隐藏特定的表。例如，您可能不想让某个职员查看包含职员薪水的表，或者不想让其他用户更改您的测试表。

您可以应用 5 种不同类型的用户特权。用户可能能够查看、删除、插入、更新或引用表或视图中的信息。此外，还可以应用这些特权的任意组合。对表不具有任何这些特权的用户将完全无法使用该表。

注： 授予用户特权之后，这些特权将在被授予特权的用户登录到数据库时生效。授予特权时，如果该用户已登录到数据库，那么这些特权直到用户执行下列操作时才会生效：

- 第一次访问那些特权所应用于的表或对象，或者
- 断开连接，然后重新连接到数据库。

4.2.2 用户角色

也可以将特权授予称为“角色”的实体。

角色是一组可以作为一个单元授予用户的特权。您可以创建角色以及将用户指定到特定角色。可以对单一用户指定多个角色，也可以对单一角色指定多个用户。

注：

- 同一个字符串不能同时用作用户名和角色名。
- 一旦获得用户角色，获得该角色的用户登录到数据库时就会生效。授予角色时，如果该用户已登录到数据库，那么该角色直到用户断开连接并重新连接到数据库之后才会生效。

系统角色

solidDB 提供以下系统角色。系统角色名称是保存的用户名。

表 15. 系统角色

保留的名称	描述
PUBLIC	此角色将特权授予所有用户。将特定表的用户特权授予 <i>PUBLIC</i> 角色之后，所有当前用户以及将来用户都将对这个表拥有指定的用户特权。系统自动授予所有用户此角色。
SYS_ADMIN_ROLE	这是数据库管理员的缺省角色。此角色对所有表、索引和用户拥有特权，并且有权使用“solidDB 远程控制”。这也是数据库创建者角色。
_SYSTEM	这是所有系统表和视图的模式名。
SYS_CONSOLE_ROLE	此角色有权使用“solidDB 远程控制”，但不具有其他管理特权。
SYS_SYNC_ADMIN_ROLE	这是数据同步功能的管理员角色。
SYS_SYNC_REGISTER_ROLE	此角色仅用于向主数据库注册或注销副本数据库。

4.2.3 SQL 语句示例

以下是用于管理用户、角色和用户特权的 SQL 语句的一些示例。

创建用户

```
CREATE USER username IDENTIFIED BY password;
```

只有管理员才有权执行此语句。以下示例创建名为 CALVIN 并且密码为 HOBBS 的新用户。

```
CREATE USER CALVIN IDENTIFIED BY HOBBS;
```

删除用户

```
DROP USER username;
```

只有管理员才有权执行此语句。以下示例将删除名为 CALVIN 的用户。

```
DROP USER CALVIN;
```

更改密码

使用 ALTER USER 命令可更改您的密码。管理员也可以更改其他用户的密码。

过程

使用以下语法更改指定用户的密码:

```
ALTER USER username IDENTIFIED BY new_password
```

示例

以下语句将用户 CALVIN 的密码更改为 GUBBES。

```
ALTER USER CALVIN IDENTIFIED BY GUBBES
```

创建角色

```
CREATE ROLE rolename;
```

以下示例创建名为 GUEST_USERS 的新用户角色。

```
CREATE ROLE GUEST_USERS;
```

删除角色

```
DROP ROLE role_name;
```

以下示例将删除名为 GUEST_USERS 的用户角色。

```
DROP ROLE GUEST_USERS;
```

将特权授予用户或角色

```
GRANT user_privilege ON table_name TO username or role_name ;
```

用户对表可能拥有的特权是 SELECT、INSERT、DELETE、UPDATE、REFERENCES 和 ALL。ALL 将上面提到的全部 5 种特权授予用户或角色。未授权前新用户不具有任何特权。

以下示例将表 TEST_TABLE 的 INSERT 和 DELETE 特权授予 GUEST_USERS 角色。

```
GRANT INSERT, DELETE ON TEST_TABLE TO GUEST_USERS;
```

EXECUTE 特权使用户有权执行存储过程:

```
GRANT EXECUTE ON procedure_name TO username or role_name ;
```

以下示例将存储过程 SP_TEST 的 EXECUTE 特权授予用户 CALVIN。

```
GRANT EXECUTE ON SP_TEST TO CALVIN;
```

通过对用户指定角色将特权授予用户

```
GRANT role_name TO username ;
```

以下示例将您对 GUEST_USERS 角色定义的特权授予用户 CALVIN。

```
GRANT GUEST_USERS TO CALVIN;
```

撤销用户或角色的特权

```
REVOKE user_privilege ON table_name FROM username or role_name ;
```

以下示例撤销 GUEST_USERS 角色对表 TEST_TABLE 拥有的 INSERT 特权。

```
REVOKE INSERT ON TEST_TABLE FROM GUEST_USERS;
```

通过撤销用户的角色来撤销特权

```
REVOKE role_name FROM username ;
```

以下示例撤销 CALVIN 所拥有的 GUEST_USERS 角色特权。

```
REVOKE GUEST_USERS FROM CALVIN;
```


将管理员特权授予用户

```
GRANT SYS_ADMIN_ROLE TO username ;
```

以下示例将管理员特权授予 CALVIN，他现在对所有表拥有所有特权。

```
GRANT SYS_ADMIN_ROLE TO CALVIN;
```

您可能想授权用户执行数据同步操作。为此，请执行以下命令：

```
GRANT SYS_SYNC_ADMIN_ROLE TO HOBBS
```

注：

如果自动落实方式处于关闭状态，那么需要落实工作。要落实工作，请使用以下 SQL 语句：`COMMIT WORK`；如果自动落实方式处于打开状态，那么将自动落实事务。

更改管理员的用户名和密码

可以使用 `ALTER USER` 命令更改数据库系统管理员的密码。不能使用 `ALTER USER` 命令更改管理员用户名；可以将 `SYS_ADMIN_ROLE` 授予另一位用户，然后删除原始管理员用户帐户来更改用户名。

关于此任务

当创建 `solidDB` 数据库时，也会随之创建数据库系统管理员帐户。数据库创建者具备 `SYS_ADMIN_ROLE` 用户角色。

过程

- 更改管理员的密码

要更改管理员密码，请发出以下命令：

```
ALTER USER username identified by new_password
```

- 更改管理员的用户名

要更改管理员用户名：

1. 请使用以下命令创建新用户：

```
CREATE USER username IDENTIFIED BY password
```

2. 请使用以下命令授予新用户 `SYS_ADMIN_ROLE` 角色：

```
GRANT SYS_ADMIN_ROLE TO username
```

3. 请使用以下命令删除原始管理员帐户：

```
DROP USER username
```

相关参考

第 149 页的 A.6, 『ALTER USER』

第 197 页的 A.24, 『CREATE USER』

第 214 页的 A.50, 『GRANT』

第 209 页的 A.45, 『DROP USER』

4.3 管理表

solidDB 提供了一个动态数据字典, 后者允许您以联机方式创建、删除和更改表。您可以使用 SQL 命令来管理 solidDB 数据库表。

在 solidDB 目录中, 您可以找到一个名为 sample.sql 的 SQL 脚本, 该脚本提供了有关管理表的示例。您可以使用 solsql 来运行该脚本。

以下是用于管理表的 SQL 语句的一些示例。有关 solidDB SQL 语句的正式定义, 请参阅第 133 页的附录 A, 『语句』。

如果要查看数据库中所有表的名称, 请发出 SQL 语句 `SELECT * FROM TABLES`。(“TABLES”是系统定义的视图。) TABLE_NAME 列包含表名。

4.3.1 访问系统表

solidDB 系统表用于存储 solidDB 服务器信息, 其中包括用户信息。根据您的用户角色和访问权的不同, 您能够访问特定的系统表。例如, DBA 能够查看所有关于所有存储过程的信息, 其中包括过程定义文本(即, CREATE PROCEDURE 语句)。常规用户能够查看存储过程, 其中包括他们创建的过程的过程定义文本。常规用户如果对存储过程具有执行访问权, 但不是该存储过程的创建者, 那么能够查看一些关于该存储过程的信息, 但无法查看过程定义文本。要获取系统表的列表, 请参阅第 319 页的附录 E, 『数据库系统表和系统视图』。

下表提供了用户角色和用户访问权对特定系统表及其数据具有的查看访问权和/或对象授权特权。

注意, 此表中的“具有访问权的用户”是指具有下列任何一种权限的常规用户: INSERT、UPDATE、DELETE 或 SELECT 访问权。*

表 16. 查看表并授予访问权

任务	DBA	所有者	具有访问权的用户*	不具有访问权的用户
查看 SYS_TABLES	全部(无限制)	全部(无限制)	全部(无限制)	全部(无限制)
查看 SYS_TABLES 中的用户表	全部(无限制)	仅限于所有者的表	用户对其具有 INSERT、UPDATE、DELETE、SELECT 或 REFERENCES 访问权的所有表。	无法查看任何表。

表 16. 查看表并授予访问权 (续)

任务	DBA	所有者	具有访问权的用户*	不具有访问权的用户
查看 SYS_COLUMNS	全部 (无限制)	所有者的表中的列	用户对其具有 INSERT、UPDATE、DELETE、SELECT 或 REFERENCES 访问权的表中的列。	无法查看任何列。
查看 SYS_PROCEDURES (不包括过程定义文本 - 即, CREATE PROCEDURE 语句的文本)	全部 (无限制)	那些由用户 (所有者) 创建的过程。	用户在其中具有执行访问权的过程。	无法查看任何过程。
查看 SYS_PROCEDURES 中的过程定义文本	全部 (无限制)	那些由用户 (所有者) 创建的过程。	注意, 执行访问权不允许用户查看过程定义文本。	无法查看任何过程或过程定义文本。
能够授予对过程的访问权	是	是	否	否
查看 SYS_TRIGGERS	全部 (无限制)	那些由用户 (所有者) 创建的触发器。	无	无法查看任何触发器。
查看 SYS_TRIGGERS 中的触发器定义文本	全部 (无限制)	那些由用户 (所有者) 创建的触发器。	无	无法查看任何触发器。

4.3.2 SQL 语句示例

以下是用于管理表的 SQL 语句的一些示例。

创建表

```
CREATE TABLE table_name (column_name column_type
    [, column_name column_type]...);
```

所有用户都有权创建表。

以下示例创建名为 TEST 的新表, 该表包含列类型为 INTEGER 的列 I 以及列类型为 VARCHAR 的列 TEXT。

```
CREATE TABLE TEST (I INTEGER, TEXT VARCHAR);
```

除去表

```
DROP TABLE table_name;
```

只有特定表的创建者或者具有 SYS_ADMIN_ROLE 角色的用户才有权除去表。

以下示例将除去名为 TEST 的表。

```
DROP TABLE TEST;
```

注:

对于目录和模式：ANSI 的 SQL 标准定义了关键字 RESTRICT 和 CASCADE。当删除目录或模式时（如果您使用关键字 RESTRICT），如果该目录或模式含有其他数据库对象（例如表），则该目录或模式不可删除。使用关键字 CASCADE 允许您删除仍包含数据库对象的目录或模式 - 将自动删除该目录或模式所包含的数据库对象。如果您未指定 RESTRICT 或 CASCADE，那么缺省行为是 RESTRICT。

对于除目录和模式以外的数据库对象而言：在 solidDB SQL 中的大多数 DROP 语句不接受关键字 RESTRICT 和 CASCADE。并且，对于这些数据库对象，规则比简单的“纯 CASCADE”或“纯 RESTRICT”行为更复杂，但通常使用删除行为 RESTRICT 来删除对象。例如，如果您尝试删除 table1，但 table2 包含依赖于 table1 的外键，或者存在引用 table1 的发布，那么必须先删除从属表或发布，然后才能删除 table1。但是，服务器不会将 RESTRICT 行为用于所有可能的依赖关系类型。例如，如果视图或存储过程引用一个表，那么仍可以删除被引用表，该视图或存储过程在下次尝试引用该表时将失败。并且，如果一个表有相应的同步历史记录表，那么将自动删除该同步历史记录表。有关同步历史记录表的更多信息，请参阅 *solidDB Advanced Replication Guide*。

将列添加到表中

```
ALTER TABLE table_name ADD COLUMN column_name column_type;
```

只有特定表的创建者或者具有 SYS_ADMIN_ROLE 角色的用户才有权对表添加或删除列。

以下示例对表 TEST 添加列类型为 CHAR(1) 的列 C。

```
ALTER TABLE TEST ADD COLUMN C CHAR(1);
```

从表中删除列

```
ALTER TABLE table_name DROP COLUMN column_name;
```

无法删除作为唯一约束或主键组成部分的列。有关主键的详细信息，请参阅4.4，『管理索引』。

以下示例语句从表 TEST 中删除列 C。

```
ALTER TABLE TEST DROP COLUMN C;
```

注：

如果自动落实方式处于关闭状态，那么必须先落实工作，然后才能修改所更改的表中的数据。在更改表之后，要落实工作，请使用以下 SQL 语句：

```
COMMIT WORK;
```

如果自动落实方式处于打开状态，那么将自动落实所有语句，其中包括 DDL（数据定义语言）语句。

4.4 管理索引

索引用于提高表访问速度。数据库引擎使用索引来直接访问表中的行。如果没有索引，那么该引擎就必须在表中的所有内容中查找期望的行。您可以对单一表创建任意数目的索引；但是，添加索引确实会降低对该表执行的写操作（例如插入、删除和更新）的速度。有关创建索引以提高性能的细节信息，请参阅第 126 页的7.3，『使用索引来提高查询性能』。

索引分为两类：非唯一索引和唯一索引。唯一索引是所有键值都唯一的索引。要创建唯一索引，必须在创建索引时使用 UNIQUE 约束。

您可以使用 SQL 语句来创建和删除索引。

4.4.1 SQL 语句示例

以下是用于管理索引的 SQL 命令的一些示例。

对表创建索引

```
CREATE [UNIQUE] INDEX index_name ON base_table_name  
  column_identifier [ASC | DESC]  
  [, column_identifier [ASC | DESC]] ...
```

只有特定表的创建者或者具有 SYS_ADMIN_ROLE 角色的用户才有权创建或删除索引。

以下示例对表 TEST 的列 I 创建名为 X_TEST 的索引。

```
CREATE INDEX X_TEST ON TEST (I);
```

对表创建唯一索引

```
CREATE UNIQUE INDEX index_name ON table_name (column_name);
```

以下示例对表 TEST 的列 I 创建名为 UX_TEST 的唯一索引。

```
CREATE UNIQUE INDEX UX_TEST ON TEST (I);
```

删除索引

```
DROP INDEX index_name;
```

以下示例将删除名为 X_TEST 的索引。

```
DROP INDEX X_TEST;
```

注:

在创建或删除索引之后，必须先落实（或回滚）工作，然后才能在所创建或删除的索引所基于的表中修改数据。

4.4.2 主键索引

要从表中检索单一特定记录，我们必须能够唯一地标识该记录。solidDB 使用“主键”来唯一地标识每个表中的每个记录。主键是包含与唯一约束相同属性的列或列组合。每个表都具有主键 - 或者是显式的主键，或者是隐式的主键。

solidDB 将根据该主键的字段自动创建“主键索引”。与任何索引一样，主键索引能够提高对表中数据的访问速度。但是，与其他索引不同，主键索引还控制记录在数据库中的存储顺序。（这称为“集群”。）记录将根据主键值的升序顺序进行存储。

如果表的创建者未指定主键，那么 solidDB 将自动为表创建主键。为了确保该主键的唯一性，服务器将使用隐式的内部行标识。您可以使用符号性伪列名“ROWID”来检索该行标识的值并将其用于查询。

注:

在 solidDB 中，不可能在创建表之后添加显式的主键。如果用户未指定主键，那么最有效的查询方法将不可用于该表（除非使用 ROWID）。并且，不能在引用完整性约束中将这样的表用作被引用表。有鉴于此，强烈建议您始终在创建表时定义主键。

一旦定义主键（无论是由表创建者定义还是由服务器定义），服务器就不允许在该表中插入具有重复主键值的行。

无法删除主键索引。

4.4.3 辅键索引

由于索引能够提高搜索速度，因此对于搜索时常用的每个属性（或者属性组合），最好为表创建一个索引。除主索引以外的所有索引都称为“辅助索引”。

一个表可以包含任意数目的索引，但每个索引都必须具有唯一的列组合、列顺序以及值顺序（升序或降序）。例如，在以下所示代码中，第三个索引与第一个索引重复，这将生成错误消息或因信息重复而浪费磁盘空间。

```
CREATE INDEX i1 ON TABLE t1 (col1, col2);
-- The following is ok because although the columns are the same as in
-- index i1, the order of the columns is different.
CREATE INDEX i2 ON TABLE t1 (col2, col1);
-- The following is not ok because index i3 would be exactly the
-- same as index i1.
CREATE INDEX i3 ON TABLE t1 (col1, col2); -- ERROR.
-- The following is ok because although the columns and
-- column order are the same, the order of the index values
-- (ASCending vs. DESCending) is different.
CREATE INDEX i3b ON TABLE t1 (col1, col2) DESC;
```

注意，如果一个索引是另一个索引的“前导子集”（即，索引 2 中所有 N 个列的列、列顺序和值顺序都正好与索引 1 的前 N 个列相同），那么只需要创建作为超集的索引。例如，假定您已对 DEPARTMENT + OFFICE + EMP_NAME 这一组合创建索引。此索引不仅可用于按 department、office 和 emp_name 执行搜索，还可以仅按 department 或者仅按 department 和 office 执行搜索。因此，不需要单独对部门名创建另一个索引或者单独对部门和办公室创建另一个索引。对于 ORDER BY 操作而言，情况亦如此；如果 ORDER BY 条件与现有索引的子集匹配，那么服务器可以使用该索引。

注意，如果定义了主键或唯一约束，那么该键或约束将作为索引实现。因此，不需要创建作为主键“前导子集”或现有唯一约束“前导子集”的索引；此类索引都是多余的。

注意，使用辅助索引执行搜索时，如果服务器找到该索引键中请求的所有数据，那么服务器不需要查找该表中完整的行。（这仅适用于“读”操作，即 SELECT 语句。如果用户更新该表中的值，那么必须更新该表中的数据行以及索引中的值。）

4.4.4 避免索引重复

solidDB 包含用于避免索引重复的保护机制。有时候，在重新创建索引（DROP/CREATE）时，如果创建的其他索引将导致原始索引变为重复索引，那么重新创建索引操作将失败。

例如，您可以创建一个包括五个列（分别名为 A、B、C、D 和 E）的表，并在此表上创建下列索引：

- B
- AB

- BCE
- ABC

索引 B 用于搜索或过滤列 B；索引 BCE 以列 B 开头，因此，使用索引来定位列 B 的查询可以使用索引 BCE。对于索引 AB 和 ABC 而言，情况亦如此。因此，索引 B 和 AB 是重复的索引。

重复的索引具有负面作用，例如：

- 所需存储空间增加。
- 更新性能下降。
- 备份时间延长。

如果您尝试创建重复索引，但创建索引失败，则 solidDB 将发出以下错误：

SOLID 表错误 13199: 索引定义重复

有关更多信息，请参阅《IBM solidDB 管理员指南》中的附录『错误代码』。

4.5 引用完整性

引用完整性概念用于确保数据库表之间的关系保持一致。在其之间必须存在完整性的表叫被引用表和引用表。

使用外键执行引用完整性。外键是引用表上匹配被引用表上主键列（或其他相似的唯一列）的字段。可用外键表示类型为 1 到 n 的概念关系，例如，“一个职员属于一个部门”。例如，如果引用表有到被引用表的外键，则引用完整性确保您不能在该引用表中添加记录，除非在被引用表中有相关的记录。

外键通过引用约束定义进行维护。这些约束指定了违反约束时，solidDB 必须要采取的引用操作。例如，从被引用表中删除具有引用的主键的行时，将发生违反约束的情况。

4.5.1 主键与候选键

为了加强引用完整性，被引用表必须要含有一个主键（优先）或候选键。

主键是包含与唯一约束相同属性的列或列组合。由于主键用于识别表中的行，所以主键必须要唯一，且必须要有 NOT NULL 属性。一个表虽然只有一个主键，但是可以有多个唯一键。主键是可选的，且可在创建表或替换表时定义。

使用 CREATE TABLE 语句中的主键约束语法来定义主键。例如：

```
CREATE TABLE customers (  
  cust_id INTEGER PRIMARY KEY,  
  name CHAR(24),  
  city CHAR(40));
```

此外，您还可以对一个列或一组列定义唯一索引，并对这些列实施 NOT NULL 约束。实际上，这将生成一个候选键。首选的方法是使用显式的主键，因为这有助于提高派生连接时的性能。

4.5.2 外键

外键是表中的一列（或一组列），它引用（或“关联”）一个被引用表中的唯一值。外键列中的每一个值都必须是在被引用表中有一个匹配值。

要确保引用表中的每个记录都正好引用被引用表中的一个记录，被引用表中的被引用列必须具有主键约束或者同时具有唯一约束和非空约束。只有唯一索引并不足够。

示例 1:

例如，以银行为背景，一个表可能包含客户信息 (Customers)，另一个表可能包含帐户信息 (Accounts)。每个帐户都与特定客户相关，每位客户都使用唯一标识 (CUST_ID) 进行身份识别。有些客户可能有不止一个帐户。该 CUST_ID 可作为 Customers 表的主键。这些 Accounts 表也包含 CUST_ID 信息以识别拥有特定帐户的客户，这使得根据帐户信息来查找客户信息成为可能。Accounts 表中 CUST_ID 的副本是外键；它引用 Customers 表的主键中与之相匹配的值。

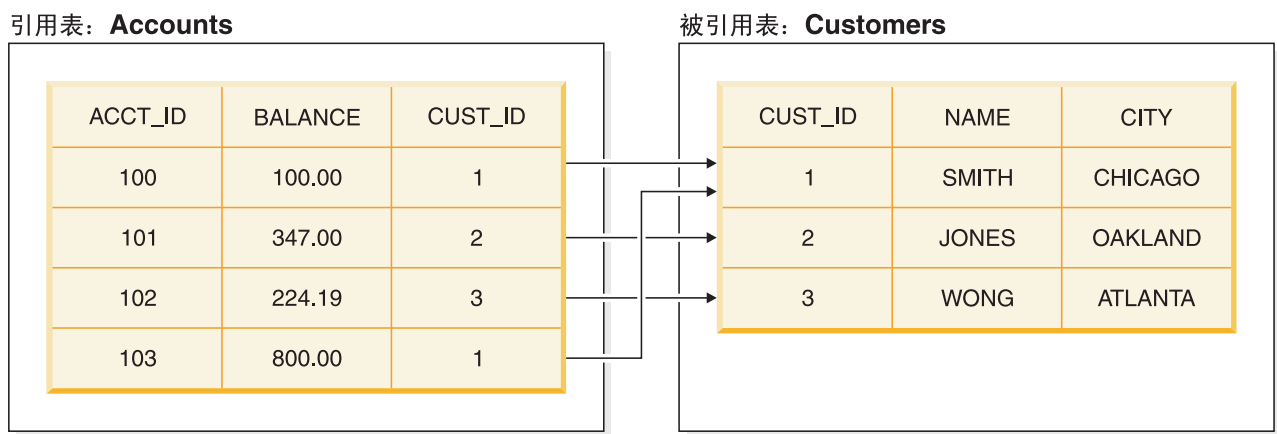


图 2. 示例: 带有引用约束的表

在上例中，可用以下语句创建引用表 Accounts:

```
CREATE TABLE accounts (  
    acct_id INTEGER PRIMARY KEY,  
    balance FLOAT,  
    cust_id INTEGER REFERENCES customers);
```

在 REFERENCES 子句中，仅指定了被引用表，而未指定被引用列。缺省情况下，假定被引用列是主键。这是帮助避免指定被引用列时发生错误的首选方式。

上例中，主键和外键使用同一个列。但是，主键和外键可由多个列组成。由于每个外键值都必须正好与相应的主键值匹配，因此外键所含的列数目和列数据类型必须和主键相同，且外键和主键的列顺序也必须相同。

外键还可有多个与主键不同的列名。外键和主键也可以有不同的缺省值。但是，由于被引用表中的值必须唯一，因此缺省值的使用场合并不多，并且很少用于主键所包含的列。缺省值也不常用于外键列。

虽然主键值必须唯一，但外键值不必唯一。例如，银行的单一客户可以有多个帐户。出现在 Customers 表中主键列的帐户标识 (ACCT_ID) 必须唯一；然而，相同的

CUST_ID 可能在 ACCOUNTS 表的外键列中多次出现。如上图所示，客户 SMITH 有不止一个帐户，因此她的 CUST_ID 在 ACCOUNTS 表的外键列中出现多次。

示例 2:

在有些情况中，一个表中的外键可以引用同一表中的主键 - 在此情况下，同一表既是被引用表也是引用表。例如，在职员表中，每个职员的记录都有可能含有一个包括该职员经理标识 (MGR_ID) 的字段。经理本身可以存储在同一个表中。因此，该表的 MGR_ID 可以是引用该表职员标识 (EMP_ID) 的外键。下图解释了该示例。

自引用表

EMP_ID	MGR_ID	EMP_NAME
1	NULL	ANNAN
10	1	WONG
20	1	SMITH
147	10	JONES
162	20	RAMA

图 3. 自引用约束

在此示例中，Rama 的经理是 Smith (Rama 的 MGR_ID 是 20，而 Smith 的 EMP_ID 是 20)。Smith 向 Annan 汇报工作 (Smith 的 MGR_ID 是 1，而 Annan 的 EMP_ID 是 1。) Jones 的经理是 Wong，而 Wong 的经理是 Annan。如果 Annan 是该公司总裁，她手下没有经理，则外键 (MGR_ID) 的值为 NULL。

如果主键包含多个列，那么应该在各个列之后定义主键。例如：

```
CREATE TABLE DEPT (  
    DIVNO INTEGER,  
    DEPTNO INTEGER,  
    DNAME VARCHAR,  
    PRIMARY KEY (DIVNO, DEPTNO));
```

类似的语法也可以用于外键。然而，往往需要使用还包括约束名的 CONSTRAINT 语法来定义外键。如果您定义了一个约束名，您可以在表创建后使用 ALTER TABLE 语句动态除去该约束。

创建含有约束名的表的示例 (emp_fk1):

```
CREATE TABLE EMP (  
    EMPNO INTEGER PRIMARY KEY,  
    DIVNO INTEGER,
```

```
DEPTNO INTEGER,  
ENAME VARCHAR,  
CONSTRAINT emp_fk1 FOREIGN KEY (DIVNO, DEPTNO) REFERENCES DEPT);
```

注： 与其他完整性约束类似，通过使用 ALTER TABLE 语句，您可以动态地对引用完整性（外键）进行命名和处理（删除或添加）。有关更多信息，请参阅第 97 页的 4.5.4, 『动态约束管理』。

定义外键时，将始终对外键列创建索引。每当更新或删除引用的记录时，服务器都将检查是否不存在丢失引用的引用记录。外键索引提高外键检查操作的性能。

相关参考

第 188 页的 A.22, 『CREATE TABLE』

第 145 页的 A.4, 『ALTER TABLE』

4.5.3 引用动作

当违反引用约束时，solidDB 会采取措施来维护引用完整性。例如以下几种情况违反了引用约束：

- 含有无效外键值的行插入到了引用表中。
- 引用表中的外键更新后为无效的值。
- 含有引用的主键的行从被引用表中删除。
- 更新了被引用表中引用的主键。

引用操作可定义为 CREATE TABLE 或 AFTER TABLE 语句中 table_constraint_definition 的一部分。当违反约束时，以下操作（选项）可用：

NO ACTION

此选项将限制操作或者回滚违反引用完整性约束的操作。NO ACTION 选项允许更改临时违反引用约束的表；如果您要求表状态永远不违反任何约束（甚至暂时违反任何约束），请使用 RESTRICT 选项。

CASCADE

在被引用表上执行操作的事例中，该选项会将引用表上的操作级联到引用表上。这包括删除所有引用行（级联删除）以及更新所有引用外键值（级联更新）。

在 CASCADE 引用操作中，不允许循环。尝试创建由具有级联动作的外键组成的循环，将导致错误。

对于任何两个表，最多可以在它们之间定义一条级联更新路径。此限制不适用于级联删除。

SET DEFAULT

在被引用表上执行操作的事例中，该选项会将引用列设置为预定义的缺省值。

SET NULL

在被引用表上执行操作的事例中，该选项会将引用列设置为 NULL。

RESTRICT

该选项限制所有违反约束的操作。

引用完整性动作有时允许对表进行临时违反引用约束的更改。NO ACTION 选项允许此类违例。

如果未指定任何选项，那么将采用缺省值 NO ACTION。

相关参考

第 188 页的A.22, 『CREATE TABLE』

第 145 页的A.4, 『ALTER TABLE』

4.5.4 动态约束管理

可以使用 ALTER TABLE 子句来动态地管理约束。可以使用的次子句是:

- ADD CONSTRAINT。此子句对表添加命名约束。
- DROP CONSTRAINT。此子句除去表的命名约束。

注:

在 solidDB 中，如果使用了关键字 CONSTRAINT，那么约束名是必需的。

- CHECK。此约束允许您对表或表列指定规则。每个规则都是一个条件，此条件对于它所应用于的表中的任何一行都不能为 false。否则，将无法更新该表。

规则是布尔表达式。例如，规则可以检查值的范围以及是否相等，此外，规则也可以是简单的比较。您可以在一个语句中执行多项检查。可用的表达式和运算符如下所示:

表 17. 表达式和运算符

表达式	说明
<	小于
>	大于
=	等于
<=	小于或等于
>=	大于或等于
<>	不等于
AND	逻辑与
ANY	在随后的列表或者指定的表中
BETWEEN	介于
IN	在随后的列表或者指定的表中
MAX	最大值
MIN	最小值
NOT	逻辑非
OR	逻辑或
XOR	逻辑异或

- **UNIQUE**。UNIQUE 约束要求表中任意两行的给定列或一组列都不能包含相同的值。您可以在表级或列级创建唯一约束。注：主键包含唯一约束。
- **FOREIGN KEY**。FOREIGN KEY 约束要求外键列中的每个值在所引用的表中都必须有匹配的值。

注：

对于未命名的约束，solidDB 将自动生成名称。如果要查看名称，那么请使用 `solidd -x hidddenames` 命令。

有关约束语法信息和示例，请参阅第 133 页的附录 A，『语句』中的 CREATE TABLE 和 ALTER TABLE 这两节。

4.6 管理数据库对象

4.6.1 简介

solidDB 允许您使用目录和模式对数据进行分组。solidDB 对模式的使用符合 SQL 标准，而 solidDB 对目录的使用是对 SQL 标准的扩展。

目录和模式允许您以分层方式对数据库对象（表、序列等）进行分组。这使您能够将相关的项放入同一个组。例如，可以将所有与记帐系统相关的表放入一个组（例如目录），并将所有与人力资源系统相关的表放入另一个组。此外，还可以按用户对数据库对象进行分组。例如，可以将 Jane Smith 使用的所有表放入单一模式。

目录是层次结构的最高（最宽）层。模式名是中间层。特定数据库对象（例如表）是层次结构的最低（最窄）层。因此，单一目录可以包含多个模式，并且每个模式都可以包含多个表。

对象名在一个组中必须唯一，但不必在各个组之间唯一。例如，Jane Smith 的模式和 Robin Trower 的模式都可以包含名为“bills”的表。这两个表相互无关。尽管它们同名，但它们可以包含不同的结构和不同的数据。同样，目录“accounting_catalog”和目录“human_resources_catalog”都可以包含名为“david_jones”的模式。那些模式尽管同名，但相互无关。

如果要指定特定的表，并且该表名在数据库中不唯一，那么可以通过指定目录、模式和表名来标识该表，例如：

```
accounting_catalog.david_jones.bills
```

后面的内容将更详细地讨论语法。

如果未指定完整的名称（即是，省略了模式或者省略了模式和目录），服务器将使用当前/缺省目录和模式名来确定要使用的表。

通常，您可以将目录想像成逻辑数据库。模式通常与用户相对应。后面的内容将对此进行更详细的讨论。

4.6.2 目录

一个物理数据库文件可以包含不止一个逻辑数据库。每个逻辑数据库都是一组完整而独立的数据库对象，例如表、索引、过程和触发器等。每个逻辑数据库都是一个目录。注意，solidDB 目录不仅仅限于索引（在传统的库卡目录概念中，索引用于定位未包含全部项内容的项）。

目录使您能够以逻辑方式对数据库进行分区，以便实现下列目标：

- 对数据进行组织，以便满足业务、用户和应用程序的需求。
- 通过使用逻辑数据库来指定多个主数据库或副本数据库，以便在一个物理数据库服务器中进行同步。有关在多主数据库环境中实现同步的更多详细信息，请参阅 *IBM solidDB Advanced Replication User Guide* 中的“ulti-master synchronization model”。

4.6.3 模式

一个目录可以包含一个或多个模式。模式是持久数据库对象，用于提供部分或整个数据库的定义。它代表一组与特定模式名相关联的数据库对象。这些对象包括表、视图、索引、存储过程、触发器和序列。模式使您能够为每个用户提供同一个逻辑数据库（即，单一目录）中的数据库对象（例如表）供该用户独自使用。如果未对数据库对象指定模式，那么缺省模式是创建对象的用户的用户标识。

4.6.4 唯一地标识目录和模式中的对象

模式使两个不同用户能够在同一个物理数据库甚至同一个逻辑数据库中创建同名的表。例如，假定单一物理数据库包含两个不同的目录 `employee_catalog` 和 `inventory_catalog`。并且，假定每个目录都包含两个不同的模式 `smith` 和 `jones`，并且同一个 `Smith` 同时拥有这两个“smith”模式，同一个 `Jones` 同时拥有这两个“jones”模式。如果 `Smith` 和 `Jones` 在他们的每个模式中创建名为 `books` 的表，那么共有 4 个名为“books”的表，并且您可以通过下列方法来访问这些表：

```
employee_catalog.smith.books  
employee_catalog.jones.books  
inventory_catalog.smith.books  
inventory_catalog.jones.books
```

正如您所见，目录名和模式名可用于“限定”（唯一地标识）数据库对象（例如表）的名称。在所有 DML 语句中，都可以使用以下语法来限定对象名：

```
catalog_name.schema_name.database_object
```

或者

```
catalog_name.user_id.database_object
```

例如：

```
SELECT cust_name FROM accounting_dept.smith.overdue_bills;
```

您可以使用模式名来限定一个或多个数据库对象，无论是否指定了目录名均如此。语法为：

```
schema_name.database_object_name
```

或者

```
user_id.database_object_name
```

例如:

```
SELECT SUM(sales_tax) FROM jones.invoices;
```

要将模式名与数据库对象配合使用, 必须已创建该模式。

缺省情况下, 如果创建数据库对象时未指定模式名, 那么将使用该数据库对象的创建者的用户标识对其进行限定。例如:

```
user_id.table_name
```

要设置目录和模式上下文, 请使用 SET CATALOG 或 SET SCHEMA 语句。

如果未使用 SET CATALOG 来设置目录上下文, 那么将通过使用缺省目录名来解析所有数据库对象名。

注: 在创建新数据库或者将旧数据库转换为新格式时, 将提示用户对数据库系统目录指定缺省目录名。用户可以访问缺省目录名, 而不必知道这个指定的缺省目录名。例如, 用户可以指定以下语法来访问系统目录:

```
""._SYSTEM.table
```

solidDB 将把作为目录名指定的空字符串 ("") 转换为缺省目录名。solidDB 还能够将 _SYSTEM 模式自动解析为系统目录, 即使用户未提供目录名亦如此。

下列 SQL 语句提供了有关创建目录和模式的示例。有关 solidDB SQL 语句的正式定义, 请参阅第 133 页的附录 A, 『语句』。

4.6.5 SQL 语句示例

以下是用于管理数据库对象的 SQL 语句的一些示例。

创建目录

```
CREATE CATALOG catalog_name
```

只有数据库的创建者或者具有 SYS_ADMIN_ROLE 角色的用户才有权创建或删除目录。

以下示例创建名为 C 的目录并假定用户标识为 SMITH:

```
CREATE CATALOG C;  
SET CATALOG C;  
CREATE TABLE T (i INTEGER);  
SELECT * FROM T;  
--The name T is resolved to C.SMITH.T
```

设置目录和模式上下文

以下示例将目录上下文设置为 C, 并将模式上下文设置为 S。

```
SET CATALOG C;  
SET SCHEMA S;  
CREATE TABLE T (i INTEGER);  
SELECT * FROM T;  
-- The name T is resolved to C.S.T
```

删除目录

```
DROP CATALOG catalog_name
```

以下示例将删除名为 C 的目录。


```
DROP CATALOG C;
```

创建模式

```
CREATE SCHEMA schema_name
```

任何数据库用户都可以创建模式；但是，用户必须有权创建与模式相关的对象（例如，CREATE PROCEDURE 和 CREATE TABLE 等）。

注意，创建模式并不表示使新模式成为当前/缺省模式。如果要使新模式成为当前模式，那么必须显式地使用 SET SCHEMA 语句来设置该模式。

以下示例创建名为 FINANCE 的模式并假定用户标识为 SMITH:

```
CREATE SCHEMA FINANCE;
CREATE TABLE EMPLOYEE (EMP_ID INTEGER);
-- "注意": 雇员表适用于 SMITH.EMPLOYEE, 不
-- 适用于 FINANCE.EMPLOYEE。创建一个不清晰的模式, 使得
-- 新模式为当前/缺省模式。SET SCHEMA FINANCE;
CREATE TABLE EMPLOYEE (ID INTEGER);
SELECT ID FROM EMPLOYEE;
-- In this case, the table is qualified to FINANCE.EMPLOYEE
```

删除模式

```
DROP SCHEMA schema_name
```

以下示例将删除名为 FINANCE 的模式。

```
DROP SCHEMA FINANCE;
```

5 管理事务

本节将说明如何管理事务、如何处理并行控制和锁定以及如何选择耐久性级别。

5.1 定义只读事务和读/写事务

要将一个事务定义为只读事务或可读可写事务，请使用下列 SQL 命令：

```
SET TRANSACTION { READ ONLY | READ WRITE }
```

可以将下列选项与此命令配合使用。

- READ ONLY

使用此选项来定义只读事务。

- READ WRITE

使用此选项来定义可读可写事务。这是缺省选项。

注：要检测事务之间的冲突，请使用标准的 ANSI SQL 命令 SET TRANSACTION ISOLATION LEVEL 对事务定义“可重复读”或“可序列化”隔离级别。有关详细信息，请参阅《IBM solidDB 管理员指南》中的选择事务隔离级别一节。

除非使用自动落实功能，否则事务必须以 COMMIT WORK 或 ROLLBACK WORK 命令结束。

5.2 并行控制和锁定

并行控制的用途是防止两个不同的用户（或者同一用户的两个不同链接）尝试更新同一数据。并行控制也可以防止一个用户在另一用户更新相同数据时查看过时数据。

以下示例解释了需要并行控制的原因。在两个示例中，假设您的支票帐户的余额为 \$1,000。那天您存入 \$300 并花费 \$200。结算时，您的帐户余额应为 \$1,100。

- **示例 1：无并行控制**

1. 上午 11:00 点，出纳员 #1 查询了您的帐户，并看到您的余额为 \$1,000。该出纳员从中扣除 \$200 的支票款项，但不能立即保存已更新的帐户余额 (\$800)。
2. 上午 11:01 点，出纳员 #2 查询了您的帐户，看到您的帐户中的余额仍然为 \$1,000。出纳员 #2 加入了 \$300 存款，并将新的帐户余额保存为 \$1,300。
3. 上午 11:09 点，银行出纳员 #1 返回该终端，完成输入并保存了更新值（经计算为 \$800）。值 \$800 将覆盖 \$1,300。
4. 当日结算时，帐户余额将是 \$800 而不是正确的值 \$1,100（\$1000 + 300 - 200）。

- **示例 2：并行控制**

1. 当出纳员 #1 开始在您的帐户上操作时，会锁定该帐户。
2. 当出纳员 #1 在更新您的帐户时，如果出纳员 #2 要读取或更新您的帐户，出纳员 #2 将无法访问该帐户并会接收到错误消息。
3. 在出纳员 #1 完成更新后，出纳员 #2 可以继续操作。

4. 当日结算时，帐户余额为 \$1,100 (\$1,000 - \$200 + \$300)。

在示例 1 中，帐户更新是同时进行的，而非按顺序执行，其中一次更新会覆盖另一次更新。在示例 2 中，为了防止两个用户同时更新数据（可能会导致互相覆盖对方的更新），系统使用并行控制机制。

solidDB 提供两种不同的并行控制机制，*悲观并行控制*和*乐观并行控制*。

悲观并行控制机制基于锁定。锁定是用于限制其他用户访问数据的机制。一个用户锁定记录后，该锁定将禁止其他用户更改该记录（在某些情况下，还将禁止进行读取）。乐观并行控制机制虽然不挂起锁定，但可以防止使用时间戳覆盖数据。

5.2.1 PESSIMISTIC 并行控制与 OPTIMISTIC 并行控制

solidDB 提供两种不同的并行控制机制，*悲观方式*和*乐观方式*。

- **悲观并行控制**（或**悲观锁定**）之所以称之为“悲观”是因为系统假定最坏情况 - 系统假定两个或两个以上的用户将要同时更新同一个记录，然后通过锁定该记录来消除这种可能性，而不考虑实际发生冲突的可能性有多低。

当用户访问任何行时，程序会立即挂起该锁定，使得不可能有两个或以上的用户同时更新该行。根据锁定方式不同（*共享式*、*互斥式*或*更新*），其他用户即使在存在锁定的情况下也可能读取到数据。有关锁定方式的更多详细信息，请参阅第 107 页的『*锁定方式：共享、互斥和更新*』。

- **乐观并行控制**（或**乐观锁定**）假定尽管有可能发生冲突，但发生的几率很小。此方法并不是在每次使用每个记录时锁定该记录，而是，系统仅仅检查有关两个用户实际尝试同时更新同一个记录的指示。如果检测到这种现象，那么将废弃其中一个用户的更新并通知该用户。

例如，如果 User1 更新了记录，而 User2 只想读取该记录，则 User2 将能够读取磁盘中的数据并执行操作，而不检查该数据的锁定状态。如果 User1 已读取该数据并进行更新，但尚未落实事务，那么 User2 可能会看到略微过期的信息。

乐观锁定仅在基于磁盘的表（D 表）上有效。

solidDB 使用多版本执行乐观并行控制。

1. 每次当服务器读取记录以便尝试对其更新时，服务器都将创建该记录的版本号副本并存储该副本以供将来参考。
2. 当准备落实该事务时，服务器将它读取的原始版本号和当前已落实的数据的版本号进行比较。
 - 如果两个版本号相同，则表明没有其他用户更改该记录，因此系统可以写更新后的值。
 - 如果磁盘上的原始读取值和当前读取值不同，则表明其他用户自从读取数据后已更改了该数据，当前的操作可能已经过期。因此，系统废弃该数据的版本，终止事务，并返回一个错误消息。
 - 检查版本号的步骤称为**验证**。验证可在落实期间执行（正常验证）或在写每个语句的时候执行（提早验证）。在 solidDB 中，提早验证是缺省方法 (**General.TransactionEarlyValidate=yes**)。

每次更新记录时，同时更新版本号。

solidDB 可以临时存储每个数据行的多个版本，而不是给每个用户提供读取数据时磁盘数据的版本。每个用户的事务都将看到数据库在该事务启动时所处的状态。这样，每个用户看到的数据都在该事务的执行期间保持一致，并且各个用户能够同时访问数据库。有关多版本的更多详细信息，请参阅《IBM solidDB 管理员指南》中的 *solidDB Bonsai Tree* 多版本和并行控制。

注：尽管有时人们将乐观并行控制机制称为乐观锁定，但它并非真实的锁定模式 - 当使用乐观并行控制时，系统不挂任何锁定。使用术语“锁定”是因为乐观并行控制防止更新重叠，与悲观锁定的目的相同。

当您使用乐观锁定时，直到写更新后的数据时您才会检测到冲突。使用悲观锁定方法时，您在尝试读取数据时就会立即检测到冲突。

以银行为例，悲观锁定就像是在银行门口设置一名警卫，他在您尝试进入银行时检查您的帐号；如果某人（您的配偶或者持有您签署的支票的商人）正在银行内访问您的帐户，那么您直到该人员办理业务完成并离开后才能进入银行。另一方面，乐观锁定允许您随时进入银行并尝试办理业务，但存在以下风险：当您准备离开银行时，门口的警卫通知您，您的业务与别人的业务有冲突，您必须再次办理该业务。

使用悲观锁定方式时，第一个请求执行锁定的用户将获取锁定。一旦您获取锁定，其他用户或连接就无法覆盖您的锁定。在 solidDB 中，该锁定将保持到事务结束为止，或者，对于长期表锁定而言，该锁定将保持到您显式地将其释放为止。

缺省并行控制机制

缺省并行控制机制取决于表的类型：

- 基于磁盘的表（D 表）使用缺省乐观模式。
- 主存储器表（M 表）始终是悲观表。

您可以覆盖乐观锁定方法并改为指定悲观锁定方法。您可以对各个表执行此任务。在一个表遵循乐观锁定规则的同时，另一个表可以遵循悲观锁定规则。两表可用于同一事务甚至同一语句；solidDB 在内部处理这些操作。

锁定和性能

乐观锁定能够实现快速性能和高度并行（多个用户同时进行访问），只需偶尔拒绝写初始时可接受但在最后却与另一个用户的更改有冲突的数据即可。

无论是否确定有两个或以上的用户尝试访问同一记录，悲观锁定需要每次操作都有额外开销。此开销并不大，但由于更新每一行时都要执行锁定，因此会不断累积。此外，每当用户尝试访问某一行时，系统都还必须检查是否已有另一个用户或连接锁定了所要请求的行。

例如，如果两个银行出纳员同时访问一个记录且出纳员 #1 获得了锁定，则无论出纳员 #2 要和出纳员 #1 一样对相同记录同时执行操作的概率多么低，出纳员 #2 都需要检查该锁定。检查每项已用记录会耗费时间。此外，很重要的一点是，在该检查期间，不能有任何其他出纳员尝试与出纳员 #2 运行同一检查（否则，他们可能都会看到在 10:59:59 时尚未使用记录 X，然后可能都尝试在 11:00:00 锁定该记录）。因此，检查锁定这一操作本身甚至也要求执行另一次锁定，以防止两个用户同时更改锁定。

选择并行控制机制

在大多数情况下，乐观并行控制的效率更高且提供的性能更好。在悲观锁定和乐观锁定间做选择时，请考虑以下几点：

- 在需要执行大量的更新操作，且多个用户极有可能尝试在同时更改数据的情况下，悲观锁定方法很有用。

例如，如果每项操作可以同时更新大量记录（银行可能在每个月结束时对每个帐户添加利息收入），且两个应用程序都同时在执行同一操作，则它们之间将发生冲突。

悲观并行控制也更适合包含频繁更新小型表的应用程序。在此类所谓的热点中，发生冲突的可能性很高，从而导致乐观并行控制耗费大量工时来回滚有冲突的事务。

如果您正将一个应用程序迁移至另一个使用悲观锁定的 DBMS 中，您也应当将悲观模式与 solidDB 一同使用。在 solidDB 中使用悲观模式意味着您不必更改该应用程序。

- 在冲突概率很低的情况下 - 记录多用户少，或更新少、大部分都是只读类型的操作 - 乐观锁定方法很有用。

5.2.2 锁定和锁定方式

锁定是一种用于防止两个或以上的用户同时执行有冲突操作的机制。如果至少其中一个操作涉及通过 UPDATE、DELETE、INSERT 和 ALTER TABLE 等语句更新数据，那么操作将发生冲突。如果所有操作都是只读操作（例如 SELECT），那么不会发生冲突。

solidDB 不允许用户显式地指定行级锁定。没有 LOCK RECORD 命令；服务器将自动执行所有行级锁定。服务器还将自动执行表级锁定。如果需要显式地设置表级锁定，那么可以使用 LOCK TABLE 命令来完成此任务。

表级锁定与行级锁定

solidDB 允许使用表级锁定和行级锁定。

行级锁定

行级锁定挂在事务中的语句定义的单独记录（行）上。一旦用户访问任何的行，系统马上就会挂起锁定。

行级锁定始终都是隐式的，solidDB 在必要时设置锁定。您不能手动锁定或解锁行级锁定。

表级锁定

可将表级锁定想象为元数据锁定；这些锁定防止并发用户同时更改数据模式（DDL 操作）或在表中记录正处于更改状态时更改模式。

例如，如果您正在更新某位用户的家庭电话号码，您不希望另一个用户在此时删除该电话号码列。如果系统允许其他用户在您完成操作前删除该电话号码列，您的事务会尝试在已不存在的列中写已更新的电话号码，从而导致数据损坏。

大部分表级锁定是隐式锁定；在必要的时候服务器本身会设置这些锁定。例如，当服务器认识到特定的操作（例如不带 WHERE 子句的 UPDATE 语句）将影响表中每条记录而且认为锁定整个表的效率最高，并且尚未存在对该表有冲突的锁定时，服务器本身可以锁定整个表。此外，当您获得表中记录的锁定时，您也会隐式地获得整个表的锁定（通常为共享锁定）。这防止一个用户在更新表中数据时另一位用户删除该表或修改该表的结构。

使用 LOCK TABLE 和 UNLOCK TABLE 命令您也可以手动锁定或解锁表级锁定。

表级锁定始终都是悲观锁定；服务器会对表挂起实际锁定，而不是仅仅查看版本信息。即使将表设置为使用乐观锁定方法，情况亦如此。

在使用高级复制的设置中，表级锁定通常与维护方式操作配套使用。有关更多信息，请参阅 *IBM solidDB Advanced Replication User Guide* 中的维护方式说明。

锁定方式：共享、互斥和更新

根据锁定模式不同，当有用户锁定某个记录时，该锁定会阻止其他用户更改甚至读取该记录。

有三种锁定方式：

- SHARED

行级共享锁定允许多个用户读取数据，但是不允许任何用户更改该数据。

表级共享锁定允许多个用户在该表上执行读取和写操作，但是不允许任何用户执行 DDL 操作。

多个用户可同时挂起共享锁定。

- EXCLUSIVE

互斥锁定仅允许一个用户/连接更新特定的数据（插入、更新和删除）。当一个用户对某行或某个表使用了互斥锁定时，则不能挂起其他任何类型的锁定。

- UPDATE

更新锁定始终是行级锁定。当用户使用 SELECT... FOR UPDATE 语句来访问某一行时，将使用更新方式锁定来锁定该行。这表示其他用户不能阅读或更新该行，从而保证了当前用户可在稍后更新该数据。

更新锁定与互斥锁定类似。二者之间的主要区别是，即使另一用户已对同一记录挂起共享锁定，您也可以对该记录挂起更新锁定。这允许更新锁定的持有人在不排除其他用户的情况下读取数据。然而，一旦更新锁定的持有人更改该数据，该更新锁定会转换为互斥锁定。

此外，相对于共享锁定，更新锁定是非对称的。您可以从已经具有共享锁定的记录中获取一个更新锁定，但是反过来却不行。由于更新锁定将阻止后续的读锁定，因此，将更新锁定转换为互斥锁定更为方便。

共享锁定和互斥锁定不能混用。如果 User1 已在一个记录上挂起了一个互斥锁定，则 User2 无法在该记录上获取共享锁定或互斥锁定。

同一特定类别中的所有锁定功能都是平等的（例如，共享锁定）。

- 无论是否拥有特权，所有用户都是平等的：即，使用 DBA 挂起锁定并不强于或弱于所有其他用户挂起的锁定。
- 执行这些挂起锁定的语句的所有方法都是平等的：执行锁定可作为以下对象的组成部分。
- 无论锁定是作为交互式类型的语句的部分来执行，还是从编译的远程应用程序中调用，亦或在将 solidDB 与共享内存访问或链接库访问配合使用时从本地应用程序调用，又或者该锁定作为一个存储过程或触发器中的语句的结果挂起，这些都不重要。

某些锁定可以升级。例如，如果您正在使用滚动游标并且对一个记录挂起共享锁定，接着在同一事务中更新该记录，那么该共享锁定可能会升级为互斥锁定。仅当该表上不存在任何其他锁定（共享锁定或互斥锁定）时，才有可能获取互斥锁定；如果您和另一个用户对同一个记录挂起共享锁定，那么服务器在该用户将其共享锁定删除之前无法将您的共享锁定升级为互斥锁定。

表级锁定的锁定方式

在悲观表和乐观表中，均可使用 EXCLUSIVE 和 SHARED 锁定方式。缺省情况下，乐观表和悲观表的锁定均为共享模式；除非您正在更改该表，否则表上的锁定通常都为共享锁定。

当您在执行 ALTER TABLE 操作时，您获得该表的共享锁定。这将允许其他用户继续从该表中读取数据，但不允许他们对该表进行更改。如果其他用户想要在同一时间在同一表中执行 DDL 操作（例如 ALTER TABLE），他们要么需要等待，要么接收到错误信息。

此外，在高级复制安装时，即便一些表的锁定为乐观模式，这些使用可选的 PESSIMISTIC 关键字运行的 solidDB 语句也需使用 EXCLUSIVE 表级锁定。

锁定持续时间和超时

缺省情况下，自获取锁定时开始，就会挂起该锁定，直到事务完成（通过落实或回滚来实现）。如果您试图获取另一用户已锁定（以共享方式或互斥方式）的记录上的互斥锁定，则您不能获取锁定；相反，您的事务将失败，并返回错误。您可以定义 solidDB 是否应立即令您的事务失败，或是在令其失败前等待并在指定的时间（以秒计）再次尝试。这可使用锁定超时设置进行控制。

锁定超时设置是引擎等待释放锁定所需的时间（以秒计）。缺省情况下，solidDB 的锁定超时设置为 30 秒。如果事务很短，短暂的等待使您能够继续执行活动，否则锁定会阻塞这些活动。

当达到锁定超时的时间间隔时，solidDB 将终止超时的语句。例如，如果 User1 正在查询表中的特定行，而 User2 尝试更新同一行的数据，则在 User1 查询完成（或超时）后更新才会继续。如果 User1 的查询结束，且 User2 的查询未超时，则会对 User2 的更新事务发布锁定。如果 User1 未能在 User2 的查询超时前完成，服务器将终止 User2 的语句。

缺省锁定超时用 **General.LockWaitTimeOut** 参数控制。在高级复制设置中，您可能还需要使用 **General.TableLockWaitTimeout** 参数为表级锁定设置缺省锁定时间。

可以使用以下特定于事务或节的命令来覆盖缺省超时：

- LOCK TABLE WAIT –为特定表（仅限 D 表）的表级锁定设置超时
- SET LOCK TIMEOUT – 为表级和行级锁定设置超时

SET LOCK TIMEOUT 不更改那些已使用 LOCK TABLE WAIT 设置其表级超时的表的超时。

注: LOCK TABLE WAIT 机制不适用于 M 表。例如，如果在“会话 1”中您锁定了表 DEPARTMENT (LOCK TABLE DEPARTMENT IN EXCLUSIVE MODE)，那么尝试将值插入“会话 2”的表中 (INSERT INTO DEPARTMENT VALUES ...)将立即返回错误 10014 Resource is locked.。

锁定超时中的等待机制仅适用于悲观锁定。不存在“正在等待乐观锁定”这种情况。如果某个用户在您读取数据后将其更改，那么无论等待多长时间都无法避免已发生的冲突。实际上，由于乐观并行方法不挂起锁定，因此没有可等待的“乐观锁定”。

LONG 互斥锁定

solidDB 允许您阻止在落实锁定事务的过程中释放互斥锁定。使用 LOCK TABLE 命令中的 LONG 选项设置这些 LONG 互斥锁定类型。

例如:

```
LOCK TABLE emp IN LONG EXCLUSIVE MODE
```

如果锁定事务异常中止或回滚，那么将释放所有锁定，其中包括 LONG 锁定。您必须使用 UNLOCK 命令显式解除 LONG 锁定。只有在 EXCLUSIVE 方式下，才允许挂起长 (LONG) 持续时间锁定。不支持 LONG 共享锁定。

事务隔离级别和锁定持续时间

更新锁定和互斥锁定在事务完成前一直挂起。共享锁定（“读取锁定”）在事务完成前也需一直挂起，但事务隔离级别会影响共享锁定的行为。例如，可序列化隔离级别将执行附加的检查。它还检查是否未对结果集添加该事务应该看到的新行。换言之，它将不允许其他用户插入符合该事务中结果集的条件行。

示例:

如果 SERIALIZABLE 事务有诸如 UPDATE customers SET x = y WHERE area_code = 415; 的更新命令，则 solidDB 不允许其他用户输入 area_code=415 的记录，直到序列化事务落实为止。

注: solidDB 挂起共享锁定直到事务完成这一实现与某些其他服务器不同。如果事务隔离级别足够低，某些服务器将在事务结束前释放共享锁定。其他数据库服务器也可能允许您延长读取/共享锁定的持续时间，以保证在单个事务中每次看到的数据都相同。

此外，在其他服务器中，事务隔离级别不仅可能影响记录的锁定时间长度，还会影响您所看到的内容。例如，在同时允许 READ COMMITTED（有时将其称为“脏读取”）和 READ COMMITTED 的系统上，隔离级别不仅影响由于您锁定特定记录而造成其他用户能查看或不能查看的内容，而且还影响您所看到的内容。

5.2.3 设置并行控制

并行控制方法和锁定方式可用 solidDB SQL 语句和配置参数控制。

将并行（锁定）方式设置为乐观或悲观

可在所有的表或特定的表上将基于磁盘的表的并行方式设置为乐观模式或悲观模式。内存表始终是悲观表。

在缺省情况下，D 表使用乐观锁定。

- 要为特定的表设置并行方式，请使用 `ALTER TABLE <table_name> SET OPTIMISTIC|PESSIMISTIC` 命令。

例如：

```
ALTER TABLE MyTable1 SET PESSIMISTIC;  
ALTER TABLE MyTable2 SET OPTIMISTIC;
```

- 要控制所有表的缺省并行方式，请将 `General.Pessimistic` 参数设置为“是”或“否”（缺省是“否”）。

例如：

```
[General]  
Pessimistic=yes
```

`General.Pessimistic` 参数仅在服务器启动时生效。如果您手动编辑 `solid.ini` 文件，则直到服务器重启前，该更改都不可见。

由于 `General.Pessimistic` 的值可以更改，因此表的并行控制方式也可能会更改。一个表可在服务器的一个实例期间使用乐观并行控制并在另一个实例期间使用悲观并行控制。

当您将 `General.Pessimistic` 参数设置为“是”时，服务器将针对以下内容缺省为悲观锁定：

- 任何新建的表，以及
- 从未使用 `AFTER TABLE` 命令显式地设置并行控制方式的任何现有表。

如果您使用 `AFTER TABLE` 命令设置表的锁定方式，则 `AFTER TABLE` 命令优先执行。

相关主题

- 『设置混合并行控制』

设置混合并行控制

有了 D 表，您可以使用混合并行控制方法。通过对各个表设置乐观或悲观并行控制，可以使用混合并行控制。

缺省情况下，`solidDB` 在 D 表中使用乐观并行控制。M 表始终是悲观表。

混合并行控制是行级悲观锁定与乐观并行控制的组合。通过逐个表地打开行级锁定，您可以指定单一事务同时使用这两种并行控制方法。此功能既可用于只读事务，也可用于读写事务。

要给各个表设置乐观或悲观并行控制，请使用以下命令：

```
ALTER TABLE base_table_name SET {OPTIMISTIC | PESSIMISTIC}
```

注：

当使用 solidDB 进行高级复制时，共享方式下的悲观表级锁定可与已同步的表一同使用。此功能为用户提供了以悲观方式运行某些同步操作（甚至对乐观表运行这些操作）的选项。例如，以悲观方式在副本数据库中执行 REFRESH 时，solidDB 将以共享方式锁定所有的表；以后，在必要时，服务器可以将这些锁定“提升”为互斥表锁定。这是在指定可选关键字 PESSIMISTIC 的情况下使用几个同步语句完成的。读操作不使用任何锁定。

读取并行方式

如何读取表的并行方式取决于并行方式的设置模式。

- 如果使用 ALTER TABLE 命令显式设置了该表的并行方式，则会在 SYS_TABLEMODES 系统表中记录该并行方式。

使用以下命令检查 SYS_TABLEMODES 中的值：

```
SELECT SYS_TABLEMODES.ID, SYS_TABLEMODES.MODE, SYS_TABLES.TABLE_NAME
FROM SYS_TABLEMODES, SYS_TABLES
WHERE SYS_TABLEMODES.ID = SYS_TABLES.ID
AND SYS_TABLES.TABLE_NAME = '<table_name>';
```

例如：

```
SELECT SYS_TABLEMODES.ID, SYS_TABLEMODES.MODE, SYS_TABLES.TABLE_NAME
FROM SYS_TABLEMODES, SYS_TABLES
WHERE SYS_TABLEMODES.ID = SYS_TABLES.ID
AND SYS_TABLES.TABLE_NAME = 'TESTTABLE2';
```

ID	MODE	TABLE_NAME
10002	PESSIMISTIC	TESTTABLE2

1 rows fetched.

如果未使用 ALTER TABLE 命令设置并行方式，则 SYS_TABLEMODES 系统表不包含该表并行方式的信息。

- 如果未使用 ALTER TABLE 命令设置该表的并行模式，请使用以下命令检查 **General.Pessimistic** 参数的设置：

```
ADMIN COMMAND 'describe parameter General.Pessimistic';
```

如果从服务器启动开始，solid.ini 文件的值一直未改变，且未通过 ADMIN COMMAND 覆盖该值，您也可以在 solid.ini 文件中查看该参数设置。

设置锁定超时

可使用 SET LOCK TIMEOUT 和 TABLE WAIT SET 命令修改锁定超时设置。缺省情况下，锁定超时设置为 30 秒。

- 使用 LOCK TABLE WAIT <timeout_in_seconds> 为表级锁定设置超时。

注：LOCK TABLE WAIT 命令仅在基于磁盘的表上生效。

- 使用 SET LOCK TIMEOUT <timeout_in_seconds> 在会话中为表级和行级锁定设置超时。

注：SET LOCK TIMEOUT 不更改那些已使用 LOCK TABLE WAIT 设置其表级超时的表的超时。

缺省情况下，超时的粒度以秒计。通过在值后面添加“MS”，可以设置详细程度为毫秒的锁定超时，例如：

```
LOCK TABLE emp,dept IN SHARED MODE WAIT 10MS;
```

或者

```
SET LOCK TIMEOUT 10MS;
```

如果未指定“MS”，锁定超时则以秒计。

注：最大超时是 1000 秒（略长于 15 分钟）。服务器不接受更大的值。

为乐观表设置锁定超时

当您使用 `SELECT FOR UPDATE` 时，将锁定您选中的行，即使该表的锁定方式为乐观模式亦如此。如果该语句成功执行，您可以在这些行中执行不间断的更新和删除操作，不用担心因冲突而失败。

如果未授权该锁定，在当前事务中竞争行写访问权（例如，`ELECT FOR UPDATE`、`UPDATE` 或 `DELETE`）的语句将立即失效。这相当于零超时。要想使用锁定等待，您可以将乐观锁定超时设置为非零值。

```
SET OPTIMISTIC LOCK TIMEOUT {seconds| milliseconds MS}
```

在会话中设置超时，`SELECT FOR UPDATE` 语句将阻塞（等待），直到超时失效或解锁。

对于立即返回的 `DELETE` 或 `UPDATE` 语句，超时无效 - 不管是成功还是冲突错误都如此。

`SET OPTIMISTIC LOCK TIMEOUT` 语句立即生效。如有必要，可以将超时值设为零以达到重设缺省行为的目的。

5.3 选择事务持久性级别

solidDB 提供两种持久性级别：严格和宽松。严格持久性表示落实事务时，服务器立即将信息写入事务日志文件。宽松持久性表示服务器可能不会在落实事务时立即写入信息；例如，服务器可能等到不怎么忙的时候或者能够通过单一写操作写入多个事务时才写入该信息。

如果您能够承受丢失少量最新数据，且性能对您而言至关重要，可以使用宽松持久性。宽松持久性适用于每个事务都并不关键的情况。例如，如果您正在监视系统性能，并且要存储有关响应时间的数据，那么您可能只对平均响应时间感兴趣，在这种情况下，即使丢失少量数据，也不会对平均响应时间产生显著影响。由于测量性能这一操作本身将影响性能（通过用尽处理器能力和 I/O 带宽之类的资源），因此，您可能希望性能跟踪操作本身具有高性能（低成本）而非高精度。宽松持久性适用于这种情况。

另一方面，如果您正在跟踪财务数据，例如账单支付情况，那么您可能想确存储 100% 的已落实数据并确保这些数据可恢复。在这种情况下，请使用严格持久性。

仅当您能够承受丢失少量最新事务这一情况时，才应该使用宽松持久性。否则，请使用严格持久性。如果您不确定严格持久性还是宽松持久性更合适，那么请使用严格持久性。

5.3.1 设置事务持久性级别

有四种方法设置事务持久性级别，以下以降顺序描述这四种方法：

1. **SET TRANSACTION DURABILITY** – 事务级设置

```
SET TRANSACTION DURABILITY { RELAXED | STRICT }
```

例如：

```
SET TRANSACTION DURABILITY STRICT;
```

SET TRANSACTION DURABILITY 命令为一个事务指定事务持久性。您必须在语句开始就发布 **SET TRANSACTION DURABILITY** 命令。持久性设置仅适用于当前事务，直到该事务落实或中止。

2. **SET DURABILITY** – 会话级设置

```
SET DURABILITY { RELAXED | STRICT }
```

例如：

```
SET DURABILITY STRICT;
```

SET DURABILITY 命令为会话指定事务持久性。会话是指从连接到服务器开始直到与服务器断开连接为止的时间段。每个用户都有一个不同的会话，即使各个会话在时间方面有所重叠亦如此。（单一用户可能建立多个会话，例如，通过运行 **sql** 的多个副本数据库或通过编写对同一服务器建立多个连接的程序）。当您使用 **SET DURABILITY** 语句时，您将指定只用于该命令发布的会话的事务持久性。您所作的选择不会影响任何其他用户、您自己当前的任何其他已打开的会话或者您的任何将来会话。每个用户会话都可以根据不丢失任何数据对于该会话而言的重要程度来设置自己的事务持久性级别。

此语句的效果将持续到会话结束或者发出另一个 **SET DURABILITY** 命令为止。

3. **Logging.DurabilityLevel** - 服务器缺省设置

```
[Logging]  
DurabilityLevel=3
```

该参数设置影响所有用户。

Logging.DurabilityLevel 参数可以动态更改。要在服务器运行期间更改缺省值设置，您需要发布以下命令：

```
ADMIN COMMAND 'parameter Logging.DurabilityLevel={1 | 2 | 3}'
```

此命令会立即生效。

4. **Logging.DurabilityLevel** - 服务器出厂值

如果您不使用以上任何的方式设置事务持久性级别，服务器将使用宽松持久性 (**Logging.DurabilityLevel=1**)。

6 SQL 诊断与故障诊断

solidDB 为 SQL 的故障诊断提供了下列诊断工具:

- SQL 信息工具和 EXPLAIN PLAN FOR 语句用于调整您的应用程序，并识别该应用程序中效率不高的 SQL 语句
- 用于存储过程和触发器的跟踪工具

您可以使用这些工具来观察性能、诊断问题以及生成高质量的问题报告。这些报告使您能够按照产品类别（例如 solidDB ODBC API、solidDB ODBC 驱动程序或 solidDB JDBC 驱动程序）来隔离问题，从而精确指出问题根源。

6.1 观察性能

您可以使用“SQL 信息”工具来提供有关 SQL 语句的信息，并可以使用 SQL 语句 EXPLAIN PLAN FOR 来显示 SQL 优化器为给定 SQL 语句选择的执行计划。通常，如果您需要与 IBM 软件支持机构联系，那么将要求您提供 SQL 语句、EXPLAIN PLAN 输出以及在信息级别 8 运行的 EXPLAIN PLAN 的“SQL 信息”输出以便生成更详尽的跟踪输出。

6.1.1 “SQL 信息”工具

“SQL 信息”工具将为 solidDB 所处理的每条 SQL 语句生成信息。

要生成“SQL 信息”，请在“SQL 信息”工具处于启用状态的情况下运行应用程序。可用以下方式启用“SQL 信息”工具:

- `Sql.Info=<info_level>` 参数
- `ADMIN COMMAND 'trace on info <info_level>'` 命令
- `SET SQL INFO ON LEVEL info_level FILE file_name` 语句

定义跟踪等级 (*info_level*) 为 0（没有跟踪）至 8（solidDB 信息来自每个已访存行）之间的整数。

表 18. “SQL 信息”级别

信息等级	描述
0	无输出
1	SQL 格式的表、索引和视图信息
2	SQL 执行图形（仅供技术人员使用）
3	某些 SQL 估算信息，solidDB 选择的键名
4	全部 SQL 估算信息，solidDB 选择的键信息
5	还包括来自已废弃的键的 solidDB 信息

表 18. “SQL 信息”级别 (续)

信息等级	描述
6	solidDB 表级信息
7	来自所访存的每一行的 SQL 信息
8	来自所访存的每一行的 solidDB 信息

跟踪信息在缺省情况下导出至 solidDB 工作目录中的 `soltrace.out` 文件。您也可以使用 `SQL.InfoFileName` 参数指定输出文件。当 `soltrace.out` 文件可能含有来自几个源的信息时，建议使用这种方式。

示例

```
[SQL]
信息=1
InfoFileName = solidsql_trace.txt
```

以下命令在第 3 等级上打开“SQL 信息”工具，将跟踪信息导出至工作目录中的 `my_query.txt` 文件中。该“SQL 信息”工具仅为执行该语句的客户端打开。

```
SET SQL INFO ON LEVEL 1 FILE 'my_query.txt'
```

以下 SQL 语句关闭“SQL 信息”工具：

```
SET SQL INFO OFF
```

6.1.2 EXPLAIN PLAN FOR 语句

EXPLAIN PLAN FOR 语句用于显示 SQL 优化器为给定 SQL 语句选择的执行计划。

EXPLAIN PLAN FOR 语句的语法如下所示：

```
EXPLAIN PLAN FOR sql_statement
```

执行计划是 solidDB 执行该语句时要执行的一系列基本操作以及这些操作的顺序。执行计划中的每个操作都称作单元”。

表 19. EXPLAIN PLAN FOR 单元

单元	描述
JOIN UNIT*	连接单元用于连接两个或更多个表。此连接可以使用循环连接或合并连接完成。
TABLE UNIT	表单元用于从表或索引中访存数据行。
ORDER UNIT	排序单元用于对行执行排序，以便进行分组或满足 ORDER BY。此排序可以在内存中进行，也可以使用外部磁盘排序器进行。
GROUP UNIT	组单元用于执行分组和聚集计算（SUM 和 MIN 等等）。
UNION UNIT*	并集单元用于执行 UNION 操作。此单元可以使用循环连接或合并连接完成。

表 19. EXPLAIN PLAN FOR 单元 (续)

单元	描述
INTERSECT UNIT*	交集单元用于执行 INTERSECT 操作。此单元可以使用循环连接或合并连接完成。
EXCEPT UNIT*	差集单元用于执行 EXCEPT 操作。此单元可以使用循环连接或合并连接完成。

* 对于只引用单个表的查询，也会生成此单元。在那种情况下，不会在此单元中执行连接；它仅仅是传递行，而不对其进行处理。

EXPLAIN PLAN FOR 语句返回的表包含下列各列。

表 20. EXPLAIN PLAN FOR 表格列

列名	描述
ID	输出行号，仅用于确保各行唯一。
UNIT_ID	这是 SQL 解释器中的内部单元标识。每个单元都具有不同的标识。单元标识是稀疏的编号序列，这是因为，SQL 解释器还为优化阶段除去的那些单元生成单元标识。如果多行具有相同的单元标识，那么意味着那些行属于同一个单元。为了进行格式化，来自一个单元的信息可能会分为多行。
PAR_ID	单元的父单元标识。父标识号引用 UNIT_ID 列中的标识。
JOIN_PATH	对于连接、并集、交集和差集单元而言，存在一条连接路径，此路径指定要在此单元中连接哪些表以及各个表的连接顺序。连接路径号引用 UNIT_ID 列中的单元标识。它表示此单元的输入来自该单元。表的连接顺序就是连接路径的列示顺序。列示的第一个表是循环连接中最外层的表。
UNIT_TYPE	单元类型是执行图单元类型。
INFO	INFO 列是为其他信息保留的。例如，它可能包含索引使用情况、数据库表名以及 solidDB 用于选择行的约束。注意，这里列示的约束可能与 SQL 语句中指定的那些约束不匹配。

在不同类型的单元的 INFO 列中，可能存在下列文本。

表 21. 单元 INFO 列中的文本

单元类型	INFO 列中的文本	描述
TABLE UNIT	<i>tablename</i>	表单元指表 <i>tablename</i> 。
TABLE UNIT	<i>constraints</i>	列示传递到数据库引擎的约束。例如，如果连接中的约束值无法事先确定，那么显示的约束值将是 NULL。
TABLE UNIT	SCAN TABLE	通过执行完全表扫描操作来搜索行。

表 21. 单元 INFO 列中的文本 (续)

单元类型	INFO 列中的文本	描述
TABLE UNIT	SCAN <i>indexname</i>	使用索引 <i>indexname</i> 来搜索行。如果能够在索引中找到所有选择的列，那么扫描索引有时比扫描整个表速度快，原因是索引包含的磁盘块较少。
TABLE UNIT	PRIMARY KEY	使用主键来搜索行。这与 SCAN 的不同之处在于，由于存在对主键属性的限制约束，因此不扫描整个表。
TABLE UNIT	INDEX <i>indexname</i>	使用索引 <i>indexname</i> 来搜索行。对于每个匹配的索引行，将单独地访存实际数据行。
TABLE UNIT	INDEX ONLY <i>indexname</i>	使用索引 <i>indexname</i> 来搜索行。选择的所有列都包含在索引中，因此不会通过读取表来单独地访存实际数据行。
JOIN UNIT	MERGE JOIN	通过进行合并连接来连接表。
JOIN UNIT	3-MERGE JOIN	通过进行 3 路合并连接来合并表。
JOIN UNIT	LOOP JOIN	通过进行循环连接来连接表。
ORDER UNIT	NO ORDERING REQUIRED	不需要执行排序，已按正确的顺序从 solidDB 检索行。
ORDER UNIT	EXTERNAL SORT	使用外部排序器对行进行排序。
ORDER UNIT	FIELD <i>n</i> USED AS PARTIAL ORDER	对于单值结果集，使用内部排序器（内存排序器）来执行排序，并且，从 solidDB 检索的行通过列号 <i>n</i> 进行不完全排序。不完全排序可以帮助内部排序器避免多次处理数据。
ORDER UNIT	<i>n</i> FIELDS USED FOR PARTIAL SORT	使用内部排序器（内存排序器）来执行排序，并且，从 solidDB 检索的行通过 <i>n</i> 个字段进行不完全排序。不完全排序可以帮助内部排序器避免多次处理数据。
ORDER UNIT	NO PARTIAL SORT	使用内部排序器进行排序。各个行按随机顺序从 solidDB 检索到排序器。
UNION UNIT	MERGE JOIN	通过进行合并连接来连接表。
UNION UNIT	3-MERGE JOIN	通过进行 3 路合并连接来合并表。
UNION UNIT	LOOP JOIN	通过进行循环连接来连接表。
INTERSECT UNIT	MERGE JOIN	通过进行合并连接来连接表。
INTERSECT UNIT	3-MERGE JOIN	通过进行 3 路合并连接来合并表。
INTERSECT UNIT	LOOP JOIN	通过进行循环连接来连接表。
EXCEPT UNIT	MERGE JOIN	通过进行合并连接来连接表。

表 21. 单元 INFO 列中的文本 (续)

单元类型	INFO 列中的文本	描述
EXCEPT UNIT	3-MERGE JOIN	通过进行 3 路合并连接来合并表。
EXCEPT UNIT	LOOP JOIN	通过进行循环连接来连接表。

示例 1

```
EXPLAIN PLAN FOR SELECT * FROM TENKTUP1 WHERE
UNIQUE2_NI BETWEEN 0 AND 99;
```

表 22. EXPLAIN PLAN FOR, 示例 1

ID	UNIT_ID	PAR_ID	JOIN_PATH	UNIT_TYPE	INFO
1	2	1	3	JOIN UNIT	
2	3	2	0	TABLE UNIT	TENKTUP1
3	3	2	0		FULL SCAN
4	3	2	0		UNIQUE2_NI <= 99
5	3	2	0		UNIQUE2_NI >= 0
6	3	2	0		

执行图形

JOIN UNIT 2 从 TABLE UNIT 3 获取输入。

表 TENKTUP1 的 TABLE UNIT 3 使用约束 UNIQUE2_NI <= 99 和 UNIQUE2_NI >= 0 执行完全表扫描操作。



图 4. 执行图 1

示例 2

```
EXPLAIN PLAN FOR SELECT * FROM TENKTUP1, TENKTUP2
WHERE TENKTUP1.UNIQUE2 > 4000 AND TENKTUP1.UNIQUE2 < 4500
AND TENKTUP1.UNIQUE2 = TENKTUP2.UNIQUE2;
```

表 23. EXPLAIN PLAN FOR, 示例 2

ID	UNIT_ID	PAR_ID	JOIN_PATH	UNIT_TYPE	INFO
1	6	1	9	JOIN UNIT	MERGE JOIN
2	6	1	10		
3	9	6	0	ORDER UNIT	NO ORDERING REQUIRED
4	8	9	0	TABLE UNIT	TENKTUP2
5	8	9	0		PRIMARY KEY
6	8	9	0		UNIQUE2 < 4500
7	8	9	0		UNIQUE2 > 4000
8	8	9	0		
9	10	6	0	ORDER UNIT	NO ORDERING REQUIRED
10	7	10	0	TABLE UNIT	TENKTUP1
11	7	10	0		PRIMARY KEY
12	7	10	0		UNIQUE2 < 4500
13	7	10	0		UNIQUE2 > 4000
14	7	10	0		

执行图形

JOIN UNIT 6: 使用合并连接算法对来自排序单元 9 和 10 的输入进行连接。

ORDER UNIT 9: 对来自 TABLE UNIT 8 的输入进行排序。由于已按正确的顺序来检索数据，因此无需执行实际的排序操作。

ORDER UNIT 10: 对来自 TABLE UNIT 7 的输入进行排序。由于已按正确的顺序来检索数据，因此无需执行实际的排序操作。

TABLE UNIT 8: 使用主键从表 TENKTUP2 中访存行。使用约束 UNIQUE2 < 4500 和 UNIQUE2 > 4000 来选择行。

TABLE UNIT 7: 使用主键从表 TENKTUP1 中访存行。使用约束 UNIQUE2 < 4500 和 UNIQUE2 > 4000 来选择行。

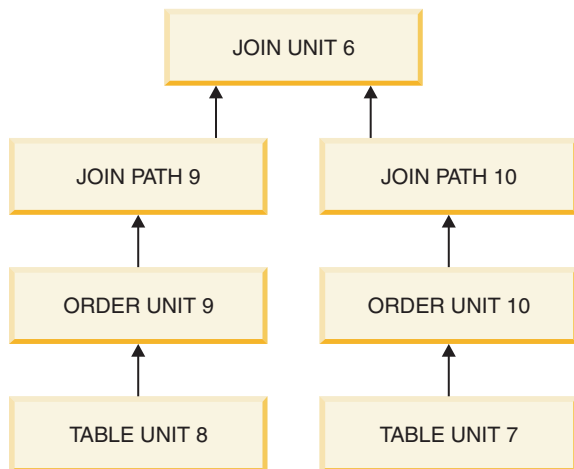


图 5. 执行图 2

6.2 用于存储过程和触发器的跟踪工具

在调试存储过程或触发器时，您可能想添加“跟踪”命令，以便查看代码的哪些部分正在执行。您也可能想跟踪该过程或触发器中的每个语句。下面两节将说明如何执行这些操作。

6.2.1 用户可定义的过程代码的跟踪输出

在存储过程或触发器中，可以使用以下命令将“跟踪”输出发送到 `soltrace.out` 文件：

```
WRITETRACE (entry VARCHAR)
```

可以使用以下命令来打开或关闭此输出：

```
ADMIN COMMAND 'usertrace { on | off }
user username { procedure | trigger | table } entity_name'
```

“`entity_name`”是要对其打开或关闭跟踪功能的过程、触发器或表的名称。如果指定了关键字“`table`”，那么将跟踪该表的所有触发器。

您可以对指定的过程、指定的触发器或者所指定表的所有触发器打开或关闭跟踪。

仅当指定的用户调用过程/触发器时，才会激活跟踪功能。这一点很有用，例如，在跟踪高级复制主数据库中传播的过程调用时，情况即如此。

打开跟踪功能将对此用户的所有过程/触发器调用打开跟踪功能，而不仅仅是对打开跟踪功能的连接中执行的调用打开跟踪功能。如果有多个使用同一个用户名的连接，那么将跟踪所有那些连接中的所有调用。此外，将对传播到主数据库并在其中执行的调用以及对副本数据库执行的调用执行跟踪。

6.2.2 执行跟踪过程

在必须跟踪存储过程或触发器中的每个语句时，您不会希望花费时间为每个 SQL 语句编写 `WRITETRACE` 语句。而是，您只需打开“`PROCTRACE`”，它将跟踪所指定存储过

程或触发器中的每个语句。与 USERTRACE 相同，您可以对指定的过程、指定的触发器或者所有与特定表相关联的触发器打开 PROCTRACE。语法为：

```
ADMIN COMMAND 'proctrace { on | off }
user username { procedure | trigger | table } entity_name'
```

“entity_name”是要对其打开或关闭跟踪功能的过程、触发器或表的名称。

仅当指定的用户调用过程/触发器时，才会激活跟踪功能。这一点很有用，例如，在跟踪高级复制主数据库中传播的过程调用时，情况即如此。

打开跟踪功能将对此用户的所有过程/触发器调用打开跟踪功能，而不仅仅是对打开跟踪功能的连接中执行的调用打开跟踪功能。如果有多个使用同一个用户名的连接，那么将跟踪所有那些连接中的所有调用。此外，将对传播到主数据库并在其中执行的调用以及对副本数据库执行的调用执行跟踪。

如果指定了关键字“table”，那么将跟踪该表的所有触发器。

示例：

```
"create procedure trace_sample(i integer)
returns(j integer)
begin
    j := 2*i;
    return row;
end";
commit work;

admin command 'proctrace on user DBA procedure TRACE_SAMPLE';
call trace_sample(2);
```

示例的输出：

```
23.01 17:25:17 ---- PROCEDURE 'DBA.DBA.TRACE_SAMPLE' TRACE BEGIN ----
0001:CREATE PROCEDURE TRACE_SAMPLE(I INTEGER)
0002:RETURNS(J INTEGER)
0003:BEGIN
    --> I:=2
    --> J:=NULL
    --> SQLSUCCESS:=1
    --> SQLERRNUM:=NULL
    --> SQLERRSTR:=NULL
    --> SQLROWCOUNT:=NULL

0004:      J := 2*I;
    --> J:=4
0005:      RETURN ROW;
0006:END
23.01 17:25:17 ---- PROCEDURE 'DBA.DBA.TRACE_SAMPLE' TRACE END ----
```

6.3 测量和提高 START AFTER COMMIT 语句的性能

6.3.1 调整 START AFTER COMMIT 语句的性能

您可以通过 SSC-API 和管理命令来控制后台任务（有关详细信息，请参阅《IBM solidDB 共享内存访问和链接库访问用户指南》）。对于执行通过 START AFTER COMMIT 启动的语句的任务，将使用任务类型 SSC_TASK_BACKGROUND。您可以对此任务类型指定更高或更低的优先级，也可以暂挂此任务类型。

注意，可能存在多个这样的任务，但您无法逐个地对其进行控制。换言之，如果对 SSC_TASK_BACKGROUND 调用 SSCSuspendTaskClass，那么将暂挂所有后台任务。

6.3.2 分析 START AFTER COMMIT 语句的故障

对于可以同时存在的未落实 START AFTER COMMIT 语句数，存在一个限制。（“未落实”表示尚未落实在其中执行 START AFTER COMMIT 语句的事务。在此时间点，START AFTER COMMIT 语句的主体 - 例如，过程调用 - 甚至尚未开始执行。）如果达到最大数目，那么发出下一个 START AFTER COMMIT 时，将返回错误。您可以在 solid.ini 中使用名为 MaxStartStatements 的参数来配置最大数目（有关详细信息，请参阅《IBM solidDB 管理员指南》中有关此参数的描述）。

如果某个语句无法启动，那么故障原因将记录到系统表 SYS_BACKGROUNDJOB_INFO 中。只有失败的 START AFTER COMMIT 语句才会记录到此表中。有关此表的详细信息，请参阅第 320 页的 E.1.4，『SYS_BACKGROUNDJOB_INFO』。

用户可以通过使用 SQL SELECT 语句或者通过调用系统过程 SYS_GETBACKGROUNDJOB_INFO 从 SYS_BACKGROUNDJOB_INFO 表中检索信息。存储过程 SYS_GETBACKGROUNDJOB_INFO 将返回与 START AFTER COMMIT 语句的给定作业标识相匹配的行。有关 SYS_GETBACKGROUNDJOB_INFO 的更多详细信息，请参阅第 372 页的 G.2.1，『SYS_GETBACKGROUNDJOB_INFO』。

如果您希望在语句未能启动时接收到通知，那么可以等待系统事件 SYS_EVENT_SACFAILED。请参阅第 373 页的 H.1，『其他事件』中有关此事件的描述以获取详细信息。应用程序可以等待此事件并使用作业标识从系统表 SYS_BACKGROUNDJOB_INFO 中检索错误消息。

7 使用 SQL 微调性能

本章讨论了可用于提高 solidDB 性能的 SQL 相关技术。

有关优化高级复制数据同步的技巧，请参阅 *IBM solidDB Advanced Replication User Guide* 中的性能监视及微调。

7.1 调整 SQL 语句和应用程序

调整 SQL 语句通常是最有效的提高数据库性能的方法，在涉及复杂查询的应用程序中尤其如此。

务必在调整 RDBMS 之前调整应用程序，原因如下：

- 在应用程序设计期间，您可以控制所要处理的 SQL 语句和数据
- 您可以提高性能，即使不熟悉将要使用的 RDBMS 的内部工作亦如此
- 如果应用程序调整得不够好，那么它即使在调整得很好的 RDBMS 上也无法很好地运行

您应该知道应用程序所处理的数据、所使用的 SQL 语句以及应用程序对数据执行的操作。例如，通过保持 SELECT 语句简单（避免使用不必要的子句和谓词），可以提高查询性能。

7.1.1 评估应用程序性能

为了确定应用程序中性能不佳的区域，solidDB 提供了下列用于观察数据库性能的诊断工具：

- “SQL 信息”工具
- EXPLAIN PLAN FOR 语句

这些工具可以帮助您调整应用程序以及标识其中效率不高的 SQL 语句。有关如何使用这些工具的更多信息，请参阅第 115 页的 6，『SQL 诊断与故障诊断』。

此外，下列命令可以提供对于评估性能而言非常有用的信息。

- **ADMIN COMMAND 'status'**

此命令返回来自服务器的统计信息。有关详细信息，请参阅《*IBM solidDB 管理员指南*》中有关此命令的内容。

- **ADMIN COMMAND 'perfmon'**

此命令返回来自服务器的详细性能统计信息。有关使用该命令生成性能报告的详细信息，请参阅《*IBM solidDB 管理员指南*》中的性能计数器 (*perfmon*) 一节。

- **ADMIN COMMAND 'trace'**

此命令对 SQL 语句和网络通信打开跟踪功能。要了解完整的语法，请参阅第 133 页的 A.1，『ADMIN COMMAND』中的跟踪选项语法。

7.1.2 使用存储过程语言

使用存储过程可以在两个方面提高一些操作的速度:

- 存储过程中的语句将被解析并编译一次，然后以经过编译的形式存储。存储过程外部的语句将在每次执行时被预解析和编译。因此，如果要多次执行语句，那么将这些语句放入存储过程将会降低开销（解析和编译）。
- 如果单一存储过程包含多条语句，那么与每条语句逐个地从客户机传递到服务器相比，调用一次存储过程所需进行的网络“往返”次数较少。

7.2 优化单表 SQL 查询

solidDB 提供简单 SQL 优化功能，用于提高特定类型的单表 SQL 查询的性能。性能改进适用于 SELECT、DELETE 和 UPDATE 语句。此功能不适用于 INSERT 语句。

用 **SQL.SimpleSQLOpt** 参数启用/禁用简单 SQL 优化。在缺省情况下，此功能处于打开状态，并且 `solid.ini` 文件未包含 **SQL.SimpleSQLOpt** 参数。要禁用此功能，必须在 `solid.ini` 文件中添加下列各行:

```
[SQL]
SimpleSQLOpt=No
```

在此文件中添加这些行之后，您可以始终通过指定 **SimpleSQLOpt=Yes** 或从 [SQL] 一节中除去此参数来启用此功能。

打开“简单 SQL 优化”功能后，solidDB 将自动优化符合下列条件的单表 SQL 查询:

- 语句只访问一个表。
- 语句未包含视图、子查询和 UNION INTERSECT 等内容。
- 语句未使用 ROWNUM。
- 语句未使用用于检索序号的 solidDB 序列对象。

与其他优化技术相同，简单 SQL 优化功能能够提高大多数查询的速度，但会导致少数查询类型的性能下降。如果您使用简单 SQL 优化功能后特定查询的运行速度显著下降，那么可以将此功能关闭。

7.3 使用索引来提高查询性能

您可以使用索引来提高查询的性能。如果一个查询的 WHERE 子句引用了已建立索引的列，那么该查询可以使用索引。如果该查询仅选择已建立索引的列，那么该查询可以直接从索引中读取已建立索引的列值，而不是从表中进行读取。

如果一个查询的 SELECT 列表中的所有字段都包含在索引中，那么 solidDB 优化器就可以使用该索引，而不必执行附加的查找操作来读取完整的记录。同样，如果 WHERE 子句的所有字段都包含在索引中，那么优化器可以使用该索引 - 如果该索引中的信息足以证明该记录不符合 WHERE 子句的条件，那么优化器可以避免查找完整的记录。

例如，假定 WHERE 子句引用两个或更多个列，例如:

```
WHERE col1 = x AND col2 >= a AND col2 <=b
```

并且，假定有一个索引同时包含 col1 和 col2，并假定该索引将 col1 或 col2 作为键的前导列。例如，如果对 col2 + col3 + col1 建立索引，那么此索引将包含这些列，并且其中一个列（col2）是该键的前导列。如果用户的查询是：

```
SELECT col1, col4
FROM table1
WHERE col1 = x AND col2 >= a AND col2 <=b;
```

那么，除非符合搜索条件，否则不需要查找完整的记录。归根结底，如果未符合搜索条件，那么我们不关心 col4 的值，因此不需要查找完整的记录。

如果一个表具有主键，那么 solidDB 将按照该主键的值顺序对磁盘中的行进行排序。由于这些行按主键的顺序进行物理排序，所以，主键本身用作索引，并且适用于索引的优化技巧也适用于主键。

如果该表不具有用户指定的主键，那么各行将使用 ROWID 进行排序。插入每一行时都将对其指定 ROWID，并且，每个记录都将获得比之前插入的记录更大的 ROWID。因此，在不具有用户指定的主键的表中，记录将按照那些行的插入顺序存储。有关主键的更多信息，请参阅第 91 页的 4.4.2，『主键索引』。

具有行值构造器约束的搜索将会优化成尽可能使用索引。为了提高效率，solidDB 将使用索引来解析格式为 (A, B, C) >= (1, 2, 3) 的行值构造器约束，其中，运算符可以是下列任何一项：<、<=、>= 和 >。（服务器不会使用索引来解析包含 =、!= 或 <> 运算符的行值构造器约束。服务器可以使用索引来解析使用 =、!= 或 <> 的其他类型约束。）有关行值构造器的更多信息，请参阅第 18 页的 2.5.5，『行值构造器』。

索引能够提高从表中选择少量行的查询的性能。对于选择的表行所占百分比低于 15% 的查询，您应该考虑使用索引。

7.3.1 全表扫描

如果查询无法使用索引，那么 solidDB 必须执行全表扫描才能执行该查询。这涉及按顺序读取一个表中所有的行。将对每一行执行检查，以确定它是否符合该查询的 WHERE 子句的条件。与使用全表扫描方法来查找单一行相比，使用索引的查询查找该行的速度要快得多。另一方面，如果一个查询要选择表中 15% 以上的行，那么使用全表扫描方法时的速度要比使用索引时的速度快。

您应该使用 EXPLAIN PLAN 语句来检查每个查询。执行此检查时，请使用真实的数据，因为最佳的计划依赖于实际数据量以及该数据的特征。EXPLAIN PLAN 语句的输出允许您检测是否确实使用某个索引，必要时，您可以重做该查询或索引。全表扫描方法通常会导致 SELECT 查询响应时间过长以及磁盘活动过多。

要诊断性能下降问题，可以参照 《IBM solidDB 管理员指南》中的性能计数器 (perfmon) 一节，使用 ADMIN COMMAND 'perfmon' 请求有关文件操作的统计数据。

执行全表扫描时，将读取该表中的每个块。对于每个块，将读取该块中存储的每一行。执行使用索引进行的查询时，将按各行在索引中的出现顺序来读取那些行，而不考虑那些行所在的块。如果某个块包含多个所选行，那么将多次读取该块。因此，在某些情况下，如果结果集相对较大，那么全表扫描操作所要求进行的 I/O 活动将少于使用索引进行的查询。

7.3.2 并置型索引

索引可以由不止一个列组成。这样的索引称为“并置型索引”。我们建议您尽可能使用并置型索引。

SQL 语句是否使用并置型索引由该 SQL 语句的 WHERE 子句中包含的列确定。如果一个查询的 WHERE 子句引用了并置型索引的开头部分，那么它可以使用该索引。索引的开头部分是指 CREATE INDEX 语句中指定的第一个列或前几个列。

示例:

```
CREATE INDEX job_sal_deptno ON emp(job, sal, deptno);
```

下列查询可以使用此索引:

```
SELECT * FROM emp WHERE job = 'clerk' and sal =  
800 and deptno = 20;  
SELECT * FROM emp WHERE sal = 1250 and job = salesman;  
SELECT job, sal FROM emp WHERE job = 'manager';
```

以下查询的 WHERE 子句未包含该索引的第一列，因此无法使用该索引:

```
SELECT * FROM emp WHERE sal = 6000;
```

选择要建立索引的列

以下列表提供选择要建立索引的列时应该遵循的准则:

- 应该对 WHERE 子句中经常使用的列创建索引。
- 应该对经常用于连接表的列创建索引。
- 应该对 ORDER BY 子句中经常使用的列创建索引。
- 应该对表中包含很少相同值或唯一值的列创建索引。
- 不应该对小型的表（仅使用几个块的表）创建索引，这是因为完全表扫描操作可能比使用索引执行的查询速度快。
- 有可能时，请选择按最合适的顺序对行进行排序的主键。
- 如果在 WHERE 子句中只是经常使用并置索引的某一列，那么请将该列放在 CREATE INDEX 语句的最前面。
- 如果在 WHERE 子句中经常使用并置索引的多个列，那么请将最有选择性的列放在 CREATE INDEX 语句的最前面。

7.4 等待事件

在许多程序中，可能必须等待特定条件成立才能执行特定任务。在某些情况下，可以使用“while”循环来检查条件是否成立。solidDB 提供了“事件”，在某些情况下，事件使您能够避免因循环等待条件而浪费 CPU 时间。

一个或多个客户机或线程可以等待一个事件，另一个客户机或线程可以发出该事件。例如，可能有多个线程等待传感器获取新数据。另一个与该传感器配合使用的线程可以发出表明数据可用的事件。有关事件的更多信息，请参阅第 73 页的 3.5，『事件』以及第 133 页的附录 A，『语句』的各章节，其中包括第 156 页的 A.10，『CREATE EVENT』。

7.5 优化批处理插入和更新

您应该设计支持按主键顺序运行批处理插入的数据库模式。数据库文件中的数据实际上按照表中主键所定义的顺序存储。如果未定义主键，那么数据将按写入数据库的顺序存储在数据库文件中。数据库操作（即，读和写）始终在页级访问数据。用 `IndexFile.BlockSize` 参数定义数据库的页面大小（块大小）。

如果按照执行主键的顺序来执行批处理写操作，那么服务器的高速缓存算法就能够对数据库文件写操作进行分组。这样，就可以在一次物理磁盘 I/O 操作中将大量的行写入磁盘。在最坏的情况下，如果插入顺序与主键顺序不同，那么每个插入或删除操作都要求重写数据库页，但只更改一行。

因此，确保批处理写操作的表具有与批处理写操作的访问顺序匹配的主键极为重要。此类数据库模式能够显著提高操作性能。

例如，根据以下命令创建一个表：

```
CREATE TABLE USAGE_EVENT (  
    EVENT_ID INTEGER NOT NULL PRIMARY KEY,  
    DEVICE_ID INTEGER NOT NULL,  
    EVENT_DATA VARCHAR NOT NULL);
```

在这个表中，`EVENT_ID` 是序号。插入和删除操作按照 `EVENT_ID` 列指定的顺序执行，从而最大程度地提高效率。

如果主键的第一列是 `DEVICE_ID`，但数据按 `EVENT_ID` 顺序写入数据库，那么对这个表执行的批处理写操作的性能将显著下降。在这种情况下，随着表增大，完成批处理写操作所需的文件 I/O 操作次数也会增加。

7.5.1 提高批处理插入和更新速度

您可以提高对 `solidDB` 执行的大规模批处理插入和更新的速度。提高速度的准则如下所示：

1. 检查是否在自动落实方式处于关闭状态的情况下运行应用程序。

`solidDB` ODBC 驱动程序的缺省设置是 `AUTOCOMMIT`。这是符合 ODBC 规范的标准设置。要将应用程序设置为关闭自动落实方式，请调用 `SQLSetConnectOption` 函数，如以下示例所示：

```
rc = SQLSetConnectOption  
(hdbc, SQL_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF);
```

2. 不要使用大型事务。建议将初始事务大小设置为 500 行。事务大小的最佳值随特定应用程序的不同而有所变化；您可能需要进行试验。
3. 要提高批处理插入的执行速度，可以将日志记录功能关闭。但是，这将提高在系统故障期间丢失数据的风险。在某些环境中，这种代价可接受。

这些准则的第 1 项和第 2 项是您提高批处理插入速度时要采取的最重要措施。实际插入速率还取决于硬件、每行的数据量以及表的现有索引。

7.6 使用优化器提示

提示是为 SQL 优化器提供伪指令的 SQL 扩展名，SQL 优化器决定要使用的查询执行计划。使用查询语句中的嵌套伪注释指定提示。优化器检测这些伪指令或提示，并将其查询执行计划基于其上。优化器提示允许在多种条件下优化应用程序到数据、查询类型和数据库。这不仅为查询偶尔遇到的性能问题提供解决方案，也将响应时间的控制权从系统转交给用户。

由于数据、用户查询和数据库存在各种情况，SQL 优化器并非始终能够选择可能的最佳执行方案，所以需要提示。例如，您可能想要强行执行合并连接，因为您知道数据已排序，而不是像优化器那样需要提示。此外，有时查询中特定的谓词会引起性能问题，而优化器无法消除这些问题。优化器可能正在使用您确定并非最优的索引。在这种情况下，您可能需要强制优化器使用能够更快生成结果的索引。

提示适用于：

- 选择合并或嵌套的循环连接
- 使用 FROM 列表中指定的固定连接顺序
- 选择内部排序或外部排序
- 选择特定索引
- 选择基于索引扫描的表扫描
- 选择在分组之前或之后进行排序

在 SELECT、UPDATE 或 DELETE 关键字后，您可以将提示以静态字符串的形式放置在 SQL 语句中。在 INSERT 关键字后面，不允许使用提示。

在提示规范中存在错误时，整个 SQL 语句将失败并返回错误消息。

启用和禁用提示

使用 **SQL.EnableHints** 参数启用和禁用提示。在缺省情况下，启用提示（使用 **SQL.EnableHints=yes**）。

相关参考

第 215 页的 A.53, 『 HINT 』

第 219 页的 A.53.1, 『 solidDB-支持提示 』

7.7 对性能不佳问题进行诊断

在 solidDB 中, 有多个不同的区域会导致性能下降。为了解决性能问题, 您需要确定根本原因。下表列示了性能不佳问题的常见症状、可能的原因以及本章中可以帮助您解决相应问题的章节。

表 24. 对性能不佳问题进行诊断

症状	诊断	解决方案
单一查询的响应时间过长。对数据库进行的其他并发访问受影响。磁盘可能比较忙。	<ul style="list-style-type: none">• 查询未充分地使用索引。• 优化器作出的决策不够优化。• 未定义外部排序, 并且大量的内部排序导致将过多内容交换到磁盘。	<p>如果缺少索引定义, 那么请创建新索引或者修改现有索引, 以便满足低速查询的索引需求。有关更多详细信息, 请参阅 使用索引提高查询功能。</p> <p>对低速查询运行 EXPLAIN PLAN FOR 语句, 并验证查询优化器是否正在使用索引。有关更多详细信息, 请参阅 EXPLAIN PLAN FOR 语句。</p> <p>如果优化器未选择最优的查询执行方案, 那么请使用优化器提示来覆盖优化器决策。有关更多详细信息, 请参阅 使用优化器提示。</p>
所有查询的响应时间都过长。增加并发用户数导致性能以超出线性情况的幅度下降。即使剔除所有用户并重新连接, 性能也没有改善。	高速缓存大小不足。	增大高速缓存大小。对于每个并发用户, 至少分配 0.5MB 高速缓存, 或者分配数据库大小的 2-5%。有关更多详细信息, 请参阅 《IBM solidDB 管理员指南》 中的 定义数据库高速缓存大小 一节。
所有查询和写操作的响应时间都过长。即使剔除所有用户并重新连接, 性能仅仅暂时有所改善。磁盘非常忙。	Bonsai 树太大, 在高速缓存中装不下。	请确保不存在无意中长时间运行的事务。验证是否所有事务 (包括只读事务) 都及时地落实。有关更多详细信息, 请参阅 《IBM solidDB 管理员指南》 中的 通过落实事务减少 Bonsai 树大小 。
随着数据库大小增大, 批处理写操作的性能下降。磁盘 I/O 量过大。	<ul style="list-style-type: none">• 落实到数据库的数据批次过小。• 将数据写入磁盘时采用的顺序不受表的主键支持。	<p>请确保关闭自动落实功能, 并确保按照每个事务至少 100 行的水平成批落实写操作。</p> <p>请修改主键或批处理写过程, 以便按照主键顺序执行写操作。有关更多详细信息, 请参阅 优化批处理插入更新。</p>
服务器进程覆盖区过度增大, 导致操作系统执行交换操作。磁盘非常忙。ADMIN COMMAND 'report' 输出表明当前存在大量活动语句。	使用 SQL 语句完成后, 未将其关闭并删除。	请确保及时地关闭并删除客户机应用程序不再使用的语句。

附录 A. 语句

本部分提供了 solidDB SQL 语句的描述及示例。

solidDB 使用基于 ANSI X3H2 和 IEC/ISO 9075 SQL 标准的 SQL 语法。完全支持 SQL-89 Level 2 标准，包括重要的 ANSI X3H2-1992 (SQL-92) 扩展。也支持完整 SQL-92、SQL-99 和 SQL-2003 标准的许多功能。

A.1 ADMIN COMMAND

```
ADMIN COMMAND 'command_name'
command_name ::= ABORT | ASSERTEXIT | BACKUP |
BACKGROUNDJOB | BACKUPLIST | CHECKPOINTING | CLEANBGJOBINFO |
CLOSE | DESCRIBE | ERRORCODE | ERROREXIT | ERRORMESSAGE | FILESPEC |
HELP | HOTSTANDBY | INDEXUSAGE | INFO | LOGMESSAGE | LOGREADER | MAKECP | MEMORY |
MESSAGES | MONITOR | NETBACKUP | NETBACKUPLIST | NETSTAT | NOTIFY |
OPEN | PARAMETER | PASSTHROUGH STATUS | PERFMON | PERFMON DIFF | PID | PROCTRACE |
PROTOCOLS | REPORT | RUNMERGE | SAVE | SHUTDOWN | SQLLIST | STARTMERGE |
STATUS | THROWOUT | TID | TRACE | TRACEMESSAGE | USERID | USERLIST |
USERTRACE | VERSION
```

用法

ADMIN COMMAND 是 solidDB-它是用于执行管理命令的指定 SQL 扩展名。

将 ADMIN COMMAND 与 solidDBSQL 编辑器 (**solsql**) 配合使用

当与 solidDB SQL 编辑器 (**solsql**) 配合使用时，*command_name* 必须用引号引起。例如：

```
ADMIN COMMAND 'backup'
```

将 ADMIN COMMAND 与 solidDB 远程控制 (**solcon**) 配合使用

当与 solidDB 远程控制 (**solcon**) 合使用时，ADMIN COMMAND 语法仅包括 *command_name*，没有引号。例如：

```
备份
```

缩写

也可使用 ADMIN COMMAND 的缩写。例如：

```
ADMIN COMMAND 'bak'
```

要访问缩写命令的列表，请执行

```
ADMIN COMMAND 'help'
```

此结果集包含两列：RC 和 TEXT：

- RC（返回码）列是命令返回码。如果成功执行该命令，那么返回的值为 0。
- TEXT 列是命令应答。

重要的用法注意事项

- **ADMIN COMMAND** 的所有选项都不是事务性选项，因此无法回滚。
- **ADMIN COMMAND** 和启动事务

虽然 **ADMIN COMMAND** 不具有事务性，但如果不存在已打开的事务，那么这些命令将启动一个新事务。（它们不会落实或回滚任何已打开的事务。）此效果通常并不重要。但是，这可能会影响事务的“开始时间”，并且有时可能会产生意外的效果。**solidDB** 的并行控制基于版本控制系统；您看到的数据库是它在事务启动时所处的状态。

例如，如果您发出 **ADMIN COMMAND** 而未另外进行落实，然后离开一小时；那么当您返回时，下一个 **SQL** 命令面向的可能是 1 小时前的数据库，即，数据库还处于您最初使用 **ADMIN COMMAND** 启动事务时的状态。

- **错误代码**

仅当命令语法或参数值不正确时，**ADMIN COMMAND** 中的错误代码才会返回错误。仅当所请求的操作可以启动时，命令才会返回 **SQLSUCCESS (0)**。操作本身的结果将写入结果集。结果集有两列：**RC** 和 **TEXT**。**RC**（返回码）列包含操作的返回码：0 表示成功，不同的数字值表示不同的错误。因此，有必要检查 **ADMIN COMMAND** 语句的代码以及操作的代码。

每个 **ADMIN COMMAND** 命令选项的语法描述如下所示：

表 25. **ADMIN COMMAND** 语法和选项

选项语法	描述
ADMIN COMMAND 'abort [backup netbackup]'	中止活动的本地备份或网络备份进程。备份操作不保证是原子操作，因此，已取消的操作可能会在备份目录中生成不完整的备份文件，直到执行下一次备份为止。 如果没有输入选项，该命令缺省为 ADMIN COMMAND 'abort backup' 。
ADMIN COMMAND 'assertexit' 缩写: asex	立即终止服务器而不执行适当的关闭操作。
ADMIN COMMAND 'backgroundjob' [LIST [-1] [user]] [ABORT {jobid user ALL}] [DELETE ERRORINFO {jobid user ALL}]' <i>user ::= USER {username userid}</i> 缩写: bgjob	列示并可能中止运行中的后台作业，即，已使用 START AFTER COMMIT 语句启动的 SQL 语句。 <ul style="list-style-type: none"> • LIST 选项列示所有用户或某个指定用户正在运行的作业。 • -1 选项指长列表（与 ADMIN COMMAND 'userlist -1' 相似）。 • ABORT 选项按作业标识号中止其中某些作业或按用户标识号中止所有作业。如果指定不带自变量的 ABORT，那么将中止所有用户的所有作业。 • DELETE ERRORINFO 选项删除来自 SYS_BACKGROUNDJOB_INFO 系统表的错误信息，后台作业遇到的错误存储在该表中。该选项执行的操作与不推荐使用的 ADMIN COMMAND 'CLEANBGJOBINFO' 命令所执行的操作相同。
ADMIN COMMAND 'backup [-s] [backup_directory]' 缩写: bak	创建数据库备份。此操作可以采用同步方式或异步方式（缺省）执行。可以使用 -s 选项（可选）指定同步操作。 用 General.BackupDirectory 定义缺省备份目录。备份目录也可以作为自变量指定。例如， backup abc 将在目录 abc 中创建备份。所有目录定义都相对于 solidDB 工作目录。
ADMIN COMMAND 'backuplist' 缩写: bls	显示上一次本地备份的状态列表。
ADMIN COMMAND 'checkpointing {ON OFF}' 缩写: cp	打开或关闭检查点

表 25. ADMIN COMMAND 语法和选项 (续)

选项语法	描述
ADMIN COMMAND 'cleanbgjobinfo' 缩写: cleanbgi	注: 建议您不要使用此命令。请改用 ADMIN COMMAND 'backgroundjob'。 清除包含后台过程的状态数据的 SYS_BACKGROUNDJOB_INFO 表。
ADMIN COMMAND 'close' 缩写: clo	让服务器拒绝新连接; 不允许建立新连接。
ADMIN COMMAND 'describe parameter param' 缩写: des	返回所有参数或使用 <i>param</i> 指定的参数的描述。 <i>param</i> 必须以 section_name.param_name 格式提供。节名和参数名不区分大小写。 以下示例描述参数 Com.Trace = y/n : ADMIN COMMAND 'des parameter com.trace' RC TEXT <pre>-- ---- 0 Trace 0 If set to 'yes', trace information of the network messages is written to a file 0 BOOL 0 RW/STARTUP 0 0 0 No 7 rows fetched.</pre>
ADMIN COMMAND 'errorcode {all SOLID_error_code}' 缩写: ec	返回所有错误代码或特定错误代码的描述。 <i>SOLID_error_code</i> 是代码编号, 例如 10034。 ADMIN COMMAND 'errorcode 10034'; RC TEXT <pre>-- ---- 0 Code: DBE_ERR_SEQEXIST (10034) 0 Class: Database 0 Type: Error 0 Text: Sequence already exists 4 rows fetched.</pre>
ADMIN COMMAND 'errorexit <number>' 缩写: erex	强制服务器立即退出进程并返回给定的进程退出码。
ADMIN COMMAND 'errormessage <string>' 缩写: ermsg	将用户定义的 <string> 输出到错误消息日志 (solerror.out) 中。
ADMIN COMMAND 'filespec [-d -a "<file_name> <max_file_size_in_bytes> [<device_number>"]]' 缩写: fs	显示或修改由 IndexFile.FileSpec 参数定义的数据库 (索引) 文件规范以及文件大小和当前填充率 (百分比)。 <ul style="list-style-type: none"> • -d 删除 <file_name max_file_size_in_bytes> [device_number] 指定的数据库文件 • -a 添加 <file_name> >max_file_size_in_bytes> [<device_number>] 指定的新数据库文件规范。 例如: <pre>ADMIN COMMAND 'fs -a "solid3.db 3000M"; RC TEXT -- ---- 0 Added: FileSpec_3 = solid3.db 3145728000</pre> 数据库文件规范更改在关闭时保存在 solid.ini 配置文件中。
ADMIN COMMAND 'help' 缩写: ?	显示可用的命令。
ADMIN COMMAND 'hotstandby [option]' 缩写: hsb	热备用命令。 要获取选项列表, 请参阅《IBM solidDB 高可用性用户指南》。 要获取选项列表, 请参阅《《IBM solidDB 高可用性用户指南》》中的『热备用 ADMIN COMMAND』。

表 25. ADMIN COMMAND 语法和选项 (续)

选项语法	描述
ADMIN COMMAND 'indexusage' 缩写: idxu	显示索引以显示已使用的Displays the indexes, showing the number of times each index has been used.
ADMIN COMMAND 'info [options]' 缩写: info	<p>返回服务器信息。</p> <p>输出由 25 行数据组成。</p> <p><i>options</i> 列示如下:</p> <ul style="list-style-type: none"> • numusers - 当前用户数。 • maxusers - 最大用户数。 • sernum - 服务器序列号。 • dbsize - 数据库大小 (KB)。 • logsize - 日志文件大小 (KB)。 • uptime - 服务器启动时间戳记。 • bcktime - 上次成功完成的本地备份的时间戳记。 • cptime - 上次成功完成的检查点的时间戳记。 • tracestate - 当前跟踪状态 - 有关跟踪信息, 请参阅 ADMIN COMMAND 'trace'。 • monitorstate - 当前监视功能状态, 显示为当前启用 SQL 监视功能的用户数目 (有关 SQL 监视信息, 请参阅 ADMIN COMMAND 'monitor')。 <p>如果所有用户都启用了 SQL 监视功能, 则值为 -1。</p> <ul style="list-style-type: none"> • openstate - 接受连接的当前状态Open 指数据库服务器接受新连接。 • nummerges - 合并次数。 • numlocks - 锁定次数。 • numcursors - 打开的游标数。 • numtransactions - 打开的事务数。 • memtotal - 分配内存的总量 (以字节计)。 • dbfreesize - 数据库中的剩余可用空间量 (KB)。 • dbpagesize - 数据库页面大小 (KB)。 • imdbsize - 内存表 (其中包括临时表和瞬态表) 以及那些表的索引所使用的空间量。返回值以千字节 (KB) 计, 格式为 VARCHAR。 • name - 服务器名称。您可以用 solidDB 启动选项 -n <i>name</i> 设置服务器名称。 • primarystarttime - 主角色的启动时间。 • secondarystarttime - 辅助角色的启动时间。 • dbconfigsize - 已配置的数据库大小 (MB), 用 IndexFile.FileSpec 参数设置。 • dbcreatetime dbcreationime - 数据库创建时间戳记。 • processsize psize - 系统级虚拟进程大小 (KB)。 <p>每个命令可以使用不止一个选项。值按所请求的顺序返回, 并且每个值各占一行。</p> <p>示例:</p> <pre>ADMIN COMMAND 'info dbsize logsize'; RC TEXT -- ---- 0 851968 0 573440 2 rows fetched.</pre>
ADMIN COMMAND 'logmessage <string>' 缩写: logmsg	将用户定义的 <string> 输出至消息日志 (solmsg.out)。

表 25. ADMIN COMMAND 语法和选项 (续)

选项语法	描述
ADMIN COMMAND 'logreader stop [all <partition_id>'] 缩写: lr	<p>该命令终止活动日志阅读器连接上的日志记录的传输。</p> <p>当发出该命令时, 活动日志阅读器应用程序会在访存 SYS_LOG 表的下一行时到达结果集的末尾 (SQLSTATE 0200, No data found)。</p> <p>如果使用了 LOGREADER STOP 或 LOGREADER STOP 格式, 则终止所有日志记录的传输。如果提供了 <PARTITION_ID>, 则该命令仅影响该分区上的日志阅读器操作。</p> <p>要再次访问该日志, 需要重新连接该应用程序。如果知道最后阅读位置, 则可恢复日志阅读而不丢失任何信息。如果没有指定日志位置就访问 SYS_LOG 表, 则从实时数据开始读起。</p> <p>要点: 不论日志中是否有记录在等待传输, 终止日志传输都将立即生效。</p> <p>当服务器在轻松耐久性方式 (缺省) 下运行时, 如果要在日志阅读器中查看这些记录, 那么在所有的记录写入日志前, 不要执行 LOGREADER STOP。在缺省日志记录设置下, 在最后一个写操作完成后等待 5 秒是安全的。</p>
ADMIN COMMAND 'makecp [-s]' 缩写: mcp	<p>创建检查点。</p> <p>仅拥有 SYS_ADMIN_ROLE 特权的用户可以执行该命令。</p> <p>缺省情况下, 检查点是异步的。如果指定选项 -s, 则此命令直到检查点完成后才会返回。</p>
ADMIN COMMAND 'memory' 缩写: mem	<p>返回服务器进程内存大小。报告的进程内存大小可能与操作系统报告的进程大小不同。</p>
ADMIN COMMAND 'messages [{ warnings errors }] [count]' 缩写: mes	<p>显示服务器消息。还可以定义可选的严重性和消息数。例如:</p> <p>ADMIN COMMAND 'messages warnings 100' 显示最近的 100 个警告。</p>
ADMIN COMMAND 'monitor { on off } [user { username userid }]' 缩写: mon	<p>打开和关闭服务器监视功能。</p> <p>当设置为 on 时, 用户活动和 SQL 调用将记录在 soltrace.out 文件中。</p>
ADMIN COMMAND 'netbackup [options] [DELETE_LOGS KEEP_LOGS] [connect connect str] [dir backup dir]' 缩写: nbak	<p>对数据库执行网络备份。此操作可以采用同步方式或异步方式 (缺省) 执行。使用 -s 选项指定同步操作。</p> <p>DELETE_LOGS 表示已删除源服务器中的备份日志文件。有时称其为完整备份。这是缺省值。</p> <p>KEEP_LOGS 表示在源服务器中保存备份的日志文件。有时称其为副本备份。使用 KEEP_LOGS 相当于将 General.NetBackupDeleteLog 参数设置为 no。</p> <p>用 General.NetBackupConnect 和 General.NetBackupDirectory 参数定义缺省连接字符串和缺省 netbackup 目录。</p> <p>与该命令一起输入的这些选项将覆盖配置文件中指定的值。</p> <p>目录定义相对于 solidDB 工作目录。</p>
ADMIN COMMAND 'netbackuplist' 缩写: nbls	<p>显示最近对数据库服务器执行的网络备份的状态列表。</p>
ADMIN COMMAND 'netstat' 缩写: net	<p>显示服务器设置和网络状态。</p>
ADMIN COMMAND 'notify user { username user id ALL } message' 缩写: not	<p>此命令将事件随事件标识 NOTIFY 一起发送给指定的用户。当语句超时长度不足以断开连接或更改事件注册时, 此标识用来取消事件等待线程。</p> <p>以下示例将通知消息发送给用户标识为 5 的用户; 然后, 该事件获取消息参数的值。</p> <p>ADMIN COMMAND 'notify user 5 Canceled by admin'</p>

表 25. ADMIN COMMAND 语法和选项 (续)

选项语法	描述
ADMIN COMMAND 'open' 缩写: ope	为新连接打开服务器; 允许建立新连接。
ADMIN COMMAND 'parameter [-r] [-t] [name=[[* value][temporary]]' 缩写: par	<p>显示和设置服务器参数值。</p> <p>如果您不使用任何选项运行该命令, 则会显示所有参数。</p> <p>输出可包含三列。例如:</p> <pre>0 PassThrough SqlPassthroughRead Force Conditional None</pre> <ul style="list-style-type: none"> • 第一列显示当前可能已经动态更改过的值 (Force)。 • 第二列显示启动时在 .ini 文件中设置的值。(Conditional) • 第三列显示出厂值。(None) • -r 表示仅返回当前参数值。 • -t 表示不会将已更改的值存储在 solid.ini 文件中 (和 temporary 相同)。 • name 可以是节名, 也可以是一个以节名开头的参数名 (section_name.parameter_name)。节名和参数名之间必须有句点。 • = [* value][temporary] <ul style="list-style-type: none"> - 如果指定带有星号 (*) 的参数值, 该参数将设置为它的出厂值。 - 如果未指定 value, 该参数设置为它的启动值。 - temporary 表示不会将已更改的值存储在 solid.ini 文件中。 <p>例如:</p> <ul style="list-style-type: none"> • 'parameter general' 显示 [General] 节中的所有参数。 • 'parameter general.readonly' 显示 [General] 节中的参数 Readonly。 • 'parameter com.trace=yes' 打开通信跟踪功能。 • 'parameter com.trace=' 将通信跟踪设置为其启动值。 • 'parameter com.trace=*' 将通信跟踪设置为其出厂值。
ADMIN COMMAND 'passthrough status' 缩写: pt	<p>提供了有关 SQL 传递连接的下列状态信息:</p> <ul style="list-style-type: none"> • NO REMOTE SERVER - 未定义远程服务器对象 • NOT CONNECTED - 未连接, 没有错误 • CONNECTED - 已连接 • LOGIN FAILED - 登录失败 • CONNECTION BROKEN - 连接断开

表 25. ADMIN COMMAND 语法和选项 (续)

选项语法	描述
ADMIN COMMAND 'perfmon [- c - r] [print_options] [name_prefix_list]' 缩写: pmon	<p>大约每隔一分钟就会返回过去几分钟的服务器性能计数器。绝大多数的值显示为每秒事件的平均数目。无法以每秒事件数显示的计数器（例如，数据库大小）将以绝对值显示。</p> <ul style="list-style-type: none"> • -c - 输出每张快照的实际计数器值。 • -r - 以原始方式输出计数器值，仅包括最新的计数器值，而不进行任何格式化。不输出计数器名称。如果实际监视操作由另外某个从服务器检索计数器值的外部程序执行，那么此选项非常有用。可使用 -xnames 选项检索计数器名称。 • print_options <ul style="list-style-type: none"> - -xtime - 输出时间（以秒计）。 - -xtimediff - 输出与上次 pmon 调用的时差（以毫秒计）。 - -xnames - 输出输出的列名。 - -xdiff - 指示与最新的 ADMIN COMMAND 'perfmon' 执行之差，而不是绝对值。 • name_prefix_list - 将输出对象限制为特定计数器类型，由计数器名称中的第一个单词指示。例如，要输出所有与“文件”相关的计数器，则 name_prefix_list 应该是 file。此外，您还可以指定多个前缀。 <p>以下示例返回所有信息：</p> <p>ADMIN COMMAND 'perfmon'</p> <p>以下示例返回所有名称以前缀 file 和 cache 开头的计数器的值。</p> <p>ADMIN COMMAND 'perfmon -c file cache'</p>
ADMIN COMMAND 'perfmon diff [start stop] [filename][interval]' 缩写: pmon diff	<p>启动能按指定的时间间隔将所有 perfmon 计数器输出到一个文件上的服务器任务。</p> <ul style="list-style-type: none"> • filename 是输出文件名称。性能数据以逗号分隔值格式输出；第一行包括计数器名称，且每个后续行包括每个取样时间的性能数据。 <p>缺省文件名称为 pmondiff.out。</p> <ul style="list-style-type: none"> • interval 是以毫秒为单位的时间间隔，以该时间间隔收集性能数据。 <p>缺省时间间隔为 1000 毫秒。</p> <p>以下命令启动以 500 毫秒的时间间隔将性能数据输出至 myd.csv 文件的任务。</p> <p>ADMIN COMMAND 'pmon diff start myd.csv 500'</p>
ADMIN COMMAND 'pid' 缩写: pid	<p>返回服务器进程标识。</p>
ADMIN COMMAND 'proctrace { on off } user username { procedure trigger table } entity_name' 缩写: ptrc	<p>此命令用于在存储过程和触发器中打开跟踪功能。</p> <p>username 是您要跟踪其过程调用（或触发器）的用户名。如果多个连接正在使用同一用户名，则将对来自所有那些连接的调用进行跟踪。此外，如果您正在使用高级复制功能，那么将不仅跟踪对副本数据库执行的调用，还将跟踪传播到主数据库并接着对主数据库执行的调用。</p> <p>entity_name 是要对其打开或关闭跟踪功能的过程、触发器或表的名称。如果您指定了过程或触发器名称，那么将对所指定过程或触发器中的每个语句生成输出。如果指定了表名，那么将对该表的所有触发器生成输出。仅当指定的用户名调用过程/触发器时，才会激活跟踪功能。</p> <p>有关 proctrace 的更多详细信息，请参阅《《IBM solidDB SQL 指南》》中的『用于存储过程和触发器的跟踪工具』一节。</p> <p>另请参阅 ADMIN COMMAND 'usertrace'.</p>

表 25. ADMIN COMMAND 语法和选项 (续)

选项语法	描述
ADMIN COMMAND 'protocols' 缩写: prot	返回可用通信协议的列表, 并且每种协议各占一行。 示例 (Windows 环境): ADMIN COMMAND 'protocols'; RC TEXT -- ---- 0 NmPipe np 0 TCP/IP tc 2 rows fetched.
ADMIN COMMAND 'report filename' 缩写: rep	将服务器信息的报告生成到 <i>filename</i> 定义的文件中。
ADMIN COMMAND 'runmerge' 缩写: rm	运行索引合并操作。
ADMIN COMMAND 'save parameters [filename]' 缩写: save	将当前配置参数值的集合保存至文件。如果未指定文件名, 那么将重写缺省的 solid.ini 文件。将在每个检查点隐式地执行此操作。
ADMIN COMMAND 'shutdown [force]' 缩写: sd	停止 solidDB。 如果使用了 force 选项, 那么将中止活动事务并强制用户断开连接。
ADMIN COMMAND 'sqlist top number_of_statements'	此命令输出当前正在运行的语句中运行时间最长的 SQL 语句的列表。此列表包含所选数目的语句。
ADMIN COMMAND 'startmerge' 缩写: sm	启动完成操作并等待此操作完成。
ADMIN COMMAND 'status' 缩写: sta	显示服务器统计信息。
ADMIN COMMAND 'status backup netbackup' 缩写: sta backup netbackup	显示最近启动的本地备份或网络备份的状态。状态可以是下列其中一项: <ul style="list-style-type: none"> • 如果上次备份成功或者尚未请求执行备份, 那么输出是 0 SUCCESS。 • 如果正在进行备份 (例如, 已启动但尚未就绪), 那么输出是 14003 ACTIVE。 • 如果即将完成备份, 那么输出是 14003 STOPPING。 • 如果上次备份失败, 那么输出是 <i>errorcode</i> ERROR, 其中 <i>errorcode</i> 显示失败原因。
ADMIN COMMAND 'throwout {username userid all}' 缩写: to	令所有用户或特定用户从 solidDB 中退出。要退出指定用户, 请指定用户名或用户标识作为自变量。要剔除所有用户, 请使用关键字 ALL 作为自变量。
ADMIN COMMAND 'tid' 缩写: tid	此命令返回服务器中当前用户线程的标识 (4 位代码)。

表 25. ADMIN COMMAND 语法和选项 (续)

选项语法	描述
<pre>ADMIN COMMAND 'trace { on off } sql est estplans rpc sync flowplans rexec batch logreader passthrough xa hac info <level> func proc all active' 缩写: tra</pre>	<p>打开或关闭服务器跟踪功能。</p> <p>缺省跟踪文件的名称是 soltrace.out。</p> <p>跟踪选项包括:</p> <ul style="list-style-type: none"> • sql - SQL 消息。 • est - SQL 估算程序信息 • estplans - SQL 执行计划 • rpc - 网络通信。 • sync - 同步消息 • flowplans - 与高级复制相关的 SQL 语句计划 • rexec - 远程过程调用信息 • batch - 后台作业和延迟过程调用信息 • logreader - 将以下信息记录到跟踪文件 soltrace.out 中。 <ul style="list-style-type: none"> - 日志阅读器读操作启动。 - 日志阅读器游标启动出错。共输出 14 种不同的错误情况。 - 日志阅读器读操作停止。 - 执行某些系统更改后, 发生异常的读停止。 - 有关所返回的日志记录数以及读进度的高级信息。 <p>每条信息都通过用户标识进行了标记, 以便对来自不同用户的操作进行分隔。</p> <ul style="list-style-type: none"> • passthrough - 提供有关 SQL 传递连接和 ODBC 驱动程序装入的跟踪信息, 如下所示: <ul style="list-style-type: none"> - 装入 ODBC 驱动程序: 装入的驱动程序名和状态 - 连接至后端的状态: 连接/重新连接/断开连接/中断 • xa - 分布式事务信息 • hac - 高可用性控制器 (HAC); 跟踪信息将输出到 HAC 工作目录上的 hactrace.out 中。 注: 要在 HAC 上启动跟踪, 您必须在 HAC 连接上发出命令。例如, 利用 solsql, 通过使用 solidhac.ini 配置文件中 HAController.Listen 参数定义的端口连接至 HAC。 • info <level> - SQL 执行跟踪 (级别可以为 0 到 8)。 • func - 函数执行信息 • proc - 存储过程执行信息 • all - 把 SQL 消息和网络通信消息都写入该跟踪文件中。 • active - 列示所有活动的跟踪
<pre>ADMIN COMMAND 'tracemessage <string>' 缩写: trcmsg</pre>	<p>将用户定义的 <string> 输出到跟踪消息日志 (soltrace.out) 中。</p>
<pre>ADMIN COMMAND 'userid' 缩写: uid</pre>	<p>返回当前连接的用户标识号。</p> <p>标识的生存期就是用户会话的生存期。在用户注销后, 此编号可以重复使用。</p> <pre>ADMIN COMMAND 'userid' RC TEXT -- ---- 0 8 1 rows fetched.</pre> <p>例如, 可在 ADMIN COMMAND 'throwout' 命令中使用用户标识来断开特定用户的连接。</p>

表 25. ADMIN COMMAND 语法和选项 (续)

选项语法	描述
<p>ADMIN COMMAND 'userlist [-l] [name id]' 缩写: ul</p>	<p>此命令显示当前登录到数据库的用户列表和每位用户不同的数据库操作和设置的信息。选项 -l (long) 显示更详细的输出。</p> <p>在未使用 -l 选项的情况下, 会显示以下信息: 用户名、用户标识、类型、机器标识、登录时间和应用程序信息 (如果可用)。</p> <p>在使用 -l 选项的情况下, 会显示以下信息:</p> <ul style="list-style-type: none"> • 标识 - 数据库中的用户会话识别号 (用户标识)。该用户标识的生存期就是该用户会话的生存期。在用户注销后, 此编号可以重复使用。 • Type - 客户机类型。可能的值是: <ul style="list-style-type: none"> - Java, 即使用 JDBC 的客户机。 - ODBC, 即使用 ODBC 的客户机。 - SQL, 即 solidDB SQL 编辑器 (solsql)。 • 机器 - 客户端计算机名称 (主机名) 及其 IP 地址 (如果可用)。 • 登录时间 - 客户计算机登录时间戳记 • 应用程序信息 - 客户计算机的 SOLAPPINFO (ODBC) 环境变量值, 或者 JDBC 连接属性 solid_appinfo 的值。 • 最后活动时间 - 客户机上次向服务器发送请求的时间。 • 自动落实 - 值 0 表示已关闭自动落实方式; 当前的事务将一直处于打开状态, 直到发出 COMMIT 或 ROLLBACK 语句为止。 <p>值 1 表示自动落实方式已打开; 每个语句都会自动落实。</p> <ul style="list-style-type: none"> • RPC 压缩 - 指示数据传输压缩功能是处于打开还是关闭状态。 • 透明故障转移 - 该字段指示是否正在使用透明故障转移 (TF) 功能 (HotStandby 配置)。因为 solidDB 工具不支持 TF, 所以当您使用 solsql 或 solcon 时只能在此字段中看到“No”值。 • 透明集群 - 透明集群指示是否已对此连接启用 HSB 中的负载均衡功能。 • 事务活跃 - 此字段指示连接中是否存在已打开的未落实事务, 值 1 表示存在, 值 0 表示不存在。如果已对连接打开自动落实功能, 那么在大多数情况下, 此值为 0。 • 事务持续时间 - 此字段指示当前打开的事务的持续时间。执行 COMMIT 或 ROLLBACK 之后, 此值将变为 0。 • 事务隔离 - 此字段指示事务的事务隔离级别。隔离级别确定如何使正在执行的事务中的数据对其他事务可视。 • 事务耐久性 - 此字段指示当前打开的事务的耐久性。 • 事务安全性 - 此字段指示当前打开的事务的安全性 (用 HotStandby.SafenessLevel 设置)。 • 事务自动落实 - 此字段指示当前打开的事务是否自动落实。如果对当前事务关闭事务自动落实方式 (值为 0), 那么当前事务将一直处于打开状态, 直到发出 COMMIT 或 ROLLBACK 语句为止。在此之后, 新语句将启动新的事务。 <p>如果对当前事务打开自动落实方式 (值为 1), 那么将自动落实每个语句。</p>

表 25. ADMIN COMMAND 语法和选项 (续)

选项语法	描述
<p>..续..</p> <pre>ADMIN COMMAND 'userlist [-1] [name id]' 缩写: ul</pre>	<ul style="list-style-type: none"> • 当前目录 - 指示当前目录名。 • 当前模式 - 指示当前模式名。 • <i>Sortgrouby</i> - 指示不存在有关结果组数的显式信息时, 如何执行 GROUP BY 语句。共有两个可能的值: <ul style="list-style-type: none"> - ADAPTIVE - 如果实际结果组的数目超出 GROUP BY 的中央内存数组所能容纳的行数, 那么对 GROUP BY 输入进行预先排序。 - STATIC - 每当 GROUP BY 列表至少包含两项内容时, 对 GROUP BY 输入进行预先排序。否则, 不对 GROUP BY 输入进行预先排序。 • 简单优化器规则 - 指示是否正在使用简单优化器规则 (SQL.SimpleOptimizerRules)。可能的值为 Yes/No/Default。 • 语句最大时间 - 指示特定于连接的语句最大执行时间 (以秒计)。此设置在您指定新的最大时间之前将一直有效。时间为零表示没有最大时间。这是缺省值。 • 锁定超时 - 指示通过 SET LOCK TIMEOUT 语句设置的超时。 • 乐观锁定超时 - 指示通过 SET OPTIMISTIC LOCK TIMEOUT 语句设置的超时。 • 空闲超时 - 指示通过 SET IDLE TIMEOUT 语句设置的超时。 • 连接路径范围 - 指示通过 SET SQL JOINPATHSPAN 语句设置的连接路径范围值。 • <i>RPC seqno</i> - 内部协议消息序号。 • <i>SQL sortarray</i> - 特定于用户的内部排序数组的大小。 • <i>SQL unionsfromors</i> - 此值指示最多可以将多少个 OR 运算符转换为 UNION。并集的执行速度更快, 但需要更多的内存。 • <i>EVENT QUEUE LENGTH</i> - 指示事件队列中已发出的事件数。 • 连接空闲超时 - 指示连接空闲超时设置 • <i>Stmt id</i> - 当前语句标识号。编号特定于会话, 每个不同的语句都会分配到一些编号。 • <i>Stmt 状态</i> - 内部语句执行状态。 • <i>Stmt 查行</i> - 在当前语句中检索或插入的行数。 • <i>Stmt 开始时间</i> - 当前语句开始日期和时间。 • <i>Stmt 最后一次活动时间</i> - 最新语句的时间戳记。 • <i>Stmt 持续时间</i> - 内部语句持续时间 (以秒计)。注: 此值与外部可视的语句等待时间无关。通常, 语句持续时间远大于等待时间。 • <i>Stmt SQL 字符串</i> - 当前 SQL 语句字符串。
<pre>ADMIN COMMAND 'usertrace { on off } user username { procedure trigger table } entity_name' 缩写: utrc</pre>	<p>此命令用于在存储过程和触发器中打开用户跟踪功能。此命令将对所指定过程或触发器中的每个 WRITETRACE 语句生成输出。</p> <ul style="list-style-type: none"> • <i>username</i> 是您要跟踪其过程调用 (或触发器) 的用户名。如果多个连接正在使用同一用户名, 那么将对来自所有那些连接的调用进行跟踪。此外, 如果您正在使用高级复制功能, 那么将不仅跟踪对副本数据库执行的调用, 还将跟踪传播到主数据库并接着对主数据库执行的调用。 • <i>entity_name</i> 是您要对其打开或关闭跟踪的过程、触发器或表的名称。如果指定了表名, 将对表上的所有触发器生成输出。仅当指定的用户调用过程/触发器时, 才会激活跟踪功能。 <p>有关 usertrace 的更多详细信息, 请参阅《《IBM solidDB SQL 指南》》中的『用于存储过程和触发器的跟踪工具』一节。</p> <p>另请参阅 ADMIN COMMAND 'proctrace'。</p>

表 25. ADMIN COMMAND 语法和选项 (续)

选项语法	描述
ADMIN COMMAND 'version' 缩写: ver	显示服务器版本信息以及与使用中的 solidDB 软件许可证相关的信息。

A.2 ADMIN EVENT

```
ADMIN EVENT 'command'
command_name ::=
    REGISTER { event_name [ , event_name ... ] | ALL } |
    UNREGISTER { event_name [ , event_name ... ] | ALL } |
    WAIT
event_name ::= the name of a system event
```

用法

ADMIN EVENT 语句是 solidDB 所特有的 SQL 扩展，它允许您向系统生成的事件注册并等待该事件，而不必编写和调用存储过程。您不能使用 ADMIN EVENT 等待用户事件。如果要等待用户事件，那么必须编写并调用存储过程。

当您使用外部事件存储过程时，必须显式注册并等待事件。在等待事件前，您必须向该事件注册。

提示: ADMIN EVENT 与 WAIT 在存储过程中的工作方式有所不同。在存储过程中，显式注册是可选的。

- 您用此命令无法向同步事件（以“SYNC_”开始）注册，这是因为 ADMIN EVENT 'wait' 命令无法返回变量结果集。而是，必须使用存储过程来处理同步事件。
- 一旦连接开始等待事件，该连接在该事件发出前将无法执行任何操作。
- 您可以向多个事件注册。在等待期间，无法指定要等待的事件的类型。此等待将一直持续到您接收到所注册的任何事件为止。
- ADMIN EVENT 命令未提供用于发出事件的选项。
- 要使用 ADMIN EVENT，您必须具有 administrator 特权或者获得 SYS_ADMIN_ROLE 角色。

示例

```
ADMIN EVENT 'register sys_event_hsbstateswitch';
ADMIN EVENT 'wait';
ADMIN EVENT 'unregister sys_event_hsbstateswitch';
```

系统发出该事件后，您将看到类似于以下的内容：

```
ENAME                POSTSRVTIME          UID NUMDATAINFO TEXTDATA
-----
SYS_EVENT_HSBSTATSWITCH 2003-10-28 18:10:14 -1 NULL          PRIMARY ACTIVE
```

1 rows fetched.

相关参考

第 163 页的 A.14, 『CREATE PROCEDURE』

相关信息

第 73 页的 3.5, 『事件』

A.3 ALTER REMOTE SERVER

```
ALTER REMOTE SERVER SET USERNAME <username> | PASSWORD <password>
```

用法

ALTER REMOTE SERVER 语句修改 SYS_SERVER 系统表中的后端登录信息。登录数据用于 Universal Cache 中的 SQL 传递。

在缺省情况下, 用户名和密码存储为大写字母。要区分大小写, 请在单引号中输入用户名和密码。

示例

```
ALTER REMOTE SERVER SET USERNAME dba PASSWORD dba
CREATE REMOTE SERVER PASSWORD 'PwD123'
```

另请参阅

第 183 页的 A.17, 『CREATE [OR REPLACE] REMOTE SERVER』

第 203 页的 A.36, 『DROP REMOTE SERVER』

A.4 ALTER TABLE

```
ALTER TABLE base_table_name
{
  ADD [COLUMN] column_identifier data_type
  [DEFAULT literal | NULL] [NOT NULL] |
  ADD CONSTRAINT constraint_name dynamic_table_constraint |
  DROP CONSTRAINT constraint_name |
  ALTER [ COLUMN ] column_name
  {DROP DEFAULT | {SET DEFAULT literal | NULL} } |
  {{ADD | DROP} NOT NULL }
  DROP [COLUMN] column_identifier |
  RENAME [COLUMN]
  column_identifier column_identifier |
  MODIFY [COLUMN] column_identifier data-type |
  MODIFY SCHEMA schema_name |
  SET HISTORY COLUMNS (c1, c2, c3) |
  SET {OPTIMISTIC | PESSIMISTIC} |
  SET STORE {DISK | MEMORY} |
  SET [NO]SYNCHISTORY |
  SET TABLE NAME new_base_table_name
}
```

其中:

```
dynamic_table_constraint::=
{FOREIGN KEY (column_identifier [, column_identifier] ...)
REFERENCES table_name [(column_identifier [, column_identifier] ...) ]
[referential_triggered_action] |
CHECK (check_condition) | UNIQUE (column_identifier)
```

```
referential_triggered_action ::=
    ON {UPDATE | DELETE} {CASCADE | SET NULL | SET DEFAULT |
    RESTRICT | NO ACTION}
```

用法

以下情况您可以使用 ALTER TABLE 语句：

- 修改表的结构；添加、除去、修改和重命名列。

服务器允许您使用 ALTER TABLE 命令来更改列的宽度。您随时可以增大列宽（即，无论表是否为空 [未包含任何行]，都可以增大列宽）。但是，表不为空时，ALTER TABLE 命令不允许减小列宽；您只能减小空表的列宽。

注：无法删除作为唯一或主键组成部分的列。

- 将表设置为使用乐观或悲观并行控制。

可以使用 ALTER TABLE *base_table_name* SET {OPTIMISTIC|PESSIMISTIC} 语句将基于磁盘的表（D 表）设置为乐观表或悲观表。缺省情况下，所有 D 表都是乐观表。M 表始终是悲观表。可以用 **General.Pessimistic** 参数来设置 D 表在数据库范围内的缺省值。

- 更改表的模式和所有者。

可以使用 ALTER TABLE *base_table_name* MODIFY SCHEMA *schema_name* 语句来更改表的所有者。此语句将所有权限（其中包括创建者权限）授予新的所有者。旧所有者对表具有的访问权（创建者权限除外）将保留。

- 更改是在内存中还是在磁盘中存储表。

可以将基于磁盘的表更改为基于内存的表，反之亦然。您只能对空表执行此操作。如果尝试将表更改为它已使用的存储方式（例如，如果尝试将内存表更改为使用内存存储方式），那么该命令将没有任何效果，并且不会发出错误消息。

- 更改引用约束设置。

有关更多信息，请参阅第 93 页的 4.5，『引用完整性』。

- 为表定义同步设置。

有关更多信息，请参阅 A.4.1，『ALTER TABLE ... SET HISTORY COLUMNS』和第 147 页的 A.4.2，『ALTER TABLE ... SET SYNCHISTORY』。

示例

```
ALTER TABLE table1 ADD x INTEGER;
ALTER TABLE table1 RENAME COLUMN old_name new_name;
ALTER TABLE table1 MODIFY COLUMN xyz SMALLINT;
ALTER TABLE table1 DROP COLUMN xyz;
ALTER TABLE table1 SET STORE MEMORY;
ALTER TABLE table1 SET PESSIMISTIC;
ALTER TABLE table2 ADD COLUMN col_new CHAR(8) DEFAULT 'VACANT' NOT NULL;
ALTER TABLE table2 ALTER COLUMN col_new SET DEFAULT 'EMPTY';
ALTER TABLE table2 ALTER COLUMN col_new DROP DEFAULT;
ALTER TABLE dept_tab1 ADD CONSTRAINT div_check CHECK(division_id < 12);
ALTER TABLE dept_tab1 DROP CONSTRAINT div_check;
```

A.4.1 ALTER TABLE ... SET HISTORY COLUMNS

```
ALTER TABLE table_name SET HISTORY COLUMNS ( col1, col2, colN ...)
```

用法

为了进一步优化同步历史记录过程，在为同步历史记录设置表之后，可以使用 SET HISTORY COLUMNS 语句来指定主数据库及其相应同步表中的哪个列更新将在历史记录表中创建条目。如果未使用此语句来指定特定的列，那么在更新相应同步表时，主数据库中对所有列执行的所有更新操作都会在历史记录表中创建新条目。通常，对于用于搜索条件或者用于进行连接的列，我们建议您使用 ALTER TABLE ... SET HISTORY COLUMNS。

用于主数据库

在主数据库中使用 SET SYNCHISTORY 和 SET HISTORY COLUMNS，以便对表启用增量发布功能。

用于副本数据库

在副本数据库中使用 SET SYNCHISTORY 和 SET HISTORY COLUMNS，以便对表启用增量 REFRESH 功能。

注：必须先执行 ALTER TABLE ... SET SYNCHISTORY 语句，这样 ALTER TABLE ... SET HISTORY COLUMNS 才有可能成功。执行 ALTER TABLE ... SET NOSYNCHISTORY 还将撤销 ALTER TABLE ... SET HISTORY COLUMNS 的效果。

示例

```
ALTER TABLE myLargeTable SET HISTORY COLUMNS (accountid);
```

返回值

表 26. ALTER TABLE SET HISTORY COLUMNS 返回值

错误代码	描述
13047	无权执行操作。
13100	表方式组合不合法
13134	表不是基本表
25038	表由发布 <i>publication_name</i> 引用；不允许执行删除或更改操作
25039	表由对发布 <i>publication_name</i> 的预订引用；不允许执行删除或更改操作。

相关参考

A.4.2, 『ALTER TABLE ... SET SYNCHISTORY』

A.4.2 ALTER TABLE ... SET SYNCHISTORY

```
ALTER TABLE table_name SET {SYNCHISTORY | NOSYNCHISTORY}
```

用法

SET SYNCHISTORY / NOSYNCHISTORY 子句指示服务器将 solidDB 高级复制体系结构的增量发布机制用于指定的表。缺省情况下，SYNCHISTORY 处于关闭状态。对指

定的表将此语句设置为 SYNCHISTORY 时，将自动创建一个影子表以便存储主表中已更新或删除的行的旧版本。可将此影子表称为同步历史记录表，或者简称为历史记录表。

当副本数据库根据主数据库中的发布执行增量 REFRESH 时，将引用历史记录表中的数据。例如，如果您从主表中删除包含 Ms. Smith 的电话账单的记录，那么记录的备份存储到同步历史记录表中。副本数据库执行刷新时，主数据库将检查历史记录表并告知副本数据库已删除 Smith 女士的记录。然后，副本数据库也可以删除该记录。如果已删除或更改的记录所占的百分比相当小，那么相对于从主数据库下载整个表，增量更新的速度更快。如果用户执行完全刷新而不是增量刷新，将不会使用历史记录表。在这种情况下，仅仅将主数据库中的表数据复制到副本数据库。

在没有任何副本数据库需要该数据来执行 REFRESH 请求时数据库会自动删除版本化数据。

您必须使用此命令来打开同步历史记录，这样一个表才能参与主数据库/副本数据库同步。您甚至可以对目前仍有数据存在的表使用此命令；但是，仅当现有发布未引用指定的表时，才能使用 ALTER TABLE SET SYNCHISTORY。

必须在主数据库和副本数据库中的表中都指定 SET SYNCHISTORY。

要检查是否已对某个表打开 SYNCHISTORY，请检查 SYS_TABLEMODES 系统表。MODE 列包含 SYNCHISTORY 信息。

例如，使用以下查询：

```
SELECT mode
FROM SYS_TABLES, SYS_TABLEMODES
WHERE table_name = 'MY_TABLE' AND SYS_TABLEMODES.ID = SYS_TABLES.ID;
MODE
----
SYNCHISTORY
1 rows fetched.
```

SYS_TABLEMODES 只显示已显式设置方式的表的方式。SYS_TABLEMODES 不显示处于缺省方式的表的方式。如果未对该表设置 SYNCHISTORY（或 NOSYNCHISTORY），那么该查询将返回一个空的结果集。

用于主数据库

在主数据库中使用 SET SYNCHISTORY，以便对表启用增量发布功能。

用于副本数据库

在副本数据库中使用 SET SYNCHISTORY，以便对表启用增量 REFRESH 功能。

注：如果副本数据库是只读的（未对发布的所复制部分进行更改），那么不需要执行 ALTER TABLE ... SET SYNCHISTORY 语句。然而，必须使用以下命令同时将 SYS_SYNC_KEEPLOCALCHANGES 参数设置为 yes：

```
set sync parameter SYS_SYNC_KEEPLOCALCHANGES 'Yes';
```

示例

```
ALTER TABLE myLargeTable SET SYNCHISTORY;
ALTER TABLE myVerySmallTable SET NOSYNCHISTORY;
```

返回值

表 27. ALTER TABLE SET SYNCHISTORY 返回值

错误代码	描述
13047	无权执行操作。
13100	表方式组合不合法
13134	表不是基本表
25038	表由发布 <i>publication_name</i> 引用; 不允许执行删除或更改操作
25039	表由对发布 <i>publication_name</i> 的预订引用; 不允许执行删除或更改操作。

相关参考

第 146 页的 A.4.1, 『ALTER TABLE ... SET HISTORY COLUMNS』

A.5 ALTER TRIGGER

```
ALTER TRIGGER trigger_name_attr SET {ENABLED | DISABLED}  
trigger_name_attr ::= [ catalog_name. [ schema_name. ] ] trigger_name
```

用法

ALTER TRIGGER 语句修改触发器属性。有效属性是 ENABLED 和 DISABLED（分别用于启用和禁用触发器）。

ALTER TRIGGER DISABLED 语句将导致 solidDB 在发出激活的 DML 语句时忽略触发器。使用此命令，您还可以启用当前处于不活动状态的触发器或者禁用当前处于活动状态的触发器。

要更改表的触发器，您必须是该表的所有者或者具有 DBA 权限的用户。

示例

```
ALTER TRIGGER trig_on_employee SET ENABLED;
```

A.6 ALTER USER

```
ALTER USER username IDENTIFIED BY new_password
```

用法

ALTER USER 语句修改指定用户的密码。只有具有管理员权限的用户才可以更改其他用户的密码。

示例

```
ALTER USER MANAGER IDENTIFIED BY O2CPTG;
```

相关任务

第 85 页的『更改密码』

使用 ALTER USER 命令可更改您的密码。管理员也可以更改其他用户的密码。

第 87 页的『更改管理员的用户名和密码』

可以使用 ALTER USER 命令更改数据库系统管理员的密码。不能使用 ALTER USER 命令更改管理员用户名；可以将 SYS_ADMIN_ROLE 授予另一位用户，然后删除原始管理员用户帐户来更改用户名。

A.6.1 ALTER USER (副本)

```
ALTER USER replica_user SET MASTER master_name USER user_specification
```

其中:

```
user_specification ::= { master_user IDENTIFIED BY master_password | NONE }
```

```
ALTER USER username SET {PUBLIC | PRIVATE}
```

用法

ALTER USER *replica_user* ... 语句将副本数据库用户标识映射到指定的主数据库用户标识。始终是将副本数据库用户标识映射到主数据库用户标识。

用户标识的映射用于在多主数据库或多层同步环境中实现安全性。在此类环境中，难以在地理上相互分隔的独立数据库中维护相同的用户名和密码。

只有具有 DBA 权限或 SYS_SYNC_ADMIN_ROLE 角色的用户才能映射用户。要实现映射，管理员必须知道主数据库用户名和密码。如果指定 NONE，那么将除去该映射。

所有副本数据库都负责预订 SYNC_CONFIG 系统发布以更新用户信息。在此过程中，将使用 MESSAGE APPEND SYNC_CONFIG 命令将公用主数据库用户名和密码下载到副本数据库。通过将副本数据库用户标识与主数据库用户标识映射，系统根据登录到副本数据库的本地用户标识来确定当前活动的主数据库用户。如果在 SYNC_CONFIG 装入期间，系统未检测到映射，则该系统将通过主数据库和副本数据库中匹配的用户标识和密码来确定当前活动的主数据库用户。

有关通过映射来实现安全性的更多详细信息，请参阅 *IBM solidDB Advanced Replication User Guide* 中的[通过访问权限和角色实现安全性](#)。

您也可以限制在 SYNC_CONFIG 装入期间下载到副本数据库的主数据库用户帐户。请使用以下命令将用户状态更改为私有用户或公共用户来实现该目的:

```
ALTER USER username SET PRIVATE | PUBLIC
```

缺省值为 PUBLIC。如果对一个用户设置了 PRIVATE 选项，则该用户的信息将不会包括在 SYNC_CONFIG 预订中，即使在 SYNC_CONFIG 请求中指定那些信息亦如此。只有具有 DBA 权限或 SYS_SYNC_ADMIN_ROLE 角色的用户才能更改用户的状态。

这允许管理员确保不会把具有管理权限的用户标识发送到副本数据库。例如，为了确保安全，管理员可能想确保 DBA 密码永远不公开。

用于主数据库

在主数据库中，您可以将用户标识设置为 PUBLIC 或 PRIVATE。

用于副本数据库

在副本数据库中，您可以将副本数据库用户标识映射到主数据库用户标识上。

示例

以下语句使用 DBA 用户的密码将副本数据库用户标识 *smith_1* 映射到主数据库用户标识 *dba*。

```
ALTER USER SMITH_1 SET MASTER MASTER_1 USER DBA IDENTIFIED BY DBA
```

以下语句将主数据库的用户 *admin1* 的状态设置为 PRIVATE，这定义了不应将 *admin1* 的用户帐户下载到任何副本数据库上。

```
ALTER USER admin1 SET PRIVATE;
```

以下语句将主数据库的用户 *salesman* 的状态设置为 PUBLIC，这定义了应将 *salesman* 的用户帐户下载到所有的副本数据库上。

```
ALTER USER salesman SET PUBLIC;
```

返回码

表 28. ALTER USER 返回值

错误代码	描述
13047	无权执行操作。
13060	找不到用户名 <i>xxx</i> 。
25020	此数据库不是主数据库。
25062	用户 <i>user_id</i> 未映射到主数据库用户 <i>user_id</i> 。
25063	用户 <i>user_id</i> 已映射到主数据库用户 <i>user_id</i> 。

A.7 CALL

```
CALL procedure_name [(parameter [, parameter ...])] [AT node-def]
```

其中：

```
node-def ::= DEFAULT | <replica name> | <master name>
```

用法

CALL 语句用于调用存储过程。AT node_ref 子句仅受使用远程存储过程的高级副本配置支持；它可以从主数据库节点对它的其中一个副本数据库节点执行此调用，反之亦然。

过程调用以同步方式执行；它将在该调用执行完毕后返回。

注： 如果使用 START AFTER COMMIT（例如 START AFTER COMMIT UNIQUE CALL FOO AT REPLIC1）来执行过程调用，那么过程调用将在后台以异步方式执行。这是 START AFTER COMMIT 语句的功能，而非过程调用的功能。

与远程存储过程一起使用

AT node_ref 子句仅能用于高级副本配置中的远程存储过程。仅当从主数据库节点对它的其中一个副本数据库节点或者从其中一个副本数据库节点对主控节点执行此调用时，此调用才有效。

DEFAULT 表示使用当前副本上下文。仅当使用带有 FOR EACH REPLICA 选项的 START AFTER COMMIT 语句在后台启动过程调用时，才会定义当前副本上下文。如果未设置缺省节点，将返回错误 Default node not defined。您可以在存储过程中以及在使用 START AFTER COMMIT 启动的语句中使用 DEFAULT。

远程存储过程无法返回结果集；它只能返回错误代码。

单一 CALL 语句只能调用单一节点上的单一过程。如果要调用单一节点上的多个过程，那么必须执行多个 CALL 语句。如果您要在多个节点上执行同一过程（即过程名相同），您需要使用 START AFTER COMMIT FOR EACH REPLICA 语句或执行多次调用。

事务

执行远程过程调用（无论此调用是否由 START AFTER COMMIT 启动）的事务与调用该过程的事务相互独立。调用者无法回滚或落实远程过程调用。在所调用节点中执行的过程负责发出它自己的落实或回滚语句。

此外，远程过程调用并不持久。如果服务器在发出远程过程调用后立即关闭，那么该调用将丢失。在恢复阶段，不会执行该调用。

远程过程的返回值

调用远程存储过程时，您无法获取所返回的完整结果集。您只能获取该存储过程的返回值（单个值）或错误代码。

注：如果该远程过程在后台执行（使用 START AFTER COMMIT），那么不会将任何返回值返回给用户，甚至不会返回错误代码。

访问权

有关更多信息，请参阅A.7.1，『执行远程存储过程调用时的访问权』。

示例

```
CALL proctest;
CALL proctest('some string', 14);
CALL remote_proc AT replica2;
CALL RemoteProc(?,?) AT MyReplica1;
```

START AFTER COMMIT FOR EACH REPLICA 语句的示例:

```
START AFTER COMMIT FOR EACH REPLICA WHERE NAME LIKE 'REPLICA%'
UNIQUE CALL MYPROC AT DEFAULT
```

A.7.1 执行远程存储过程调用时的访问权

以远程方式调用存储过程时，您必须考虑访问权 - 即，调用者在远程服务器上是否有权执行此过程？

情况 1: 如果已使用 SET SYNC USER 命令来设置同步用户。

调用者将“同步用户”的用户名和密码发送到远程服务器，远程服务器尝试使用该用户名和密码来执行该过程。在这种情况下，该用户名和密码在远程服务器（即，要在其上执行该存储过程的服务器）中必须存在，并且该用户必须对数据库以及所调用的过程具有适当的访问权。

情况 2: 未设置同步用户：

调用者在调用远程过程时，将向远程服务器发送下列信息：

如果调用者是主数据库，而远程服务器是副本数据库（M → R）：

- 主数据库的名称（SYS_SYNC_REPLICAS.MASTER_NAME）。
- 副本数据库标识（SYS_SYNC_REPLICAS.ID）。
- 调用者的用户名。
- 调用者的用户标识。

如果调用者是副本数据库，并且远程过程是主数据库（R → M）：

- 主数据库的名称（SYS_SYNC_MASTERS.NAME）。
- 副本数据库标识（SYS_SYNC_MASTERS.REPLICA_ID）。
- 主数据库用户标识（副本数据库刷新数据时使用的用户标识。在 SYS_SYNC_USERS 表中，必须存在从本地副本数据库用户到主数据库用户的映射。）

在调用的节点上，将执行下列操作：

如果远程节点是副本数据库（M → R）：

- 根据从调用者接收到的主数据库名称，从 SYS_SYNC_MASTERS 表中获取主数据库标识（主数据库本身并不知道它在副本数据库中的标识）。从 SYS_SYNC_USERMAPS 表中，根据主数据库用户名和主数据库标识来获取副本数据库用户标识。请选择第一个对该过程具有访问权的用户。
- 如果在 SYS_SYNC_USERMAPS 中没有匹配的行，那么根据从调用者接收到的主数据库标识和主数据库用户名从 SYS_SYNC_USERS 表中获取名称（NAME）和密码（PASSWD），然后尝试使用该名称和密码来执行该过程。

如果远程节点是主数据库（R → M）

- 尝试使用从副本数据库接收到的用户标识来执行该过程。

如果副本数据库允许从任何主数据库执行调用，那么它应该在 solid.ini 文件中定义它自己的连接字符串信息，例如：

```
[Synchronizer]
ConnectStrForMaster=tcp replicahost 1316
```

副本数据库将任何消息转发到主数据库时，都会自动地将连接字符串发送到主数据库。当主数据库从副本数据库接收到连接字符串时，如果新值与先前值不同，那么新值将替换先前值。

主数据库可以使用以下语句对副本数据库设置连接字符串（如果副本数据库尚未执行任何消息传递操作，并且主数据库需要调用副本数据库而且知道连接字符串已更改）：

```
SET SYNC CONNECT <connect-info> TO REPLICA <replica-name>
```

A.8 COMMIT WORK

```
COMMIT [WORK]
```

用法

COMMIT 或 COMMIT WORK 语句在数据库中进行永久性更改，从而终止事务。要废弃更改，请使用 ROLLBACK 命令。

如果您未显式地落实事务，且该程序未自动为您落实事务，将回滚该事务。例如，缺省情况下，solidDB SQL 编辑器 (**solsql**) 不落实事务。

相关参考

第 252 页的 A.75, 『ROLLBACK WORK』

A.9 CREATE CATALOG

```
CREATE CATALOG catalog_name
```

用法

在 solidDB 数据库中，CREATE CATALOG 创建一个新的目录。在 solidDB 中，对目录的使用是 SQL 标准的扩展。

目录使您能够以逻辑方式对数据库进行分区以便对数据进行组织，从而满足业务或应用程序的需求。

solidDB 数据库文件可以包含不止一个逻辑数据库。每个逻辑数据库都是一组完整而独立的数据库对象，例如表、索引、触发器、存储过程等。每个逻辑数据库都是作为数据库目录实现的。因此，solidDB 可以有一个或多个目录。

在创建新数据库或者将旧数据库转换为新格式时，将提示您输入缺省目录名。这个缺省目录名用于向后兼容 V3.x 以前的 solidDB 数据库。

一个目录可以具有零个或多个模式名。缺省模式名是创建该目录的用户的用户标识。

一个模式可以具有零个或多个数据库对象名。数据库对象可以由模式或用户标识限定。

目录名用于对数据库对象名进行限定。

要点: 目录名不能包含空格。

在所有 DML 语句中，都可以对数据库对象名进行限定，如下所示：

```
catalog_name.schema_name.database_object
```

或者

```
catalog_name.user_id.database_object
```

如果使用目录名，还必须使用模式名。反之则不然；如果已通过执行适当的 SET CATALOG 语句来指定缺省目录，那么可以在不使用目录名的情况下使用模式名。

```
catalog_name.database_object -- 不合法
schema_name.database_object -- 合法
```

只有具有 DBA 权限的用户 (SYS_ADMIN_ROLE) 才能为数据库创建目录。

创建目录并不会自动使该目录成为当前缺省目录。如果您已创建新目录并且要让后续命令在该目录中执行,还必须执行第 259 页的 A.79.3,『SET CATALOG』语句。例如:

```
CREATE CATALOG MyCatalog;
CREATE SCHEMA smith; -- 不在 MyCatalog 中
SET CATALOG MyCatalog;
CREATE SCHEMA jones; -- 在 MyCatalog 中
```

要使用模式,必须在创建数据库对象名之前使用第 184 页的 A.19,『CREATE SCHEMA』语句创建模式名。但是,可以在没有模式名的情况下创建数据库对象名。在这种情况下,数据库对象仅由 user_id 限定。

在程序中,可以使用第 199 页的 A.28,『DROP CATALOG』语句设置目录上下文。

可以使用第 199 页的 A.28,『DROP CATALOG』从数据库中删除目录。删除目录名时,必须在删除该目录之前删除所有与该目录名相关联的对象。

目录名的解析规则

- 标准名称 (catalog_name.schema_name.database_object_name) 不要求进行任何名称解析,但需要验证。
- 如果未使用 SET CATALOG 来设置目录上下文,那么始终在使用缺省目录名作为目录名的情况下解析所有数据库对象名。在这种情况下,将使用模式名解析规则来解析数据库对象名。有关此类规则的更多信息,请参阅第 184 页的 A.19,『CREATE SCHEMA』。
- 如果已设置目录上下文,并且在该上下文中无法使用 catalog_name 来解析目录名,那么 database_object_name 解析将失败。
- 要访问数据库系统目录,用户不需要知道系统目录名。用户可以指定 ""_SYSTEM.table"。solidDB 会将用作目录名的空字符串转换为缺省目录名。solidDB 还能够将 _SYSTEM 模式自动解析为系统目录,即使未提供目录名亦如此。

示例

```
CREATE CATALOG C;
SET CATALOG C;
CREATE SCHEMA S;
SET SCHEMA S;
CREATE TABLE T (i INTEGER);
SELECT * FROM T;
-- the name T is resolved to C.S.T

-- Assume the userid is SMITH
CREATE CATALOG C;
SET CATALOG C;
CREATE TABLE T (i INTEGER);
SELECT * FROM T;
--The name T is resolved to C.SMITH.T

-- Assume there is no Catalog context set.
-- Meaning the default catalog name is BASE or the setting
-- of the base catalog.
CREATE SCHEMA S;
SET SCHEMA S;
```

```

CREATE TABLE T (i INTEGER);
SELECT * FROM T;
--The name T is resolved to <BASE>.S.T

CREATE CATALOG C1;
SET CATALOG C1;
CREATE SCHEMA S1;
SET SCHEMA S1;
CREATE TABLE T1 (c1 INTEGER);

CREATE CATALOG C2;
SET CATALOG C2;
CREATE SCHEMA S2;
SET SCHEMA S2;
CREATE TABLE T1 (c2 INTEGER)

SET CATALOG BASE;
SET SCHEMA USER;
SELECT * FROM T1;
-- This select will give an error as it
-- cannot resolve the T1.

```

A.10 CREATE EVENT

```

CREATE EVENT event_name
  [(parameter_name datatype
    [parameter_name data_type...])]

```

其中:

event_name 是任何能够识别该事件的用户定义的字母数字字符串

parameter_name 参数的用户定义名

data_type 是 SQL 数据类型

用法

CREATE EVENT 语句在 solidDB 数据库中创建用户定义的事件对象。

在您创建了一个用户定义的事件后，您必须使用存储过程注册、发布及等待事件。

示例

```

CREATE EVENT ALERT1(I INTEGER, C CHAR(4));

```

相关参考

第 144 页的 A.2, 『ADMIN EVENT』

第 200 页的 A.29, 『DROP EVENT』

第 246 页的 A.69, 『POST EVENT』

第 249 页的 A.71, 『REGISTER EVENT』

第 280 页的 A.87.2, 『data_type』

相关信息

第 73 页的 3.5, 『事件』

A.11 CREATE FUNCTION

```
CREATE FUNCTION function_name [(parameter_definition
    [, parameter_definition ...])]
    RETURN[S] data_type
    [SQL_data_access_indication]
    [LANGUAGE SQL]
    BEGIN
        function_body
    END
```

parameter_definition::= *parameter_name*
 data_type[=*literal*]

SQL_data_access_indication::= CONTAINS SQL | READS SQL DATA | NO SQL

用法

CREATE FUNCTION 语句创建用户定义的存储函数。存储函数可用 solidDB 专有 SQL 过程语言编写。

通过编写与内置函数行为方式相同的函数，您可以使用存储 SQL 函数扩展服务器功能。

存储函数接受字面值、变量和变量标记符 ("?")或列名作为输入参数。不支持输出参数。

参数

function_name

识别函数名。函数名也可包括目录名和模式：

[catalog_name[.schema].]function_name

名称必须是有效的 SQL 标识。函数名在模式内必须唯一。尽管对其没有限制，但是不应给用户定义的函数取与内置函数相同的名称。

parameter_definition::= *parameter_name data_type*[=*literal*]

识别该函数的输入参数的数值，然后指定其名称和数据类型，以及每个参数可选的缺省值。必须为该函数期望接收的每个参数指定列表中的一个入口。共有 90 个参数可供指定。

示例:

```
CREATE FUNCTION week_number (s TIMESTAMP)
...
```

您可以创建没有参数的函数。例如:

```
CREATE FUNCTION howdy() RETURNS VARCHAR20
...
```


data_type 必须是有效的 solidDB 第 280 页的 A.87.2, 『data_type』。
literal 定义参数的缺省值。该值必须是字面值。

存储函数接受字面值、变量和变量标记符 ("?") 或列名作为输入参数。如果使用了列名, 会从当前行的变量实例中检索该列的值 (例如在 WHERE 子句中)。不支持输出参数。

RETURN[S] *data_type*

标识函数输出的数据类型。

data_type 必须是有效的 solidDB 第 280 页的 A.87.2, 『data_type』。

SQL_data_access_indication

定义该函数是否发布任何 SQL 语句, 如果发布, 请指明类型。

CONTAINS SQL

指示不读取或不修改 SQL 数据的 SQL 语句可通过该函数执行。

READS SQL DATA

指示该函数仅执行只读的 SQL 语句。

NO SQL

指示该函数不能执行任何 SQL 语句。

注: 系统不会针对所声明的数据访问指示来检查函数内容的有效性。

LANGUAGE SQL

该可选子句指示 CREATE FUNCTION 语句正用于注册用 solidDB SQL 过程语言编写的新函数。有关如何创建外部 C 编程语言编写的函数的详细信息, 请参阅第 159 页的 A.12, 『CREATE FUNCTION (外部)』。

function_body

函数体必须是有效的 solidDB SQL 过程语句。有关更多详细信息, 请参阅第 163 页的 A.14, 『CREATE PROCEDURE』。

要了解有效的表达式, 请参阅第 280 页的 A.87.3, 『表达式』。

返回值

- SQL_ERROR
- SQL_SUCCESS

限制

- 返回值不能是复杂表达式; 它只能是一个变量或一个字面值。
- 不允许在函数中使用 COMMIT。
- 不支持 RETURNS NULL ON NULL INPUT。如果任何实参是 NULL, 那么将自动强制使用 NULL 函数值。

示例 1: 创建并使用简单函数 hello

创建该函数:

```
CREATE FUNCTION hello (s VARCHAR(20))
RETURNS VARCHAR(50)
BEGIN
    RETURN 'Hello';
END
```

使用该函数:

```
SELECT hello('world');

HELLO('world')
-----
Hello
```

示例 2: 创建函数并使用它返回周数

创建该函数:

```
CREATE FUNCTION week_number (s TIMESTAMP)
RETURNS INTEGER
BEGIN
    DECLARE week INTEGER;
    week := floor(( dayofyear(s) - dayofweek(s) + 7)/7);
    RETURN week;
END
```

使用该函数:

```
SELECT quota, week_number(sales_date)AS week, week_number(now())
AS current_week
FROM sales ORDER BY quota DESC LIMIT 1;

QUOTA WEEK    CURRENT_WEEK
-----
467    23          43

SELECT quota FROM sales
WHERE current_week(sales_date) = 23 ORDER BY quota DESC LIMIT 1;

QUOTA
-----
467

INSERT INTO report VALUES (?, week_number(now()), week_number(?));
```

相关参考

A.12, 『CREATE FUNCTION (外部)』

第 200 页的 A.30, 『DROP FUNCTION』

第 214 页的 A.50, 『GRANT』

A.12 CREATE FUNCTION (外部)

```
CREATE FUNCTION function_name [(parameter_definition
[, parameter_definition ...])]
RETURN[S] data_type
[SQL_data_access_indication]
LANGUAGE C EXTERNAL NAME 'external_function_library_identifier'
```

parameter_definition::= *parameter_name data_type*[=literal]

SQL_data_access_indication::= CONTAINS SQL | READS SQL DATA | NO SQL

用法

TCREATE FUNCTION (外部) 语句注册用户定义的外部存储函数。可用 C 编程语言编写 solidDB 外部存储函数。

通过编写 C 程序, 您可以用外部函数扩展服务器的功能。

外部函数的用法与内置函数或用户定义的函数的用法相同。solidDB 使用标准动态库界面（Windows 中的 DDL、Linux 和 UNIX 中的共享库）来访问外部函数。系统会把线程控制传递给外部库例程，直至其返回。此外，外部函数的数据库界面与 ODBC 应用程序的相似；唯一不同的是，它的连接句柄 (hdbc) 会传递到外部例程，并且，它能立即开始使用数据库界面而不用建立连接。

如果要允许使用外部存储函数，您必须使用共享内存访问 (SMA) 或链接库访问 (LLA) 将您的应用程序链接至 solidDB。

存储函数接受字面值、变量和变量标记符 ("?")或列名作为输入参数。不支持输出参数。

外部函数是标量函数，在每次调用时都返回单一值。

参数

function_name

识别函数名。函数名也可包括目录名和模式：

[catalog_name[.schema].]function_name

名称必须是有效的 SQL 标识。函数名在模式内必须唯一。尽管对其没有限制，但是不应给用户定义的函数取与内置函数相同的名称。

parameter_definition:= parameter_name data_type [=literal]

识别该函数的输入参数的数值，然后指定其名称和数据类型，以及每个参数可选的缺省值。必须为该函数期望接收的每个参数指定列表中的一个入口。共有 90 个参数可供指定。

示例：

```
CREATE FUNCTION week_number (s TIMESTAMP)
...
```

您可以创建没有参数的函数。例如：

```
CREATE FUNCTION howdy() RETURNS VARCHAR20
...
```

data_type 必须是有效的 solidDB 第 280 页的 A.87.2，『*data_type*』。

literal 定义参数的缺省值。该值必须是字面值。

存储函数接受字面值、变量和变量标记符 ("?")或列名作为输入参数。如果使用了列名，会从当前行的变量实例中检索该列的值（例如在 WHERE 子句中）。不支持输出参数。

RETURN[S] data_type

标识函数输出的数据类型。

data_type 必须是有效的 solidDB 第 280 页的 A.87.2，『*data_type*』。

SQL_data_access_indication

定义该函数是否发布任何 SQL 语句，如果发布，请指明类型。

CONTAINS SQL

指示不读取或不修改 SQL 数据的 SQL 语句可通过该函数执行。

READS SQL DATA

指示该函数仅执行只读的 SQL 语句。

NO SQL

指示该函数不能执行任何 SQL 语句。

注：系统不会针对所声明的数据访问指示来检查函数内容的有效性。

LANGUAGE C

指示正使用 CREATE FUNCTION 语句注册以 C 编程语言编写的代码为基础的新函数。

EXTERNAL NAME '*external_function_library_identifier*'

识别由用户编写的代码名称，该代码实现正在定义的函数。名称必须引在单引号中。不允许有多余空格。

external_function_library_identifier::=[*path*]*library_name*!*C_routine_name*

[*path*]*library_name*

标识函数的绝对路径（可选）和名称。

例如，在 UNIX 系统上，'/u/jchui/mylib/myfunc.so' 会导致 solidDB 在 /u/jchui/mylib 中查找 myfunc.so 库。

在 Windows 操作系统中，'d:\mylib\myfunc.dll' 会导致 solidDB 装入来自 myfunc.dll 目录的文件 d:\mylib。

!*C_routine_name*

识别将要调用的 C 例程（函数）入口点名称。惊叹号 (!) 作为库名和例程名称之间的定界符。

'!proc8' 会指示 solidDB 在 [*path*]*library_name* 指定的位置中查找库，并指示它在该库中使用入口点 *proc8*。

注：在 Windows 环境中，您必须使用调用约定 `__stdcall` 来调用 C 例程。这是因为 `__stdcall` 是标准的 DLL 界面调用约定。此外，保证您的 DLL 导出您想要调用的例程（例如，包括您 DLL 项目中相应的 `.def` 文件中的例程）。

注：当发出 CREATE FUNCTION 语句时，不需要存在库（以及该库中的函数）。但是，当使用函数时，库以及该库中的函数必须存在并且可供访问。

返回值

- SQL_ERROR
- SQL_SUCCESS

限制

- 假设外部函数的调用接口无防护且线程安全。
- 假定所有函数都是非确定性函数。在语法中允许使用关键字 [NOT] DETERMINISTIC，但 DETERMINISTIC 设置无效。
- 不允许在函数中使用 COMMIT [WORK]。
- 仅有一种参数传递约定可用；返回值会作为输入参数之后的隐式输出参数传递给调用的 C 函数。

- 您不应启动与调用函数并行执行的新线程，因为所提供的连接句柄不能使用与常规 ODBC 连接句柄相同的方式来保证多线程安全。如果启动了新线程，那么这些线程必须创建新连接。如果新线程尝试与函数本身共享同一连接，那么将返回 `SQLException: 句柄无效错误`。

示例

提示: 有关更多示例，请参阅 `solidDB` 安装目录中的 `samples/procedures` 目录。

下列示例注册了外部函数 `EXTFUNCCUBEVOLUME`，此外部函数通过将立方体的边长用作输入参数来计算该立方体的体积:

```
CREATE FUNCTION EXTFUNCCUBEVOLUME (edge float)
RETURN FLOAT
LANGUAGE C
EXTERNAL NAME 'example!lib!extfunccubevolume'
```

外部函数 `EXTFUNCCUBEVOLUME` 的 C 程序

```
#include <solidodbc3.h>

int extfunccubevolume(
    double* edge, /* input */
    double* retval, /* return value */
    SQLLEN nullind[2],
    SQLHDBC hdbc, /* not used in this function */
    char sqlst[6],
    char qualName[],
    char diagMsg[71])
{
    if (nullind[0] == SQL_NULL_DATA) {
        /* NULL input implies NULL return value */
        nullind[1] = SQL_NULL_DATA;
        return (SQL_SUCCESS);
    }
    if (*edge < 0.0) {
        /* error: edge cannot be < 0 */
        strcpy(diagMsg, "extfunccubevolume: parameter 'edge' cannot be < 0");
        strcpy(sqlst, "22003"); /* Numeric values out of range */
        return (SQL_ERROR);
    }
    *retval = (*edge) * (*edge) * (*edge);
    nullind[1] = sizeof(double);
    return (SQL_SUCCESS);
}
```

使用外部函数 `EXTFUNCCUBEVOLUME`

```
SELECT extfunccubevolume(edge) FROM cubic_containers WHERE material = 'STEEL';
```

相关参考

第 157 页的 A.11, 『CREATE FUNCTION』

第 200 页的 A.30, 『DROP FUNCTION』

第 214 页的 A.50, 『GRANT』

A.13 CREATE INDEX

```
CREATE [UNIQUE] INDEX index_name
ON base_table_name
(column_identifier [ASC | DESC]
[, column_identifier [ASC | DESC]] ...)
```

用法

`CREATE INDEX` 语句为表创建基于所指定列的索引。

关键字 `UNIQUE` 指定正在建立索引的列必须包含唯一的值。如果指定了不止一个列，那么这些列的组合必须包含唯一的值，但各个列不必包含唯一的值。

例如，如果对 `LAST_NAME` 与 `FIRST_NAME` 的组合创建索引，那么下列数据值可接受，这是因为，虽然存在重复的姓和重复的名，但没有任何两行包含相同的姓名值。

```
SMITH, PATTI
SMITH, DAVID
JONES, DAVID
```

关键字 `ASC` 和 `DESC` 指定是应该按升序还是降序顺序对给定的列建立索引。如果既未指定 `ASC` 也未指定 `DESC`，那么将使用升序顺序。

示例

```
CREATE UNIQUE INDEX UX_TEST ON TEST (I);
CREATE INDEX X_TEST ON TEST (I DESC, J DESC);
```

相关参考

第 180 页的 A.16, 『CREATE [OR REPLACE] PUBLICATION』

A.14 CREATE PROCEDURE

```
CREATE PROCEDURE procedure_name [(parameter_definition [, parameter_definition ...])]
  [RETURNS (output_column_definition [, output_column_definition ...])]
  [SQL_data_access_indication]
  [LANGUAGE SQL] BEGIN procedure_body END;
```

```
parameter_definition ::= [parameter_mode] parameter_name data_type
output_column_definition::= column_name column_type
```

```
SQL_data_access_indication::= CONTAINS SQL | READS SQL DATA | MODIFIES SQL DATA | NO SQL
```

```
procedure_body ::= [declare_statement; ...][procedure_statement; ...]
```

```
parameter_mode ::= IN | OUT | INOUT
```

```
declare_statement ::= DECLARE variable_name data_type
```

```
procedure_statement ::= prepare_statement | execute_statement |
  fetch_statement | control_statement | post_statement |
  wait_event_statement | wait_register_statement | exec_direct_statement |
  writetrace_statement | sql_dml_or_ddl_statement
```

```
prepare_statement ::= EXEC SQL PREPARE
  { cursor_name | CURSORNAME( { string_literal | variable } ) }
  sql_statement
```

```
execute_statement ::=
  EXEC SQL EXECUTE cursor_name
    [USING (variable [, variable ...])]
    [INTO (variable [, variable ...])] |
  EXEC SQL CLOSE cursor_name |
  EXEC SQL DROP cursor_name |
  EXEC SQL {COMMIT | ROLLBACK} WORK |
  EXEC SQL SET TRANSACTION {READ ONLY | READ WRITE} |
  EXEC SQL WHENEVER SQLERROR {ABORT | ROLLBACK [WORK], ABORT}
  EXEC SEQUENCE sequence_name.CURRENT INTO variable |
  EXEC SEQUENCE sequence_name.NEXT INTO variable |
  EXEC SEQUENCE sequence_name SET VALUE USING variable
```

```
fetch_statement ::= EXEC SQL FETCH cursor_name
```

```
cursor_name ::= literal
```

```

post_statement ::= POST EVENT event_name [(parameters)]

wait_event_statement ::=
  WAIT EVENT
  [event_specification ...]
  END WAIT

event_specification ::=
  WHEN event_name [(parameters)]
  BEGIN statements
  END EVENT

wait_register_statement ::=
  REGISTER EVENT event_name |
  UNREGISTER EVENT event_name

writetrace_statement ::= WRITETRACE(string)

control_statement ::=
  SET variable_name = value | variable_name ::= value |
  WHILE expression
  LOOP procedure_statement... END LOOP |
  LEAVE |
  IF expression THEN procedure_statement ...
  [ ELSEIF procedure_statement ... THEN] ...
  ELSE procedure_statement ... END IF |
  RETURN | RETURN SQLERROR OF cursor_name | RETURN ROW |
  RETURN NOROW

exec_direct_statement ::=
  EXEC SQL [USING (variable [, variable ...])]
  [CURSORNAME(variable)]
  EXECDIRECT sql_dml_or_ddl_statement |
  EXEC SQL cursor_name
  [USING (variable [, variable ...])]
  [INTO (variable [, variable ...])]
  [CURSORNAME(variable)]
  EXECDIRECT sql_dml_or_ddl_statement

```

A.14.1 用法

存储过程是在服务器中执行的简单程序或过程。用户可以创建包含多条 SQL 语句或完整事务的过程，并可以通过单一调用语句来执行该过程。使用存储过程有助于降低网络流量，并使您能够更严格地控制访问权和数据库操作。

过程由 `CREATE PROCEDURE name body` 语句创建，由 `DROP PROCEDURE name` 删除。

过程由 `CALL name [parameter ...]` 语句调用。

存储过程语法是根据 SQL-99 规范和动态 SQL 建模的专有语法。过程包含控制语句和 SQL 语句。

用空白主体可创建存储过程。

可以在过程中使用所有 SQL DML 和 DDL 语句。例如，该过程因此可以创建表或落实事务。过程中的每个 SQL 语句都是原子语句。

“热备用”配置

在 HotStandby 配置中，除非在过程声明中通过使用 SQL 标准子句 *SQL Data Access Indication* 将 SQL 存储过程指定为只读过程，否则所有 SQL 存储过程都将在主数据库中执行。

```
<SQL-data-access-indication> ::=  
    NO SQL |  
    READS SQL DATA |  
    CONTAINS SQL |  
    MODIFIES SQL DATA
```

为了避免不必要地移交只读过程和函数，可以声明下列其中一个值：

- NO SQL
- READS SQL DATA
- CONTAINS SQL

只有 MODIFIES SQL DATA（这是缺省值）才会进行事务移交。

<SQL-data-access-indication> 子句介于 RETURNS 子句（可选）和过程主体之间。

例如：

```
"CREATE PROCEDURE PHONEBOOK_SEARCH  
(IN FIRST_NAME VARCHAR, LAST_NAME VARCHAR)  
RETURNS (PHONE_NR NUMERIC, CITY VARCHAR)  
READS SQL DATA  
BEGIN  
-- procedure_body  
END";  
  
---
```

访问权

过程由创建者拥有。可将指定的访问权授予其他用户。过程运行时，它具有创建者对数据库对象拥有的访问权。

在存储过程落实

对于存储过程内的语句而言，“自动落实”功能的工作方式与存储过程外部的语句不同。对于存储过程外部的 SQL 语句而言，当自动落实功能处于打开状态时，在每个语句后面都有隐式的 COMMIT WORK 操作。但是，对于存储过程而言，只有在存储过程返回到调用者之后才会执行隐式的 COMMIT WORK。注意，这并不表示存储过程具有“原子性”。如上所述，存储过程可以包含自己的 COMMIT 和 ROLLBACK 命令。过程返回后执行的隐式 COMMIT WORK 将仅落实在下列其中一项之后执行的存储过程语句部分：

- 过程中的上一个 COMMIT WORK
- 过程中的上一个 ROLLBACK WORK
- 过程启动（如果在该过程中未执行任何 COMMIT 或 ROLLBACK 命令的话）

如果从一个存储过程中调用另一个存储过程，那么只在最外部的过程调用结束后执行隐式的 COMMIT WORK。在嵌套过程调用完成后，不会执行隐式的 COMMIT WORK。

例如，在以下脚本中，将仅在 CALL outer_proc(); 语句后执行隐式的 COMMIT WORK：

```
"CREATE PROCEDURE inner_proc  
BEGIN  
...  
END";
```



```

CREATE PROCEDURE outer_proc
BEGIN
    ...
    EXEC SQL PREPARE cursor1 CALL inner_proc();
    EXEC SQL EXECUTE cursor1;
    ...
END";
    CALL outer_proc();

```

A.14.2 parameter_modes

parameter_mode ::= IN | OUT | INOUT

存储过程支持三种不同的参数方式：输入参数、输出参数和输入/输出参数。

- 输入参数从调用程序传递到存储过程。*parameter_mode* 值为 IN。这是缺省行为。
- 输出参数从存储过程返回到调用程序。*parameter_mode* 值为 OUT。
- 输入/输出参数将值传递到过程并将一个值返回给调用过程。*parameter_mode* 为 INOUT。

请参阅下表，对各种参数方式进行比较：

表 29. 参数方式的比较

功能	IN	OUT	INOUT
缺省/指定	缺省。	必须指定。	必须指定。
操作	将值传递到子程序。	将值返回给调用者。	将初始值传递到子程序；将更新后的值返回给调用者。
动作	形参，作用类似于常量。	形参，作用类似于未初始化的变量。	形参，作用类似于已初始化的变量。
赋值	形参，不能给它分配值。	形参，不能用于表达式；必须给它分配值。	形参，应该给它分配值。
参数类型	实参，可以是常量、已初始化的变量、字面值或表达式。	实参，必须是变量。	实参，必须是变量。

在编程接口中，输出参数与变量绑定，如下所示：

- 在 JDBC 中，使用方法 `CallableStatement.registerOutParameter()` 进行绑定。
- 在 ODBC 中，使用函数 `SQLBindParameter()` 进行绑定，其中，第三个自变量 `InputOutputType` 的类型可以是：
 - `SQL_PARAM_INPUT`
 - `SQL_PARAM_OUTPUT`
 - `SQL_PARAM_INPUT_OUTPUT`

A.14.3 prepare_statement

```

prepare_statement ::= EXEC SQL PREPARE
    { cursor_name | CURSORNAME( { string_literal | variable } ) }
    sql_statement

```

首先，请使用 `EXEC SQL PREPARE` 语句来准备 SQL 语句

cursor_name 规范是必须指定的游标名。它可以是事务中的任何唯一游标名。如果该过程不是完整的事务，那么该过程外部其他打开的游标可能具有相冲突的游标名。

A.14.4 execute_statement

```
execute_statement ::=
EXEC SQL EXECUTE cursor_name
  [USING (variable [, variable ...])]
  [INTO (variable [, variable ...])] |
EXEC SQL CLOSE cursor_name |
EXEC SQL DROP cursor_name |
EXEC SQL {COMMIT | ROLLBACK} WORK |
EXEC SQL SET TRANSACTION {READ ONLY | READ WRITE} |
EXEC SQL WHENEVER SQLERROR {ABORT | ROLLBACK [WORK], ABORT}
EXEC SEQUENCE sequence_name.CURRENT INTO variable |
EXEC SEQUENCE sequence_name.NEXT INTO variable |
EXEC SEQUENCE sequence_name SET VALUE USING variable

cursor_name ::= literal
```

执行预编译 SQL 语句

要执行 SQL *statement*，请使用以下语句：

```
EXEC SQL EXECUTE cursor [opt_using] [opt_into]
```

其中，可选的 *opt-using* 规范的语法为：

```
USING (variable_list)
```

其中，*variable_list* 包含以逗号分隔的过程变量或参数列表。这些变量是该 SQL 语句的输入参数。SQL 输入参数由 PREPARE 语句中的标准问号语法标记。如果该 SQL 语句未包含输入参数，那么将忽略 USING 规范。

可选的 *opt_into* 规范的语法为：

```
INTO (variable_list)
```

其中，*variable_list* 包含 SQL SELECT 语句的列值要存储到的变量。INTO 规范仅对 SQL SELECT 语句有效。

执行 UPDATE、INSERT 和 DELETE 语句后，将有一个附加的变量可用于检查该语句的结果。变量 SQLROWCOUNT 包含上一个语句所影响的行数。

使用事务

使用以下语句来终止事务：

```
EXEC SQL {COMMIT | ROLLBACK} WORK
```

使用以下语句来控制事务的类型：

```
EXEC SQL SET TRANSACTION {READ ONLY | READ WRITE}
```

关闭和删除游标

使用游标完毕后，您可以关闭该游标或关闭并删除该游标。

如果可能复用游标并想提高性能，您应该仅关闭该游标。您关闭游标时，会释放在执行阶段期间分配的所有内存，但是游标保留准备状态。

删除游标会释放所有已分配的资源。下次您使用已删除的游标时，需要做准备。

检查错误

在过程主体中执行的每个 EXEC SQL 语句的结果都将存储到变量 SQLSUCCESS 中。将为每个过程自动生成此变量。如果上一个 SQL 语句成功，那么 SQLSUCCESS 中存储的值为 1。如果该 SQL 语句失败，那么 SQLSUCCESS 中存储的值为 0。

```
EXEC SQL WHENEVER SQLERROR {ABORT | [ROLLBACK [WORK], ABORT]}
```

以上语句使您不必在过程中执行的每个 SQL 语句之后执行 IF NOT SQLSUCCESS THEN 测试。如果在存储过程中包括此语句，那么将对所执行的语句的所有返回值检查错误。如果所执行的语句返回错误，那么该过程将自动中止。您可以选择将事务回滚。

最近失败的 SQL 语句的错误字符串将存储到变量 SQLERRSTR 中。

A.14.5 fetch_statement

```
fetch_statement ::= EXEC SQL FETCH cursor_name
```

EXEC SQL FETCH 语句会访问行。如果访问成功完成，列值将存储到 EXECUTE 或 EXECDIRECT 语句的 *opt_into* 规范所定义的变量中。

A.14.6 post_statement

```
post_statement ::= POST EVENT event_name [( parameters) ]  
                [UNIQUE | DATA UNIQUE]
```

POST EVENT 语句用于发布系统事件和用户定义的事件。

UNIQUE 关键字表示对于每个用户和每个事件，在事件队列中只保留最后发出的事件。例如，在 POST EVENT EV(1) 和 POST EVENT EV(2) 之后，如果发出 EV(2) 前未处理 EV(1)，那么事件队列只包含 EV(2)。废弃事件 EV(1)。

DATA UNIQUE 表示事件参数也必须唯一。因此，执行 POST EVENT EV(1)、POST EVENT EV(2) 和 POST EVENT EV(2) 调用之后，事件队列将包含 EV(1) 和 EV(2)。废弃第一个 EV(2)。

只允许在存储过程中使用 POST EVENT 语句。

相关信息

第 73 页的 3.5, 『事件』

A.14.7 wait_register_statement

```
wait_register_statement ::=  
    REGISTER EVENT | UNREGISTER EVENT event_name
```

REGISTER EVENT 语句告诉服务器您希望以后每次遇到该指定事件时都受到提醒，即使尚未等待此事件亦如此。通过分开 REGISTER EVENT 和 WAIT EVENT 命令，您可以立即开始对事件进行排队，同时等待到以后实际地开始处理它们为止。

在等待事件前，您不需要对每个事件注册。等待一个事件时，如果尚未以显式方式对该事件注册，那么将以隐式方式对该事件注册。因此，仅当您希望立即开始对事件进行排队，但在完成排队前不想开始等待这些事件时，才需要以显式方式注册事件。

UNREGISTER EVENT 语句注销指定事件。

仅在存储过程中允许使用 REGISTER EVENT 和 UNREGISTER EVENT 语句。

相关信息

第 73 页的 3.5, 『事件』

A.14.8 wait_event_statement

```
wait_event_statement ::=
    WAIT EVENT
    [event_specification ...]
    END WAIT

event_specification ::=
    WHEN event_name [(parameters)]
    BEGIN statements
    END EVENT
```

WAIT EVENT 语句令过程等待事件发生。系统定义的事件和用户定义的事件都可使用 WAIT EVENT。

如果要让存储过程停止等待事件，可以使用从客户机应用程序中的独立线程中调用的 ODBC 函数 SQLCancel()。此函数将取消执行语句。此外，您也可以创建并发送特定的用户事件。要等待这个附加的事件，必须修改等待存储过程。客户机应用程序将识别此事件并退出等待循环。

提示：对于系统事件，您还可以使用 ADMIN EVENT 语句等待事件，而不必使用存储过程。然而，您不能使用 ADMIN EVENT 发出事件。

```
ADMIN EVENT 'register sys_event_hsbstateswitch';
ADMIN EVENT 'wait';
```

相关信息

第 73 页的 3.5, 『事件』

A.14.9 control_statement

```
control_statement ::=
    SET variable_name = value | variable_name ::= value |
    WHILE expression
        LOOP procedure_statement... END LOOP |
    LEAVE |
    IF expression THEN procedure_statement ...
        [ ELSEIF procedure_statement ... THEN] ...
        ELSE procedure_statement ... END IF |
    RETURN | RETURN SQLERROR OF cursor_name | RETURN ROW |
    RETURN NOROW
```

在过程中，可以使用下列控制语句：

表 30. 控制语句

控制语句	描述
设置 <code>variable = expression</code>	对变量赋值。值可以是字面值（例如，10 或“文本”），也可以是另一个变量。可将参数视为常规变量。
<code>variable ::= expression</code>	另一种用于对变量赋值的语法。
while <code>expr</code> loop <code>statement-list</code> end loop	表达式为 true 时进行循环。
leave	离开最内部的 while 循环并从关键字 end loop 后的下一个语句开始继续执行该过程。
if <code>expr</code> then <code>statement-list1</code> else <code>statement-list2</code> end if	如果表达式 <code>expr</code> 为 true，那么执行 <code>statements-list1</code> ，否则执行 <code>statement-list2</code> 。
if <code>expr1</code> then <code>statement-list1</code> elseif <code>expr2</code> then <code>statement-list2</code> end if	如果 <code>expr1</code> 为 true，那么执行 <code>statement-list1</code> 。如果 <code>expr2</code> 为 true，那么执行 <code>statement-list2</code> 。此语句可以选择包含多个 elseif 语句以及一个 else 语句。
return	返回输出参数的当前值并退出过程。如果过程包含 <code>return row</code> 语句，那么 return 的行为与 <code>return norow</code> 类似。
return sqlerror of <code>cursor-name</code>	返回与游标相关联的 sqlerror 并退出过程。
return row	返回输出参数的当前值并继续执行过程。return row 并不退出过程，而是将控制权返回给调用者。
return norow	返回结果集的末尾并退出过程。

注：用 RETURNS 子句定义的过程通常至少返回一行。即便当该行填充的是 NULL 时，也是如此。

为了不返回空白行，可在过程主体上使用 RETURN NOROW。它会立即返回且没有结果集 (SQL_NO_DATA)。语句 RETURN NOROW 可在一系列 RETURN ROW 语句后使用，因此该过程会返回使用 RETURN ROW 生成的行。

A.14.10 writetrace_statement

`writetrace_statement ::= WRITETRACE(string)`

WRITETRACE 语句函数将跟踪输出（字符串）发送到 soltrace.out 跟踪文件。在调试存储过程中的问题时，此函数很有用。

仅当已打开跟踪功能时，才会写输出。

有关存储过程中的跟踪功能和如何开启跟踪功能的更多信息，请参阅第 121 页的 6.2，『用于存储过程和触发器的跟踪工具』。

A.14.11 exec_direct_statement

```
exec_direct_statement ::=  
EXEC SQL [USING (variable [, variable ...])]   
[CURSORNAME(variable)]   
EXECDIRECT sql_dml_or_ddl_statement |   
EXEC SQL cursor_name   
[USING (variable [, variable ...])]   
[INTO (variable [, variable ...])]   
[CURSORNAME(variable)]   
EXECDIRECT sql_dml_or_ddl_statement
```

EXECDIRECT 语句允许您在存储过程中执行语句，不必事先准备那些语句。这将减少所需的代码量。如果该语句是游标，那么仍需要将其关闭并删除；只能跳过 PREPARE 语句。

使用 EXECDIRECT 的注意事项：

- 如果该语句指定了游标名，必须使用 EXEC SQL DROP 语句来删除该游标。
- 如果未指定游标名，您不需要删除该语句。
- 如果该语句是访存游标，必须指定 INTO... 子句。
- 如果指定了 INTO 子句，必须指定游标名；否则 FETCH 语句将能够指定应该从中访存行的游标名。您可以同时打开不止一个游标。

A.14.12 过程堆栈函数

下列函数可用于分析过程堆栈的当前内容：PROC_COUNT()、PROC_NAME(N) 和 PROC_SCHEMA(N)。

PROC_COUNT() 返回过程堆栈中的过程数目。这包括当前过程。

PROC_NAME(N) 返回堆栈中的第 N 个过程名。第一个过程的位置为零。

PROC_SCHEMA(N) 返回过程堆栈中第 N 个过程的模式名。

A.14.13 动态游标名

```
CURSORNAME(  
    prefix -- VARCHAR  
)
```

CURSORNAME() 函数允许您动态地生成游标名，而不是以硬编码方式指定游标名。

注：

严格来说，CURSORNAME() 不是函数，尽管语法十分相似。CURSORNAME(arg) 并不会实际地返回任何内容；而是，它根据给定的自变量设置当前语句的游标名。但是，将其称为函数比较方便，因此我们就这样做。

游标名在连接中必须唯一。对于递归存储过程而言，由于每次调用都使用相同的游标名，因此将出现问题。当递归过程调用它自己时，第二次调用将发现第一次调用已创建具有第二次调用所要使用的名称的游标。

为了解决此问题，我们必须动态地生成唯一的游标名，并且必须能够在声明和使用游标时使用那些名称。为了能够生成唯一的名称并将那些名称用作游标，我们使用两个函数：

- GET_UNIQUE_STRING
- CURSORNAME

GET_UNIQUE_STRING 函数的名称反映了它的功能 - 生成唯一的字符串。CURSORNAME 函数（实际上，这是一个伪函数）允许您使用动态生成的字符串作为游标名的组成部分。

注意，每次调用 GET_UNIQUE_STRING 时都将返回不同的输出，即使输入相同亦如此。另一方面，如果输入每次都相同，那么 CURSORNAME 每次都返回相同的输出。

以下是使用 GET_UNIQUE_STRING 和 CURSORNAME 来动态生成游标名的示例。系统会把动态生成的游标名分配给占位符“cname”，后者将用于 PREPARE 后的每个语句。

```
DECLARE autoname VARCHAR;  
Autoname := GET_UNIQUE_STRING('CUR ');  
EXEC SQL PREPARE cname CURSORNAME(autoname) SELECT * FROM TABLES;  
EXEC SQL EXECUTE cname USING(...) INTO(...);  
EXEC SQL FETCH cname;  
EXEC SQL CLOSE cname;  
EXEC SQL DROP cname;
```

CURSORNAME() 只能用于 PREPARE 语句和 EXEC DIRECT 语句。它不能用于 EXECUTE、FETCH、CLOSE 和 DROP 等语句。

通过使用 CURSORNAME() 功能和 GET_UNIQUE_STRING() 函数，可以在递归存储过程中生成唯一的游标名。如果该过程调用它自己，那么每次在存储过程中调用此函数时，此函数都将返回唯一的字符串，该字符串可以在 PREPARE 语句中用作游标名。请参阅以下内容，以获取可以在存储过程中使用的代码的一些示例。

注意，如果不更改输入 autoname，那么每次调用 CURSORNAME(autoname) 都将返回同一个值 - 即，同一个游标名。

A.14.14 示例

CREATE PROCEDURE

```
"create procedure test2(tableid integer)  
  returns (cnt integer)  
begin  
  exec sql prepare c1 select count(*) from sys_tables where id > ?;  
  exec sql execute c1 using (tableid) into (cnt);  
  exec sql fetch c1;  
  exec sql close c1;  
  exec sql drop c1;  
end";
```

使用显式的 RETURN 语句

此示例使用显式的 RETURN 语句来返回多行（每次返回一行）。

```
"create procedure return_tables  
  returns (name varchar)  
begin  
  exec sql execdirect create table table_name (lname char (20));  
  exec sql whenever sqlerror rollback, abort;
```

```

exec sql prepare c1 select table_name from sys_tables;
exec sql execute c1 into (name);
while sqlsuccess loop
    exec sql fetch c1;
    if not sqlsuccess
        then leave;
    end if
    return row;
end loop;
exec sql close c1;
exec sql drop c1;
end";

```

使用 EXECDIRECT

```

-- This example shows how to use "execdirect".
"CREATE PROCEDURE p
BEGIN
    DECLARE host_x INT;
    DECLARE host_y INT;

    -- Examples of execdirect without a cursor. Here we create a table
    -- and insert a row into that table.
    EXEC SQL EXECDIRECT create table foo (x int, y int);
    EXEC SQL EXECDIRECT insert into foo(x, y) values (1, 2);

    SET host_x = 1;

    -- Example of execdirect with cursor name.
    -- In this example, "c1" is the cursor name; "host_x" is the
    -- variable whose value will be substituted for the "?";
    -- "host_y" is the variable into which we will store the value of the
    -- column y (when we fetch it).
    -- Note: although you do not need a "prepare" statement, you still
    -- need close/drop.
    EXEC SQL c1 USING(host_x) INTO(host_y) EXECDIRECT
        SELECT y from foo where x=?;
    EXEC SQL FETCH c1;
    EXEC SQL CLOSE c1;
    EXEC SQL DROP c1;
END";

```

使用 CURSORNAME

此示例说明 CURSORNAME() 伪函数的用法。此示例仅提供了存储过程主体的部分内容，而未提供完整的存储过程。

```

-- Declare a variable that will hold a unique string that we can use
-- as a cursor name.
DECLARE autoname VARCHAR ;
Autoname := GET_UNIQUE_STRING('CUR_') ;
EXEC SQL PREPARE curs_name CURSORNAME(autoname) SELECT * FROM TABLES;
EXEC SQL EXECUTE curs_name USING(...) INTO(...);
EXEC SQL FETCH curs_name;
EXEC SQL CLOSE curs_name;
EXEC SQL DROP curs_name;

```

使用 GET_UNIQUE_STRING 和 CURSORNAME

该示例在一个递归存储过程中使用 GET_UNIQUE_STRING 和 CURSORNAME 函数。

以下存储过程演示如何在递归过程中使用这两个函数。游标名“curs1”似乎是硬编码的，但它实际上是已映射到动态生成的名称。


```

-- Demonstrate GET_UNIQUE_STRING and CURSORNAME functions in a
-- recursive stored procedure.
-- Given a number N greater than or equal to 1, this procedure
-- returns the sum of the numbers 1 - N. (This can also be done in a loop,
-- but the purpose of the example is to show the use of the
-- CURSORNAME function in a recursive procedure.)
"CREATE PROCEDURE Sum1ToN(n INT)
RETURNS (SumSoFar INT)
BEGIN
    DECLARE SumOfRemainingItems INT;
    DECLARE nMinusOne INT;
    DECLARE autoname VARCHAR;

    SumSoFar := 0;
    SumOfRemainingItems := 0;
    nMinusOne := n - 1;

    IF (nMinusOne > 0) THEN
        Autoname := GET_UNIQUE_STRING('CURSOR_NAME_PREFIX ');
        EXEC SQL PREPARE curs1 CURSORNAME(autoname) CALL Sum1ToN(?);
        EXEC SQL EXECUTE curs1 USING(nMinusOne) INTO(SumOfRemainingItems);
        EXEC SQL FETCH curs1;
        EXEC SQL CLOSE curs1;
        EXEC SQL DROP curs1;
    END IF;

    SumSoFar := n + SumOfRemainingItems;
END";

```

在 CREATE PROCEDURE 中使用 EXECDIRECT

```

CREATE TABLE table1 (x INT, y INT);
INSERT INTO table1 (x, y) VALUES (1, 2);

"CREATE PROCEDURE FOO
RETURNS (r INT)
BEGIN
    DECLARE autoname VARCHAR;
    Autoname := GET_UNIQUE_STRING('CUR_');
    EXEC SQL curs_name INTO(r) CURSORNAME(autoname) EXECDIRECT
        SELECT y FROM TABLE1 WHERE x = 1;
    EXEC SQL FETCH curs_name;
    EXEC SQL CLOSE curs_name;
    EXEC SQL DROP curs_name;
END";

CALL foo();
SELECT * FROM table1;

```

为同步消息创建唯一的名称

为同步消息创建唯一的名称:

```

DECLARE Autoname VARCHAR;
DECLARE Sqlstr VARCHAR;
Autoname := get_unique_string('MSG_');
Sqlstr := 'MESSAGE' + autoname + 'BEGIN';
EXEC SQL EXECDIRECT Sqlstr;
...
Sqlstr := 'MESSAGE' + autoname + 'FORWARD';
EXEC SQL EXECDIRECT Sqlstr;

```

使用 GET_UNIQUE_STRING

该示例演示了如何使用 GET_UNIQUE_STRING() 函数从递归存储过程内部创建唯一的名称。

```

CREATE TABLE table1 (i int, beginMsg VARCHAR, endMsg VARCHAR);

-- This is a simplified example of recursion.
-- Note that the messages that are composed are not actually used! This is
-- not a true example of synchronization; it is only an example of
-- generating unique message names. The "count" parameter is the
-- number of times that you want this function to call
-- itself (not including the initial call).
"CREATE PROCEDURE repeater(count INT)

BEGIN

DECLARE Autoname VARCHAR;
DECLARE MsgBeginStr VARCHAR;
DECLARE MsgEndStr VARCHAR;

Autoname := GET_UNIQUE_STRING('MSG_');
MsgBeginStr := 'MESSAGE ' + Autoname + ' BEGIN';
MsgEndStr := 'MESSAGE ' + Autoname + ' END';

EXEC SQL c1 USING (count, MsgBeginStr, MsgEndStr) EXECDIRECT
    INSERT INTO table1 (i, beginMsg, endMsg) VALUES (?, ?, ?);
EXEC SQL CLOSE c1;
EXEC SQL DROP c1;

-- Once you have composed the SQL statement as a string,
-- you can execute it one of two ways:
-- 1) by using the EXECDIRECT feature or
-- 2) by preparing and executing the SQL statement.
-- In this example, we use EXECDIRECT.
EXEC SQL EXECDIRECT MsgBeginStr;
EXEC SQL EXECDIRECT MsgEndStr;
-- Do something useful here.

-- The recursive portion of the function.
IF (count > 1) THEN
    SET count = count - 1;
    -- Note that we can also use our unique name as a cursor name,
    -- as shown below.
    EXEC SQL Autoname USING (count) EXECDIRECT CALL repeater(?);
    EXEC SQL CLOSE Autoname;
    EXEC SQL DROP Autoname;
END IF

RETURN;
END";

CALL repeater(3);

-- Show the message names that we composed.
SELECT * FROM table1;

```

这个 SELECT 语句的输出将类似于:

```

I   BEGINMSG                ENDMSG
-- -----                -
1  MESSAGE MSG_019 BEGIN    MESSAGE MSG_019 END
2  MESSAGE MSG_020 BEGIN    MESSAGE MSG_020 END
3  MESSAGE MSG_021 BEGIN    MESSAGE MSG_021 END

```

A.15 CREATE PROCEDURE (外部)

```

CREATE PROCEDURE procedure_name [(parameter_definition [, parameter_definition ...])]
    [RETURNS (output_column_definition [, output_column_definition ...])]
    [SQL_data_access_indication]
    LANGUAGE C

```

```
EXTERNAL NAME 'external_procedure_library_identifier'
```

```
parameter_definition ::= [parameter_mode] parameter_name data_type  
parameter_mode ::= IN | OUT | INOUT
```

```
output_column_definition ::= column_name column_type
```

```
SQL_data_access_indication ::= CONTAINS SQL | READS SQL DATA | MODIFIES SQL DATA | NO SQL
```

用法

CREATE PROCEDURE (外部) 语句定义使用 C 编程语言编写的外部过程。

使用操作系统提供的标准动态库界面 (Linux 和 UNIX 中的共享库及 Windows 中的 DLL) 访问外部过程。如果要允许使用外部存储过程, 您必须使用共享内存访问 (SMA) 或链接库访问 (LLA) 将您的应用程序链接至 solidDB。

用与调用 SQL 存储过程同样的 CALL 语句语法调用外部过程; 系统会把线程控制传递给外部库例程, 直至其返回。此外, 外部过程的数据库界面与 ODBC 应用程序的相似, 唯一不同的是, 它的连接句柄 (hdbc) 会传递给外部例程, 并且, 它能立即开始使用数据库界面而不用建立连接。

外部过程的输出可作为单一行或输出参数的结果集来返回。

参数

procedure_name

识别过程名称。过程名称还可包括目录名及模式:

```
[catalog_name[.schema].]procedure_name
```

名称必须是有效的 SQL 标识。过程名称在模式内必须唯一。

```
parameter_definition ::= [parameter_mode] parameter_name data_type
```

识别过程的参数, 并指定方式、可选参数名称和参数的数据类型。必须为过程期望的每个参数指定列表中的一个入口。

```
parameter_mode ::= IN | OUT | INOUT
```

识别参数方式。

IN 识别作为过程的输入参数的参数。将控制权交回给调用 SQL 应用程序时, 对过程中的参数所做的任何更改都不可用于该调用 SQL 应用程序。缺省值为 IN。

OUT

识别作为过程的输出参数的参数。

INOUT

识别作为过程的输入参数和输出参数的参数。

```
parameter_name
```

指定参数名称。参数名称对过程而言必须是唯一的。

```
data_type
```

指定参数的数据类型。数据类型必须是有效的 solidDB 第 280 页的 A.87.2, 『data_type』。

RETURNS (*output_column_definition*)

确定过程的输出。

output_column_definition ::= column_name column_type

column_name

指定输出列的名称。

column_type

指定输出列的数据类型。数据类型必须是有效的 solidDB 第 280 页的 A.87.2, 『data_type』。

注: 如果使用 RETURNS 关键字, 那么参数方式只能是 IN。

SQL_data_access_indication

定义过程是否发出任何 SQL 语句, 如果发出, 还会定义发出类型。

CONTAINS SQL

指示该过程包含 SQL 语句, 但这些语句不读取或修改 SQL 数据。

READS SQL DATA

指示该过程包含读取 SQL 数据但不能修改任何数据的 SQL 语句。

MODIFIES SQL DATA

指示该过程包含修改 SQL 数据的 SQL 语句。

NO SQL

指示该过程不执行任何 SQL 语句。

注: 系统不会针对所声明的数据访问指示来检查过程内容的有效性。

LANGUAGE C

指示 CREATE PROCEDURE 语句正用于注册基于 C 编程语言编写的代码的新过程。

EXTERNAL NAME '*external_procedure_library_identifier*'

识别用户编写的代码名称, 该代码实现正处于定义中的过程。名称必须引在单引号中。不允许有多余空格。

external_procedure_library_identifier ::= [*path*]
library_name!*C_routine_name*

[*path*]*library_name*

识别过程的完全路径 (可选) 和名称。

例如, 在 UNIX 系统上, '/u/jchui/mylib/myproc.so' 会导致 solidDB 在 /u/jchui/mylib 中查找 myproc.so 库。

在 Windows 操作系统中, 'd:\mylib\myproc.dll' 会引起 solidDB 装入来自 d:\mylib 目录的文件 myproc.dll。

!*C_routine_name*

识别将要调用的 C 例程 (过程) 的入口点名称。感叹号 (!) 作为库名和例程名称之间的定界符。

'!proc8' 会指示 solidDB 在 [*path*]*library_name* 指定的位置中查找库, 并指示它在该库中使用入口点 *proc8*。

注：在 Windows 环境中，您必须使用调用约定 `__stdcall` 来调用 C 例程。这是因为 `__stdcall` 是标准的 DLL 界面调用约定。此外，保证您的 DLL 导出您想要调用的例程（例如，包括您 DLL 项目中相应的 `.def` 文件中的例程）。

注：当发出 `CREATE PROCEDURE` 语句时，不需要存在库（以及该库中的过程）。但是，当调用过程时，库以及该库中的过程必须存在并且可供访问。

返回值

- `SQL_ERROR`
- `SQL_SUCCESS`

限制

- 假设外部过程的调用界面无防护且线程安全。
- 假定所有的过程都为非确定性的过程；过程取决于一些影响结果的状态值。
- 您不应启动与调用过程并行执行的新线程，因为所提供的连接句柄不能使用与常规的 ODBC 连接句柄相同的方式来保证多线程安全。如果已启动新线程，它们不能同时访问该连接，在初始外部过程返回后也如此。

示例

提示：有关更多示例，请参阅 `solidDB` 安装目录中的 `samples/procedures` 目录。

以下示例注册外部存储过程 `EXTPROCEXAMPLE1`。

```
CREATE PROCEDURE EXTPROCEXAMPLE1(IN P1 INTEGER, INOUT P2 INTEGER, OUT P3 CHAR(10))
READS SQL DATA
LANGUAGE C
EXTERNAL NAME 'example1lib!extprocexample1'
```

过程的 C 程序

```
#include <solidodbc3.h>
/* UTF-8 sequence for a char can be up to 4 bytes! */
#define CHARMAXBYTELEN(charlen) (4 * (charlen) + 1)
int extprocexample1(int* p1, int* p2, char p3[CHARMAXBYTELEN(10)],
                    SQLLEN nullind[3],
                    SQLHDBC hdbc,
                    char sqlst[6],
                    char qualName[],
                    char diagMsg[71])
{
    SQLHSTMT hstmt = SQL_NULL_STMT;
    SQLRETURN rc;

    nullind[2] = SQL_NULL_DATA;
    if (nullind[0] == SQL_NULL_DATA || nullind[1] == SQL_NULL_DATA) {
        goto null_input_detected;
    }
    rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    if (rc != SQL_SUCCESS) {
        goto failure;
    }
    rc = SQLPrepare(hstmt, "SELECT P2,P3 FROM EXTPROCEXAMPLE1_TAB "
                    "WHERE P1 = ? AND P2 > ?", SQL_NTS);
    if (rc != SQL_SUCCESS) {
        goto failure;
    }
    rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT,
```

```

                                SQL_C_SLONG, SQL_INTEGER,
                                0, 0, p1, sizeof(*p1), NULL);
if (rc != SQL_SUCCESS) {
    goto failure;
}
rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT,
                      SQL_C_SLONG, SQL_INTEGER,
                      0, 0, p2, sizeof(*p2), NULL);
if (rc != SQL_SUCCESS) {
    goto failure;
}
rc = SQLBindColumn(hstmt, 1, SQL_C_SLONG, p2, 0, NULL);
if (rc != SQL_SUCCESS) {
    goto failure;
}
rc = SQLBindColumn(hstmt, 2, SQL_C_CHAR, p3, CHARMAXBYTELEN(10), NULL);
if (rc != SQL_SUCCESS) {
    goto failure;
}
rc = SQLExecute(hstmt);
if (rc != SQL_SUCCESS) {
    goto failure;
}
rc = SQLFetch(hstmt);
if (rc == SQL_NO_DATA) {
    goto no_data_found;
}
if (rc != SQL_SUCCESS) {
    goto failure;
}
nullind[1] = sizeof(int);
nullind[2] = sizeof(int);
retcode = 1;
cleanup:;
if (hstmt != SQL_NULL_HANDLE) {
    SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
}
return retcode;
null_input_detected:;
no_data_found:;
retcode = SQL_SUCCESS;
nullind[1] = SQL_NULL_DATA;
nullind[2] = SQL_NULL_DATA;
goto cleanup;
failure:;
retcode = SQL_ERROR;
if (hstmt != SQL_NULL_HANDLE) {
    rc = SQLError(SQL_NULL_HANDLE, SQL_NULL_HANDLE, hstmt,
                 sqlst, &nativeerror, diagMsg, 71, NULL);
    if (rc == SQL_NO_DATA) {
        goto generic_failure;
    }
} else {
    rc = SQLError(SQL_NULL_HANDLE, hdbc, SQL_NULL_HANDLE,
                 sqlst, &nativeerror, diagMsg, 71, NULL);
    if (rc == SQL_NO_DATA) {
        goto generic_failure;
    }
}
goto exit_cleanup;
generic_failure:;
memcpy(sqlst, "1C000", 6);
memcpy(diagMsg, "Generic error", 14);
goto exit_cleanup;
}

```

相关参考

第 163 页的 A.14, 『CREATE PROCEDURE』

第 201 页的 A.33, 『DROP PROCEDURE』

A.16 CREATE [OR REPLACE] PUBLICATION

```
"CREATE [OR REPLACE] PUBLICATION publication_name
  [(parameter_definition [,parameter_definition...])]
BEGIN
  main_result_set_definition...
END";
```

```
main_result_set_definition ::=
RESULT SET FOR main_replica_table_name
```

```
BEGIN
  SELECT select_list
  FROM master_table_name
  [ WHERE search_condition ] ;
  [ [DISTINCT] result_set_definition... ]
END
```

```
result_set_definition ::=
RESULT SET FOR replica_table_name
```

```
BEGIN
  SELECT select_list
  FROM master_table_name
  [ WHERE search_condition ] ;
  [[ DISTINCT] result_set_definition... ]
END
```

其中:

search_condition 可以引用 *parameter_definitions* 和/或先前 (较高) 级别上定义的副本表列。

用法

CREATE [OR REPLACE] PUBLICATION 语句创建或替代发布。发布用于定义一组数据, 通过执行刷新 (REFRESH) 操作, 您可以将这些数据从主数据库复制到副本数据库。发布始终在事务方面具有一致性, 即, 它的数据已通过一个事务从主数据库中读取, 并且该数据已通过另一个事务写入副本数据库。

要点: 除非主数据库正在使用 READ COMMITTED 隔离级别, 否则从发布读取的数据在内部处于一致状态。

SELECT 子句的搜索条件可以包含发布的输入自变量作为参数。参数名必须以冒号作为前缀。

发布可以包含来自多个表的数据。发布的各个表可以相互独立, 也可以在相互之间存在关系。如果各个表之间存在关系, 那么必须对结果集进行嵌套。对于发布的内部结果集的 SELECT 语句的 WHERE 子句而言, 它必须引用外部结果集的表的列。

如果发布的外部结果集与内部结果集之间存在 N-1 关系, 那么必须在结果集定义中使用关键字 DISTINCT。

replica_table_name 可以与 *master_table_name* 不同。发布定义在主控表与副本表之间提供了映射。如果有多个副本表，那么所有副本表都应该使用同一个名称，即使该名称与主控表中使用的名称不同亦如此。主控表与副本表中的列名必须相同。

最初的下载始终是完整发布，即将该发布中包含的所有数据都发送到副本数据库。同一发布的后续下载（刷新）可以是增量发布，即，它们只包含上次刷新后更改的数据。为了能够使用增量发布，在主数据库和副本数据库中都必须对该发布所包含的表设置 `SYNCHISTORY ON`。有关详细信息，请参阅第 147 页的 A.4.2，『ALTER TABLE ... SET SYNCHISTORY』和 第 202 页的 A.35，『DROP PUBLICATION REGISTRATION』。

如果使用了可选的关键字“OR REPLACE”，并且该发布已存在，那么将由新定义替换该关键字。由于未删除并重新创建该发布，因此副本数据库不需要重新注册，并且根据该发布进行的后续刷新可以是增量刷新（而不必是完全刷新）这完全取决于对该发布所作的更改。

为了避免副本数据库在您更新发布时根据该发布进行刷新，您可以将目录的同步方式暂时设置为维护方式。但是，在替换发布时，维护方式的使用不是必要条件。

如果替换现有的发布，那么将在副本数据库请求执行刷新时将该发布的新定义发送到每个副本数据库。副本数据库自身不需要显式地重新向该发布注册。

将现有发布替换为新定义时，可以更改结果集定义。但是，不能更改该发布的参数。更改参数的唯一方法是删除该发布并创建新发布，这还意味着副本必须要重新注册，且该副本在下次请求刷新时将完全刷新而不是增量刷新。

替换现有发布时，与该发布相关的特权将保持不变。不必重新创建这些特权。

在任何可以执行 `CREATE PUBLICATION` 命令的情况下，都可以执行 `CREATE OR REPLACE PUBLICATION` 命令。

要点： 如果使用 `CREATE OR REPLACE PUBLICATION` 来更改现有高级复制发布的内容，那么必须确保从副本数据库中除去无效的行。

用于主数据库

您在主数据库中定义发布，以使副本数据库能够根据该发布执行刷新。

用于副本数据库

不需要在副本数据库中定义发布。发布预订功能只依赖于主数据库中的定义。如果在副本数据库中执行 `CREATE OR REPLACE` 命令，那么将把发布定义存储到副本数据库，但该发布定义不用于任何事情。

注： 如果一个数据库既是副本数据库（对于其上方的主数据库而言）又是主数据库（对于其下方的副本数据库而言），那么您可能要在该数据库中创建发布定义。

示例

示例 1:

以下示例发布通过将客户的地区码用作搜索条件从客户表中检索数据。对于每个客户，还将检索该客户的订单和发票（1-N 关系）以及该客户的指定售货员（1-1 关系）。


```

"CREATE PUBLICATION PUB_CUSTOMERS_BY_AREA
  (IN_AREA_CODE VARCHAR)
BEGIN
  RESULT SET FOR CUSTOMER
  BEGIN
    SELECT * FROM CUSTOMER
    WHERE AREA_CODE = :IN_AREA_CODE;
  RESULT SET FOR CUST_ORDER
  BEGIN
    SELECT * FROM CUST_ORDER
    WHERE CUSTOMER_ID = CUSTOMER.ID;
  END
  RESULT SET FOR INVOICE
  BEGIN
    SELECT * FROM INVOICE
    WHERE CUSTOMER_ID = CUSTOMER.ID;
  END
  DISTINCT RESULT SET FOR SALESMAN
  BEGIN
    SELECT * FROM SALESMAN
    WHERE ID = CUSTOMER.SALESMAN_ID;
  END
  END
END";

```

注:

:IN_AREA_CODE 中的冒号 (:) 指定引用同名的发布参数。

示例 2:

开发者决定在发布 P 所引用的表 T 中添加新列 C。必须对主数据库和所有副本数据库都进行此修改。

在主数据库中执行的任务如下所示:

```

-- 防止其他用户进行同步操作
-- 对于该目录
SET SYNC MAINTENANCE MODE ON;
ALTER TABLE T ADD COLUMN C INTEGER;
COMMIT WORK;
CREATE OR REPLACE PUBLICATION P ... (还将列 C 添加到发布)
COMMIT WORK;
SET SYNC MAINTENANCE MODE OFF;

```

在副本数据库中执行的任务如下所示:

```

-- 防止其他用户进行同步操作-- 对于该目录
SET SYNC MAINTENANCE MODE ON;
ALTER TABLE T ADD COLUMN C INTEGER;
COMMIT WORK;
SET SYNC MAINTENANCE MODE OFF;

```

返回值

表 31. CREATE PUBLICATION 返回值

错误代码	描述
13047	无权执行操作。您不具有删除此发布或创建新发布所需的特权。
13120	名称对于发布而言过长。
25015	语法错误: <i>error_message</i> (第 <i>line_number</i> 行)

表 31. CREATE PUBLICATION 返回值 (续)

错误代码	描述
25021	此数据库不是主数据库或副本数据库。只能在主数据库或副本数据库中创建发布。（在实践中，只应该在主数据库中创建发布。）
25033	发布 <i>publication_name</i> 已存在。
25049	在预订层次结构中，找不到被引用表 <i>table_name</i> 。
25061	表 <i>table_name</i> 的 WHERE 条件必须引用发布的外层表。

A.17 CREATE [OR REPLACE] REMOTE SERVER

```
CREATE [OR REPLACE] REMOTE SERVER [USERNAME <username> PASSWORD <password>]
```

用法

CREATE REMOTE SERVER 语句在 SYS_SERVER 系统表中创建用于 Universal Cache 中的 SQL 传递的后端登录信息。

CREATE OR REPLACE REMOTE SERVER 语句修改 SYS_SERVER 系统表中的登录信息，从而覆盖现有的输入。

在大多数情况下，InfoSphere™ CDC 技术用于把后端登录数据传输到前端。当首个从 solidDB 到后端数据服务器的预定上的制作镜像或刷新启动时，solidDB 实例的 InfoSphere CDC 会递归来自后端的 InfoSphere CDC 实例并使用 CREATE REMOTE SERVER 语句将其存储在 solidDB 系统表 SYS_SERVER 中。存储在 SYS_SERVER 表中的密码已隐藏。

在下面所描述的事例中，InfoSphere CDC 技术不传输后端登录数据。

- 如果后端的 InfoSphere CDC 实例在使用能够自动访问数据库的用户标识的情况下运行，那么无需存储登录数据。
- 如果后端数据服务器是 DB2® for z/OS® 或 DB2 for iSeries®, 则不能访存该登录数据。要避免错误，您需要将 InfoSphere CDC for solidDB 系统参数 **retrieve_credentials** 设置为 FALSE。
- 如果您将 InfoSphere CDC for solidDB 安装和预定从 V6.3 进行升级，那么 InfoSphere CDC for solidDB V6.3 不存储 SYS_TABLE 表中的后端登录数据。

在后面的两个事例中，使用 CREATE REMOTE SERVER（或 ALTER REMOTE SERVER）语句以手动方式定义了登录数据。

在缺省情况下，用户名和密码存储为大写字母。要区分大小写，请在单引号中输入用户名和密码。

示例

```
CREATE REMOTE SERVER USERNAME dba PASSWORD dba
CREATE REMOTE SERVER USERNAME 'AdMin' PASSWORD 'Pwd123'
CREATE OR REPLACE REMOTE SERVER USERNAME dba PASSWORD 'Pwd123'
```

另请参阅

第 145 页的 A.3, 『ALTER REMOTE SERVER』

第 203 页的 A.36, 『DROP REMOTE SERVER』

A.18 CREATE ROLE

CREATE ROLE *role_name*

用法

用 CREATE ROLE 语句创建新用户角色。

角色是一组可以作为一个单元授予用户的特权。您可以创建角色以及将用户指定到特定角色。可以对单一用户指定多个角色，也可以对单一角色指定多个用户。

注:

- 同一个字符串不能同时用作用户名和角色名。
- 一旦获得用户角色，获得该角色的用户登录到数据库时就会生效。授予角色时，如果该用户已登录到数据库，那么该角色直到用户断开连接并重新连接到数据库之后才会生效。

solidDB 提供以下系统角色。系统角色名称是保存的用户名。

表 32. 系统角色

保留的名称	描述
PUBLIC	此角色将特权授予所有用户。将特定表的用户特权授予 <i>PUBLIC</i> 角色之后，所有当前用户以及将来用户都将对这个表拥有指定的用户特权。系统自动授予所有用户此角色。
SYS_ADMIN_ROLE	这是数据库管理员的缺省角色。此角色对所有表、索引和用户拥有特权，并且有权使用“solidDB 远程控制”。这也是数据库创建者角色。
_SYSTEM	这是所有系统表和视图的模式名。
SYS_CONSOLE_ROLE	此角色有权使用“solidDB 远程控制”，但不具有其他管理特权。
SYS_SYNC_ADMIN_ROLE	这是数据同步功能的管理员角色。
SYS_SYNC_REGISTER_ROLE	此角色仅用于向主数据库注册或注销副本数据库。

示例

```
CREATE ROLE GUEST_USERS;
```

A.19 CREATE SCHEMA

CREATE SCHEMA *schema_name*

用法

CREATE SCHEMA 语句会创建新模式。

模式是数据库用户的数据库对象（例如表、视图、索引、事件、触发器、序列和存储过程）的集合。缺省模式名是用户标识；每个用户都有一个缺省模式。solidDB 中的模式使用遵循 SQL 标准。

模式名用于对数据库对象名进行限定。在所有 DML 语句中，都对数据库对象名进行限定，如下所示：

```
catalog_name.schema_name.database_object_name
```

或者

```
user_id.database_object_name
```

要以逻辑模式对数据库进行分区，用户可以在创建模式前创建目录。有关创建目录的详细信息，请参阅第 154 页的 A.9，『CREATE CATALOG』。在创建新数据库或者将旧数据库转换为新格式时，将提示您输入缺省目录名。

要使用模式，必须在创建数据库对象名（例如表名或过程名）之前创建模式名。但是，可以在没有模式名的情况下创建数据库对象名。在这种情况下，数据库对象仅由 user_id 限定。

要在 DML 语句中指定数据库对象名，可以通过对名称进行全限定以显式方式指定该名称，也可以通过使用以下语句设置模式名上下文以隐式方式指定该名称：

```
SET SCHEMA schema_name
```

创建模式并不会自动使该模式成为当前缺省模式。如果您已创建新模式并且要让后续命令在该模式中执行，还必须执行第 261 页的 A.79.9，『SET SCHEMA』语句。例如：

```
CREATE SCHEMA MySchema;  
CREATE TABLE t1; -- 不在 MySchema 中  
SET SCHEMA MySchema;  
CREATE TABLE t2; -- 在 MySchema 中
```

可以使用第 204 页的 A.39，『DROP SCHEMA』语句从数据库中删除模式。删除模式名时，必须在删除该模式之前删除所有与该模式名相关联的对象。

可以使用 SET SCHEMA USER 语句除去模式上下文。

模式名的解析规则：

- 标准名称（*schema_name.database_object_name*）不要求进行任何名称解析，但需要验证。
- 如果未使用 SET SCHEMA 来设置模式上下文，那么始终在使用用户标识作为模式名的情况下解析所有数据库对象名。
- 如果无法根据该模式名来解析数据库对象名，那么将根据所有现有的模式名来解析数据库对象名。
- 如果名称解析操作找不到或者找到多个匹配的数据库对象名，那么 solidDB 服务器将发出名称解析冲突错误。

示例

```
-- Assume the userID is SMITH.
CREATE SCHEMA FINANCE;
CREATE TABLE EMPLOYEE (EMP_ID INTEGER);
SET SCHEMA FINANCE;
CREATE TABLE EMPLOYEE (ID INTEGER);
SELECT ID FROM EMPLOYEE;
-- In this case, the table is qualified to FINANCE.EMPLOYEE
SELECT EMP_ID FROM EMPLOYEE;
-- This will give an error as the context is with FINANCE and
-- table is resolved to FINANCE.EMPLOYEE

--The following are valid schema statements: one with a schema context,
--the other without.
SELECT ID FROM FINANCE.EMPLOYEE;
SELECT EMP_ID FROM SMITH.EMPLOYEE
--The following statement will resolve to schema SMITH without a schema
--context
SELECT EMP_ID FROM EMPLOYEE;
```

A.20 CREATE SEQUENCE

```
CREATE [DENSE] SEQUENCE sequence_name
[START WITH constant] [INCREMENT BY constant]
```

用法

序列器对象是用于获取序号的对象。

DENSE 序列将确保各个序号之间不存在间隔。序号分配与当前事务绑定。如果事务回滚，那么序号分配也将回滚。紧密序列的缺点是，该序列在当前事务结束前不可供其他事务使用。

稀疏序列将确保所返回的值的唯一性，但这些值不与当前事务绑定。如果一个事务分配稀疏序号并接着回滚，那么该序号仅仅会丢失。

密集序列和稀疏序列均从 1 开始编号，除非 START WITH *constant* 另作指定。

序号是 8 字节的值。序列值可以存储在 BIGINT、BINARY 或 INT 数据类型中。

- 建议您使用 BIGINT。
- 8 字节 BINARY 值可以存储完整的序号，但将 BINARY 值作为整数数据类型使用时，并不是很方便。
- 由于 4 字节整数无法容纳 8 字节序号，因此存储在 INT 变量中的序列值将丢失信息。

当 8 字节数字序号存储为 4 字节整数时（在存储过程中或应用程序中），省略最高的 4 字节。当序号超过 $2^{31} - 1$ (=2147483647) 时，将导致数据丢失。

以下示例示范该行为：

```
CREATE SEQUENCE seq1;

-- Set the sequence number to 2^31 - 1,
-- then return that value and the "next" value (2^31).
"CREATE PROCEDURE set_seq1_to_2G
RETURNS (x INT, y INT)
BEGIN
DECLARE int1 INTEGER;
```

```

int1 := 2147483647;
EXEC SEQUENCE seq1 SET VALUE USING int1;
EXEC SEQUENCE seq1 CURRENT INTO x;
EXEC SEQUENCE seq1 NEXT INTO y;
END";

COMMIT WORK;

CALL set_seq1_to_2G();

```

此调用的返回值如下所示:

```

      x           y
2147483647      -2147483648

```

x 的值正确, 但 y 的值是负数, 而不是正确的正数。

使用序列器对象代替独立的表的优点是, 序列器对象已专门针对高速执行进行微调, 所需的开销低于常规 UPDATE 语句。

在 SQL 语句中, 可以对序列值进行递增以及使用这些值。由 INCREMENT BY *constant* 定义增量。

在 SQL 中, 还可以使用下列构造:

```

sequence_name.CURRVAL
sequence_name.NEXTVAL

```

要检索当前序列值, 需要“选择”访问权。要分配新的序列值, 需要“更新”访问权。这些访问权的授予和撤销方式与表访问权相同。

存储过程中的序列

此外, 还可以在存储过程中使用序列。

表 33. 存储过程中的序列

存储过程语句	用法
EXEC SEQUENCE <i>sequence_name</i> .CURRENT INTO <i>variable</i>	检索新序列值
EXEC SEQUENCE <i>sequence_name</i> .NEXT INTO <i>variable</i>	检索当前序列值
EXEC SEQUENCE <i>sequence_name</i> SET VALUE USING <i>variable</i>	设置序列值 提示: 另外, 您可以选择使用 SET SEQUENCE <name> VALUE <bigint value> 语句。

示例

```

CREATE DENSE SEQUENCE SEQ1;
INSERT INTO ORDER (id) VALUES (SEQ1.NEXTVAL);
INSERT INTO ORDER (id) VALUES (SEQ1.NEXTVAL);

"CREATE PROCEDURE get_my_seq
RETURNS (val INTEGER)
BEGIN
EXEC SEQUENCE my_sequence.NEXT INTO (val);
END";

```

A.21 CREATE SYNC BOOKMARK

```

CREATE SYNC BOOKMARK bookmark_name

```

用法

CREATE SYNC BOOKMARK 语句在主数据库中创建书签。该语句只在 solidDB 高级复制设置中有效。

书签 代表用户定义的数据库版本。它是 solidDB 数据库的持久快照，用于为执行特定同步任务而提供参考。书签通常用于通过 EXPORT SUBSCRIPTION 命令从主数据库中导出数据，以便将这些数据导入到副本数据库。通过导出和导入数据，您就能够在数据库大于 2GB 时更有效地根据主数据库创建副本数据库。

要创建书签，您必须具有管理 DBA 特权或者 SYS_SYNC_ADMIN_ROLE 角色。对于可以在数据库中创建的书签的数目而言，没有任何限制。您只能在主数据库中创建书签。如果尝试在副本数据库中创建书签，那么系统将发出错误。

如果已使用 第 147 页的 A.4.2, 『ALTER TABLE ... SET SYNCHISTORY』命令对表进行同步历史记录设置，那么书签将保留该表的历史记录信息。因此，不再需要书签之后，请使用 第 206 页的 A.42, 『DROP SYNC BOOKMARK』语句将其删除。否则，额外的历史记录数据将增加磁盘空间耗用量。

创建新书签时，系统将关联其他属性（例如书签的创建者、创建日期和时间以及唯一的书签标识）。此元数据在系统表 第 340 页的 E.2.10, 『SYS_SYNC_BOOKMARKS』中进行维护。

用于主数据库

使用 CREATE SYNC BOOKMARK 语句在主数据库中创建书签。

用于副本数据库

CREATE SYNC BOOKMARK 语句不可用于副本数据库。

示例

```
CREATE SYNC BOOKMARK BOOKMARK_AFTER_DATALOAD;
```

返回值

表 34. CREATE SYNC BOOKMARK 返回值

错误代码	描述
25066	书签已存在。
13047	无权执行操作。

A.22 CREATE TABLE

```
CREATE [ { [GLOBAL] TEMPORARY | TRANSIENT } ] TABLE base_table_name  
(column_element [, column_element] ...) [STORE {MEMORY | DISK}]  
[LIKE table_name]
```

```
base_table_name ::= base_table_identifier | schema_name.base_table_identifier |  
                  catalog_name.schema_name.base_table_identifier
```

```
column_element ::= column_definition | table_constraint_definition
```

```

column_definition ::= column_identifier
                    data_type [DEFAULT literal | NULL] [NOT NULL]
                    [column_constraint_definition [column_constraint_definition] ...]

column_constraint_definition ::= [CONSTRAINT constraint_name]
                                UNIQUE | PRIMARY KEY |
                                REFERENCES ref_table_name [(referenced_columns)] |
                                CHECK (check_condition)

table_constraint_definition ::= [CONSTRAINT constraint_name]
                                UNIQUE (column_identifier [, column_identifier] ...) |
                                PRIMARY KEY (column_identifier [, column_identifier] ...) |
                                CHECK (check_condition) |
                                {FOREIGN KEY (column_identifier [, column_identifier] ...)
                                REFERENCES table_name [(referenced_columns)]
                                [referential_triggered_action] }
referential_triggered_action ::=
                                ON {UPDATE | DELETE} {CASCADE | SET NULL | SET DEFAULT |
                                RESTRICT | NO ACTION}

```

用法

要创建表，请使用 CREATE TABLE 语句。CREATE TABLE 语句要求指定已创建的列的列表、数据类型、每个列中值的大小（如果适用）以及其他选项，例如创建主键。

STORE (表类型)

STORE 子句指示是应该将表存储在内存还是磁盘中。CREATE TABLE 语句优先于 **General.DefaultStoreIsMemory** 参数指定的存储位置。

内存表可以是持久（常规）表、临时表或瞬态表。

所有临时表和瞬态表都必须是内存表。您不需要指定 STORE MEMORY 子句；如果省略 STORE 子句，那么系统会自动将临时表和瞬态表创建为内存表。对于临时表和瞬态表而言，会忽略 **General.DefaultStoreIsMemory** 参数。如果您尝试以显式方式将临时表或瞬态表创建为基于磁盘的表，将发生错误。

为了遵循 SQL:1999 的临时表标准，支持关键字“GLOBAL”。在 solidDB 中，所有临时表都是全局表，而与是否使用了 GLOBAL 关键字无关。

基于磁盘的表和内存表的行大小

在基于磁盘的表中，行（不包括 BLOB）的最大大小约为页面大小的 1/3。在内存表中，行（包括 BLOB）的最大大小与页面大小大致相同。（磁盘页面和内存页面存在少量开销，因此所有页面不可用于用户数据。）用 **IndexFile.BlockSize** 参数定义页面大小（块大小），缺省值为 16 KB。

服务器并不使用简单规则来确定 BLOB 存储器，但一般规则是，每个 BLOB 在该行所在的页面中占用 256 字节，而该 BLOB 的其余内容则存储在单独的 BLOB 页面中。如果 BLOB 不足 256 字节，那么它将完全存储在主磁盘页面中，而不会存储在 BLOB 页面中。

每一行最多只能包含 1000 列。

引用完整性

创建表时，您应该始终定义主键。如果未定义主键，那么 solidDB 将自动创建一个主键。这将导致数据在磁盘上按意外的顺序排列，并可能导致性能下降。适当的主键能够提高使用该主键的查询的运行速度。

可以在列级和表级定义约束。对于列级，使用 NOT NULL 定义的约束指定在该列中不允许插入 NULL 值。UNIQUE 指定不允许任意两行具有相同的值。PRIMARY KEY 确保一个或多个列（它们是主键）不允许任意两行具有相同的值，并且不允许任何 NULL 值；因此，PRIMARY KEY 相当于 UNIQUE 与 NOT NULL 的组合。带有 FOREIGN KEY 的 REFERENCES 子句为引用完整性约束指定表名和一组列。这表示在这个表中插入或更新数据时，该数据必须与被引用表和被引用列中的值匹配。

CHECK 关键字用于对可以插入到某列中的值进行限制（例如，限制为只允许插入特定整数范围内的值）。如果定义了此关键字，那么检查约束功能将对该列中插入或更新的任何数据执行验证检查。如果数据违反约束，那么将不允许进行修改。例如：

```
CREATE TABLE table1 (salary DECIMAL CHECK (salary >= 0.0));
```

check_condition 是布尔表达式，用于指定列的检查约束。检查约束由谓词 >、<、=、<>、<=、>= 以及关键字 BETWEEN、IN、LIKE（可以包含通配符）和 IS [NOT] NULL 定义。表达式（与 WHERE 子句的语法类似）可以由关键字 AND 和 OR 进行限定。例如：

```
...CHECK (col1 = 'Y' OR col1 = 'N')...  
...CHECK (last_name IS NOT NULL)...
```

可以在列级或表级定义 UNIQUE 和 PRIMARY KEY 约束。它们还将对指定的列自动创建唯一索引。

FOREIGN KEY 用于指定所列示的列是此表中的外键。

REFERENCES 指定由外键所引用的表和列。虽然列级约束可以使用 REFERENCES 子句，但只有表级约束可以使用 FOREIGN KEY ... REFERENCES 子句。

要将 REFERENCES 约束与外键配合使用，外键的定义必须始终包含足够的列，以便唯一地标识被引用表中的某一行。外键包含的列的数目和类型（数据类型）必须与被引用表中的主键相同，并且顺序也必须相同；但是，外键的列名和缺省值可以与该主键不同。

以下规则适用于这些约束：

- *check_condition* 不能包含子查询、聚集函数、主变量或参数。
- 列检查约束只能引用它们的定义所基于的列。
- 如果先前已在语句中定义了该表中所有的列，那么表检查约束可以引用该表中的任何列。
- 表只能有一个主键约束，但可以有一个或多个唯一约束。
- CREATE TABLE 语句中的 UNIQUE 和 PRIMARY KEY 约束可以用于创建索引。在 ALTER TABLE 语句中，无法删除作为唯一键或主键组成部分的列。您可能想改为使用 CREATE INDEX 语句来创建索引 - 这样创建的索引将具有名称，并且您可以将其删除。CREATE INDEX 语句还提供了其他一些功能，例如能够创建非唯一索引以及指定索引是按升序顺序还是降序顺序排序。
- 持久表类型、瞬态表类型和临时表类型的引用完整性规则并不相同。

- 临时表可以引用另一个临时表，但是可能不会引用任何其他类型的表（瞬态表或持久表）。除临时表之外，其他类型的表都不能引用临时表。
- 瞬态表可以引用其他瞬态表和持久表。它们不能引用临时表。临时表和持久表都不能引用瞬态表。

创建和现有表相同的表

您可以使用 `CREATE new_table_name LIKE existing_table_name` 构造创建一个与现有表具有相同定义的表，而不是在 `CREATE TABLE` 语句中提供表定义。

例如:

```
CREATE TABLE MY_TABLE1(A INTEGER NOT NULL PRIMARY KEY, B INT) STORE MEMORY;
CREATE TABLE MY_TABLE2 LIKE MY_TABLE1;
```

以下信息将从现有的表复制到新表中:

- 列
- 列数据类型
- 缺省值
- NOT NULL 约束
- 主键
- 表类型（内存表或基于磁盘的表）

不复制以下信息:

- 数据
- 唯一键和辅键
- 外键
- 检查约束

要创建和现有表一样的表，您必须具有现有表的访问权和将表创建至当前目录和模式中的权利。

示例

```
CREATE TABLE DEPT (DEPTNO INTEGER NOT NULL, DNAME VARCHAR, PRIMARY KEY(DEPTNO));
CREATE TABLE DEPT2 (DEPTNO INTEGER NOT NULL PRIMARY KEY, DNAME VARCHAR);
CREATE TABLE DEPT3 (DEPTNO INTEGER NOT NULL UNIQUE, DNAME VARCHAR);
CREATE TABLE DEPT4 (DEPTNO INTEGER NOT NULL, DNAME VARCHAR, UNIQUE(DEPTNO));
CREATE TABLE EMP (DEPTNO INTEGER, ENAME VARCHAR, FOREIGN KEY (DEPTNO)
REFERENCES DEPT (DEPTNO)) STORE DISK;
CREATE TABLE EMP2 (DEPTNO INTEGER, ENAME VARCHAR, CHECK (ENAME IS NOT NULL),
FOREIGN KEY (DEPTNO) REFERENCES DEPT (DEPTNO)) STORE MEMORY;
CREATE GLOBAL TEMPORARY TABLE T1 (C1 INT);
CREATE TRANSIENT TABLE T2 (C1 INT);
```

A.23 CREATE TRIGGER

```
CREATE TRIGGER trigger_name ON table_name time_of_operation
triggering_event [REFERENCING column_reference]
BEGIN trigger_body END
```

其中:

```

trigger_name      ::= literal
table_name        ::= literal
time_of_operation ::= BEFORE | AFTER
triggering_event  ::= INSERT | UPDATE | DELETE
column_reference  ::= {OLD | NEW} column_name [AS] col_identifier
                  [, REFERENCING column_reference ]

trigger_body ::=
                [declare_statement;...]
                [trigger_statement;...]

old_column_name   ::= literal
new_column_name   ::= literal
col_identifier    ::= literal

```

用法

触发器提供用于在特定操作（INSERT、UPDATE 或 DELETE）发生时执行一系列 SQL 语句的机制。触发器主体包括用户想执行的 SQL 语句。触发器主体使用存储过程语言编写而成（请参阅 第 163 页的 A.14, 『CREATE PROCEDURE』）。

您可以在表上创建一个或更多的触发器，每个触发器都定义为用于激活特定的 INSET、UPDATE 或 DELETE 命令。当用户修改表中的数据时，将激活与该命令相对应的触发器。

只能将直接插入 SQL 或存储过程与触发器配合使用。如果您在该触发器中使用存储过程，则必须用 CREATE PROCEDURE 创建该过程。从触发器主体中调用的过程可以调用其他触发器。

要创建触发器，您必须是 DBA 或者正在定义的触发器所基于的表的所有者。

触发器由 CREATE TRIGGER 语句创建，可用 DROP TRIGGER 语句从系统目录中删除触发器。也可以用 ALTER TRIGGER 语句禁用触发器。

示例

```

"CREATE TRIGGER TRIGGER_BI ON TRIGGER_TEST
  BEFORE INSERT
  REFERENCING NEW BI AS NEW_BI
BEGIN
  EXEC SQL PREPARE BI INSERT INTO TRIGGER_OUTPUT VALUES (
    'BI', TRIG_NAME(0), TRIG_SCHEMA(0));
  EXEC SQL EXECUTE BI;
  SET NEW_BI = 'TRIGGER_BI';
END";

```

相关参考

第 149 页的 A.5, 『ALTER TRIGGER』

第 208 页的 A.44, 『DROP TRIGGER』

相关信息

第 59 页的 3.3.3, 『触发器与过程』

第 69 页的 3.3.9, 『触发器示例』

A.23.1 trigger_name

trigger_name 识别触发器。它至多包含 254 个字符。

A.23.2 BEFORE | AFTER 子句

BEFORE | AFTER 子句指定是在调用 DML 语句之前还是之后执行触发器。在某些情况下，BEFORE 和 AFTER 子句可互换。但是，在另一些情况下，其中一个子句优先于另一个子句。

- 在执行数据验证时，例如检查域约束和引用完整性时，使用 BEFORE 子句更有效率。
- 使用 AFTER 子句时，将处理由于调用 DML 语句而变为可用的表行。相反，AFTER 子句还在调用 DELETE 语句后确认数据删除。

对于每个表，可以定义多达 6 个触发器，即，为操作（INSERT、UPDATE 和 DELETE）与时间（BEFORE 和 AFTER）的每个组合定义一个触发器：

- BEFORE INSERT
- BEFORE UPDATE
- BEFORE DELETE
- AFTER INSERT
- AFTER UPDATE
- AFTER DELETE

以下示例显示对 table1 定义的 BEFORE INSERT 触发器 trig01。

```
"CREATE TRIGGER TRIG01 ON table1
  BEFORE INSERT
  REFERENCING NEW COL1 AS NEW_COL1
BEGIN
  EXEC SQL PREPARE CUR1
    INSERT INTO T2 VALUES (?);
  EXEC SQL EXECUTE CUR1 USING (NEW_COL1);
  EXEC SQL CLOSE CUR1;
  EXEC SQL DROP CUR1;
END"
```

以下是对每个 DML 操作使用 CREATE TRIGGER 命令的 BEFORE 和 AFTER 子句的示例（包括含义和优点）：

- UPDATE 操作

在处理 UPDATE 之前，BEFORE 子句可以验证修改后的数据是否遵循完整性约束规则。如果将 REFERENCING NEW AS *new_column_identifier* 子句与 BEFORE UPDATE 子句配合使用，那么更新后的值可供所触发的 SQL 语句使用。在触发器中，您可以在执行 UPDATE 前设置缺省列值或派生的列值。

AFTER 子句可以对新修改的数据执行操作。例如，更新分公司的地址后，可以计算该分公司的销售额。

如果将 REFERENCING OLD AS *old_column_identifier* 子句与 AFTER UPDATE 子句配合使用，那么调用更新前存在的值可供所触发的 SQL 语句使用。

- INSERT 操作

在执行 INSERT 之前，BEFORE 子句可以验证新数据是否遵循完整性约束规则。对于所触发的 SQL 语句，作为参数传递的列值可视，但插入的行不可视。在触发器中，您可以在执行 INSERT 前设置缺省列值或派生的列值。

AFTER 子句可以对新插入的数据执行操作。例如，插入销售订单后，可以计算总订购量以确定客户是否符合折扣条件。

对于所触发的 SQL 语句，作为参数传递的列值以及插入的行可视。

- DELETE 操作

BEFORE 子句可以对将要删除的数据执行操作。对于所触发的 SQL 语句，作为参数传递的列值以将要删除的已插入行可视。

AFTER 子句可用于确认删除数据。对于所触发的 SQL 语句，作为参数传递的列值可视。对于触发 SQL 语句，删除的行可视。

A.23.3 INSERT | UPDATE | DELETE 子句

INSERT | UPDATE | DELETE 子句指示当尝试用户操作（INSERT、UPDATE 和 DELETE）时要执行的触发器操作。

与处理触发器相关的语句在由于对表调用 DML（INSERT、UPDATE 和 DELETE）语句而引起的落实和自动落实之前执行。如果触发器主体或者触发器主体中调用的过程尝试执行 COMMIT 或 ROLLBACK，那么 solidDB 将返回相应的运行时错误。

INSERT 指定该触发器由对表执行的 INSERT 激活。系统会将装入 N 行数据的行为视为 N 次插入。

注:

如果尝试在触发器处于启用状态的情况下装入数据，那么可能会对性能产生一定影响。根据业务需求的不同，您可能希望先禁用触发器，接着装入数据，然后在装入完成后再次启用触发器。有关详细信息，请参阅第 149 页的 A.5，『ALTER TRIGGER』。

DELETE 指定该触发器由对表执行的 DELETE 激活。

UPDATE 指定该触发器由对表执行的 UPDATE 激活。使用该 UPDATE 子句时应遵循以下规则:

- 在触发器的 REFERENCES 子句中，不能在 BEFORE 次子句中多次引用某个列（指定该列的别名）并在 AFTER 次子句中引用该列一次。并且，如果同时在 BEFORE 和 AFTER 次子句中引用该列，那么在这两个次子句中，该列的别名不能相同。
- solidDB 服务器允许对同一个表执行递归更新，并且不禁止对同一行执行递归更新。

solidDB 服务器不会检测不同触发器的操作导致更新相同数据的情况。例如，假定对表 Table1 中不同的列 Col1 和 Col2 定义了两个更新触发器（一个是 BEFORE 触发器，另一个是 AFTER 触发器）。尝试对 Table1 中所有的列执行更新时，将激活这两个触发器都。这两个触发器都调用存储过程，那些存储过程将更新第二个表 Table2 中的同一个列 Col3。第一个触发器将 Table2.Col3 更新为 10，第二个触发器将 Table2.Col3 更新为 20。

同样，当激活触发器的 UPDATE 的结果与触发器本身的操作有冲突时，solidDB 服务器不检测这一情况。例如，考虑以下 SQL 语句:

```
UPDATE t1 SET c1 = 20 WHERE c3 = 10;
```

如果触发器由此 UPDATE 激活，然后调用包含以下 SQL 语句的过程，则该过程将覆盖激活该触发器的 UPDATE 的结果：

```
UPDATE t1 SET c1 = 17 WHERE c1 = 20;
```

注：以上示例可能会导致以递归方式执行触发器，您应该尝试避免这种情况。

A.23.4 table_name

table_name 是创建的触发器所基于的表的名称。solidDB 服务器允许删除对其定义了从属触发器的表。删除表时，还将删除所有从属对象，其中包括触发器。然而仍可能会发生运行时错误。例如，假定您创建了两个表 A 和 B。如果过程 SP-B 将数据插入表 A，然后删除表 A，并且表 B 包含需要调用 SP-B 的触发器，那么用户将接收到运行时错误。

A.23.5 trigger_body

trigger_body 包含触发器触发时要执行的语句。*trigger_body* 定义相当于存储过程定义。有关创建存储过程主体的详细信息，请参阅第 163 页的 A.14，『CREATE PROCEDURE』。

注：创建包含空主体的触发器在语法上有效，尽管用处不大。

触发器主体还可以调用任何向 solidDB 服务器注册的过程。solidDB 过程调用规则遵循标准的过程调用实践。

您必须显式地检查业务逻辑错误并引发错误。

A.23.6 REFERENCING 子句

创建基于 INSERT/UPDATE/DELETE 操作的触发器时，REFERENCING 子句可选。此子句提供了一种方法来引用当前列标识（对于 INSERT 和 DELETE 操作），并通过指定 UPDATE 操作所应用于的列的别名来引用旧列标识以及经过更新的新列标识。

您必须指定 OLD 或 NEW *column_identifier* 才能对其进行访问。除非您使用 REFERENCING 次子句来定义 *column_identifier*，否则 solidDB 服务器不允许对其进行访问。

{OLD | NEW} column_name AS col_identifier

REFERENCING 子句的 {OLD | NEW} *column_name* AS *col_identifier* 次子句允许您在执行 UPDATE 操作前后引用列值。这将生成一组旧列值和新列值，这些值可以传递到一个存储过程；一旦进行传递，该过程就包含用于确定这些参数值的逻辑（例如，域约束检查）。

使用 OLD AS 子句来指定 UPDATE 之前存在的旧表标识的别名。使用 NEW AS 子句来指定 UPDATE 之后存在的新表标识的别名。

如果同时引用同一个列的旧值和新值，那么必须使用不同的 *column_identifiers*。

每个作为 NEW 或 OLD 引用的列都应该有一个不同的 REFERENCING 次子句。

触发器中的语句原子性确保触发器中执行的操作对该触发器中的后续 SQL 语句可视。例如，如果在触发器中执行 INSERT 语句，然后还在同一个触发器中执行 SELECT，那么插入的行将可视。

对于 AFTER 触发器而言，插入或更新的行将在 AFTER 插入触发器中可视，但删除的行对于该触发器中执行的 SELECT 而言不可视。对于 BEFORE 触发器而言，插入或更新的行在该触发器中不可视，但删除的行可视。对于 UPDATE 而言，更新前的值在 BEFORE 触发器中可用。

下表对触发器中的语句原子性作了摘要，指示了该行对触发器主体中的 SELECT 语句是否可视。

表 35. 触发器中的语句原子性

操作	BEFORE 触发器	AFTER 触发器
INSERT	行不可视	行可视
UPDATE	先前值不可视	新值可视
DELETE	行可视	行不可视

A.23.7 触发器用途注释和限制

- 要使用触发器所调用的存储过程，请提供该触发器的定义所基于的表的目录、模式/所有者和名称，并指定是对该表启用还是禁用触发器。
- 要对表创建触发器，您必须具有 DBA 权限或者是正在定义的触发器所基于的表的所有者。
- 缺省情况下，对于表、操作（INSERT、UPDATE 和 DELETE）与时间（BEFORE 和 AFTER）的每个组合，最多可以定义一个触发器。这表示每个表最多可以有 6 个触发器。

注： 触发器将应用于每一行。这意味着，如果执行 10 次插入，那么触发器将执行 10 次。

- 不能对视图定义触发器（即使该视图基于单个表亦如此）。
- 如果将会影响到触发器所依赖的列，那么不能更改触发器定义所基于的表。
- 不能对系统表创建触发器。
- 不能执行引用了已删除或已更改的对象的触发器。为了预防此错误，请执行下列操作：
 - 重新创建任何已删除的引用的对象。
 - 将任何已更改的引用的对象复原到触发器所知的原始状态。
- 在触发器语句中，可以使用保留字，但必须将其引在双引号中。例如，以下 CREATE TRIGGER 语句引用了名为“DATA”的列（DATA 是一个保留字）。

```
"CREATE TRIGGER TRIG1 ON TMPT BEFORE INSERT
REFERENCING NEW "DATA" AS NEW_DATA
BEGIN
END"
```


相关信息

第 59 页的 3.3.3, 『触发器与过程』

第 61 页的 3.3.4, 『触发器与事务』

A.24 CREATE USER

```
CREATE USER user_name IDENTIFIED BY password
```

用法

CREATE USER 语句创建具有给定密码的新用户。要执行该语句, 您必须有管理员特权。

示例

以下示例创建了名为 HOBBS、密码为 CALVIN 的用户。

```
CREATE USER HOBBS IDENTIFIED BY CALVIN;
```

相关参考

第 149 页的 A.6, 『ALTER USER』

第 209 页的 A.45, 『DROP USER』

A.25 CREATE VIEW

```
CREATE VIEW viewed_table_name [( column_identifier  
    [, column_identifier ]... )]  
AS query-specification
```

用法

用 CREATE VIEW 语句创建一个视图。视图如同一个虚拟表, 可以提供不同的方式查看一个或多个表中的数据; 可将其称为结果表规范。该规范是一个 SELECT 语句, 每当在 SQL 语句中引用视图时, 都会运行该语句。视图中有列和行, 和表一样。所有视图都可以像表那样用于数据检索。

示例

```
CREATE VIEW TEST_VIEW  
    (VIEW_I, VIEW_C, VIEW_ID)  
AS SELECT I, C, ID FROM TEST;
```

相关参考

第 209 页的 A.46, 『DROP VIEW』

A.26 DELETE

```
DELETE FROM table_name [WHERE search_condition]  
DELETE FROM table_name WHERE CURRENT OF cursor_name
```

用法

共有两种 DELETE 语句:

- 已搜索 DELETE 语句, DELETE FROM *table_name* [WHERE *search_condition*], 删除给定表中的所有行或 第 282 页的 A.87.5, 『搜索条件』 指定的行。
- 已定位 DELETE 语句, DELETE FROM *table_name* WHERE CURRENT OF *cursor_name*, 删除游标的当前行。

table_name 可以是表名, 也可以是别名。

已搜索 DELETE 的示例

```
DELETE FROM TEST WHERE ID = 5;
DELETE FROM TEST;
```

已定位 DELETE 的示例

```
DELETE FROM TEST WHERE CURRENT OF MY_CURSOR;
```

A.27 DESCRIBE

```
DESC[RIBE] [type option] {[context_specifier.]object} [format option]
```

其中:

```
type option := CATALOG | SCHEMA | TABLE | VIEW | PROCEDURE | FUNCTION | TRIGGER
```

```
context_specifier := catalog | [catalog.]schema | [[catalog.]schema.]table
```

```
object := catalog | schema | table | view | procedure | function | trigger
```

```
format option := [NICE (default) | RAW ]
```

用法

DESCRIBE 语句描述数据库对象 (例如目录、模式、表、视图、过程或函数): 它输出指定表或视图的列定义, 或者指定函数或过程的规范。

DESCRIBE 的结果取决于输出格式: NICE 或 RAW。

- NICE 格式简要且简化。它可用于对数据库的元数据进行快速临时查询。
- RAW 格式更具有针对性且更详细。它输出的信息通常可通过使用 solidDB Data Dictionary (soldd) 或复杂 SQL 查询获取。

NICE 和 RAW 格式均不适用于所有类型。

- DESCRIBE CATALOG *my_catalog* 采用 NICE 格式输出指定目录的表名、创建者、所有者和模式列表。
- DESCRIBE SCHEMA *my_schema* 采用 NICE 格式输出模式名称、它所属的目录名、创建者和所有者。
- DESCRIBE TABLE *my_table* 采用 NICE 格式输出表名称、所用的存储器类型、父表列表、子表列表、列、列类型、列精度、列可空性、列是否为主键以及列是否为辅键。在 RAW 格式中, 将与相关索引及约束的创建语句一起输出该创建语句。
- DESCRIBE VIEW *my_view* 采用 NICE 格式输出视图的列, 采用 RAW 格式输出创建语句。
- DESCRIBE PROCEDURE *my_procedure* 采用 RAW 格式输出创建语句。
- DESCRIBE FUNCTION *my_function* 采用 RAW 格式输出创建语句。
- DESCRIBE TRIGGER *my_trigger* 采用 RAW 格式输出创建语句。

注：在许多情况下，只有一种输出格式可用。在这些情况下，不管用户指定哪种输出格式，都会产生唯一可用的输出。

示例

以下示例采用 NICE 格式在 PERSON 表中输出信息。

```
DESCRIBE PERSON;
Table name : PERSON
Table type : In-memory
Memory usage: 7935 KB (total), 7925 KB (active), 6192 KB (rows), 1733 KB (indexes).

Parent tables : CITY
Child tables  : OWNER, IN_RELATION

COLUMN      TYPE      PRECISION  NULLABLE  PRIMARY KEY  SECONDARY KEY
-----
FIRSTNAME   VARCHAR   30         YES       NO           NULL
LASTNAME    VARCHAR   60         NO        YES          NULL
HOME_CITY   INT       NULL       YES       NO           'CITY'
```

1 rows fetched.

以下示例采用 RAW 格式输出名为 PERSON 的表的信息。

```
DESCRIBE PERSON RAW;

CREATE TABLE "DBA"."DBA"."PERSON" (
  "FIRSTNAME" VARCHAR(30),
  "LASTNAME"  VARCHAR(60) NOT NULL,
  "HOME_CITY" INTEGER,
  FOREIGN KEY("HOME_CITY")
    REFERENCES "DBA"."DBA"."CITY"("CITY"),
  PRIMARY KEY ("LASTNAME")
);

1 rows fetched.
```

另请参阅

第 224 页的 A.56, 『LIST』

A.28 DROP CATALOG

```
DROP CATALOG catalog_name [CASCADE | RESTRICT]
```

用法

DROP CATALOG 语句从数据库中删除指定的目录。

如果您没有指定 RESTRICT 或 CASCADE，则在使用此语句之前，必须删除目录中的所有数据库对象。

RESTRICT 关键字表示只有之前已删除目录中的所有数据库对象时，才可以删除该目录。

CASCADE 关键字表示如果目录包含数据库对象（例如表），将自动删除这些对象。如果您使用了 CASCADE 关键字，并且其他目录中的对象引用了目录中正删除的对象，则通过删除那些引用对象或者将其更新为消除引用来自动解决那些引用。

只有数据库创建者或有 SYS_ADMIN_ROLE 特权的用户才能创建或删除目录。如果不具有 SYS_ADMIN_ROLE 特权，即便是该目录的创建者也不能删除该目录。

示例

```
DROP CATALOG C1;  
DROP CATALOG C2 CASCADE;  
DROP CATALOG C3 RESTRICT;
```

相关参考

第 154 页的 A.9, 『CREATE CATALOG』

A.29 DROP EVENT

```
DROP EVENT event_name  
DROP EVENT [[catalog_name.]schema_name.]event_name
```

用法

DROP EVENT 语句从数据库中除去用户定义的事件。无法删除系统事件。

示例

```
DROP EVENT event_test;
```

以下示例使用目录、模式、和事件名称来指定要断开的事件。

```
DROP EVENT HR_database.smith_schema.event1;
```

相关参考

第 156 页的 A.10, 『CREATE EVENT』

A.30 DROP FUNCTION

```
DROP FUNCTION [catalog-name[.schema].]function_name
```

用法

DROP FUNCTION 语句移除指定的用户定义的存储 SQL 函数或来自数据库的外部函数。

相关参考

第 157 页的 A.11, 『CREATE FUNCTION』

第 159 页的 A.12, 『CREATE FUNCTION (外部)』

A.31 DROP INDEX

DITA

```
DROP INDEX index_name  
DROP INDEX [[catalog_name.]schema_name.]index_name
```

用法

DROP INDEX 语句从数据库中除去指定的索引。

无法除去主键索引。

示例

```
DROP INDEX test_index;  
-- Using catalog, schema, and index name  
DROP INDEX bank_accounts.bankteller.first_name_index;
```

A.32 DROP MASTER

```
DROP MASTER master_name
```

用法

DROP MASTER 语句用于从副本数据库中删除主数据库定义。执行此断开后，副本数据库将无法与主数据库进行同步。DROP master 语句仅可以使用于副本数据库中。

如果 *master_name* 是保留字，那么必须将其引在双引号中。

要使用 DROP MASTER 语句，必须开启自动落实设置。

注：停止使用主数据库的首选方法是注销副本数据库。仅当无法执行 MESSAGE APPEND UNREGISTER REPLICA 语句时，才应该使用 DROP MASTER 语句。有关注销副本数据库的详细信息，请参阅第 228 页的 A.58，『MESSAGE APPEND』。

返回值

表 36. DROP MASTER 返回值

错误代码	描述
13047	无权执行操作。
25007	找不到主数据库 <i>master_name</i> 。
25019	此数据库不是副本数据库。
25056	不允许自动落实。
25065	在主数据库 <i>master_name</i> 中找到未完成的消息 <i>message_name</i> 。

示例

```
DROP MASTER "MASTER";  
DROP MASTER MY_MASTER;
```

A.33 DROP PROCEDURE

```
DROP PROCEDURE procedure_name  
DROP PROCEDURE [[catalog_name.]schema_name.]procedure_name
```

用法

DROP PROCEDURE 语句从数据库中除去指定的过程。

示例

```
DROP PROCEDURE PROCTEST;  
  
DROP PROCEDURE telecomm_database.technician1.add_new_IP_address;
```

A.34 DROP PUBLICATION

DROP PUBLICATION *publication_name*

用法

DROP PUBLICATION 语句删除主数据库中的发布定义。此外，还将自动删除所有对已删除的发布进行的预订。

用于主数据库

在主数据库中发布时，从主数据库中除去发布定义并且副本数据库不可以从中刷新发布定义。

用于副本数据库

不需要在副本数据库中定义发布，因此 DROP PUBLICATION 语句一般不会发出到副本数据库上。如果您已在副本数据库上创建发布，DROP PUBLICATION 将从副本数据库中除去发布定义。

返回值

表 37. DROP PUBLICATION 返回值

错误代码	描述
25010	找不到发布 <i>publication_name</i> 。
13111	不明确实体名 <i>name</i>

示例

```
DROP PUBLICATION customers_by_area;
```

相关参考

第 180 页的 A.16, 『CREATE [OR REPLACE] PUBLICATION』

A.35 DROP PUBLICATION REGISTRATION

DROP PUBLICATION *publication_name* REGISTRATION

用法

DROP PUBLICATION REGISTRATION 语句用于从副本数据库中删除某个发布的注册。该发布定义仍存在于主数据库中，但用户将无法根据该发布进行刷新。此外，还将自动删除所有对已注册的发布进行的预订。

DROP PUBLICATION REGISTRATION 语句仅能发出在副本数据库上。

返回值

表 38. *DROP PUBLICATION REGISTRATION* 返回值

错误代码	描述
13047	无权执行操作。
25019	此数据库不是副本数据库。
25025	未定义节点名。
25071	未向发布 <i>publication_name</i> 注册。

示例

```
DROP PUBLICATION customers_by_area REGISTRATION;
```

A.36 DROP REMOTE SERVER

DROP REMOTE SERVER

用法

DROP REMOTE SERVER 语句会除去 SYS_SERVER 系统表中的后端登录信息。登录数据用于 Universal Cache 中的 SQL 传递。

另请参阅

第 183 页的 A.17, 『CREATE [OR REPLACE] REMOTE SERVER』

第 145 页的 A.3, 『ALTER REMOTE SERVER』

A.37 DROP REPLICA

DROP REPLICA *replica_name*

用法

DROP REPLICA 语句用于从主数据库中删除副本数据库。执行此操作后，所删除的副本数据库将无法与主数据库进行同步。DROP REPLICA 语句仅能发出在主数据库上。

如果 *replica_name* 是保留字，那么必须将其引在双引号中。

要使用 DROP REPLICA 语句，必须开启自动落实设置。

注：停止使用副本数据库的首选方法是注销副本数据库。仅当无法执行 MESSAGE APPEND UNREGISTER REPLICA 语句时，才应该使用 DROP REPLICA 语句。有关注销副本数据库的详细信息，请参阅第 228 页的 A.58, 『MESSAGE APPEND』。

返回值

表 39. *DROP REPLICA* 返回值

错误代码	描述
13047	无权执行操作。
25009	找不到副本数据库 <i>replica_name</i> 。
25020	此数据库不是主数据库。
25056	不允许自动落实。
25064	在副本数据库 <i>replica_name</i> 中找到未完成的消息 <i>message_name</i> 。

示例

```
DROP REPLICA salesman_smith ;  
DROP REPLICA "REPLICA";
```

A.38 DROP ROLE

```
DROP ROLE role_name
```

用法

`DROP ROLE` 语句从数据库中除去指定的角色。

您不能断开 solidDB 系统角色（`SYS_ADMIN_ROLE` 等等）。

示例

```
DROP ROLE GUEST_USERS;
```

A.39 DROP SCHEMA

```
DROP SCHEMA schema_name [CASCADE | RESTRICT]  
DROP SCHEMA [catalog_name.] schema_name [CASCADE | RESTRICT]
```

用法

`DROP SCHEMA` 语句从数据库中删除指定的模式。

如果您不指定 `RESTRICT` 或 `CASCADE` 关键字，那么在使用此语句之前，必须删除所有与指定的 *schema_name* 相关联的对象。

`RESTRICT` 关键字表示只有在使用此语句之前删除所有与指定的 *schema_name* 相关联的对象，才可以删除模式。

`CASCADE` 关键字表示将自动删除所指定模式中的所有数据库对象（例如表）。如果使用了 `CASCADE` 关键字，并且其他模式中的对象引用了模式中正删除的对象，那么将通过删除那些引用对象或者将其更新为消除引用来自动解决那些引用。

示例

```
DROP SCHEMA finance;
DROP SCHEMA finance CASCADE;
DROP SCHEMA finance RESTRICT;
DROP SCHEMA forecasting_db.securities_schema CASCADE;
```

A.40 DROP SEQUENCE

```
DROP SEQUENCE sequence_name
DROP SEQUENCE [[catalog_name.]schema_name.]sequence_name
```

用法

DROP SEQUENCE 语句从数据库中除去指定的序列。

示例

```
DROP SEQUENCE SEQ1;
-- Using catalog, schema, and sequence name
DROP SEQUENCE bank_db.checking_acct_schema.account_num_seq;
```

A.41 DROP SUBSCRIPTION

用于副本数据库:

```
DROP SUBSCRIPTION publication_name [{(parameter_list) | ALL}]
[COMMITBLOCK number_of_rows] [OPTIMISTIC | PESSIMISTIC]
```

用于主数据库:

```
DROP SUBSCRIPTION publication_name [{(parameter_list) | ALL}]
REPLICA replica_name
```

用法

DROP SUBSCRIPTION 语句可通过删除用于检索主数据库数据的预订，来删除副本数据库中不再需要的数据。要发出 DROP SUBSCRIPTION 语句，必须将自动落实设置为关闭。

当在主数据库上发出时，DROP SUBSCRIPTION 语句移除指定副本的预订。

当在副本数据库上发出时，DROP SUBSCRIPTION 语句移除该副本数据库上的预订。

缺省情况下，将通过一个事务来删除预订的数据。如果数据量较大，例如达到数万行，那么建议您定义 COMMITBLOCK。如果使用了 COMMITBLOCK 选项，那么将通过不止一个事务来删除数据。这将确保此操作的性能优良。

在副本数据库中，您可以定义 DROP SUBSCRIPTION 语句在最初执行时使用表级悲观锁定方式。如果指定了悲观方式，那么直到删除操作完成为止，才会解除对受影响的表进行的所有其他并发访问的阻塞。如果使用了乐观方式，则 DROP SUBSCRIPTION 可能会因为并行冲突而失败。

此外，也可以从主数据库中删除预订。在这种情况下，副本数据库名称将包括在命令中。您可以在 SYS_SYNC_REPLICAS 表中找到所有已在主数据库中注册的副本数据库的名称。此操作将只删除有关此副本数据库的预订的内部信息。副本数据库中的实际数据将保留不变。

如果副本数据库不再使用某个预订，并且副本数据库尚未删除该预订本身，那么最好从主数据库中删除该预订。删除旧预订将释放数据库中的旧历史记录数据。删除预订后，将自动地从主数据库中删除此历史记录数据。

如果已从主数据库中删除副本数据库的预订，那么该副本数据库下次刷新时将接收到全部数据。

在这种情况下删除预订时，如果已删除对发布的最后一个预订，那么 `DROP SUBSCRIPTION` 还将删除发布注册。否则，必须使用 `DROP PUBLICATION REGISTRATION` 语句或 `MESSAGE APPEND UNREGISTER PUBLICATION` 显式地删除注册。

可以将 `DROP SUBSCRIPTION` 语句定义为最初执行时使用表级悲观锁定方式。如果指定了悲观方式，那么直到导入操作完成为止，才会解除对受影响的表进行的所有其他并发访问的阻塞。如果使用了乐观方式，则 `DROP SUBSCRIPTION` 可能会因为并行冲突而失败。

当一个事务获取对表的互斥锁定时，`General.TableLockWaitTimeout` 参数设置决定该事务在互斥锁定或共享锁定释放前等待的时间。

返回值

表 40. `DROP SUBSCRIPTION` 返回值

错误代码	描述
13047	无权执行操作。
25004	不支持动态参数。
25009	找不到副本数据库 <code>replica_name</code> 。
25010	找不到发布 <code>publication_name</code> 。
25019	此数据库不是副本数据库。
25020	此数据库不是主数据库。
25041	找不到对发布 <code>publication_name</code> 的预订。
25056	不允许自动落实。

示例

从主数据库中删除预订：

```
DROP SUBSCRIPTION customers_by_area('south')
FROM REPLICA salesman_joe
```

A.42 DROP SYNC BOOKMARK

`DROP SYNC BOOKMARK bookmark_name`

用法

DROP SYNC BOOKMARK 语句删除定义在主数据库上的书签。将数据导出到文件时，通常使用书签。在成功地将文件从主数据库导入到副本数据库之后，建议您删除用于将数据导出到文件的书签。

要删除书签，您必须具有管理特权（SYS_ADMIN_ROLE 或 SYS_SYNC_ADMIN_ROLE）。DROP SYNC BOOKMARK 语句不可用于副本数据库。

如果保留书签，那么在主数据库中，将为每个书签跟踪主数据库中对数据所作的所有后续更改（包括删除和更新），以便执行增量刷新。

如果未删除书签，那么对于主数据库中注册的每个书签，历史记录信息将占用磁盘空间，并且将引发不必要的磁盘 I/O。这可能会导致性能下降。

注：只有将导出的数据导入到所有期望的副本数据库之后，并且所有副本数据库都至少进行一次同步之后，才应该删除书签。您应该仅当已没有要执行导入的副本数据库，并且那些副本数据库在导入后已根据该发布执行一次刷新之后，才删除书签。

删除书签时，solidDB 将使用下列规则来删除历史记录：

- 查找对该表发送到任何副本数据库的最旧 REFRESH。
- 查找最旧的书签。
- 确定最旧的 REFRESH 与最旧的书签哪个更旧。
- 删除历史记录中直到所确定的较旧者（最旧的 REFRESH 或最旧的书签）为止的所有行。

返回值

表 41. DROP SYNC BOOKMARK 返回值

错误代码	描述
25067	找不到同步化器书签 <i>bookmark_name</i> 。
13047	无权执行操作。

示例

```
DROP SYNC BOOKMARK new_database;  
DROP SYNC BOOKMARK database_after_dataload;
```

A.43 DROP TABLE

```
DROP TABLE base_table_name [CASCADE [CONSTRAINTS]]  
DROP TABLE [[catalog_name.]schema_name.]table_name [CASCADE  
[CONSTRAINTS]]
```

用途

DROP TABLE 语句从数据库中除去指定的表。

[CASCADE [CONSTRAINTS]] 表示引用在删除的表中除去主要、唯一键的所有引用完整性约束已除去。表示从引用已删除表的表格中除去所有外键规范。然而，在引用表中没有更改列值。

注：缺省情况下，用 RESTRICT 类型删除行为删除对象，即正如把 RESTRICT 引用操作定义在相关外建中。

然而，存在以下异常情况：

- 如果该表有同步历史记录表，那么将自动删除该同步历史记录表。
- 如果该表有索引，那么无需事先删除那些索引；在删除表时，将自动删除那些索引。

示例

```
DROP TABLE table1;
-- Using catalog, schema, and table name
DROP TABLE domains_db.demand_schema.bad_address_table;
--remove foreign key constraints in referencing tables
DROP TABLE table2 CASCADE CONSTRAINTS;
```

A.44 DROP TRIGGER

```
DROP TRIGGER [[catalog_name.]schema_name.]trigger_name
```

用法

DROP TRIGGER 语句从系统目录中断开（或删除）对表定义的触发器。要删除表的触发器，您必须是该表的所有者或者拥有 DBA 权限。

示例：删除触发器

```
DROP TRIGGER update_acct_balance;
-- Using schema and trigger name
DROP TRIGGER savings_accounts.update_acct_balance;
-- Using catalog, schema, and trigger name
DROP TRIGGER accounts.savings_accounts.update_acct_balance;
```

示例：删除并重新创建触发器

```
DROP TRIGGER TRIGGER_BI;
COMMIT WORK;

"CREATE TRIGGER TRIGGER_BI ON TRIGGER_TEST
BEFORE INSERT
REFERENCING NEW BI AS NEW_BI
BEGIN
EXEC SQL PREPARE BI_INSERT INTO TRIGGER_OUTPUT VALUES(
'BI_NEW', TRIG_NAME(0), TRIG_SCHEMA(0));
EXEC SQL EXECUTE BI;
SET NEW_BI = 'TRIGGER_BI_NEW';
END";
COMMIT WORK;

INSERT INTO TRIGGER_TEST(XX) VALUES ('XX');
COMMIT WORK;

SELECT * FROM TRIGGER_TEST;
SELECT * FROM TRIGGER_OUTPUT;
COMMIT WORK;
```

A.45 DROP USER

```
DROP USER user_name
```

用途

DROP USER 语句从数据库中除去指定的用户。在使用此语句之前，必须删除所有与指定的 *user_name* 相关联的对象；DROP USER 语句不是级联操作。

示例

```
DROP USER HOBBS;
```

A.46 DROP VIEW

```
DROP VIEW view_name  
DROP VIEW [[catalog_name.]schema_name.]view_name
```

用法

DROP VIEW 语句从数据库中除去指定的视图。

示例

```
DROP VIEW sum_of_acct_balances;  
-- Using schema and view name  
DROP VIEW acct_manager_schema.sum_of_acct_balances;  
-- Using catalog, schema, and view name  
DROP VIEW account_db.acct_manager_schema.sum_of_acct_balances;
```

A.47 EXPLAIN PLAN FOR

```
EXPLAIN PLAN FOR sql_statement
```

用法

EXPLAIN PLAN FOR 语句显示为指定 SQL 语句选择 SQL 优化器的执行计划。

执行计划是 solidDB 执行该语句时要执行的一系列基本操作以及这些操作的顺序。执行计划中的每个操作都称作单元”。

表 42. EXPLAIN PLAN FOR 单元

单元	描述
JOIN UNIT*	连接单元用于连接两个或更多个表。此连接可以使用循环连接或合并连接完成。
TABLE UNIT	表单元用于从表或索引中访存数据行。
ORDER UNIT	排序单元用于对行执行排序，以便进行分组或满足 ORDER BY。此排序可以在内存中进行，也可以使用外部磁盘排序器进行。
GROUP UNIT	组单元用于执行分组和聚集计算（SUM 和 MIN 等等）。
UNION UNIT*	并集单元用于执行 UNION 操作。此单元可以使用循环连接或合并连接完成。

表 42. EXPLAIN PLAN FOR 单元 (续)

单元	描述
INTERSECT UNIT*	交集单元用于执行 INTERSECT 操作。此单元可以使用循环连接或合并连接完成。
EXCEPT UNIT*	差集单元用于执行 EXCEPT 操作。此单元可以使用循环连接或合并连接完成。

* 对于只引用单个表的查询，也会生成此单元。在那种情况下，不会在此单元中执行连接；它仅仅是传递行，而不对其进行处理。

EXPLAIN PLAN FOR 语句返回的表包含下列各列。

表 43. EXPLAIN PLAN FOR 表格列

列名	描述
ID	输出行号，仅用于确保各行唯一。
UNIT_ID	这是 SQL 解释器中的内部单元标识。每个单元都具有不同的标识。单元标识是稀疏的编号序列，这是因为，SQL 解释器还为优化阶段除去的那些单元生成单元标识。如果多行具有相同的单元标识，那么意味着那些行属于同一个单元。为了进行格式化，来自一个单元的信息可能会分为多行。
PAR_ID	单元的父单元标识。父标识号引用 UNIT_ID 列中的标识。
JOIN_PATH	对于连接、并集、交集和差集单元而言，存在一条连接路径，此路径指定要在此单元中连接哪些表以及各个表的连接顺序。连接路径号引用 UNIT_ID 列中的单元标识。它表示此单元的输入来自该单元。表的连接顺序就是连接路径的列示顺序。列示的第一个表是循环连接中最外层的表。
UNIT_TYPE	单元类型是执行图单元类型。
INFO	INFO 列是为其他信息保留的。例如，它可能包含索引使用情况、数据库表名以及 solidDB 用于选择行的约束。注意，这里列示的约束可能与 SQL 语句中指定的那些约束不匹配。

示例

EXPLAIN PLAN FOR 从表中选择 *;

A.48 EXPORT SUBSCRIPTION

```
EXPORT SUBSCRIPTION publication_name [(publication_parameters)]
  TO 'filename'
  USING BOOKMARK bookmark_name;
  [WITH [NO] DATA];
```

其中:

publication_name 和 *bookmark_name* 是必须存在于数据库中的标识。

filename 代表引在单引号中的字面值。通过指定同一个文件名，可以将多个发布导出到单一文件。

例如，在将数据导出到未包含主数据库数据并需要部分数据的现有数据库时，WITH DATA 选项会使用数据创建副本数据库。

例如，在将预订导入到已包含数据的数据库时（例如，使用现有主数据库的备份副本时），WITH NO DATA 选项仅用元数据创建副本数据库。

缺省情况下，导出文件使用 WITH DATA 选项进行创建并包含所有行。如果指定了多个发布，那么导出的文件将同时具有 WITH DATA 和 WITH NO DATA 选项。

用法

EXPORT SUBSCRIPTION 语句将把一个版本的数据从主数据库导出到文件。然后，您可以使用 IMPORT 语句将文件中的该数据导入到副本数据库。TheEXPORT SUBSCRIPTION 语句不可用于副本数据库。

EXPORT SUBSCRIPTION 语句有多种用途。例如：

- 根据现有的主数据库创建大型副本数据库（大于 2GB）。

此过程要求先将包含或未包含数据的预订导出到一个文件，然后将该预订导入到副本数据库。

- 将特定版本的数据导出到副本数据库。

为了提高性能，您可以选择“导出”数据，而不是使用 MESSAGE APPEND REFRESH 语句来把该数据发送到副本数据库。

- 导出元数据信息，而不导出实际的行数据。

您可以创建一个已包含现有数据并且只需要与某个发布相关联的模式和版本信息的副本数据库。

与 MESSAGE APPEND REFRESH 语句不同（执行此语句时，副本数据库请求执行刷新），您请求直接从主数据库执行导出。导出输出将保存到用户指定的文件，而不是保存到 solidDB 消息。

EXPORT SUBSCRIPTION 语句从主数据库导出发布，作为对一组输入参数值（如果它们在该发布中使用）的刷新。该语句基于给定的书签，这意味着导出数据与此书签保持一致。导出数据时，EXPORT SUBSCRIPTION 语句将包括完全发布中所有在该书签之前的行。但是，由于导出操作基于给定的书签，因此后续 REFRESH 将是增量刷新。

如果在主数据库中创建用于导出和导入数据的书签，那么在下列情况下，该书签必须存在：

- 对主数据库执行 EXPORT SUBSCRIPTION 语句时。

如果此时不存在该书签，那么将生成错误消息 25067，即表明找不到该书签。

- 对期望的所有副本数据库执行 IMPORT 语句，以及那些副本数据库接收到它们的第一组数据（REFRESH）时。

在文件导入期间，不需要连接到主数据库，并且不会检查该书签是否存在。但是，如果该书签在副本数据库接收到它的第一个 REFERSH 时不存在，那么该 REFERSH

将失败并发出错误消息 25067，并且导入数据不可用。补救措施是，在主数据库中创建一个新书签，重新导出数据，然后重新导入数据。

有关更多详细信息，请参阅 *IBM solidDB Advanced Replication User Guide* 中的导出和导入预订。

用法规则

以下规则适用于 EXPORT SUBSCRIPTION 语句：

- 执行导出时，每个预订只允许使用一个文件。您可以使用同一个文件名将多个预订包括到同一个文件中。
- 导出文件的文件大小依赖于底层操作系统。如果相应的平台（例如 SUN 或 HP）允许超过 2GB，那么可以写入大于 2GB 的文件。这意味着，副本数据库（接收方）也应该使用兼容的平台和文件系统。否则，副本数据库将无法接受导出文件。如果主数据库和副本数据库的操作系统都支持大小大于 2GB 的文件，那么将允许使用大于 2GB 的导出文件。
- 导出文件可以包含不止一个预订。您可以使用 WITH DATA 或 WITH NO DATA 选项来导出预订。包含多个预订的导出文件可以同时采用 WITH DATA 和 WITH NO DATA 选项。
- 使用 WITH NO DATA 选项将预订导出到文件时，将只把元数据（即，与该发布相对应的模式和版本信息）导出到文件。
- 使用 EXPORT SUBSCRIPTION 语句时，请关闭自动落实方式。

返回值

表 44. EXPORT SUBSCRIPTION 返回值

错误代码	描述
25056	不允许自动落实。
25067	找不到书签。
25068	打开导出文件 <i>file_name</i> 失败。
25010	找不到发布 <i>name</i> 。

示例

```
EXPORT SUBSCRIPTION FINANCE_PUBLICATION(2004) TO 'FINANCE.EXP'  
USING BOOKMARK BOOKMARK_FOR_FINANCE_DB WITH NO DATA;
```

相关参考

第 180 页的 A.16，『CREATE [OR REPLACE] PUBLICATION』

第 187 页的 A.21，『CREATE SYNC BOOKMARK』

A.49 EXPORT SUBSCRIPTION TO REPLICA

```
EXPORT SUBSCRIPTION publication_name [(publication_parameters)]  
TO REPLICA replica_node_name  
USING BOOKMARK bookmark_name  
[COMMITBLOCK number_of_rows]
```

其中:

publication_name 和 *bookmark_name* 是必须存在于数据库中的标识。

COMMIT BLOCK 关键字指定通过一个事务在副本数据库中落实的所导出数据行数。

用法

EXPORT SUBSCRIPTION TO REPLICA 语句将发布所指定的大量数据从主数据库导出到副本数据库。在 EXPORT 操作完成后，副本数据库可以使用 MESSAGE APPEND REFRESH 语句以递增方式刷新该预订的数据。EXPORT SUBSCRIPTION TO REPLICA 语句不可用于副本数据库。

由于 EXPORT SUBSCRIPTION TO REPLICA 语句不使用基于磁盘的高级复制消息 (MESSAGE) 将数据从主数据库发送到副本数据库，因此最大程度地降低了操作期间对磁盘的使用，从而显著提高将大量数据从主数据库发送到副本数据库的效率。

从主数据库中导出的发布数据是 REFRESH 以及该发布所使用的一组输入参数值。

EXPORT SUBSCRIPTION TO REPLICA 语句基于给定的书签，这意味着，导出数据在此书签之前保持一致。导出数据时，EXPORT SUBSCRIPTION 语句将包括完全发布中所有在该书签之前的行。但是，由于导出操作基于给定的书签，因此后续 REFRESH 将是增量刷新。

如果在主数据库中创建用于导出数据的书签，那么对主数据库执行 EXPORT SUBSCRIPTION 语句时，该书签必须存在。如果此时不存在该书签，那么将生成错误消息 25067，即表明找不到该书签。

导出大量的行时，指定落实块有助于提高操作性能。但是，在落实块处于活动状态的情况下执行导出操作时，建议不要让其他应用程序使用副本数据库。

返回值

表 45. EXPORT SUBSCRIPTION TO REPLICA 返回值

错误代码	描述
25056	不允许自动落实。
25067	找不到书签。
25010	找不到发布 <i>name</i> 。

示例

```
EXPORT SUBSCRIPTION FINANCE_PUBLICATION(2004) TO REPLICA replica_1  
USING BOOKMARK BOOKMARK_FOR_FINANCE_DB COMMITBLOCK 10000;
```


相关参考

第 180 页的 A.16, 『CREATE [OR REPLACE] PUBLICATION』

第 187 页的 A.21, 『CREATE SYNC BOOKMARK』

A.50 GRANT

```
GRANT {ALL | grant_privilege [, grant_privilege]...}
      ON table_name
TO {PUBLIC | user_name [, user_name]... |
   role_name [, role_name]... }
[WITH GRANT OPTION]

GRANT role_name TO user_name

grant_privilege ::= DELETE | INSERT | SELECT |
                 UPDATE [( column_identifier [, column_identifier]... )] |
                 REFERENCES [( column_identifier [, column_identifier]... )]

GRANT EXECUTE ON function_name
      TO {PUBLIC | user_name [, user_name]... | role_name [, role_name]... }

GRANT EXECUTE ON procedure_name
      TO {PUBLIC | user_name [, user_name]... | role_name [, role_name]... }

GRANT {SELECT | INSERT} ON event_name
      TO {PUBLIC | user_name [, user_name]... | role_name [, role_name]... }

GRANT {SELECT | UPDATE} ON sequence_name
      TO {PUBLIC | user_name [, user_name]... | role_name [, role_name]... }
```

用法

GRANT 语句用于执行下列操作:

- 将特权授予指定的用户或角色
- 通过将所指定角色的特权授予用户, 将特权授予指定的用户

将角色的特权授予用户时, 该角色可以是您创建的角色, 也可以是系统定义的角色, 例如 SYS_SYNC_ADMIN_ROLE 或 SYS_ADMIN_ROLE。

如果使用了可选的 WITH GRANT OPTION, 那么接收到特权的用户可以将该特权授予其他用户。

示例

```
GRANT GUEST_USERS TO CALVIN;
GRANT INSERT, DELETE ON TEST TO GUEST_USERS;
```

相关参考

第 250 页的 A.72, 『REVOKE』

相关信息

第 84 页的 4.2, 『管理用户特权和角色』

A.51 GRANT PASSTHROUGH

```
GRANT PASSTHROUGH {READ | WRITE}
TO {PUBLIC | user_name [, user_name]... | role_name [, role_name]... }
[WITH GRANT OPTION]
```

支持环境

此命令需要使用 SQL 传递功能。

用法

GRANT PASSTHROUGH 语句授予对 SQL 传递的访问权。

示例

```
GRANT PASSTHROUGH READ TO cdcuser1, cdcuser2
GRANT PASSTHROUGH WRITE TO cdcuser1, cdcuser2
```

另请参阅

第 250 页的 A.73, 『REVOKE PASSTHROUGH』

A.52 GRANT REFRESH

```
GRANT REFRESH ON publication_name TO {PUBLIC |
user_name,
[ user_name ] ... | role_name , [ role_name ] ...}
```

用法

该 GRANT REFRESH 语句将一个发布的访问权授予主数据库中定义的用户或角色。该 GRANT REFRESH 语句将仅应用于高级的复制配置, 且仅在主数据库上执行。

注: 关键字 REFRESH 可以用来等同于关键字 SUBSCRIBE; 然而在 GRANT 语句中已经不建议使用关键字 SUBSCRIBE。

返回值

表 46. GRANT REFRESH 返回值

错误代码	描述
13137	授权/撤销方式不合法。
13048	不具有 GRANT OPTION 特权。
25010	找不到发布 <i>name</i> 。

示例

```
GRANT REFRESH ON customers_by_area TO salesman_jones;
GRANT REFRESH ON customers_by_area TO all_salesmen;
```

A.53 HINT

```
--(* vendor (SOLID), product (Engine), option(hint)
--hint *)--
```

```
hint::=
[MERGE JOIN |
TRIPLE MERGE JOIN |
LOOP JOIN |
```

```

JOIN ORDER FIXED |
INTERNAL SORT |
EXTERNAL SORT |
INDEX [REVERSE] table_name.index_name |
PRIMARY KEY [REVERSE] table_name |
FULL SCAN table_name |
[NO] SORT BEFORE GROUP BY |
UNION FOR OR |
OR FOR OR |
LOOP FOR OR]

```

用法

提示用于向 SQL 优化器指定伪指令，以确定要使用的查询执行计划。使用查询语句中的嵌套伪注释指定提示。优化器检测这些伪指令或提示，并将其查询执行计划基于其上。优化器提示允许在多种条件下优化应用程序到数据、查询类型和数据库。这不仅为查询偶尔遇到的性能问题提供解决方案，也将响应时间的控制权从系统转交给用户。

提示适用于：

- 选择合并或嵌套的循环连接
- 使用 FROM 列表中指定的固定连接顺序
- 选择内部排序或外部排序
- 选择特定索引
- 选择基于索引扫描的表扫描
- 选择在分组之前或之后进行排序

在 SELECT、UPDATE 或 DELETE 关键字后，您可以将提示以静态字符串的形式放置在 SQL 语句中。在 INSERT 关键字后面，不允许使用提示。

伪注释标识

在伪注释前缀后面，跟着标识信息。您必须将供应商指定为 SOLID，将产品指定为 Engine，并将选项（即伪注释类名）指定为有效的 hint。

终结器可能位于自身所在行上，或提示的最后一行末尾。例如，以下任意一项均可接受：

```

--(* vendor (Solid), product (Engine), option(hint)
--hint
-- *)--

```

或者

```

--(* vendor (Solid), product (Engine), option(hint)
--hint *)--

```

空格的使用比较敏感。在伪注释前缀 --(* 和后缀 *)-- 中，圆括号与星号之间不能有空格。*)-- 终结器之前必须有空格，即星号之前必须有空格（请参阅以上示例）。--(* 中左括号前不要求空格。终止符 *)-- 未处于注释定界符 -- 之后时，不可以位于单独的一行上。

优化器提示中的表名称解析

优化器提示中的表名称解析与 SQL 语句中的任何表名相同。这意味着，如果查询中存在表的别名，那么必须在优化器提示中使用别名，而不能使用表名。例如：

```
SELECT
  --(* vendor(SOLID), product(Engine), option(hint)
  -- FULL SCAN emp_alias *)--
  emp_alias.emp_id, employee_name, dependent_name
FROM employee_table AS emp_alias LEFT OUTER JOIN dependent_table
AS dep_alias
  ON (dep_alias.emp_id = emp_alias.emp_id)
ORDER BY emp_alias.emp_id;
```

如果在应该指定别名时指定表名，那么您将接收到以下错误消息：

102: 未使用的优化器提示。

如果未使用别名，并且正在使用另一个模式和/或另一个目录中的表，那么在提示中，请确保在表名前指定模式和/或目录名。例如：

```
SELECT
  --(* vendor(SOLID), product(Engine), option(hint)
  -- FULL SCAN sally_schema.employee_table *)--
  emp_id, employee_name
FROM sally_schema.employee_table;
```

对 ODBC 或 JDBC 使用提示

如果您正在使用提示，并且将查询编写为字符串，然后使用 ODBC 或 JDBC 来提交该字符串，那么必须确保在该字符串中嵌入适当的换行符以标记注释结束。否则，将发生语法错误。如果未嵌入任何换行符，那么第一个注释开始后的所有语句都将像是注释。例如，假定代码如下所示：

```
strcpy(s, "SELECT --(* hint... *)-- col_name FROM table;");
```

第一个“--”后的所有内容都像是注释，因此语句似乎不完整。您必须将以上代码修改为：

```
strcpy(s, "SELECT --(* hint... *)-- \n col_name FROM table;");
```

嵌入式的换行符 \n 字符终止该注释。一种实用的调试技术是，输出字符串以确保它们看起来正确。它们应该类似于：

```
SELECT --(* hint ... *)--
column_name FROM table_name...;
```

或者

```
SELECT --(* hint ... *)--
column_name FROM table_name...;
```

子选项和多个提示

每个子查询都需要自己的提示；例如，以下是提示语法的有效用法：

```
INSERT INTO ... SELECT hint FROM ...
UPDATE hint TABLE ... WHERE column = (SELECT hint ... FROM ...)
DELETE hint TABLE ... WHERE column = (SELECT hint ... FROM ...)
```

在一个伪注释中指定多个提示时，您必须用逗号分隔提示，如下示例所示：

示例 1:

```

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
--MERGE JOIN
--JOIN ORDER FIXED *)--
*
FROM TAB1 A, TAB2 B;
WHERE A.INTF = B.INTF;

```

示例 2:

```

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
--INDEX TAB1.INDEX1
--INDEX TAB1.INDEX1 FULL SCAN TAB2 *)--
*
FROM TAB1, TAB2
WHERE TAB1.INTF = TAB2.INTF;

```

启用和禁用提示

使用 **SQL.EnableHints** 参数启用和禁用提示。在缺省情况下，启用提示（使用 **SQL.EnableHints=yes**）。

示例

```

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
--MERGE JOIN
--JOIN ORDER FIXED
-- *)--
col1, col2
FROM TAB1 A, TAB2 B;
WHERE A.INTF = B.INTF;

```

```

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
--INDEX TAB1.INDEX1
--INDEX TAB1.INDEX1 FULL SCAN TAB2
-- *)--
*
FROM TAB1, TAB2
WHERE TAB1.INTF = TAB2.INTF;

```

```

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- INDEX TAB1.IDX1 *)--
* FROM TAB1 WHERE I > 100

```

```

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- INDEX MyCatalog.mySchema.TAB1.IDX1 *)--
* FROM TAB1 WHERE I > 100

```

```

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- JOIN ORDER FIXED *)--
* FROM TAB1, TAB2 WHERE TAB1.I >= TAB2.I

```

```

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- LOOP JOIN *)--
* FROM TAB1, TAB2 WHERE TAB1.I >= TAB2.I

```

```

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- INDEX REVERSE MyCatalog.mySchema.TAB1.IDX1 *)--
* FROM TAB1 WHERE I > 100

```

```

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- SORT BEFORE GROUP BY *)--
AVG(I) FROM TAB1 WHERE I > 10 GROUP BY I2

SELECT
--(* vendor(SOLID), product(Engine), option(hint)
-- INTERNAL SORT *)--
* FROM TAB1 WHERE I > 10 ORDER BY I2

```

A.53.1 solidDB-支持提示

表 47. solidDB-支持提示

提示	定义
MERGE JOIN	<p>指示对于 FORM 子句中列示的所有表，优化器在 SELECT 查询中选择合并连接访问方案。当两个表的大小大致相同，并且数据均匀分布时，使用 MERGE JOIN 选项。当连接相同数目的行时，此选项的速度比 LOOP JOIN 快。在连接数据时，MERGE JOIN 最多支持 3 个表。将通过连接列并对列的结果进行组合，对连接表进行排序。</p> <p>当数据按连接键排序，并且嵌套的循环连接性能欠佳时，可以使用此提示。仅当两个表之间存在相等谓词时（例如，“table1.col1 = table2.col1”），优化器才会选择合并连接。否则，即使已指定了 MERGE JOIN 提示，优化器也将选择 LOOP JOIN。</p> <p>在执行合并操作前，如果数据未排序，那么 solidDB 查询执行程序将对数据进行排序。</p> <p>请记住，与不进行排序的合并连接相比，进行排序的合并连接需要耗用更多资源。</p>
TRIPLE MERGE JOIN	<p>TRIPLE MERGE JOIN 是 MERGE JOIN 的变体。它有三个在同等基础上合并在一起的表源，而非 MERGE JOIN 中的两个表源。如果可能，TRIPLE MERGE JOIN 提示会指示 SQL 解释器使用三表合并连接算法。在三个表源中，只有一个字段应该在计算 WHERE 条件后在所有的结果行中相等时，才可以使用三表合并连接算法。</p>
LOOP JOIN	<p>指示对于 FORM 子句中列示的所有表，优化器在 SELECT 查询中选择嵌套循环连接。缺省情况下，优化器不会选择嵌套循环连接。</p> <p>LOOP JOIN 对内层表和外层表都进行循环来查找内层表和外层表内各列之间的匹配项。为了提高性能，应该对连接列建立索引。</p> <p>当表较小并且放入内存比使用其他连接算法效率更高时，请使用循环连接。</p>

表 47. solidDB-支持提示 (续)

提示	定义
JOIN ORDER FIXED	<p>指定优化器进行连接时按照查询的 FROM 子句中的列顺序来使用表。这意味着，优化器不尝试重新安排连接顺序，并且不尝试查找用于完成连接的替代访问路径。</p> <p>提示: 您应当运行 EXPLAIN PLAN 输出来测试该提示，以确保生成的方案对于给定的查询而言最优。</p>
INTERNAL SORT	<p>指定查询执行程序使用内部排序器。如果期望的结果集较小（数百行，而不是数千行），例如，如果您正在执行某些聚集、对较小的结果集执行 ORDER BY 或者对较小的结果集执行 GROUP BY 等，请使用此提示。</p> <p>此提示用于避免使用开销较大的外部排序器。</p>
EXTERNAL SORT	<p>指定查询执行程序使用外部排序器。当期望的结果集较大并且在内存中放不下时，例如当期望的结果集包含数千行时，请使用此提示。</p>
INDEX [REVERSE] <i>table_name.index_name</i>	<p>对于给定的表，强制执行给定的索引扫描。在这种情况下，优化器将不会评估是否有任何其他可用于构建访问方案的索引或者表扫描是否更适合于给定的查询。</p> <p>提示: 您应当运行 EXPLAIN PLAN 输出来测试该提示，以确保生成的方案对于给定的查询而言最优。</p> <p>可选关键字 REVERSE 按逆序返回行。在这种情况下，查询执行程序首先处理索引的最后一页并开始按索引的降序（逆序）键顺序返回行。</p> <p>在 <i>tablename.indexname</i> 中，<i>tablename</i> 是标准的表名，它可包含 <i>catalogname</i> 和 <i>schemaname</i>。</p>
PRIMARY KEY [REVERSE] <i>table_name</i>	<p>对于给定的表，强制执行主键扫描。</p> <p>可选关键字 REVERSE 按逆序返回行。</p> <p>如果主键对于给定的表不可用，您将接收到运行时错误。</p>
FULL SCAN <i>table_name</i>	<p>对于给定的表，强制执行表扫描。在这种情况下，优化器将不会评估是否有任何其他可用于构建访问方案的索引或者表扫描是否更适合于给定的查询。</p> <p>提示: 您应当运行 EXPLAIN PLAN 输出来测试该提示，以确保生成的方案对于给定的查询而言最优。</p>
[NO] SORT BEFORE GROUP BY	<p>指示按 GROUP BY 列对结果集进行分组前，是否执行排序操作。</p> <p>如果所分组的项较少（数百行），请使用 NO SORT BEFORE。另一方面，如果所分组的项较多（数千行），请使用 SORT BEFORE。</p>

表 47. solidDB-支持提示 (续)

提示	定义
UNION FOR OR	<p>UNION FOR OR 提示指示 SQL 解释器使用以下类型的构造替代 A = 1 OR A = 2 样式的 OR 条件:</p> <pre>SELECT ... WHERE A = 1 UNION ALL SELECT ... WHERE A = 2</pre> <p>大多数情况下, SQL 解释器自动执行该替代操作; UNION FOR OR 提示确保始终使用 UNION 类型的执行操作。</p> <p>注: 也可以处理 A = 1 OR B = 2 类型的条件, 但是由于这些条件并不互斥, 可能会造成一些问题。基于此原因, A = 1 OR B = 2 的构造如下所示:</p> <pre>SELECT ... WHERE A = 1 UNION ALL SELECT ... WHERE B = 2 AND UtoT NOT (A = 1)</pre> <p>其中 UtoT 表示 UNKNOWN TO TRUE。</p> <p>需要 UtoT 运算符来处理含有 NULL 值的情况。如果未使用 UtoT 运算符, 含有 A = NULL 和 B = 2 值的行在 UNION 变体中将无法正确显示。</p>
OR FOR OR	<p>OR FOR OR 提示是 UNION FOR OR 的反义提示。它将防止解释器使用 UNION 类型的解决方案。</p>
LOOP FOR OR	<p>LOOP FOR OR 提示是一个介于 UNION FOR OR 和 OR FOR OR 之间的替代查询执行计划。使用 LOOP FOR OR 时, 会将 OR 值逐个传递到数据表级, 但却不能处理诸如 A = 1 OR B = 2 的条件 (请参阅 UNION FOR OR 的描述, 以了解 A = 1 OR B = 2 的详细信息)。</p>

A.54 IMPORT

```
IMPORT 'file_name' [COMMITBLOCK number_of_rows]
[{'OPTIMISTIC' | 'PESSIMISTIC'}]
```

用法

该 IMPORT 语句将来自 EXPORT SUBSCRIPTION 语句创建的数据文件中的数据导入至副本数据库。该 IMPORT 语句只可以在副本数据库上发出。

file_name 代表括在单引号中的字面值。IMPORT 命令只能接受单一文件名。因此, 所有要导入到副本数据库的数据都必须包含在一个文件中。

COMMITBLOCK 选项指示落实数据前要处理的行数。*number_of_rows* 是与可选的 COMMITBLOCK 子句配合使用的整数值, 用于指示落实块大小。使用 COMMITBLOCK 选项能够提高导入性能并频繁地释放内部事务资源。

COMMITBLOCK 大小的最佳值随服务器上各种资源的不同而有所变化。COMMITBLOCK 大小的一个不错的示例是, 对于 10,000 行, 设置为 1000。如果未指

定 COMMITBLOCK 选项，那么 IMPORT 命令将使用发布中所有的行作为一个事务。对于少量的行而言，这样做的效果可能不错，但对于数千行乃至数百万行的情况，将会是个问题。

可以将 IMPORT 定义为最初执行时使用表级悲观锁定方式。如果指定了悲观方式，那么直到导入操作完成为止对受影响的表进行的所有其他并发访问的阻塞。否则，如果使用乐观方式，那么 IMPORT 可能会由于并行冲突而失败。

当一个事务获取对表的互斥锁定时，**General.TableLockWaitTimeout** 参数设置决定该事务在互斥锁定或共享锁定释放前等待的时间。

直到副本数据库在完成导入后第一次刷新数据之后，所导入的数据才会变为有效。副本数据库第一次执行 REFRESH 时，在主数据库中必须存在用于导出文件的书签。如果该书签不存在，该 REFRESH 语句将失败。这意味着您必须在主数据库中创建新书签，重新导出数据，然后在副本数据库中重新导入该数据。

使用该 IMPORT 语句时应遵循以下规则：

- 执行导入时，每个预订只允许使用一个文件。
- 导出文件的文件大小依赖于底层操作系统。如果相应的平台（例如 SUN 或 HP）允许超过 2GB，那么可以写入大于 2GB 的文件。这意味着，副本数据库（接收方）也应该使用兼容的平台和文件系统。否则，副本数据库将无法接受导出文件。如果主数据库和副本数据库的操作系统都支持大小大于 2GB 的文件，那么将允许使用大于 2GB 的导出文件。
- 在使用 IMPORT 命令之前，请备份副本数据库。如果使用了 COMMITBLOCK 选项，并且操作失败，那么所导入的数据将仅仅是部分落实；您需要使用备份文件来还原副本数据库。
- 要使用 IMPORT 语句，必须将自动落实设置为关闭。

返回值

表 48. IMPORT 返回值

错误代码	描述
25007	找不到主数据库 <i>master_name</i> 。
25019	此数据库不是副本数据库。
25069	打开导入文件 <i>file_name</i> 失败。
13XXX	表级错误。
13124	找不到用户标识 <i>num</i> 。 例如，如果已删除用户，那么将生成此消息。
10006	并行冲突（同时正在执行其他操作）。
13047	无权执行操作。
13056	不允许对伪列执行插入。
21XXX	通信错误

表 48. *IMPORT* 返回值 (续)

错误代码	描述
25024	未定义主数据库。
25026	不是有效的主数据库用户。
25031	事务处于活动状态，操作失败。
25036	找不到发布 <i>publication_name</i> 或者发布版本不匹配。
25040	找不到用户标识 <i>user_id</i> 。 执行消息应答时，将主数据库用户映射到本地副本数据库标识的尝试失败。
25041	找不到对发布 <i>publication_name</i> 的预订。
25048	找不到发布 <i>publication_name</i> 请求信息。
25054	还没有为同步历史记录设置表 <i>table_name</i> 。
25056	不允许自动落实。
25060	列 <i>column_name</i> 在表 <i>table_name</i> 中的发布 <i>publication_name</i> 结果集中不存在。

示例

```
IMPORT 'FINANCE.EXP';
```

A.55 INSERT

```
INSERT INTO table_name insert_columns_and_source

insert_columns_and_source::=
  from_subquery
  | from_constructor
  | from_default

from_subquery ::=
  [insert_column_name_list] query expression

insert_column_name_list ::=
  ([column name [, column name]... ])

from_constructor ::=
  [insert_column_name_list] VALUES row_constructor [, row_constructor]... ]

row_constructor ::= ([insert_item [, insert_item]...])

insert_item ::= insert_value | DEFAULT | NULL

from_default ::= DEFAULT VALUES
```

用法

INSERT 语句将行插入表中。

INSERT 语句有多种变体。在最简单的实例中，按照定义或更改表时指定的顺序为新行的每一列提供一个值。在 INSERT 语句的首选格式中，作为语句的组成部分来指定各个列，并且，只要列列表的顺序与值列表的顺序匹配，那么这些列不必具有任何特定顺序。

insert_value 在过程中可以是字面值、标量函数或变量。

示例

```
INSERT INTO TEST (C, ID) VALUES (0.22, 5);
INSERT INTO TEST VALUES (0.35, 9);
```

您也可以执行多行插入。例如，要在一个语句中插入三行，可以使用以下命令：

```
INSERT INTO employees VALUES
(10021, 'Peter', 'Humlaut'),
(10543, 'John', 'Wilson'),
(10556, 'Bunba', 'Olo');
```

通过使用 DEFAULT VALUES 语句，可以插入缺省值（如下面的第二个示例所示）。等同的格式为“INSERT INTO TEST() VALUES()”。您还可以对一个列指定特定的值并对另一个列使用缺省值。下列示例演示了这些方法：

```
INSERT INTO TEST () VALUES ();
INSERT INTO TEST DEFAULT VALUES;
INSERT INTO TEST (C, ID) VALUES (0.35, DEFAULT);
INSERT INTO TEST (C, ID) SELECT A, B FROM INPUT_TO_TEST;
```

A.56 LIST

LIST <statement>

其中：

statement :=

```
CATALOGS |
USERS |
ROLES |

[catalog.]SCHEMAS |
[catalog.]MASTERS |
[catalog.]REPLICAS |
[catalog.]SUBSCRIPTIONS |
[catalog.]PUBLICATIONS |
[[catalog.]schema.]TABLES |
[[catalog.]schema.]VIEWS |
[[catalog.]schema.]PROCEDURES |
[[catalog.]schema.]FUNCTIONS |
[[catalog.]schema.]EVENTS |
[[catalog.]schema.]SEQUENCES |
[[[catalog.]schema.]table.]INDEXES |
[[[catalog.]schema.]table.]TRIGGERS
```

用法

LIST 语句可用于查看 solidDB 数据库中现有的数据库对象。LIST 语句会输出特定数据库对象类型列表。

对象类型可以作为自变量指定。任何对象名均可用通配符表达式替换（例如，foo %b_r）。通配符“%”将范围扩展到该特定级别上的每个可能的对象。

LIST 语句可与 DESCRIBE 命令配合使用，如下所示：

```
LIST my_catalog.my_schema.PROCEDURES
DESCRIBE my_catalog.my_schema.certain_procedure
```

示例

```
LIST my_schema.TABLES;
```

```
Catalog: my_catalog
Schema: my_schema
TABLES:
-----
CITY
PERSON
```

1 rows fetched.

```
LIST my_catalog.%.TABLES;
```

```
Catalog: my_catalog
Schema: my_schema
TABLES:
-----
PERSON
IN_RELATION
CITY
CAR
Schema: another_schema
TABLES:
-----
HORSENAME
```

1 rows fetched.

另请参阅

第 198 页的 A.27, 『 DESCRIBE 』

A.57 LOCK TABLE

```
LOCK lock-definition [lock-definition] [wait-option]
lock-definition ::= TABLE tablename [,tablename]
IN { SHARED | [LONG] EXCLUSIVE } MODE
wait-option ::= NOWAIT | WAIT <#seconds>
```

其中：

- **tablename**: 要锁定的表的名称。您可以通过对表名进行限定来指定该表的目录和模式。只能锁定表，无法锁定视图。
- **SHARED**: 共享方式允许其他用户对该表执行读写操作。不允许 DDL 操作。此外，共享方式禁止其他用户对同一个表发出互斥锁定。
- **EXCLUSIVE**: 如果一个 D 表使用悲观锁定方式，则互斥锁定将阻止其他用户以任何方式（例如，插入或删除数据、DLL 操作和获取锁定）访问该表（SELECT 语句不包括在内）。

如果是 M 表（通常是悲观模式）和乐观模式的 D 表的情况，互斥锁定允许其他用户在已锁定的表上执行 SELECT 和 SELECT FOR UPDATE 语句，但阻止对该表执行任何其他活动（插入或删除数据、DLL 操作和获取锁定）。

- **LONG:** 在缺省情况下，会在事务结束时释放锁定。如果指定了 **LONG** 选项，那么执行锁定的事务落实时，将不会释放该锁定。如果锁定事务异常中止或回滚，那么将释放所有锁定，其中包括 **LONG** 锁定。您必须使用 **UNLOCK** 命令显式解除 **LONG** 锁定。只有在 **EXCLUSIVE** 方式下，才允许挂起长 (**LONG**) 持续时间锁定。不支持 **LONG** 共享锁定。
- **NOWAIT:** 指定立即将控制权返回给您，即使另一用户锁定任何所指定表，亦如此。如果未获取所请求的锁定，将返回错误。
- **WAIT:** 指定系统在获得所请求锁定之前应等待的时间的超时（以秒计）。如果在该时间段内未获取所请求的锁定，将返回错误。

注: **WAIT** 命令仅在基于磁盘的表上生效。

用法

LOCK 和 **UNLOCK** 命令允许您以手动方式锁定表以及将表解锁。锁定表将限制对该对象的访问。**LONG** 选项允许您将手动互斥锁定的持续时间延长到当前事务结束之后；换言之，可以在一系列多个事务之间保持以互斥方式锁定表。

您不需要频繁地执行手动锁定。服务器的自动锁定操作通常已足够。有关一般性的锁定以及具体的服务器自动锁定的详细讨论，请参阅第 103 页的 5.2, 『并行控制和锁定』。

显式地锁定表主要是为了帮助数据库管理员在数据库中执行维护操作而不受其他用户影响。例如，在更改模式时，高级复制设置中常常使用手动锁定。有关更多详细信息，请参阅 *IBM solidDB Advanced Replication User Guide* 中的更新分布式系统的模式。

表锁定可以是 **SHARED** 锁定或 **EXCLUSIVE** 锁定。

对表挂起的 **EXCLUSIVE** 锁定将不允许任何其他用户或连接更改该表或者该表中的任何记录。如果您对一个表挂起互斥锁定，那么在释放该互斥锁定前，其他用户/连接将无法对该表执行下列任何操作:

- **INSERT**、**UPDATE** 和 **DELETE**
- **ALTER TABLE**
- **DROP TABLE**
- **LOCK TABLE** (共享方式或互斥方式)

此外，如果 **D** 表使用悲观锁定方式，那么互斥锁定还将阻止其他用户/连接执行下列操作:

- **SELECT FOR UPDATE**

互斥锁定不会阻止其他用户从该表中选择 (**SELECT**) 记录。其他大部分数据库服务器的行为有所不同 - 即，它们不允许对以互斥方式锁定的表执行 **SELECT** 操作。

共享锁定的限制性不如互斥锁定。如果您对一个表挂起共享锁定，那么在释放该共享锁定前，其他用户/连接将无法对该表执行下列任何操作:

- **ALTER TABLE**
- **DROP TABLE**
- **LOCK TABLE** (互斥方式)

对一个表使用共享锁定后，其他用户/连接可以对该表执行插入、更新、删除和选择操作。

对表挂起的共享锁定与对记录挂起的共享锁定略有不同。如果对记录挂起共享锁定，那么没有任何其他用户能够更改该记录中的数据。如果您对表挂起共享锁定，其他用户也仍可以更改该表中的数据。

不止一个用户可以同时对一个表挂起共享锁定。因此，如果您对一个表挂起共享锁定，其他用户也可以对该表挂起共享锁定。但是，一个用户对表挂起共享锁定或互斥锁定后，另一个用户将无法对该表挂起互斥锁定。

LOCK 命令在它执行时生效。如果未使用 LONG 选项，那么将在该事务结束时释放该锁定。如果使用了 LONG 选项，那么该表将一直处于锁定状态，直到您显式地将其解锁为止。如果您回滚在其中挂起了锁定的事务，则也会释放表锁定 - 仅当您落实在其中挂起 LONG 锁定的事务，LONG 锁定才会在事务之间保持。

LOCK/UNLOCK TABLE 命令仅适用于表。没有用于以手动方式对表中的各个记录进行锁定或解锁的命令。

在一个 LOCK 命令中，可以锁定不止一个表并指定不同的方式。如果该 LOCK 命令失败，那么不会锁定任何表。如果该 LOCK 命令成功，那么将挂起所请求的所有锁定。

如果用户未指定等待选项（NOWAIT 或 WAIT seconds），那么将使用缺省等待时间。该时间与死锁检测超时时间相同。WAIT 命令仅在基于磁盘的表上生效。

要使用 LOCK TABLE 命令对某个表发出锁定，必须具有对该表的插入、删除或更新特权。不存在用于将表的 LOCK 和 UNLOCK 特权授予其他用户的 GRANT 命令。

返回值

错误代码	描述
10014	资源被锁定。
13047	无权执行操作。
13011	找不到表 <tablename>。

示例

```
LOCK TABLE emp IN SHARED MODE;  
LOCK TABLE emp IN SHARED MODE TABLE dept IN EXCLUSIVE MODE;  
LOCK TABLE emp,dept IN SHARED MODE NOWAIT;  
LOCK TABLE emp IN LONG EXCLUSIVE MODE;
```

相关参考

第 276 页的 A.83, 『UNLOCK TABLE』

相关信息

第 265 页的 『SET SYNC MODE』

A.58 MESSAGE APPEND

```
MESSAGE unique_message_name APPEND
[
  PROPAGATE TRANSACTIONS
  [ { IGNORE_ERRORS | LOG_ERRORS | FAIL_ERRORS } ]
  [WHERE { property_name {=|<|<=|>|>=|<>} 'value_string' | ALL } ]
]
[ { REFRESH | SUBSCRIBE }
publication_name[(publication_parameters)]
timeout[(timeout_in_seconds)]
[FULL]
]
[REGISTER PUBLICATION publication_name]
[UNREGISTER PUBLICATION publication_name]
[REGISTER REPLICA]
[UNREGISTER REPLICA]
[SYNC_CONFIG ('sync_config_arg')]

```

用法

MESSAGE APPEND 语句在副本数据库中使用 MESSAGE BEGIN 语句向创建的信息中追加任务。任务包括以下几项:

- 将事务传播到主数据库
- 根据主数据库来刷新发布
- 为副本数据库预订注册或注销发布
- 对主数据库注册或注销副本数据库
- 从主数据库下载主数据库用户信息（用户名和密码列表）

该 MESSAGE APPEND 语句将仅应用于高级复制配置，且仅在副本数据库上发出。

PROPAGATE TRANSACTIONS 任务可以包含一个 WHERE 子句，后者用于只传播通过 SAVE PROPERTY 语句定义的事务属性与特定条件相符的事务。

使用关键字 ALL 将覆盖先前使用以下语句设置的任何缺省传播条件:

```
SAVE DEFAULT PROPAGATE PROPERTY WHERE property_name {=|<|<=|>|>=|<>} 'value'
```

这使您能够传播未包含任何属性的事务。

REGISTER REPLICA 任务将新的副本数据库添加到主数据库中的副本数据库列表。必须先向主数据库注册副本数据库，然后才能在副本数据库中执行任何其他同步功能。

在多主数据库环境中，要使每个主数据库与副本数据库同步，必须通过设置目录向每个主数据库注册副本数据库。一个副本数据库目录只能向一个主数据库目录注册。在同步环境中创建目录后，此语句执行实际的注册工作。要对副本数据库进行同步，必须为每个主数据库创建一个新目录。

注:

- 在单主数据库环境中，不需要使用目录。缺省情况下，未使用目录时，副本数据库的注册工作将通过一个基本目录自动进行，该基本目录映射到主数据库基本目录，后者的名称在数据库创建时指定。
- 单一副本数据库节点可以有多个主数据库，但对于每个主数据库目录，该节点必须有一个不同的副本数据库目录。
- 单一副本数据库目录不能有多个主数据库。

UNREGISTER REPLICA 选项从主数据库中的副本数据库列表中除去现有的副本数据库。

如果在发布中使用 REFRESH 任务，那么此任务可以包含该发布的自变量。参数必须是字面值；例如，不能使用存储过程变量。将关键字 FULL 与 REFRESH 配合使用将强制访存全部数据以便将其发送到副本数据库。所请求的发布必须已注册。关键字 REFRESH 与 SUBSCRIBE 是同义词；但是，在 MESSAGE APPEND 语句中，建议您不要使用 SUBSCRIBE。

REGISTER PUBLICATION 任务在副本数据库中注册发布，以便能够根据该发布来刷新该副本数据库。用户只能根据已注册的发布进行刷新。这样，将对发布参数进行验证，从而防止用户意外地预订不必要的预订或者请求临时预订。已注册的发布所引用的所有表都必须存在于副本数据库中。

UNREGISTER PUBLICATION 选项从主数据库中的已注册发布列表中除去现有的已注册发布。

SYNC_CONFIG 任务的输入自变量定义从主数据库返回到副本数据库的用户名的搜索模式。在此自变量中，字符串 *match_string* 可以使用遵循 LIKE 关键字约定的 SQL 通配符（例如符号 %）。有关使用 LIKE 关键字的详细信息，请参阅第 284 页的 A.89，『通配符』。

返回值

表 49. MESSAGE APPEND 返回值

错误代码	描述
13133	不是此产品的有效许可证。
25004	不支持动态参数。
25005	消息 <i>message_name</i> 已处于活动状态。
25006	消息 <i>message_name</i> 未处于活动状态。
25015	语法错误: <i>error_message</i> (第 <i>line_number</i> 行)
25018	消息状态不合法。 副本数据库中的追加消息必须介于 MESSAGE BEGIN 与 MESSAGE END 语句之间。
25024	未定义主数据库。
25025	未定义节点名。

表 49. MESSAGE APPEND 返回值 (续)

错误代码	描述
25026	不是有效的主数据库用户。
25028	消息 <i>message_name</i> 只能包含一个系统预订。
25035	消息 <i>message_name</i> 在使用中。 某个用户当前正在创建或转发此消息。
25044	SYNC_CONFIG 系统发布只接受字符自变量。
25056	不允许自动落实。
25071	未向发布 <i>publication_name</i> 注册。
25072	已向发布 <i>publication_name</i> 注册。

示例

```
MESSAGE MyMsg0001 APPEND PROPAGATE TRANSACTIONS;
MESSAGE MyMsg0001 APPEND REFRESH PUB_CUSTOMERS_BY_AREA('SOUTH');
MESSAGE MyMsg0001 APPEND REGISTER REPLICA;
MESSAGE MyMsg0001 APPEND SYNC_CONFIG ('S%');
MESSAGE MyMsg0001 APPEND REGISTER PUBLICATION pub1_customer;
```

A.59 MESSAGE BEGIN

```
MESSAGE unique_message_name BEGIN [TO master_node_name]
```

用法

MESSAGE BEGIN 语句用于显式地开始每一条从副本数据库发送至主数据库的消息。MESSAGE BEGIN 语句仅适用于高级复制配置。只可以在副本数据库上发出该语句。

每条消息都必须具有在副本数据库中唯一的名称。要构造唯一的消息名，您可以使用 GET_UNIQUE_STRING() 函数（第 287 页的 B.1, 『字符串函数』对此函数作了阐述）。处理消息后，可以重复使用该消息名。但是，如果该消息由于任何原因而失败，那么主数据库将保留失败消息的副本，如果您在删除该失败消息前尝试重复使用该消息名，那么该名称将不唯一。即使在能够重复使用现有名称的情况下，您可能也想使用新消息名。注意，同一个主数据库的两个副本数据库可以包含相同的消息名。

在向主数据库系统目录以外的主数据库目录注册副本数据库时，必须在 MESSAGE BEGIN 命令中提供主数据库节点名。主数据库节点名用于在主数据库中解析正确的目录。注意，仅当使用 REGISTER REPLICA 语句时，才能指定主数据库节点名。以后，消息会自动发送到正确的主数据库节点。

如果使用可选的“TO *master_node_name*”子句，那么必须将 *master_node_name* 括在双引号中。

注:

处理消息时，请确保自动落实方式始终处于关闭状态。

来自副本数据库的返回值

表 50. 来自副本数据库的 MESSAGE BEGIN 返回值

错误代码	描述
25005	消息 <i>message_name</i> 已处于活动状态。 已创建具有所指定名称的消息，并且它可能仍处于活动状态。在副本数据库中成功执行该消息的应答之后，将自动删除该消息。
25035	消息 <i>message_name</i> 在使用中。 某个用户当前正在创建或转发此消息。
25056	不允许自动落实。

来自主数据库的返回值

表 51. 来自主数据库的 MESSAGE BEGIN 返回值

错误代码	描述
25019	此数据库不是副本数据库。
25025	未定义节点名。
25056	不允许自动落实。

示例

```
MESSAGE MyMsg0001 BEGIN ;  
MESSAGE MyMsg0002 BEGIN TO "BerkeleyMaster";
```

A.60 MESSAGE DELETE

```
MESSAGE message_name [FROM REPLICA replica_name] DELETE
```

A.60.1 支持环境

此命令需要 solidDB 高级复制组件。

A.60.2 用法

如果消息的执行由于错误而终止，那么您可以使用此命令来显式地从数据库中删除此消息，以便从错误状态恢复。注意，删除此消息时，已在此消息中传播到主数据库的当前事务以及所有后续事务都将永久丢失。要使用此语句，您必须具有 SYS_SYNC_ADMIN_ROLE 访问权。

注:

此外，MESSAGE DELETE CURRENT TRANSACTION 命令允许您只删除有问题的事务，从而提供了更好的恢复方法。

如果要从主数据库中删除消息，那么还必须提供转发该消息的副本数据库的节点名。

删除消息时，请确保自动落实方式始终处于关闭状态。

A.60.3 用于主数据库

在主数据库中使用此语句来删除已失败的消息。务必使用以下语法来指定副本数据库：FROM REPLICA *replica_name*。

A.60.4 用于副本数据库

此语句用于在副本数据库中删除消息。

A.60.5 示例

```
MESSAGE MyMsg0000 DELETE ;  
MESSAGE MyMsg0001 FROM REPLICA bills_laptop DELETE ;
```

来自副本数据库的返回值

有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 52. 来自副本数据库的 MESSAGE DELETE 返回值

错误代码	描述
25005	消息 <i>message_name</i> 已处于活动状态。
25013	找不到消息 <i>message_name</i> 。
25035	消息 <i>message_name</i> 在使用中。 某个用户当前正在创建或转发此消息。
25056	不允许自动落实。

有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 53. 来自主数据库的 MESSAGE DELETE 返回值

错误代码	描述
13047	无权执行操作。
25009	找不到副本数据库 <i>replica_name</i> 。
25013	找不到消息 <i>message_name</i> 。
25020	此数据库不是主数据库。
25035	消息 <i>message_name</i> 在使用中。 某个用户当前正在执行此消息。
25056	不允许自动落实。

A.61 MESSAGE DELETE CURRENT TRANSACTION

```
MESSAGE message_name FROM REPLICA replica_name
DELETE CURRENT TRANSACTION
```

A.61.1 支持环境

此命令需要 solidDB 高级复制组件。

A.61.2 用法

此语句从主数据库中的给定消息中删除当前事务。要使用此语句，您必须具有 SYS_SYNC_ADMIN_ROLE 特权。

如果消息执行期间发生 DBMS 级错误（例如发生重复的插入），那么消息的执行将停止。您可以通过从消息中删除有问题的事务来解决此类错误。一旦使用 MESSAGE FROM REPLICA DELETE CURRENT TRANSACTION 删除当前事务，管理员就可以继续执行同步过程。

删除当前事务时，请确保自动落实方式始终处于关闭状态。

仅当消息处于错误状态时，才应该使用此语句；如果在其他情况下使用此语句，那么将返回错误消息。此语句是事务性操作，必须先进行落实，这样消息才能继续执行。要在落实删除后重新启动消息，请使用以下语句：

```
MESSAGE msgname FROM REPLICA replicaname EXECUTE
```

此删除操作在执行 MESSAGE FROM REPLICA EXECUTE 语句之前首先完成；即，此语句从副本数据库中启动消息，但等待到活动语句完成后才实际地执行该消息。因此，此语句以异步方式执行消息。

注：删除事务只能作为最后的手段；通常，应该将事务编写成能够防止主数据库中发生无法解决的冲突。MESSAGE FROM REPLICA DELETE CURRENT TRANSACTION 旨在用于频繁发生无法解决的冲突的开发阶段。

删除事务时，务必小心谨慎。由于后续事务可能依赖于所删除的事务的结果，因此，这将带来发生更多事务错误的风险。

A.61.3 用于主数据库

在主数据库中使用此语句来删除已失败的事务。

A.61.4 用于副本数据库

此语句不可用于副本数据库。

A.61.5 示例

```
MESSAGE somefailures FROM REPLICA laptop1 DELETE
CURRENT TRANSACTION;
COMMIT WORK;
MESSAGE somefailures FROM REPLICA laptop1 EXECUTE;
COMMIT WORK;
```

A.61.6 返回值

有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 54. MESSAGE DELETE CURRENT TRANSACTION 返回值

错误代码	描述
13047	无权执行操作。
25009	找不到副本数据库 <i>replica_name</i> 。
25013	找不到消息名 <i>message_name</i> 。
25018	消息状态不合法。 试图从未出错的消息中删除事务。
25056	不允许自动落实。

A.62 MESSAGE END

MESSAGE *unique_message_name* END

A.62.1 支持环境

此命令需要 solidDB 高级复制组件。

A.62.2 用法

必须先对消息进行“包装”并使之持久化，然后才能将其发送到主数据库。使用 MESSAGE END 命令结束消息将关闭该消息，即，您不再能够对其追加任何内容。落实事务将使消息持久化。

注:

处理消息时，请确保自动落实方式处于关闭状态。

A.62.3 用于主数据库

MESSAGE END 语句不可用于主数据库。

A.62.4 用于副本数据库

在副本数据库中使用 MESSAGE END 语句来结束消息。

示例

```
MESSAGE MyMsg001 END ;  
COMMIT WORK ;
```

以下示例提供了一条完整的消息，此消息将传播事务并根据发布 PUB_CUSTOMERS_BY_AREA 执行刷新。

```

MESSAGE MyMsg001 BEGIN ;
MESSAGE MyMsg001 APPEND PROPAGATE TRANSACTIONS;
MESSAGE MyMsg001 APPEND REFRESH PUB_CUSTOMERS_BY_AREA(" SOUTH');
MESSAGE MyMsg001 END ;
COMMIT WORK ;

```

A.62.5 来自副本数据库的返回值

有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 55. 来自副本数据库的 `MESSAGE END` 返回值

错误代码	描述
13133	不是此产品的有效许可证。
25005	消息 <code>message_name</code> 已处于活动状态。
25013	找不到消息 <code>message_name</code> 。
25018	消息状态不合法。 必须存在用于开始事务的 <code>MESSAGE BEGIN</code> 语句，并且，对于每条消息，只能执行一次 <code>MESSAGE END</code> 语句。
25026	不是有效的主数据库用户。
25035	消息 <code>message_name</code> 在使用中。 某个用户当前正在创建或转发此消息。
25056	不允许自动落实。

A.62.6 来自主数据库的返回值

有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 56. 来自主数据库的 `MESSAGE END` 返回值

错误代码	描述
25019	此数据库不是副本数据库。
25056	不允许自动落实。

A.63 MESSAGE EXECUTE

```
MESSAGE message_name EXECUTE [{OPTIMISTIC | PESSIMISTIC}]
```

A.63.1 支持环境

此命令需要 solidDB 高级复制组件。

A.63.2 用法

如果在副本数据库中执行应答消息失败，那么可以使用此语句来重新执行消息。例如，如果数据库服务器检测到 REFRESH 与正在执行的用户事务之间发生并行冲突，那么将发生以上情况。

如果您预期并行冲突经常发生，并且重新执行该消息由于并行冲突而失败，那么请在使用 PESSIMISTIC 选项进行表级锁定的情况下执行该消息；这将确保该消息执行成功。

在此方式下，对受影响的表进行的所有其他并发访问都将被阻塞到同步消息完成为止。否则，如果使用乐观方式，那么 MESSAGE EXECUTE 语句可能会由于并行冲突而失败。

当一个事务获取对表的互斥锁定时，solid.ini 配置文件的 General 一节中的 TableLockWaitTimeout 参数设置确定该事务在互斥锁定或共享锁定被释放前的等待时间。有关详细信息，请参阅《solidDB 管理指南》中有关此参数的描述。

注：

处理消息时，请确保自动落实方式始终处于关闭状态。

A.63.3 用于主数据库

此语句不可用于主数据库。请参阅第 241 页的 A.66，『MESSAGE FROM REPLICA EXECUTE』。

A.63.4 用于副本数据库

在副本数据库中使用此语句，以便在副本数据库中重新执行先前执行失败的消息。

A.63.5 结果集

MESSAGE EXECUTE 返回一个结果集。所返回的结果集与 MESSAGE GET REPLY 命令返回的结果集相同。

A.63.6 示例

```
MESSAGE MyMsg0002 EXECUTE;
```

A.63.7 返回值

有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 57. MESSAGE EXECUTE 返回值

错误代码	描述
13XXX	表级错误。
10006	并行冲突（同时正在执行其他操作）。
13047	无权执行操作。
13056	不允许对伪列执行插入。

表 57. MESSAGE EXECUTE 返回值 (续)

错误代码	描述
25005	消息 <i>message_name</i> 已处于活动状态。
25013	找不到消息名 <i>message_name</i> 。
25018	消息状态不合法。
25024	未定义主数据库。
25026	不是有效的主数据库用户。
25031	事务处于活动状态，操作失败。
25035	消息 <i>message_name</i> 在使用中。 某个用户当前正在创建或转发此消息。
25040	找不到用户标识 <i>user_id</i> 。 执行消息应答时，将主数据库用户映射到本地副本数据库标识的尝试失败。
25041	找不到对发布 <i>publication_name</i> 的预订。
25048	找不到发布 <i>publication_name</i> 请求信息。
25056	不允许自动落实。

A.64 MESSAGE FORWARD

```
MESSAGE unique_message_name FORWARD
[TO {'connect_string' | node_name | "node_name"} ]
[TIMEOUT {number_of_seconds | FOREVER} ]
[COMMITBLOCK block_size_in_rows]
[{'OPTIMISTIC' | PESSIMISTIC}]
```

A.64.1 支持环境

此命令需要 solidDB 高级复制组件。

A.64.2 用法

在使用 MESSAGE END 语句完成消息并使之持久化之后，可以使用 MESSAGE FORWARD 语句将其发送到主数据库。

仅当正在向主数据库注册新的副本数据库时（即，从副本数据库发送第一条消息到主服务器时），才需要使用关键字 TO 来指定消息的接收方。

connect_string 是有效的连接字符串，例如：

```
tcp [host_computer_name] server_port_number
```

有关连接字符串的更多信息，请参阅《solidDB 管理指南》中标题为『通信协议』的章节。

在 MESSAGE FORWARD 命令的上下文中，必须使用单引号对连接字符串进行定界。

node_name（不带引号）是并非作为保留字的有效字母数字序列。如果节点名是保留字，那么使用 "*node_name*"（括在双引号中）；在这种情况下，双引号确保将节点名视为定界标识。例如，由于单词 "master" 是保留字，因此将其用作节点名时，将其引在双引号中：

```
-- On master
SET SYNC NODE "master";
--On replica
MESSAGE refresh_severe_bugs2 FORWARD TO "master" TIMEOUT FOREVER;
```

发送的每条消息都有一条应答消息。TIMEOUT 属性定义副本服务器等待应答消息的时间长度。

如果未定义 TIMEOUT，那么将把该消息转发到主数据库，并且副本数据库不访存应答。在这种情况下，可以通过进行单独的 MESSAGE GET REPLY 调用来检索应答。

如果已发送的消息的应答包含对大型发布执行的 REFRESH，那么可以使用 COMMITBLOCK 属性来定义 REFRESH 的落实块大小（即，在一个事务中落实的行数）。这对副本数据库的性能有正面影响。使用 COMMITBLOCK 属性时，建议不要允许联机用户访问数据库。

作为 MESSAGE FORWARD 操作的组成部分，可以在副本数据库中最初执行应答消息时指定表级悲观锁定。如果指定了悲观方式，那么对受影响的表进行的所有其他并发访问都将被阻塞到同步消息完成为止。否则，如果使用乐观方式，那么 MESSAGE FORWARD 操作可能会由于并行冲突而失败。

当一个事务获取对表的互斥锁定时，solid.ini 配置文件的 General 一节中的 TableLockWaitTimeout 参数设置确定该事务在互斥锁定或共享锁定被释放前的等待时间段。有关详细信息，请参阅《solidDB 管理指南》中有关此参数的描述。

如果转发的消息由于通信错误而传递失败，那么您可以显式地使用 MESSAGE FORWARD 来重新发送该消息。一旦重新发送完毕，MESSAGE FORWARD 就将重新执行该消息。

注：

处理消息时，请确保自动落实方式始终处于关闭状态。

A.64.3 示例

转发消息并等待应答 60 秒：

```
MESSAGE MyMsg001 FORWARD TIMEOUT 60 ;
```

将消息转发到机器“mastermachine.acme.com”上运行的主服务器。不等待应答消息：

```
MESSAGE MyRegistrationMsg FORWARD TO
'tcp mastermachine.acme.com 1313';
```

转发消息，等待应答 5 分钟（300 秒），并使用最多包含 1000 行的事务将已刷新的发布的数据落实到副本数据库：

```
MESSAGE MyMsg001 FORWARD TIMEOUT 300 COMMITBLOCK 1000 ;
```

A.64.4 来自副本数据库的返回值

有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 58. 来自副本数据库的 MESSAGE FORWARD 返回值

错误代码	描述
13XXX	表级错误。
21XXX	通信错误
10006	并行冲突（同时正在执行其他操作）。
13047	无权执行操作。
13056	不允许对伪列执行插入。
25005	消息 <i>message_name</i> 已处于活动状态。
25013	找不到消息名 <i>message_name</i> 。
25018	消息状态不合法。 在副本数据库中，如果消息已结束并且结束事务已落实完毕，那么您只能使用 MESSAGE FORWARD 语句来执行该消息。
25024	未定义主数据库。 如果在 MESSAGE FORWARD 语句中的 <i>connect_string</i> 两旁使用了双引号而非单引号，那么将生成此消息。 例如，如果主数据库节点的节点名为 "master"（这是一个保留字，因此应该由双引号定界），并且该节点的连接字符串是： tcp localhost 1315 那么以下所示的 MESSAGE 语句正确： --On the replica ... --double quotes MESSAGE msg1 BEGIN TO "master"; ... --single quotes MESSAGE msg2 FORWARD TO 'tcp localhost 1315'; 注意，MESSAGE BEGIN 语句在副本服务器中定义主服务器的节点名。MESSAGE FORWARD 语句可以包含该服务器的连接字符串。
25026	不是有效的主数据库用户。
25031	事务处于活动状态，操作失败。
25035	消息 <i>message_name</i> 在使用中。 某个用户当前正在创建或转发此消息。

表 58. 来自副本数据库的 MESSAGE FORWARD 返回值 (续)

错误代码	描述
25040	找不到用户标识 <i>user_id</i> 。 执行消息应答时，将主数据库用户映射到本地副本数据库标识的尝试失败。
25041	找不到对发布 <i>publication_name</i> 的预订。
25048	找不到发布 <i>publication_name</i> 请求信息。
25052	未能将节点名设置为 <i>node_name</i> 。
25054	还没有为同步历史记录设置表 <i>table_name</i> 。
25055	仅当未注册时，才允许指定连接信息。 仅当副本数据库尚未向主数据库注册时，才允许在 MESSAGE <i>message_name</i> FORWARD TO <i>connect_info options</i> 中指定连接信息。
25056	不允许自动落实。
25057	副本数据库已向主数据库注册。
25060	列 <i>column_name</i> 在表 <i>table_name</i> 中的发布 <i>publication_name</i> 结果集中不存在。

A.64.5 来自主数据库的返回值

有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 59. 来自主数据库的 MESSAGE FORWARD 返回值

错误代码	描述
13XXX	表级错误。
13124	找不到用户标识 <i>num</i> 。 例如，如果已删除用户，那么将生成此消息。
25016	找不到消息，副本数据库标识为 <i>replica_id</i> ，消息标识为 <i>message_id</i> 。
25056	不允许自动落实。

结果集

如果 MESSAGE FORWARD 也检索应答，那么该语句将返回结果集。返回的结果集与 MESSAGE GET REPLY 语句返回的结果集相同。请参阅第 242 页的 A.68，『MESSAGE GET REPLY』。

A.65 MESSAGE FROM REPLICA DELETE

```
MESSAGE msgid FROM REPLICA replicaname DELETE;  
MESSAGE msgid FROM REPLICA replicaname DELETE CURRENT TRANSACTION;
```

只能对主数据库执行此命令。

A.66 MESSAGE FROM REPLICA EXECUTE

```
MESSAGE message_name FROM REPLICA replica_name EXECUTE
```

A.66.1 支持环境

此命令需要 solidDB 高级复制组件。

A.66.2 用法

如果消息执行期间发生 DBMS 级错误（例如发生重复的插入），或者过程通过将 SYS_ROLLBACK 参数放至事务公告牌而引发错误，那么消息的执行将停止。您可以通过修正错误原因从此类错误恢复，例如，从数据库中除去重复的行，然后执行消息。

使用 MESSAGE DELETE CURRENT TRANSACTION 来删除有错误的事务时，此删除在 MESSAGE FROM REPLICA EXECUTE 命令执行前完成；即，此语句从副本数据库中启动消息，但等待到活动语句完成后才实际地执行该消息。因此，此命令以异步方式执行消息。

注:

处理消息时，请确保自动落实方式始终处于关闭状态。

A.66.3 用于主数据库

此命令用于在主数据库中执行已失败的消息。

A.66.4 用于副本数据库

此命令不可用于副本数据库。请参阅第 235 页的 A.63，『MESSAGE EXECUTE』以了解备用方案。

A.66.5 示例

```
MESSAGE MyMsg0002 FROM REPLICA bills_laptop EXECUTE;
```

A.66.6 返回值

有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 60. MESSAGE FROM REPLICA EXECUTE 返回值

错误代码	描述
13047	无权执行操作。
25009	找不到副本数据库 <i>replica_name</i> 。

表 60. MESSAGE FROM REPLICA EXECUTE 返回值 (续)

错误代码	描述
25013	找不到消息名 <i>message_name</i> 。
25018	消息状态不合法。 试图从未出错的消息中删除事务。
25056	不允许自动落实。

A.67 MESSAGE FROM REPLICA RESTART

```
MESSAGE msgid FROM REPLICA replicaname RESTART <err-options>;
```

其中, <*err-options*> 可以是 IGNORE_ERRORS、LOG_ERRORS 或 FAIL_ERRORS。

只能对主数据库执行此命令。

此命令允许您重新执行存储在系统表中并且可以使用 SYNC_FAILED_MESSAGES 视图进行检索的已失败事务。

A.68 MESSAGE GET REPLY

```
MESSAGE unique_message_name GET REPLY
[TIMEOUT {FOREVER | seconds}]
[COMMITBLOCK block_size_in_rows]
[NO EXECUTE]
[{OPTIMISTIC | PESSIMISTIC}]
```

A.68.1 支持环境

此命令需要 solidDB 高级复制组件。

A.68.2 用法

如果 MESSAGE FORWARD 语句未接收到对已发送的消息的应答, 那么您可以通过在副本数据库中使用 MESSAGE GET REPLY 语句向主数据库单独请求获取该应答。

如果应答消息包含对大型发布执行的 REFRESH, 那么可以使用 COMMITBLOCK 属性来限制 REFRESH 的落实块大小 (即, 在一个事务中落实的行数)。这对副本数据库的性能有正面影响。使用 COMMITBLOCK 属性时, 建议不要允许联机用户访问数据库。

如果在副本数据库中执行带有 COMMITBLOCK 属性的应答消息失败, 那么无法执行该消息。您必须从副本数据库中删除失败的消息, 然后根据主数据库执行刷新。

如果指定了 NO EXECUTE, 那么当主数据库中的应答消息可用时, 将仅仅读取并存储该消息以供将来执行。否则, 将从主数据库下载该应答消息并在同一个语句中执行该消息。使用 NO EXECUTE 将允许以后在不同的事务中执行应答消息, 从而减少了通信线路中的瓶颈。

可以将应答消息定义为最初执行时使用表级悲观锁定方式。如果指定了悲观方式，那么对受影响的表进行的所有其他并发访问都将被阻塞到同步消息完成为止。否则，如果使用乐观方式，那么 MESSAGE GET REPLY 操作可能会由于并行冲突而失败。

当一个事务获取对表的互斥锁定时，solid.ini 配置文件的 General 一节中的 TableLockWaitTimeout 参数设置确定该事务在互斥锁定或共享锁定被释放前的等待时间段。有关详细信息，请参阅《solidDB 管理指南》中有关此参数的描述。

如果应答消息由于通信错误而传递失败（未指定 COMMITBLOCK），那么您可以显式地使用 MESSAGE GET REPLY 来重新发送该消息。一旦重新发送完毕，MESSAGE GET REPLY 就将重新执行该消息。

注：

处理消息时，请确保自动落实方式始终处于关闭状态。

A.68.3 用于主数据库

MESSAGE GET REPLY 不可用于主数据库。

A.68.4 用于副本数据库

在副本数据库中使用 MESSAGE GET REPLY 来访问主数据库中的消息应答。

A.68.5 示例

```
MESSAGE MyMessage001 GET REPLY TIMEOUT 120
MESSAGE MyMessage001 GET REPLY TIMEOUT 300 COMMITBLOCK 1000
```

A.68.6 来自副本数据库的返回值

事务传播过程中发生的致命错误将导致消息中止，并且将返回一个错误代码给副本数据库。要传播已中止的消息，您需要更正致命错误，然后使用 MESSAGE FROM REPLICAS EXECUTE 命令来重新启动该消息。

如果 REFRESH 在主数据库中失败，那么将在结果集中添加有关已失败的 REFRESH 的错误消息。该消息的其他部件将正常执行。对于失败的 REFRESH，必须在单独的同步消息中从主数据库执行 REFRESH。

即使 REFRESH（即，应答消息的执行）在副本数据库中失败，该消息也仍然包含在副本数据库中，并可以由 MESSAGE msg_name EXECUTE 命令重新启动。

有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 61. 来自副本数据库的 MESSAGE GET REPLY 返回值

错误代码	描述
13XXX	表级错误。
13124	找不到用户标识 num。 例如，如果已删除用户，那么将生成此消息。

表 61. 来自副本数据库的 MESSAGE GET REPLY 返回值 (续)

错误代码	描述
10006	并行冲突（同时正在执行其他操作）。
13047	无权执行操作。
13056	不允许对伪列执行插入。
21XXX	通信错误
25005	消息 <i>message_name</i> 已处于活动状态。
25013	找不到消息名 <i>message_name</i> 。
25018	消息状态不合法。 在副本数据库中，如果已把消息转发到主数据库，那么您只能使用 MESSAGE GET REPLY 语句来执行该消息。
25024	未定义主数据库。
25026	不是有效的主数据库用户。
25031	事务处于活动状态，操作失败。
25035	消息 <i>message_name</i> 在使用中。某个用户当前正在创建或转发此消息。
25036	找不到发布 <i>publication_name</i> 或者发布版本不匹配。
25040	找不到用户标识 <i>user_id</i> 。 执行消息应答时，将主数据库用户映射到本地副本数据库标识的尝试失败。
25041	找不到对发布 <i>publication_name</i> 的预订。
25048	找不到发布 <i>publication_name</i> 请求信息。
25054	还没有为同步历史记录设置表 <i>table_name</i> 。
25056	不允许自动落实。
25057	已向主数据库 <i>master_name</i> 注册。
25060	列 <i>column_name</i> 在表 <i>table_name</i> 中的发布 <i>publication_name</i> 结果集中不存在。

A.68.7 来自主数据库的返回值

有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 62. 来自主数据库的 MESSAGE GET REPLY 返回值

错误代码	描述
13XXX	表级错误。
13124	找不到用户标识 <i>num</i> 。 例如，如果已删除用户，那么将生成此消息。
25012	消息应答已超时。
25016	找不到消息，副本数据库标识为 <i>replica-id</i> ，消息标识为 <i>message-id</i> 。
25043	应答消息太长（ <i>size_of_messages</i> 个字节）。最大值设置为 <i>max_message_size</i> 个字节。
25056	不允许自动落实。

A.68.8 结果集

MESSAGE GET REPLY 返回一个结果集表。此结果集的列如下所示：

表 63. MESSAGE GET REPLY 结果集表

列名	描述
部件号	消息部件号
类型	结果集行的类型。可能的类型是： 0: 消息部件开始 1: 此类型未在使用中 2: 此消息是传播消息，该操作的状态存储在返回消息中 3: 任务 4: 预订任务 5: 刷新类型（FULL 或 INCREMENTAL） 6: MESSAGE DELETE 状态
Masterid	主数据库标识
Msgid	消息标识
Errcode	消息错误代码。如果成功，那么值为零。
Errstr	消息错误字符串。如果成功，那么值为 NULL。

表 63. MESSAGE GET REPLY 结果集表 (续)

列名	描述
Insertcount	插入到副本数据库的行数。 Type = 3: 插入总数 Type = 4: 从副本数据库历史记录复原到副本数据库基本表的行插入数 Type = 5: 从主数据库接收的插入操作数
Deletecount	Type = 3: 删除总数 Type = 4: 从副本数据库基本表复原的行删除数 Type = 5: 从主数据库接收的删除操作数
Bytecount	消息的大小（以字节计）。在从命令 MESSAGE END 接收的结果中指示。否则为 0。
Info	当前任务的信息。 Type = 0: 消息名 Type = 3: 发布名 Type = 4: 表名 Type = 5: FULL/INCREMENTAL

A.69 POST EVENT

```
post_statement ::= POST EVENT event_name [( parameters) ]
                [UNIQUE | DATA UNIQUE]
```

仅在存储过程中允许使用 POST EVENT 语句；它可用于发布系统事件和用户定义的事件。有关更多详细信息，请参阅 第 168 页的 A.14.6，『post_statement』。

相关参考

第 168 页的 A.14.6，『post_statement』

相关信息

第 73 页的 3.5，『事件』

A.70 REFRESH

```
REFRESH publication [parameters] [FULL]
[OPTIMISTIC|PESSIMISTIC]
[COMMITBLOCK number_of_rows]
[TIMEOUT {DEFAULT | FOREVER | timeout_ms} ]
```

A.70.1 用法

REFRESH 语句是无存储器刷新命令。此命令通过以流式方法传输相关联数据来节省内存。由于不将任何消息写入磁盘，因此还能节省 I/T 带宽。每条命令都将阻塞到它执行成功为止。

可选属性 OPTIMISTIC|PESSIMISTIC 定义副本表的锁定方式。

- OPTIMISTIC 方式（缺省值）指定，并行控制方法取决于表类型和隔离级别。对于 OPTIMISTIC 方式下的 D 表而言，REFRESH 将始终成功。对于一般的 M 表以及 PESSIMISTIC 方式下的 D 表而言，将使用行级锁定方法。如果无法获取锁定，那么 PESSIMISTIC 将失败并返回错误。
- PESSIMISTIC 指定在刷新期间以互斥方式锁定表，而不考虑表类型以及所选隔离级别。如果无法获取锁定，那么刷新请求将失败并返回错误。

如果对 REFRESH 请求的应答包含对大型发布执行的 REFRESH，那么可以使用 COMMITBLOCK 属性来定义 REFRESH 的落实块大小（即，在一个事务中落实的行数）。这对副本数据库的性能有正面影响。使用 COMMITBLOCK 属性时，建议不要允许联机用户访问数据库。

如果未使用 COMMITBLOCK，那么 REFRESH 的执行是当前事务的组成部分。可以通过发出 ROLLBACK 命令来撤销 REFRESH 的效果。为了使 REFRESH 的效果可耐久，必须发出 COMMIT WORK。就可以通过回滚和落实反复发出而言，REFRESH 具有幂等性，并且当数据库处于静止状态时，效果始终相同。

如果使用了 COMMITBLOCK 子句，那么将以隐式方式在副本数据库中落实每个具有指定大小的传输部件。ROLLBACK 语句将仅除去最后一个传输部件的效果。COMMIT WORK 将落实最后一个传输部件。

TIMEOUT 属性定义副本服务器等待应答消息的时间长度。如果未定义 TIMEOUT，那么将使用 FOREVER。

A.70.2 示例

同步无消息刷新:

```
REFRESH publ_states;  
PESSIMISTIC;  
COMMITBLOCK 1000;  
COMMIT WORK;
```

A.70.3 返回值

有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 64. REFRESH 返回值

错误代码	描述
13133	不是此产品的有效许可证。
25004	不支持动态参数。
25015	语法错误: <i>error_message</i> (第 <i>line_number</i> 行)
25024	未定义主数据库。
25025	未定义节点名。
25026	不是有效的主数据库用户。

表 64. REFRESH 返回值 (续)

错误代码	描述
25044	SYNC_CONFIG 系统发布只接受字符自变量。
25056	不允许自动落实。
25071	未向发布 <i>publication_name</i> 注册。
25072	已向发布 <i>publication_name</i> 注册。
13XXX	表级错误。
21XXX	通信错误
10006	并行冲突（同时正在执行其他操作）。
13047	无权执行操作。
13056	不允许对伪列执行插入。
25005	消息 <i>message_name</i> 已处于活动状态。
25018	消息状态不合法。 在副本数据库中，如果消息已结束并且结束事务已落实完毕，那么您只能使用 MESSAGE FORWARD 语句来执行该消息。
25024	未定义主数据库。 如果在 MESSAGE FORWARD 语句中的 <i>connect_string</i> 两旁使用了双引号而非单引号，那么将生成此消息。 例如，如果主数据库节点的节点名为 "master"（这是一个保留字，因此应该由双引号定界），并且该节点的连接字符串是： tcp localhost 1315 那么以下所示的 MESSAGE 语句正确： --On the replica ... --double quotes MESSAGE msg1 BEGIN TO "master"; ... --single quotes MESSAGE msg2 FORWARD TO 'tcp localhost 1315'; 注意，MESSAGE BEGIN 语句在副本服务器中定义主服务器的节点名。MESSAGE FORWARD 语句可以包含该服务器的连接字符串。
25026	不是有效的主数据库用户。
25031	事务处于活动状态，操作失败。
25035	消息 <i>message_name</i> 在使用中。 某个用户当前正在创建或转发此消息。

表 64. REFRESH 返回值 (续)

错误代码	描述
25040	找不到用户标识 <i>user_id</i> 。 执行消息应答时，将主数据库用户映射到本地副本数据库标识的尝试失败。
25041	找不到对发布 <i>publication_name</i> 的预订。
25048	找不到发布 <i>publication_name</i> 请求信息。
25052	未能将节点名设置为 <i>node_name</i> 。
25054	还没有为同步历史记录设置表 <i>table_name</i> 。
25055	仅当未注册时，才允许指定连接信息。 仅当副本数据库尚未向主数据库注册时，才允许在 MESSAGE <i>message_name</i> FORWARD TO <i>connect_info options</i> 中指定连接信息。
25056	不允许自动落实。
25057	副本数据库已向主数据库注册。
25060	列 <i>column_name</i> 在表 <i>table_name</i> 中的发布 <i>publication_name</i> 结果集中不存在。
13XXX	表级错误。
13124	找不到用户标识 <i>num</i> 。 例如，如果已删除用户，那么将生成此消息。
25056	不允许自动落实。

A.71 REGISTER EVENT

```
wait_register_statement ::= REGISTER EVENT event_name
```

REGISTER EVENT 语句告诉服务器您希望以后每次遇到该指定事件时都受到提醒，即使尚未等待此事件亦如此。

您只能在存储过程中使用 REGISTER EVENT 语句。有关更多详细信息，请参阅第 168 页的 A.14.7，『*wait_register_statement*』。

相关参考

第 168 页的 A.14.7, 『wait_register_statement』

相关信息

第 73 页的 3.5, 『事件』

A.72 REVOKE

```
REVOKE { role_name [, role_name ]... }
      FROM {PUBLIC | user_name [, user_name ]... }

REVOKE
  {ALL | revoke_privilege [, revoke_privilege]... } ON table-name
  FROM {PUBLIC | user_name [, user_name]... | role_name [, role_name]... }

revoke-privilege ::= DELETE | INSERT | SELECT |
  UPDATE [( column_identifier [, column_identifier]... )] |
  REFERENCES

REVOKE EXECUTE ON procedure_name
      FROM {PUBLIC | user_name [, user_name]... | role_name [, role_name]... }

REVOKE {SELECT | INSERT} ON event_name FROM
      {PUBLIC | user_name [, user_name]... | role_name [, role_name]... }

REVOKE {SELECT | INSERT} ON sequence_name
      FROM {PUBLIC | user_name [, user_name]... | role_name [, role_name]... }
```

用法

REVOKE 语句从用户除去角色或者从用户与角色除去特权。

注: 在 REVOKE 语句中不支持关键字 CASCADE 和 RESTRICT。

示例

以下命令从用户 HOBBS 除去角色 GUEST_USERS:

```
REVOKE GUEST_USERS FROM HOBBS;
```

以下命令从角色 GUEST_USERS 除去特权来插入到事件 TEST 中:

```
REVOKE INSERT ON TEST FROM GUEST_USERS;
```

相关参考

第 214 页的 A.50, 『GRANT』

相关信息

第 84 页的 4.2, 『管理用户特权和角色』

A.73 REVOKE PASSTHROUGH

```
REVOKE PASSTHROUGH
      FROM {PUBLIC | user_name [, user_name]... | role_name [, role_name]... }
```

支持环境

此命令需要使用 SQL 传递功能。

用法

REVOKE PASSTHROUGH 语句撤销对 SQL 传递的访问权。

示例

```
REVOKE PASSTHROUGH READ FROM cdcuser1, cdcuser2
REVOKE PASSTHROUGH WRITE FROM cdcuser1, cdcuser2
```

另请参阅

第 214 页的 A.51, 『GRANT PASSTHROUGH』

A.74 REVOKE REFRESH

```
REVOKE { REFRESH | SUBSCRIBE } ON publication_name FROM { PUBLIC |
    user_name, [ user_name ] ... |
    role_name, [ role_name ] ... }
```

A.74.1 支持环境

此命令需要 solidDB 高级复制组件。

A.74.2 用法

此语句撤销主数据库中定义的用户或角色对发布具有的访问权。

注:

关键字“REFRESH”与“SUBSCRIBE”是同义词。但是, 在 REVOKE 语句中, 已废弃关键字“SUBSCRIBE”。

A.74.3 用于主数据库

使用此语句来撤销用户或角色对发布拥有的访问权。

A.74.4 用于副本数据库

此语句不可用于副本数据库。

A.74.5 示例

```
REVOKE REFRESH ON customers_by_area FROM joe_smith;
REVOKE REFRESH ON customers_by_area FROM all_salesmen;
```

A.74.6 返回值

表 65. REVOKE REFRESH 返回值

错误代码	描述
13137	授权/撤销方式不合法。
13048	不具有 GRANT OPTION 特权。
25010	找不到发布 <i>name</i> 。

A.75 ROLLBACK WORK

ROLLBACK [WORK]

用法

ROLLBACK 或 ROLLBACK WORK 语句废弃当前事务在数据库中做的更改。此语句也会终止事务。

相关参考

第 154 页的 A.8, 『COMMIT WORK』

A.76 SAVE

SAVE [NO CHECK] [{ IGNORE_ERRORS | LOG_ERRORS | FAIL_ERRORS }]
[{ AUTOSAVE | AUTOSAVEONLY }] *sql_statement*

A.76.1 支持环境

此命令需要 solidDB 高级复制组件。

A.76.2 用法

对于需要传播到主数据库的事务，您必须将该事务的语句显式地保存到副本数据库的事务队列。要完成此任务，请在事务语句前添加 SAVE 语句。

只有主数据库用户才能保存语句。这是因为，在主数据库中执行保存的语句时，必须使用主数据库中某个用户的适当访问权来执行那些数据。在主数据库中执行保存的语句时，将使用保存该语句时在副本数据库中处于活动状态的主数据库用户的访问权。如果副本数据库中的用户已映射到主数据库中的用户，那么 SAVE 语句将使用主数据库中的用户的访问权。

与事务传播相关的错误处理的缺省行为是，失败的事务暂停执行消息；这将中止当前正在执行的事务并阻止执行同一消息中的任何后续事务。但是，您可以选择另一种错误处理行为。

下面说明 SAVE 命令的选项：

NO CHECK: 此选项表示不在副本数据库中准备语句。如果命令在副本数据库中没有意义，那么此选项很实用。例如，如果 SQL 命令调用一个存在于主数据库中但不存在于副本数据库中的存储过程，那么您不会希望副本数据库尝试准备该语句。如果使用此选项，那么该语句不能带有参数标记。

IGNORE_ERRORS: 此选项表示如果语句在主数据库中执行时失败，然后将忽略失败的语句，并且事务将中止。但是，只有该事务终止，而不是整条消息中止。主数据库将从失败事务后的第一个事务开始继续执行该消息。

LOG_ERRORS: 此选项表示如果语句在主数据库中执行时失败，然后将忽略失败的语句，并且当前事务将中止。失败的事务的语句将保存在 SYS_SYNC_RECEIVED_STMTS 系统表中，以供以后执行或调查。您可以使用 SYNC_FAILED_MESSAGES 系统视图来检查失败的事务，并可以使用 MESSAGE <msg_id> FROM REPLICA <replica_name> RESTART -statement 从该视图中重新执行那些事务。

注意，与 IGNORE_ERROR 选项相同，中止事务并不会中止整条消息。主数据库将从失败事务后的第一个事务开始继续执行该消息。

FAIL_ERRORS: 此选项表示语句失败时，主数据库将停止执行消息。这是缺省行为。

AUTOSAVE: 此选项意味着在主数据库中执行该语句，并且，如果该主数据库还是另外某个主数据库的副本数据库（即，作为中间层节点），那么自动保存该语句以便进一步传播。

AUTOSAVEONLY: 此选项意味着不在主数据库中执行该语句，而是，如果该主数据库还是另外某个主数据库的副本数据库（即，作为中间层节点），那么自动保存该语句以便进一步传播。

A.76.3 用于主数据库

此语句不可用于主数据库。

A.76.4 用于副本数据库

在副本数据库中使用此语句，以便保存要传播到主数据库的语句。

A.76.5 示例

```
SAVE INSERT INTO mytbl (col1, col2) VALUES ('calvin', 'hobbes')
SAVE CALL SP_UPDATE_MYTAB('calvin_1', 'hobbes')
SAVE CALL SP_DELETE_MYTAB('calvin')
SAVE NO CHECK IGNORE_ERRORS insert into mytab values(1,2)
```

A.76.6 返回值

有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 66. SAVE 返回值

错误代码	描述
25001	内部错误 主数据库超出保存语句所需的数据库大小限制。
25003	无法保存 SAVE 语句。
25070	在事务中，只能为一个主数据库保存语句。

A.77 SAVE PROPERTY

```
SAVE PROPERTY property_name VALUE 'value_string'
SAVE PROPERTY property_name VALUE NONE
SAVE DEFAULT PROPERTY property_name VALUE 'value_string'
SAVE DEFAULT PROPERTY property_name VALUE NONE
SAVE DEFAULT PROPAGATE PROPERTY WHERE name {=|<|<=|>|>=|<>} 'value'
SAVE DEFAULT PROPAGATE PROPERTY NONE
```


A.77.1 支持环境

此命令需要 solidDB 高级复制组件。

A.77.2 用法

可以使用以下命令对当前活动事务指定属性:

```
SAVE PROPERTY property_name VALUE 'value_string'
```

主数据库中事务的语句可以通过调用 GET_PARAM() 函数来访问这些属性。在副本数据库中, 只存在适用于以下命令的属性:

```
MESSAGE APPEND unique_message_name PROPAGATE TRANSACTIONS  
WHERE property > 'value_string'
```

在主数据库中执行事务时, 保存的属性将被放入该事务的参数公告牌。如果这个保存的属性已存在, 那么新值将覆盖先前的值。

也可以定义缺省属性, 这些属性将保存到当前连接的所有事务。用于执行此操作的语句如下所示:

```
SAVE DEFAULT PROPERTY property_name VALUE 'value_string'
```

可以使用 SAVE DEFAULT PROPAGATE PROPERTY WHERE 语句来保存缺省事务传播条件。例如, 此语句可用于设置当前连接中创建的事务的传播优先级。

可以在连接级使用 SAVE DEFAULT PROPAGATE PROPERTY WHERE *property* > '*value*' 对所有 MESSAGE *unique_message_name* APPEND PROPAGATE TRANSACTIONS 追加缺省的 WHERE 语句。如果还在 PROPAGATE 语句中输入了 WHERE 语句, 那么它将覆盖使用 DEFAULT PROPAGATE PROPERTY 设置的 WHERE 语句。

通过使用值字符串 NONE 来重新保存属性, 可以除去属性或缺省属性。

A.77.3 用于主数据库

此语句不可用于主数据库。

A.77.4 用于副本数据库

可以在副本数据库中使用这些语句为已经为了传播到主数据库而保存的事务设置属性。在主数据库中, 可以读取属性的值。

A.77.5 “PUT_PARAM()”与“SAVE PROPERTY *property_name* VALUE *property_value*”之间的区别

有关“SAVE PROPERTY”与“PUT_PARAM()”之间的区别的讨论, 请参阅有关 PUT_PARAM() 函数的描述。

A.77.6 示例

```
SAVE PROPERTY conflict_rule VALUE 'override'  
SAVE DEFAULT PROPERTY userid VALUE 'scott'  
SAVE DEFAULT PROPERTY userid VALUE NONE  
SAVE DEFAULT PROPAGATE PROPERTY WHERE priority > '2'
```

A.77.7 返回值

有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 67. *SAVE PROPERTY* 返回值

错误代码	描述
13086	参数中的数据类型无效。

A.77.8 结果集

SAVE PROPERTY 不返回结果集。

A.78 SELECT

```
SELECT [ALL | DISTINCT] select-list
      LEVEL
      FROM table_reference_list
      [WHERE search_condition]
      [GROUP BY column_name [, column_name]... ]
      [HAVING search_condition]
      [hierarchical_condition]
      [[UNION | INTERSECT | EXCEPT] [ALL] select_statement]...
      [ORDER BY expression]
      [ASC | DESC]
      [LIMIT row_count [OFFSET skipped_rows] | LIMIT skipped_rows,row_count |
      FETCH FIRST ROW ONLY | FETCH FIRST constant ROWS ONLY]
      [FOR UPDATE]
```

用法

SELECT 语句从数据库检索值。您可以从一个或多个表中选择零个或多个记录。*SELECT* 操作一般调用为查询。

LEVEL

LEVEL 是只在分层查询的上下文中才有效的伪列。如果将结果集视为由相互引用的行组成的树，那么 *LEVEL* 列将生成树层编号，并且顶层的行的层编号为“1”。

select-list

请参阅第 282 页的 A.87.4，『*query_specification*』和 第 284 页的 A.87.7，『*SELECT* 语句中的伪列』。

table_reference_list

请参阅第 283 页的 A.87.6，『*table_reference*』。

search_condition

请参阅第 282 页的 A.87.5，『搜索条件』。

hierarchical_condition

```
::= START WITH search_condition CONNECT BY [PRIOR] search_condition
```

如果表包含分层数据，那么可以使用分层查询子句按分层顺序选择行。在分层查询子句中，**START WITH** 指定层次结构的根行，**CONNECT BY** 指定层次结构中父行与子行之间的关系。**CONNECT BY** 条件不能包含子查询。

在分层查询中，条件中的一个表达式必须由 **PRIOR** 运算符限定才能引用父行。**PRIOR** 是一元运算符，其优先顺序与一元 **+** 和 **-** 算术运算符相同。在面向分层查询中当前行的父行的情况下，此运算符对紧随其后的表达式进行求值。**PRIOR** 最常用于使用相等运算符来比较列值的场合。**PRIOR** 关键字可以位于该运算符的任何一端。

ORDER BY *expression*

ORDER SIBLINGS BY 致使处于任何层的行相应地进行排序。

另请参阅第 280 页的 A.87.3，『表达式』。

LIMIT

非标准的子句 **LIMIT row_count OFFSET skipped_rows** 允许使用大小为 *row_count* 并且定位于 *skipped_rows+1* 行的滑动窗口来屏蔽结果集的部分内容。*skipped_rows* 的负数值导致错误，而 *row_count* 的负数值导致完整生成的结果集。

可以使用两种格式：例如，**LIMIT 24 OFFSET 10** 与 **LIMIT 10, 24** 等同。

FETCH

FETCH FIRST ROW ONLY 子句仅返回表的第一行。**FETCH FIRST constant ROWS ONLY** 返回用 *constant* 定义的行数。

FOR UPDATE

当用户使用 **SELECT... FOR UPDATE** 语句来访问某一行时，将使用更新方式锁定来锁定该行。这意味着没有任何其他用户能够读取或更新该行，并且将确保当前用户以后能够更新该行。有关更多详细信息，请参阅第 103 页的 5.2，『并行控制和锁定』。

示例

```
SELECT ID FROM TEST;
SELECT DISTINCT ID, C FROM TEST WHERE ID = 5;
SELECT DISTINCT ID FROM TEST ORDER BY ID ASC;
SELECT NAME, ADDRESS FROM CUSTOMERS
UNION
SELECT NAME, DEP FROM PERSONNEL;
SELECT dept, count(*) FROM person
GROUP BY dept
ORDER BY dept
LIMIT 20 OFFSET 10
```

LIMIT 和 FETCH 的示例

```
SELECT * FROM SYS_TABLES LIMIT 1;
SELECT * FROM SYS_TABLES FETCH FIRST ROW ONLY; -- same as above

SELECT * FROM SYS_TABLES WHERE TABLE_NAME LIKE 'A%' LIMIT 5;
SELECT * FROM SYS_TABLES WHERE TABLE_NAME LIKE 'A%' FETCH
FIRST 5 ROWS ONLY; -- 同上
```

START WITH 示例

```
SELECT last_name, employee_id, manager_id, LEVEL
FROM employees
START WITH employee_id = 100
CONNECT BY PRIOR employee_id = manager_id
ORDER SIBLINGS BY last_name;
```

LAST_NAME	EMPLOYEE_ID	MANAGER_ID
King	100	
Cambrault	148	100
Bates	172	148
Bloom	169	148
Fox	170	148
Kumar	173	148
Ozer	168	148
Smith	171	148
De Haan	102	100
Hunold	103	102
Austin	105	103
Ernst	104	103
Lorentz	107	103
Pataballa	106	103
Errazuriz	147	100
Ande	166	147
Banda	167	147

LEVEL 和 ORDER SIBLINGS BY 的示例

```
SELECT last_name, employee_id, manager_id, LEVEL
FROM employees
START WITH last_name = 'King'
CONNECT BY PRIOR employee_id = manager_id
ORDER SIBLINGS BY last_name
ORDER BY LEVEL;
```

LAST_NAME	EMPLOYEE_ID	MANAGER_ID	LEVEL
King	100	NULL	1
Cambrault	148	100	2
De Haan	102	100	2
Bates	172	148	3
Bloom	169	148	3
Gates	104	148	3
Hunold	103	102	3
Hope	202	172	4
Smith	201	172	4

A.79 SET

SET 命令应用于从中执行这些命令的用户会话（连接）。它们不会影响其他用户会话。

您随时可以发出 SET 语句；但是，它们并非全都会立即生效。下列语句将立即生效：

- SET CATALOG
- SET IDLE TIMEOUT
- SET LOCK TIMEOUT
- SET OPTIMISTIC LOCK TIMEOUT
- SET SCHEMA
- SET STATEMENT MAXTIME

下列语句在下一个 COMMIT WORK 之后生效:

- SET DURABILITY
- SET ISOLATION LEVEL
- SET { READ ONLY | READ WRITE | WRITE }

SET 语句无法回滚; 即使发出它们的事务中止或回滚, 它们也将保持生效。您最好在事务中的任何 DDL/DML SQL 语句之前发出 SET 语句。

设置将一直生效到会话(连接)结束或者另一个 SET 命令更改设置为止, 在某些情况下, 将一直生效到执行优先顺序更高的命令(例如, SET TRANSACTION)为止。

示例

```
SET CATALOG myCatalog;  
SET DURABILITY STRICT;  
SET IDLE TIMEOUT 30;  
SET ISOLATION LEVEL REPEATABLE READ;  
SET OPTIMISTIC LOCK TIMEOUT 30;  
SET LOCK TIMEOUT 30;  
SET LOCK TIMEOUT 500MS;  
SET READ ONLY;  
SET SCHEMA 'accounting_info';  
SET SCHEMA 'john_smith';  
SET STATEMENT MAXTIME 180;
```

另请参阅

第 271 页的 A.80, 『SET TRANSACTION』

A.79.1 SET 与 SET TRANSACTION 之间的差别

SET 和 SET TRANSACTION 命令用于设置多个事务属性, 如隔离级别、读数级别、或 SQL 传递方式。

SET 命令为会话设置属性并经常称为会话级别命令。SET TRANSACTION 命令为一个事务设置属性, 这些命令称为事务级别命令。

事务级命令遵循的规则与会话级命令不同。这些区别列示如下。

- 事务级命令在发出这些命令的事务中生效; 会话级命令在下一个事务中生效, 即在下一个 COMMIT WORK 之后生效。
- 事务级命令仅应用于当前事务; 会话级命令应用于所有后续事务 - 即, 直到会话(连接)结束或另一个 SET 命令对它们进行更改为止。
- 事务级命令必须在事务开始时执行, 即, 在任何 DML 或 DDL 语句之前执行。但是, 它们可以在其他 SET 语句之后执行。如果违反此规则, 那么将返回错误。会话级命令可以在事务期间的任何时间点执行。
- 事务级命令优先于会话级命令。但是, 事务级命令仅应用于当前事务。在当前事务已落实或中止后, 这些设置将恢复为最近一条 SET 命令(如果有的话)设置的值。
例如:

```
COMMIT WORK; -- Finish previous transaction;  
SET ISOLATION LEVEL SERIALIZABLE;  
COMMIT WORK; -- Isolation level is now SERIALIZABLE  
...  
COMMIT WORK;  
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

```
-- Isolation level is now REPEATABLE READ because
-- transaction-level settings take precedence
-- over session-level settings.
COMMIT WORK;
-- Isolation level is now back to SERIALIZABLE, since the
-- transaction-level settings applied only to that
-- transaction.
```

隔离级别设置和读级别设置的完整优先顺序层次结构如下所示。项越接近于列表顶部，优先顺序越高。

1. SET TRANSACTION ... (事务级别命令)
2. SET ... (会话级设置)
3. 服务器级别设置由 `solid.ini` 配置参数中的值 (例如, **IsolationLevel** 或 **DurabilityLevel**) 进行指定
4. 服务器的缺省设置 (参数出厂值)。

A.79.2 SET (读/写级别)

```
SET {READ ONLY | READ WRITE | WRITE}
```

SET {READ ONLY | READ WRITE | WRITE} 允许您指定是只允许连接执行读操作、允许其执行读写操作还是只允许其执行写操作。

另请参阅第 260 页的 A.79.6, 『SET ISOLATION LEVEL』。

A.79.3 SET CATALOG

```
SET CATALOG catalog_name
```

SET CATALOG 设置连接中的当前目录上下文。

A.79.4 SET DELETE CAPTURE

```
SET DELETE CAPTURE {NONE | CHANGES}
```

用法

SET DELETE CAPTURE 语句设置 solidDB Universal Cache 中的数据帐龄方式。数据帐龄从前端中除去表行，但将它们保留在后端中。

- SET DELETE CAPTURE NONE: 启用数据帐龄
- SET DELETE CAPTURE CHANGES: 禁用数据帐龄

SET DELETE CAPTURE 语句从下一个 SQL 语句开始立即生效，直到类似的语句或 SET TRANSACTION DELETE CAPTURE 将其还原为止。

另请参阅

第 272 页的 A.80.3, 『SET TRANSACTION DELETE CAPTURE』

A.79.5 SET DURABILITY

```
SET DURABILITY { RELAXED | STRICT | DEFAULT }
```

SET DURABILITY 设置事务持久性级别。

- **STRICT** 表示落实事务时，服务器立即将信息写入事务日志文件。
- **RELAXED** 表示服务器可能不会在落实事务时立即写入信息；例如，服务器可能等到不怎么忙的时候或者能够通过单一写操作写入多个事务时才写入该信息。如果使用宽松耐久性，并且服务器异常关闭，您可能会丢失部分最新事务。
- **DEFAULT** 表示已使用 **Logging.DurabilityLevel** 参数定义的耐久性级别。

有关更多详细信息，请参阅 第 112 页的 5.3，『选择事务耐久性级别』。

A.79.6 SET ISOLATION LEVEL

```
SET ISOLATION LEVEL {
    READ COMMITTED |
    REPEATABLE READ |
    SERIALIZABLE }
```

SET ISOLATION LEVEL 用于指定隔离级别。

如果已指定的工作负载服务器为辅助服务器，那么可以通过编程将其更改为主数据库。在会话级别，下列语句将工作负载连接服务器更改为主数据库：

```
SET WRITE (非标准)
SET ISOLATION LEVEL REPEATABLE READ
SET ISOLATION LEVEL SERIALIZABLE
```

如果此语句是事务的第一个语句，那么它将立即生效，否则将从下一个事务开始生效。

如果上述语句不适用，那么它将返回 **SQL_SUCCESS** 并且不执行任何操作。例如，将 **SET WRITE** 应用于独立服务器时，情况即如此。在这种情况下，**SET WRITE** 的语义与 **SET READ WRITE** 等同。

使用 **SET READ WRITE or ... READ ONLY (SQL:1999)** 语句可以撤销 **SET WRITE** 语句的效果。并且，隔离级别语句的效果也相同：

```
SET ISOLATION LEVEL READ COMMITTED
```

A.79.7 SET PASSTHROUGH

```
SET PASSTHROUGH {READ <passthrough level> [WRITE <passthrough level>]}
| {WRITE <passthrough level> | [READ <passthrough level>]}
| <passthrough level>
```

其中：

```
passthrough level ::= NONE | CONDITIONAL | FORCE | DEFAULT
```

用法

SET PASSTHROUGH 语句设置 **solidDB Universal Cache** 中的 **SQL** 传递方式。

- **NONE**: 不使用 **SQL** 传递；没有命令从前端传递到后端
- **CONDITIONAL**: **SQL** 传递由缺少的表或语法错误激活。
- **FORCE**: 使用 **SQL** 传递来将所有语句从前端传递到后端
- **DEFAULT**: 导致使用 **SQL** 传递的当前会话缺省值

SET PASSTHROUGH 语句从下一个 SQL 语句开始立即生效，直到类似的语句或 SET TRANSACTION PASSTHROUGH 将其还原为止。

另请参阅

第 273 页的 A.80.6, 『SET TRANSACTION PASSTHROUGH』

A.79.8 SET SAFENESS

```
SET SAFENESS {1SAFE | 2SAFE | DEFAULT}
```

SET SAFENESS 确定复制协议是同步的 (2-safe) 还是异步的 (1-safe)。

- 1-safe: 首先在主服务器中落实事务，然后再将此事务落实到辅助服务器中
- 2-safe: 在未得到辅助服务器确认之前不落实事务 (这是缺省值)。

SET SAFENESS 设置当前会话的安全级别。

A.79.9 SET SCHEMA

```
SET SCHEMA {'schema_name' | USER | 'user_name'}
```

用法

solidDB 支持 SQL89 样式的模式。模式用于帮助唯一地标识数据库中的实体 (表和视图)。通过使用模式，每个用户都可以创建实体，而不必担心他选择的名称是否与其他用户/模式选择的名称冲突。

要唯一地标识实体 (例如表)，您通过指定目录名和模式名对该实体进行“限定”。以下是全限定的表名的一个示例:

```
FinanceCatalog.AccountsReceivableSchema.CustomersTable
```

为了遵循 ANSI SQL-92 标准，可以将 user_name 或 schema_name 括在单引号中。

可以使用 SET SCHEMA 语句来更改缺省模式。通过使用 SET SCHEMA USER 语句，可以将模式更改为当前用户名。此外，可以将模式设置为“user_name”，这必须是数据库中的有效用户名。

用于解析实体名 [schema_name.]table_identifier 的算法如下所示:

1. 如果指定了 schema_name，那么将只在该模式中搜索 table_identifier。
2. 如果指定了 schema_name，那么
 - a. 将在缺省模式中搜索第一个 table_identifier。缺省模式最初与用户名相同，但可以使用 SET SCHEMA 语句进行更改。
 - b. 然后，在数据库中的所有模式中搜索 table_identifier。如果找到不止一个具有相同 identifier 和类型 (表和存储过程等等) 的实体，那么将返回新的错误代码 13110 (实体名 table_identifier 不明确)。

SET SCHEMA 语句只影响缺省实体名解析，而不会更改对数据库实体的任何访问权。此语句设置当前会话中由 EXECDIRECT 语句或 PREPARE 语句准备的语句中未限定名称的缺省模式名。

示例

```
SET SCHEMA 'CUSTOMERS';
```

另请参阅

目录还用于限定（唯一地标识）表和其他数据库实体的名称，因此，您可能还想参阅有关 SET CATALOG 命令的信息。

A.79.10 SET SEQUENCE

```
SET SEQUENCE <sequence_name> VALUE <value>
```

用法

SET SEQUENCE 语句将以动态的方式设置序列值。

<value> 是数字序号。序列值存储为 BIGINT 数据类型。

当序列值已设好时，下一个返回值为 <value> + 1。

示例

```
SELECT MYSEQ.NEXT;
      MYSEQ.NEXT
-----
                        81 rows fetched.SET SEQUENCE MYSEQ VALUE 5;
Command completed successfully, 0 rows affected.
```

```
SELECT MYSEQ.NEXT;
      MYSEQ.NEXT
-----
                        61 rows fetched.
```

另请参阅

第 186 页的 A.20, 『CREATE SEQUENCE』

A.79.11 SET SQL

```
SET SQL INFO {ON | OFF} [FILE {file_name | "{file_name}" | '{file_name}'}
      [LEVEL info_level]
SET SQL SORTARRAYSIZE {array-size | DEFAULT}
SET SQL JOINPATHSPAN { | DEFAULT}
SET SQL CONVERTORSTOUNIONS
      {YES [COUNT ] | NO | DEFAULT}
```

用法

所有设置都将在每次用户会话时读取（这与 solid.ini 文件中的设置不同，该文件中的设置将在 solidDB 每次启动时自动读取）。

SET SQL INFO: SET SQL INFO 命令允许您打开跟踪信息，这些信息可以帮助您调试问题或调整查询。对于 SQL INFO，缺省文件是由所有用户共享的全局 soltrace.out。如果指定了文件名，那么除非设置新文件，否则所有将来的 INFO ON 设置都将使用该文件。建议您将文件名括在单引号中，否则该文件名将转换为大写。信息输出会追加到该文件，并且系统永远不会截断该文件；所以，不再需要信息文件时，用户必须以手动方式将该文件删除。如果打开文件失败，那么将以静默方式废弃信息输出。

缺省 SQL INFO LEVEL 是 4。一种生成实用信息输出的好方法是，打开信息输出功能并指定新文件名，然后使用 EXPLAIN PLAN FOR 语法来执行 SQL 语句。此方法将提供所有必需的估算程序信息，但不会生成访存操作的输出（访存操作可能会生成巨大的输出文件）。

SET SQL SORTARRAYSIZE: 此命令设置 SQL 对查询的结果集进行排序时使用的数组大小。单位是“行”- 例如，如果指定的值为 1000，那么服务器将创建大小足以对 1000 行进行排序的数组。

SET SQL JOINPATHSPAN: 此命令已过时。虽然接受了该语法，但此命令没有任何效果。

SET SQL CONVERTORSTOUNIONS 允许将包含“OR”运算的查询转换为使用“UNION”运算的等同查询。下列运算在逻辑上等同：

```
select ... where x = 1 OR y = 1;  
select ... where x = 1 UNION select... where y = 1;
```

通过设置 CONVERTORSTOUNIONS，您告知优化器，如果基于数据量和数据分布的 UNION 可能效率更高，那么可以使用等同的 UNION 运算代替 OR 运算。SQL CONVERTORSTOUNIONS 中的 COUNT 参数（“将 OR 转换为 UNION”）指定可以转换为 UNION 运算的 OR 运算的最大数目。注意，还可以通过使用 solid.ini 配置参数 ConvertORsToUNIONS 来指定 CONVERTORSTOUNIONS（有关详细信息，请参阅《solidDB 管理指南》中有关此参数的描述）。缺省值是 100，这在几乎所有情况下都适用。

示例

```
SET SQL INFO ON FILE 'sqlinfo.txt' LEVEL 5
```

A.79.12 SET STATEMENT MAXTIME

```
SET STATEMENT MAXTIME minutes
```

SET STATEMENT MAXTIME 设置特定于连接的最大执行时间（以分钟计）。此设置在您设置新的最大时间之前将一直有效。时间为零表示没有最大时间，这也是缺省值。

A.79.13 SET SYNC

下列各章描述不同的 SET SYNC 命令。

SET SYNC master_or_replica

```
SET SYNC master_or_replica yes_or_no
```

其中：

```
master_or_replica ::= MASTER | REPLICA  
yes_or_no ::= YES | NO
```

支持环境： 此命令需要 solidDB 高级复制组件。

用法： 创建和配置用于同步功能的数据库目录时，必须使用此命令来指定该数据库是主数据库、副本数据库还是同时作为这两种数据库。只有 DBA 或者具有 SYS_SYNC_ADMIN_ROLE 角色的用户才能设置数据库角色。

如果域中存在根据来自此数据库的发布执行刷新和/或将事务传播至此数据库的副本数据库，那么此数据库目录是主数据库。如果此数据库目录可以根据主数据库中的发布来执行刷新，那么它是副本数据库目录。在多层同步中，中间层数据库扮演双重角色，即，同时作为主数据库和副本数据库。

注意，使用此命令要求已使用 `SET SYNC NODE` 命令对主数据库或副本数据库设置节点名。有关详细信息，请参阅第 267 页的『`SET SYNC NODE`』。

将数据库设置为具有双重角色时，可以将此语句使用一或两次。例如：

```
SET SYNC MASTER YES;
SET SYNC REPLICA YES;
```

注意，将数据库设置为具有双重角色时，`SET SYNC REPLICA YES` 不会覆盖 `SET SYNC MASTER YES`。只有以下显式语句才会覆盖主数据库的状态：

```
SET SYNC MASTER NO;
```

一旦被覆盖，当前数据库就被设置为只具有副本数据库角色。

示例：

```
-- configure as replica
SET SYNC REPLICA YES;
-- configure as master
SET SYNC MASTER YES;
```

返回值： 有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 68. `SET SYNC` 返回值

错误代码	描述
13047	无权执行操作。
13107	集合操作不合法。
13133	不是此产品的有效许可证。
25051	找到未完成的消息。

SET SYNC CONNECT

```
SET SYNC CONNECT 'connect_string [,connect_string]' TO MASTER
master_name
SET SYNC CONNECT 'connect_string' TO REPLICA replica_name
```

支持环境： 此命令需要 solidDB 高级复制组件。

用法： 此语句更改与数据库名称相关联的网络名。每次在副本服务器（或主服务器）所连接的数据库中更改网络名之后，请在副本服务器（或主服务器）中使用此语句。网络名由 `solid.ini` 配置文件中的 `Listen` 参数定义。

`SET SYNC CONNECT ... TO MASTER` 中的第二个连接字符串用于在主服务器发生故障时透明地从副本服务器进行故障转移以切换到备用的主服务器。连接字符串的顺序并不重要。将自动维护与当前活动的主服务器的连接。

用于主数据库: 在主数据库中使用此语句来更改副本数据库的网络名。

用于副本数据库: 在副本数据库中使用此语句来更改主数据库的网络名。

示例:

```
SET SYNC CONNECT 'tcp server.company.com 1313' TO MASTER hq_master;
```

返回值: 有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 69. SET SYNC CONNECT 返回值

错误代码	描述
13047	无权执行操作。
13107	集合操作不合法。
21300	网络协议不合法。
25007	找不到主数据库 <i>master_name</i> 。
25019	此数据库不是副本数据库。

SET SYNC MODE

```
SET SYNC MODE { MAINTENANCE | NORMAL }
```

支持环境: 此命令需要 solidDB 高级复制组件。

用法: 此命令将当前目录的同步方式设置为维护方式或正常方式。

此命令仅适用于同步操作所涉及的目录（即，主数据库目录或副本数据库目录，或者 3 层或 3 层以上层次结构中同时作为主数据库和副本数据库的目录）。

此命令仅适用于当前目录。如果要将多个目录的同步方式设置为维护方式，那么必须使用 SET CATALOG 命令对每个目录进行切换，然后对该目录发出 SET SYNC MODE MAINTENANCE 命令。

当目录的同步方式为维护方式时，下列规则适用：

- 该目录将不会发送或接收同步消息，因此将不会参与同步活动（例如，执行刷新或响应刷新请求）。
- 允许对发布所引用的表执行 DDL 命令（例如 ALTER TABLE）。
- 同步方式更改时，服务器将发送系统事件 SYNC_MAINTENANCEMODE_BEGIN 或 SYNC_MAINTENANCEMODE_END。
- 如果用 REPLACE 选项更改（删除或重新创建）主目录的发布，则在下次副本目录从更改的发布中刷新时，该发布的元数据（内部发布定义数据）将自动刷新至每个副本目录。（无论刷新发布时数据库是否处于维护同步方式，情况均如此。）
- 每个目录在参数公告牌中都有一个名为 SYNC_MODE 的只读参数，因此，应用程序可以检查该目录的方式。该参数的值是“MAINTENANCE”（如果该目录处于维护同步方式）或“NORMAL”（如果该目录未处于维护同步方式）。如果该目录不是主数据库目录或副本数据库目录，那么值为 NULL。

- 用户必须具有 DBA 或同步管理特权才能将同步方式设置为维护方式或正常方式。
- 用户同时可以有多个处于维护同步方式的目录。
- 如果打开此方式的会话断开连接，那么此方式将关闭。
- 系统将禁用常规同步历史记录操作。例如，对已打开同步历史记录的表执行删除或更新操作时，同步历史记录表将不会存储“原始”行（即，已删除或更新前的行）。但请注意，删除和更新操作将应用于同步历史记录表，例如：

```
DELETE * FROM T WHERE c = 5
```

将从历史记录表以及基本表中删除行。下表说明当同步方式设置为维护方式时，各种操作（INSERT 和 DELETE 等等）如何应用于主数据库和副本数据库中的同步历史记录表。

表 70. 不同的操作如何应用于同步历史记录表

操作	主数据库	副本数据库
INSERT	将行插入到基本表。	将行插入到基本表并将其标记为正式行。
UPDATE	同时更新基本表和历史记录。	同时更新基本表和历史记录。不更新试验性/正式状态，以使试验性行保持处于试验性状态，并使正式行保持处于正式状态。
DELETE	从基本表和历史记录中删除行。	从基本表和历史记录中删除行。
添加、更改和删除列	还对历史记录执行同一操作。	还对历史记录执行同一操作。
更改表方式	不更改历史记录方式	不更改历史记录方式
创建索引	还对历史记录创建同一索引	还对历史记录创建同一索引
创建触发器	不对历史记录创建触发器	不对历史记录创建触发器

示例:

```
SET SYNC MODE MAINTENANCE SET SYNC MODE NORMAL
```

返回值: 有关每个错误代码的详细信息，请参阅《IBM solidDB 管理员指南》中标题为『错误代码』的附录。

表 71. SET SYNC MODE 返回值

错误代码	描述
13047	无权执行操作。
13133	不是此产品的有效许可证。
25021	此数据库不是主数据库或副本数据库。此操作仅适用于主数据库和副本数据库。
25088	目录已处于维护方式。您已打开该方式。
25089	不允许关闭维护方式。另一个用户已打开该方式，因此您无法将其关闭。

表 71. SET SYNC MODE 返回值 (续)

错误代码	描述
25090	目录已处于维护方式。另一个用户已打开该方式，因此您无法将其打开。
25091	目录未处于维护方式。您尝试关闭该方式，但该方式当前未处于打开状态。

SET SYNC NODE

SET SYNC NODE {*unique_node_name* | NONE}

支持环境: 此命令需要 solidDB 高级复制组件。

用法: 指定节点名这一操作是副本数据库注册过程的组成部分。solidDB 环境的每个目录都必须具有在域中唯一的节点名。一个目录只能具有一个节点名。不同的目录不能具有相同的节点名。

在下列情况下，可以使用 SET SYNC NODE *unique_node_name* 选项将节点重命名:

- 该节点是副本数据库并且尚未向主数据库注册

和/或

- 该节点是主数据库，并且在主数据库中未注册任何副本数据库

以下是将节点重命名的示例:

```
SET SYNC NODE A; -- Now the node name is A.
SET SYNC NODE B; -- Now the node name is B.
COMMIT WORK;
SET SYNC NODE C; -- Now the node name is C.
ROLLBACK WORK; -- Now the node name is rolled back to B.
SET SYNC NODE NONE; -- Now the node has no name.
COMMIT WORK;
```

unique_node_name 必须遵循用于对数据库中其他对象（例如表）进行命名的规则。请不要将节点名括在单引号中。

如果指定 NONE，那么此命令将除去当前节点名。

如果要使用保留字（例如“NONE”）作为节点名，那么必须将该关键字括在双引号中，以确保它被视为定界标识。例如:

```
SET SYNC NODE "NONE"; -- Now the node name is "NONE"
```

您可以使用以下语句来验证所指定的节点名:

```
SELECT GET_PARAM('SYNC NODE')
```

SET SYNC NODE NONE 选项将从当前目录中除去该节点名。当您正在删除已同步的数据库并除去其注册时，请使用此选项。

注:

使用 SET SYNC NODE NONE 选项时，请确保与该节点名相关联的目录未被定义为主数据库目录和/或副本数据库目录。要除去该节点名，必须对该目录定义 SET SYNC

MASTER NO 和/或 SET SYNC REPLICA NO。如果您尝试将主数据库目录和/或副本数据库目录的节点名设置为 NONE，那么 solidDB 将返回错误消息 25082。

用于主数据库: 在主数据库中使用此语句，以便设置或删除当前目录的节点名。

用于副本数据库: 在副本数据库中使用此语句，以便设置或删除当前目录的节点名。

示例:

```
SET SYNC NODE SalesmanJones;
```

返回值: 有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 72. SET SYNC NODE 返回值

错误代码	描述
13047	无权执行操作。
13107	集合操作不合法。
25059	执行注册后，不能更改节点名。
25082	如果节点是主数据库或副本数据库，那么无法除去节点名。

SET SYNC PARAMETER

```
SET SYNC PARAMETER parameter_name 'value_as_string';  
SET SYNC PARAMETER parameter_name NONE;
```

支持环境: 此命令需要 solidDB 高级复制组件。

用法: 此语句定义通过参数公告牌提供给目录中执行的所有事务使用的持久目录级参数。每个目录都有一组不同的参数。

如果该参数已存在，那么新值将覆盖先前的值。通过将现有参数的值设置为 NONE，可以删除该参数。所有参数都存储在 SYS_BULLETIN_BOARD 系统表中。

这些参数不会传播到主数据库。

除了特定于系统的参数以外，您还可以在系统表中存储许多用于配置同步功能的系统参数。“SQL 参考”末尾列示了可用的系统参数。

用于主数据库: 在主数据库中使用 SET SYNC PARAMETER 来设置数据库参数。

用于副本数据库: 在副本数据库中使用 SET SYNC PARAMETER 来设置数据库参数。

示例:

```
SET SYNC PARAMETER db_type 'REPLICA'  
SET SYNC PARAMETER db_type NONE
```

返回值: 有关每个错误代码的详细信息，请参阅《solidDB 管理指南》中标题为『错误代码』的附录。

表 73. SET SYNC PARAMETER 返回值

错误代码	描述
13086	参数中的数据类型无效。

另请参阅: GET_PARAM

PUT_PARAM

SET SYNC PROPERTY

在主数据库中的语法:

```
SET SYNC PROPERTY <propertyname> = { 'value' | NONE } FOR REPLICA
<replicaname>
```

在副本数据库中的语法:

```
SAVE SET SYNC PROPERTY <propertyname> = { 'value' | NONE }
```

支持环境: 此命令需要 solidDB 高级复制组件。

用法: 此命令用于对副本数据库指定属性名和属性值。可以对具有属性的副本数据库进行分组，并且，使用 START AFTER COMMIT 语句时，可以指定一个组。例如，您的一些副本数据库与自行车行业相关，其他副本数据库与冲浪板行业相关，并且您想单独地更新每组副本数据库。您可以使用属性名对这些副本数据库进行分组。一个组的所有成员都具有同一个属性，并且该属性的值相同。

有关更多信息，请参阅 *solidDB Advanced Replication Guide* 中标题为“Replica Property Names”的章节。

示例: 主数据库:

```
SET SYNC PROPERTY color = 'red' FOR REPLICA replica1;
SET SYNC PROPERTY color = NONE FOR REPLICA replica1;
```

副本数据库:

```
SAVE SET SYNC PROPERTY color = 'red';
SAVE SET SYNC PROPERTY color = NONE;
```

SET SYNC USER

```
SET SYNC USER master_username IDENTIFIED BY password
SET SYNC USER NONE
```

支持环境: 此命令需要 solidDB 高级复制组件。

用法: 此语句用于定义在主数据库中注册副本数据库时执行的注册过程所使用的用户名和密码。要使用此命令，您必须具有 SYS_SYNC_ADMIN_ROLE 访问权。

注:

SET SYNC USER 语句仅用于副本数据库注册过程。除注册以外，所有其他同步操作都要求副本数据库包含有效的主数据库用户标识。如果您要对副本数据库指定另一个主数据库用户，那么必须将副本数据库中的副本数据库用户标识映射到主数据库中的主

数据库用户标识。有关详细信息，请参阅 *solidDB Advanced Replication Guide* 中标题为“Mapping Replica User ID With Master User ID”的章节。

您在主数据库中定义注册用户名。指定的名称必须具有足够的权限来执行副本数据库注册任务。要将注册权限授予主数据库中的主数据库用户，请使用 `GRANT rolename TO user` 语句对该用户指定 `SYS_SYNC_REGISTER_ROLE` 或 `SYS_SYNC_ADMIN_ROLE` 角色。

成功完成注册后，必须将同步用户复位为 `NONE`；否则，如果主数据库用户保存语句、传播消息、根据发布执行刷新或者向发布注册，那么将返回以下错误消息：

不允许所定义的用户执行此操作。

用于主数据库： 此语句不可用于主数据库。

用于副本数据库： 此语句用于在副本数据库中设置用户名。

示例：

```
SET SYNC USER homer IDENTIFIED BY marge;
SET SYNC USER NONE;
```

A.79.14 SET TIMEOUT

```
SET IDLE TIMEOUT { timeout_in_seconds |
                  timeout_in_millisecondsMS | DEFAULT }
SET LOCK TIMEOUT { timeout_in_seconds |
                  timeout_in_millisecondsMS }
SET OPTIMISTIC LOCK TIMEOUT { timeout_in_seconds |
                              timeout_in_millisecondsMS }
```

- `SET IDLE TIMEOUT` 设置特定于连接的最大超时（以秒计）。此设置在您指定新的超时之前将一直有效。如果将超时设置为 `DEFAULT`，那么表示没有最大时间。
- `SET LOCK TIMEOUT` 设置引擎等待释放锁定所需的时间（以秒计）。

在悲观并行控制方式下，`SET LOCK TIMEOUT` 语句应用于表。

缺省情况下，锁定超时为 30 秒。超时锁定的最大值为 1000 秒，超过 1000 秒的 `SET LOCK TIMEOUT` 将失败。

`SET LOCK TIMEOUT` 不更改已用 `LOCK TABLE WAIT` 设置了表级超时的表。

- 在乐观并行控制方式中，`SET OPTIMISTIC LOCK TIMEOUT` 为采用 `SELECT FOR UPDATE` 语句的行锁定设置超时。只有其他 `SELECT FOR UPDATE` 语句竞争同一表的行时，该超时才生效。在缺省情况下，超时为零（无需等待）。

缺省情况下，超时定义为秒。通过在值后面添加“MS”，可以设置详细程度为毫秒的锁定超时，例如：

```
SET LOCK TIMEOUT 500MS;
SET LOCK TIMEOUT 1500 MS;
```

提示：“MS”前的空格并不重要，并且，可以使用大写和小写字母。如果未指定“MS”，那么锁定超时将以秒计。

达到超时时间间隔时，`solidDB` 将终止发生超时的语句。有关更多信息，请参阅第 111 页的『设置锁定超时』。

A.80 SET TRANSACTION

SET TRANSACTION 命令在事务开始时生效，并会一直影响事务，直到事务落实或中止为止。如果在事务过程中发出 SET TRANSACTION 语句，那么会返回错误。

SET TRANSACTION 命令基于 ANSI SQL。但是，solidDB 实现与 ANSI 定义有一些区别。solidDB 不支持把两个 ANSI 定义的事务属性结合到一个 SET TRANSACTION 语句中，例如：

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE, READ WRITE;
```

然而，solidDB 支持在单一事务中使用多个 SET TRANSACTION 语句，例如：

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
SET TRANSACTION READ WRITE;
```

示例

```
SET TRANSACTION DURABILITY RELAXED;  
SET TRANSACTION ISOLATION REPEATABLE READ;  
SET TRANSACTION READ WRITE;
```

另请参阅

第 257 页的 A.79，『SET』

A.80.1 SET 与 SET TRANSACTION 之间的差别

SET 和 SET TRANSACTION 命令用于设置多个事务属性，如隔离级别、读数级别、或 SQL 传递方式。

SET 命令为会话设置属性并经常称为会话级别命令。SET TRANSACTION 命令为一个事务设置属性，这些命令称为事务级别命令。

事务级命令遵循的规则与会话级命令不同。这些区别列示如下。

- 事务级命令在发出这些命令的事务中生效；会话级命令在下一个事务中生效，即在下一个 COMMIT WORK 之后生效。
 - 事务级命令仅应用于当前事务；会话级命令应用于所有后续事务 - 即，直到会话（连接）结束或另一个 SET 命令对它们进行更改为止。
 - 事务级命令必须在事务开始时执行，即，在任何 DML 或 DDL 语句之前执行。但是，它们可以在其他 SET 语句之后执行。如果违反此规则，那么将返回错误。会话级命令可以在事务期间的任何时间点执行。
 - 事务级命令优先于会话级命令。但是，事务级命令仅应用于当前事务。在当前事务已落实或中止后，这些设置将恢复为最近一条 SET 命令（如果有的话）设置的值。
- 例如：

```
COMMIT WORK; -- Finish previous transaction;  
SET ISOLATION LEVEL SERIALIZABLE;  
COMMIT WORK; -- Isolation level is now SERIALIZABLE  
...  
COMMIT WORK;  
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;  
-- Isolation level is now REPEATABLE READ because  
-- transaction-level settings take precedence  
-- over session-level settings.
```

```
COMMIT WORK;
-- Isolation level is now back to SERIALIZABLE, since the
-- transaction-level settings applied only to that
-- transaction.
```

隔离级别设置和读级别设置的完整优先顺序层次结构如下所示。项越接近于列表顶部，优先顺序越高。

1. SET TRANSACTION ... (事务级别命令)
2. SET ... (会话级设置)
3. 服务器级别设置由 `solid.ini` 配置参数中的值 (例如, `IsolationLevel` 或 `DurabilityLevel`) 进行指定
4. 服务器的缺省设置 (参数出厂值)。

A.80.2 SET TRANSACTION (读/写级别)

```
SET TRANSACTION {READ ONLY | READ WRITE | WRITE}
```

SET TRANSACTION { READ ONLY | READ WRITE | WRITE} 命令基于 ANSI SQL。此命令允许用户指定是否允许事务对数据进行任何更改。

可用“热备用”配置中的动态负载均衡对事务级读写级设置进行操作。如果已指定的工作负载服务器为辅助服务器，那么可以通过编程在一个事务的持续期间将其更改为主数据库。在事务级别，下列语句会在一个事务的时间内将工作负载连接服务器更改为“主数据库”：

```
SET TRANSACTION WRITE (非标准)
```

受影响的事务是通过使用此语句启动的事务。在“主数据库”中执行事务之后，工作负载连接服务器将恢复为会话缺省模式。

A.80.3 SET TRANSACTION DELETE CAPTURE

```
SET TRANSACTION DELETE CAPTURE {NONE | CHANGES | DEFAULT}
```

用法

SET TRANSACTION DELETE CAPTURE 语句设置下一个事务的数据帐龄方式。数据帐龄从前端中除去表行，但将它们保留在后端中。

- NONE: 启用数据帐龄。
- CHANGES: 禁用数据帐龄。
- DEFAULT: 数据捕获方式恢复为使用先前的 SET DELETE CAPTURE 语句设置的方式，或者服务器的缺省值。

SET TRANSACTION DELETE CAPTURE 语句在事务开始时生效，它会一直影响事务，直到事务落实或中止为止。如果在执行事务过程中发出该语句，那么会返回错误。

另请参阅

第 259 页的 A.79.4, 『SET DELETE CAPTURE』

A.80.4 SET TRANSACTION DURABILITY

```
SET TRANSACTION DURABILITY {RELAXED | STRICT}
```

SET TRANSACTION DURABILITY { RELAXED | STRICT } 命令控制服务器是否使用严格 或宽松耐久性来执行事务日志记录。此命令是 solidDB 对 SQL 的扩展；它不是 ANSI 标准的组成部分。

必须在事务开始时给定 SET TRANSACTION DURABILITY 命令。耐久性设置仅适用于当前事务，直到该事务落实或中止。

如果新的事务耐久性设置为 STRICT，任何尚未写入磁盘的先前已落实的事务都将在 SET TRANSACTION DURABILITY STRICT 事务落实的时候写入磁盘。这表示在事务耐久性级别更改至 STRICT 的时候待处理事务不会写入磁盘中；直到使用 SET TRANSACTION DURABILITY STRICT 命令启动的当前事务落实后，该写操作才会继续。

注：SET TRANSACTION DURABILITY 中没有 DEFAULT 选项将值设置为 **Logging.DurabilityLevel** 参数指定的任何值。相反，在事务落实（或中止）后，服务器将返回执行 SET TRANSACTION DURABILITY 语句之前所设置的耐久性级别。

A.80.5 SET TRANSACTION ISOLATION LEVEL

```
SET TRANSACTION ISOLATION LEVEL {  
    READ COMMITTED |  
    REPEATABLE READ |  
    SERIALIZABLE}
```

SET TRANSACTION ISOLATION 命令基于 ANSI SQL。设置事务隔离级别为以下其一：

- 落实读
- REPEATABLE READ
- SERIALIZABLE

有关隔离级别的更多详细信息，请参阅 [选择隔离级别](#)。

A.80.6 SET TRANSACTION PASSTHROUGH

```
SET TRANSACTION PASSTHROUGH {READ <passthrough level> [WRITE <passthrough level>]}  
| {WRITE <passthrough level> | [READ <passthrough level>]}  
| <passthrough level>
```

其中：

```
passthrough level ::= NONE | CONDITIONAL | FORCE | DEFAULT
```

用法

SET TRANSACTION PASSTHROUGH 语句设置 solidDB Universal Cache 中下一个事务的 SQL 传递方式。

- NONE：不使用 SQL 传递；没有命令从前端传递到后端
- CONDITIONAL：SQL 传递由缺少的表或语法错误激活。
- FORCE：使用 SQL 传递来将所有语句从前端传递到后端
- DEFAULT：导致使用 SQL 传递的当前会话缺省值

SET TRANSACTION PASSTHROUGH 语句在事务开始时生效，它会一直影响事务，直到事务落实或中止为止。如果在执行事务过程中发出该语句，那么会返回错误。

另请参阅

第 260 页的 A.79.7, 『SET PASSTHROUGH』

A.80.7 SET TRANSACTION SAFENESS

SET TRANSACTION SAFENESS {1SAFE | 2SAFE | DEFAULT}

SET TRANSACTION SAFENESS 确定复制协议是同步的 (2-safe) 还是异步的 (1-safe)。

- 1-safe: 首先在主服务器中落实事务, 然后再将此事务落实到辅助服务器中
- 2-safe: 在未得到辅助服务器确认之前不落实事务 (这是缺省值)。

SET TRANSACTION SAFENESS 设置当前事务的安全级别。

A.81 START AFTER COMMIT

```
START AFTER COMMIT
  [FOR EACH REPLICA WHERE search_condition [RETRY retry_spec]]
  {UNIQUE | NONUNIQUE} stmt;

stmt ::= any SQL statement.
search_condition ::= search_item | search_item {AND|OR } search_item
search_item ::= {search_test | (search_condition)}
search_test ::= comparison_test | like_test
comparison_test ::= property_name { = | <> | > | >= | < | <= } value
property_name ::= name of a replica property
like_test ::= property_name [NOT] LIKE value [ESCAPE value]
value ::= literal
retry_spec ::= seconds,count
```

用法

START AFTER COMMIT 语句指定当前事务落实时要执行的 SQL 语句 (例如存储过程调用)。如果该事务回滚, 那么不会执行指定的 SQL 语句。

使用 START AFTER COMMIT 在后台启动的语句将在独立的事务中执行。该事务以自动落实方式执行, 即它一启动就无法回滚。

START AFTER COMMIT 语句返回包含一个 INTEGER 列的结果集。此整数是唯一的“作业”标识, 并可用于查询由于 SQL 语句无效、访问权不足以及副本数据库不可用等问题而未能启动的语句的状态。

如果在 <stmt> 之前使用 UNIQUE 关键字, 那么仅当尚无等同的语句正在执行或处于暂挂状态时, 才会执行该语句。语句以简单字符串比较方式进行比较。例如, call foo(1) 与 call foo(2) 不同。服务器还将考虑该语句是否已在同一个或另一个副本数据库中执行 (或者处于暂挂状态等待执行); 只有同一个副本数据库中完全相同的语句才需废弃。

要点: 使用 UNIQUE 关键字来废弃重复语句时, 会废弃最新的语句, 而最旧的语句将保持运行。很可能会出现这样一种情况: 您执行多次更新, 并且触发了多项 START AFTER COMMIT 操作, 但只有最旧的操作执行, 因此最新更新的数据可能不会立即发送到副本数据库。

NONUNIQUE 表示可以在后台同时执行重复的语句。

FOR EACH REPLICA 指定对每个满足 WHERE 子句的 search_condition 部分所指定属性条件的副本数据执行该语句。在执行该语句之前，将与该副本数据库建立连接。过程调用启动后，该过程可以使用关键字 DEFAULT 来获取“当前”副本数据库名称。

如果指定了 RETRY，并且第一次尝试时未能与副本数据库取得联系，那么将在 N 秒（由 retry_spec 中的秒数定义）之后重新执行该操作。此计数指定重试次数。

后台语句的上下文

在后台启动的语句将在发出 START AFTER COMMIT 语句的用户的上下文中执行，并且将在从中执行 START AFTER COMMIT 语句的目录和模式中执行。

在以下示例中，“CALL FOO”将在目录“katmandu”和模式“steinbeck”中执行。

```
SET CATALOG katmandu;
SET SCHEMA steinbeck;
START AFTER COMMIT UNIQUE CALL FOO;
COMMIT WORK;
SET CATALOG irrelevant_catalog;
SET SCHEMA irrelevant_schema
```

耐久性

后台语句不持久。换言之，不保证执行 START AFTER COMMIT 所启动的语句。

回滚

后台语句在启动后无法回滚。在使用 START AFTER COMMIT 启动的语句执行成功之后，无法将其回滚。

START AFTER COMMIT 语句本身可以回滚，这将导致指定的语句无法执行。例如：

```
START AFTER COMMIT UNIQUE INSERT INTO MyTable VALUES (1);
ROLLBACK;
```

在以上示例中，事务将回滚，因此将不会执行 INSERT INTO MyTable VALUES (1)。

执行顺序

后台语句以异步方式执行，它们的顺序不确定，即使在一个事务中亦如此。

示例

在后台启动本地过程。

```
START AFTER COMMIT NONUNIQUE CALL myproc;
```

如果“CALL myproc”尚未在后台运行，那么启动调用。

```
START AFTER COMMIT UNIQUE call myproc;
```

在使用属性“color”为“blue”的副本数据库的情况下，在后台启动过程。

```
START AFTER COMMIT FOR EACH REPLICA WHERE color='blue' UNIQUE CALL myproc;
```

系统认为下列语句全都是不同的语句，因此它们将执行，尽管它们都包含 UNIQUE 关键字。name 是每个副本数据库的唯一属性。

```
START AFTER COMMIT UNIQUE call myproc;  
START AFTER COMMIT FOR EACH REPLICa WHERE name='R1' UNIQUE call myproc;  
START AFTER COMMIT FOR EACH REPLICa WHERE name='R2' UNIQUE call myproc;  
START AFTER COMMIT FOR EACH REPLICa WHERE name='R3' UNIQUE call myproc;
```

但是，如果在先前语句的事务中执行以下语句，并且条件“color='blue'”与副本数据库 R1、R2 或 R3 的其中一些匹配，那么将不再对那些副本数据库执行该调用。

```
START AFTER COMMIT FOR EACH REPLICa WHERE color='blue' UNIQUE call myproc;
```

相关信息

第 21 页的 3, 『SQL 扩展』

solidDB 中的 SQL 支持能和任何以 SQL 为基础的高级系统媲美；solidDB 提供了绝大部分常规预期功能和一系列应用 solidDB（一种特定的非标准 SQL 语法）的有用扩展。程序类 SQL 扩展（例如，存储过程、存储函数和触发器）可允许将应用程序逻辑的部分移到数据库中。这些扩展可以减少网络流量，从而提高性能。

A.82 TRUNCATE TABLE

```
TRUNCATE TABLE tablename
```

用法

从调用者的视图而言，此语句在语义方面等同于 DELETE FROM *tablename*。但是，宽松隔离大大提高了效率。

限制

在执行此语句期间，在并发事务中不会维护已定义的隔离级别。除去行的效果将立即反映在所有并发事务中。因此，建议您仅将此语句用于维护用途。

当截断的表作为引用的表参与引用完整性约束时，如果正在引用的表不为空，那么 TRUNCATE 语句将失败。无论是否定义了引用操作（RESTRICT 和 CASCADE 等）都是如此。如果正在引用的表与已引用的表（树结构的表）不相同，那么限制不适用。

如果要截断收集的相关表，那么应该以仅从正在引用的表开始的表顺序发出 TRUNCATE 语句（直到正在引用的链方式），并使用仅具有已引用表的表结束。

A.83 UNLOCK TABLE

```
UNLOCK TABLE { ALL | tablename [,tablename]} 
```

其中：

tablename 是要解锁的表的名称。您还可以通过对表名进行限定来指定该表的目录和模式。

ALL 将释放所有表上的所有表级锁定。

用法

UNLOCK TABLE 语句使用带有 LONG 选项的 LOCK TABLE 命令以手动方式解除对表的锁定。LONG 选项允许您保持锁定，即使挂起该锁定的事务结束亦如此。由于此锁定没有自然的端点（事务结束除外），因此，您必须使用 UNLOCK 命令显式地释放 LONG 锁定。

UNLOCK TABLE 命令不适用于服务器的自动锁定或者未使用 LONG 选项的手动锁定。如果一个锁定是自动锁定，或者它是手动锁定但不是 LONG 锁定，服务器将在挂起该锁定的事务结束时自动释放该锁定。因此，不需要以手动方式释放那些锁定。

使用 UNLOCK TABLE 命令时，它不会立即生效；而是，那些锁定会在当前事务落实时释放。

注：

如果当前事务（从中执行 UNLOCK TABLE 命令的事务）未落实（例如，回滚此事务），不会将表解锁；它们会一直处于锁定状态，直到另一个 UNLOCK TABLE 命令成功执行并落实为止。

LOCK/UNLOCK 命令仅适用于表。没有用于以手动方式对各个记录进行锁定或解锁的命令。

如果有名为 ALL 的表，需要使用定界标识功能来指定表名。

返回值

表 74. UNLOCK TABLE 返回值

错误代码	描述
10083	表 <table_name> 未锁定。
13011	找不到表 <tablename>。

示例

```
LOCK TABLE emp IN SHARED MODE;
LOCK TABLE emp IN SHARED MODE TABLE dept IN EXCLUSIVE MODE;
LOCK TABLE emp,dept IN SHARED MODE NOWAIT;

-- Get an exclusive lock that will persist past the end of the current
-- transaction. If you can't get an exclusive lock immediately, then
-- wait up to 60 seconds to get it.
LOCK TABLE emp, dept IN LONG EXCLUSIVE MODE WAIT 60;
-- Make the schema changes (or do whatever you needed the exclusive
-- lock for).
CALL DO_SCHEMA_CHANGES_1;
COMMIT WORK;
CALL DO_SCHEMA_CHANGES_2;
UNLOCK TABLE ALL; -- at the end of this transaction, release locks.
...
COMMIT WORK;
...
UNLOCK TABLE "ALL"; -- Unlock the table named "ALL".
```


相关参考

第 225 页的 A.57, 『 LOCK TABLE 』

A.84 UNREGISTER EVENT

```
wait_register_statement ::= UNREGISTER EVENT event_name
```

UNREGISTER EVENT 语句注销指定事件。您只能在存储过程中使用 UNREGISTER EVENT 语句。

有关更多详细信息, 请参阅 第 168 页的 A.14.7, 『 wait_register_statement 』。

相关参考

第 168 页的 A.14.7, 『 wait_register_statement 』

相关信息

第 73 页的 3.5, 『 事件 』

A.85 UPDATE

```
UPDATE table_name
  SET [table_name.]column_identifier = {expression | NULL}
  [, [table_name.]column_identifier = {expression | NULL}]...
  [WHERE search_condition]
```

```
UPDATE table_name
  SET [table_name.]column_identifier = {expression | NULL}
  [, [table_name.]column_identifier = {expression | NULL}]...
  WHERE CURRENT OF cursor_name
```

用法

共有两种类型的 UPDATE 语句:

- **已搜索 UPDATE 语句**, UPDATE *table_name* ... [WHERE *search_condition*], 修改由第 282 页的 A.87.5, 『 搜索条件 』 指定的一行或多行的一列或多列的列值。
- **已定位 UPDATE 语句**, UPDATE *table_name* ... WHERE CURRENT OF *cursor_name*, 更新游标的当前行。游标的名称由名为 SQLSetCursorName 的 ODBC API 函数定义。

table_name 可以是表名, 也可以是别名。

已搜索 UPDATE 示例

```
UPDATE TEST SET C = 0.44
WHERE ID = 5
```

已定位 UPDATE 示例

```
UPDATE TEST SET C = 0.33
WHERE CURRENT OF MYCURSOR
```

A.86 WAIT EVENT

```
wait_event_statement ::=
  WAIT EVENT
  [event_specification ...]
  END WAIT
```

```

event_specification ::=
    WHEN event_name [(parameters)]
    BEGIN statements
    END EVENT

```

WAIT EVENT 语句令过程等待事件发生。系统定义的事件和用户定义的事件都可使用 WAIT EVENT。

有关更多详细信息，请参阅 第 169 页的 A.14.8，『wait_event_statement』。

相关参考

第 169 页的 A.14.8，『wait_event_statement』

相关信息

第 73 页的 3.5，『事件』

A.87 常用子句

A.87.1 check_condition

表 75. check_condition

check_condition	
check_condition	::= check_item check_item { AND OR } check_item
check_item	::= [NOT] { check_test (check_condition) }
check_test	::= comparison_test between_test like_test null_test list_test
comparison_test	::= expression { = <> < <= > >= } { expression subquery }
between_test	::= column_identifier [NOT] BETWEEN expression AND expression
like_test	::= column_identifier [NOT] LIKE value [ESCAPE value]
null_test	::= column_identifier IS [NOT] NULL
list_test	::= expression [NOT] IN ({ value [, value]... })

A.87.2 data_type

表 76. data_type

变量名	数据类型
<i>data_type</i>	<pre> ::= {BIGINT BINARY BLOB CHAR [length] CHARACTER LARGE OBJECT CHAR LARGE OBJECT CLOB DATE DECIMAL [(precision [, scale])] DOUBLE [PRECISION] FLOAT [(precision)] INTEGER LONG NATIONAL VARCHAR LONG VARBINARY LONG VARCHAR LONG WVARCHAR NCHAR LARGE OBJECT NUMERIC [(precision [, scale])] NATIONAL CHAR NATIONAL CHARACTER NATIONAL VARCHAR NCHAR NCHAR VARYING NCLOB NVARCHAR REAL SMALLINT TIME TIMESTAMP [(timestamp precision)] TINYINT VARBINARY VARCHAR [(length)] } WCHAR WVARCHAR [length] </pre>

A.87.3 表达式

表 77. 表达式

表达式	
<i>expression</i>	<pre> ::= expression_item expression_item { + - * / } expression_item </pre> <p>注: 运算符两端的空格可选。</p>
<i>expression_item</i>	<pre> ::= [+ -] { value column_identifier function case_expression cast_expression (expression) } </pre>
<i>value</i>	<pre> ::= literal USER variable </pre>
<i>function</i>	<pre> ::= set_function null_function string_function numeric_function datetime_function system_function datatypeconversion_function </pre> <p>注: 字符串、数字、日期时间和数据类型转换函数是标量函数，即，操作由函数名指示，后跟一对括号括住零个或多个指定的自变量。每个标量函数都返回单一的值。</p>

表 77. 表达式 (续)

表达式	
<i>set_function</i>	<pre> ::= COUNT (*) { AVG MAX MIN SUM COUNT } ({ ALL DISTINCT } expression)</pre>
<i>null_function</i>	<pre> ::= { NULLVAL_CHAR() NULLVAL_INT() }</pre>
<i>datatypeconversion_function</i>	<pre> ::= CONVERT_CHAR(value_exp) CONVERT_DATE(value_exp) CONVERT_DECIMAL(value_exp) CONVERT_DOUBLE(value_exp) CONVERT_FLOAT(value_exp) CONVERT_INTEGER(value_exp) CONVERT_LONGVARCHAR(value_exp) CONVERT_NUMERIC(value_exp) CONVERT_REAL(value_exp) CONVERT_SMALLINT(value_exp) CONVERT_TIME(value_exp) CONVERT_TIMESTAMP(value_exp) CONVERT_TINYINT(value_exp) CONVERT_VARCHAR(value_exp)</pre> <p>注：这些函数用于实现 ODBC 所定义的 {fn CONVERT(value, odbc_ttypename)} 转义子句。但是，您应使用在 SQL-92 中定义并完全受 solidDB 支持的 CAST(value AS sql_ttypename)。</p>
<i>case_expression</i>	<pre> ::= case_abbreviation case_specification</pre>
<i>case_abbreviation</i>	<pre> ::= NULLIF(value_exp, value_exp) COALESCE(value_exp {, value_exp }...)</pre> <p>如果 NULLIF 函数的第一个参数与第二个参数相等，那么此函数将返回 NULL；否则，此函数将返回第一个参数。此函数相当于 IF (p1 = p2) THEN RETURN NULL ELSE RETURN p1；。如果您使用一个特殊值作为标志来指示 NULL，那么 NULLIF 函数很有用。您可以使用 NULLIF 将该特殊值转换为 NULL。换言之，此函数的行为就像是 IF (p1 = NullFlag) THEN RETURN NULL ELSE RETURN p1；</p> <p>COALESCE 返回第一个非 NULL 自变量。自变量列表几乎可以具有任意长度。所有自变量应该具有相同的（或兼容的）数据类型。</p>
<i>case_specification</i>	<pre> ::= CASE [value_exp] WHEN value_exp THEN {value_exp } [WHEN value_exp THEN { value_exp } ...] [ELSE { value_exp }] END</pre>
<i>cast_expression</i>	<pre> ::= CAST (value_exp AS -data-type)</pre>

表 77. 表达式 (续)

表达式	
<i>row value constructor expression</i>	<p>行值构造器 (RVC) 是由圆括号定界的有序值序列, 例如:</p> <p>(1, 4, 9)</p> <p>('Smith', 'Lisa')</p> <p>行构造基于一系列元素/值, 与由一系列字段组成的表上的行相似。</p> <p>有关行值构造器的更多信息, 请参阅第 18 页的 2.5.5, 『行值构造器』。</p>

A.87.4 query_specification

表 78. *query_specification*

query_specification	
<i>query_specification</i>	::= SELECT [DISTINCT ALL] <i>select_list</i> <i>table_expression</i>
<i>select_list</i>	::= * <i>select_sublist</i> [{, <i>select_sublist</i> } ...]
<i>select_sublist</i>	::= <i>derived_column</i> [<i>table_name</i> <i>table_identifier</i>].*
<i>derived_column</i>	::= <i>expression</i> [[AS] <i>column_alias</i>]]
<i>table_expression</i>	::= FROM <i>table_reference_list</i> [WHERE <i>search_condition</i>] [GROUP BY <i>column_name_list</i> [[UNION INTERSECT EXCEPT] [ALL] [CORRESPONDING [BY (<i>column_name_list</i>)]] <i>query_specification</i>] [HAVING <i>search_condition</i>]

A.87.5 搜索条件

表 79. 搜索条件

搜索条件	
<i>search_condition</i>	::= <i>search_item</i> <i>search_item</i> { AND OR } <i>search_item</i>
<i>search_item</i>	::= [NOT] { <i>search_test</i> (<i>search_condition</i>) }
<i>search_test</i>	::= <i>comparison_test</i> <i>between_test</i> <i>like_test</i> <i>null_test</i> <i>set_test</i> <i>quantified_test</i> <i>existence_test</i>
<i>comparison_test</i>	::= <i>expression</i> { = <> < <= > >= } { <i>expression</i> <i>subquery</i> } 注: 运算符两端的空格可选。
<i>between_test</i>	::= <i>column_identifier</i> [NOT] BETWEEN <i>expression</i> AND <i>expression</i>

表 79. 搜索条件 (续)

搜索条件	
<i>like_test</i>	::= <i>column_identifier</i> [NOT] LIKE <i>value</i> [ESCAPE <i>value</i>]
<i>null_test</i>	::= <i>column_identifier</i> IS [NOT] NULL
<i>set_test</i>	<p>::= <i>expression</i> [NOT] IN ({ <i>value</i> [, <i>value</i>]... <i>subquery</i> })</p> <p>注: 如果 IN 子句包含只有一个项目, 不需要括号。</p> <p>::= <i>expression</i> [NOT] IN <i>value</i></p> <p>例如:</p> <p>SELECT * FROM MY_TABLE1 WHERE A IN 2; SELECT * FROM MY_TABLE1 WHERE A IN (2,3,6);</p>
<i>quantified_test</i>	::= <i>expression</i> { = <> < <= > >= } [ALL ANY SOME] <i>subquery</i>
<i>existence_test</i>	::= EXISTS <i>subquery</i>

A.87.6 table_reference

表 80. table_reference

Table_reference	
<i>table_reference_list</i>	::= <i>table_reference</i> [, <i>table-reference</i> ...]
<i>table_reference</i>	::= <i>table_name</i> [[AS] <i>correlation_name</i>] <i>derived_table</i> [[AS] <i>correlation_name</i>] [(<i>derived_column_list</i>)] <i>joined_table</i>
<i>table_name</i>	::= <i>table_identifier</i> <i>schema_name.table_identifier</i>
<i>derived_table</i>	::= <i>subquery</i>
<i>derived_column_list</i>	::= <i>column_name_list</i>
<i>joined_table</i>	::= <i>cross_join</i> <i>qualified_join</i> (<i>joined_table</i>)
<i>cross_join</i>	::= <i>table_reference</i> CROSS JOIN <i>table_reference</i>
<i>qualified_join</i>	::= <i>table_reference</i> [NATURAL] [<i>join_type</i>] JOIN <i>table_reference</i> [<i>join_specification</i>]
<i>join_type</i>	::= INNER <i>outer_join_type</i> [OUTER] UNION
<i>outer_join_type</i>	::= LEFT RIGHT FULL
<i>join_specification</i>	::= <i>join_condition</i> <i>named_columns_join</i>
<i>join_condition</i>	::= ON <i>search_condition</i>
<i>named_columns_join</i>	::= USING (<i>column_name_list</i>)
<i>column_name_list</i>	::= <i>column_identifier</i> [{ , <i>column_identifier</i> } ...]

A.87.7 SELECT 语句中的伪列

在 SELECT 语句的选择列表中，可以使用该节中所描述的伪列。

表 81. 伪列

伪列	类型	说明
ROWVER	VARBINARY(10)	表中的行的版本。 注：由于 ROWVER 都引用单行的行，因此只能用于从单一表中返回行的查询。
ROWID	VARBINARY(254)	表中的行的持久标识。 注：由于 ROWID 引用单行的行，因此只能用于从单一表中返回行的查询。
ROWNUM	DECIMAL(16,2)	行号指示从表或一组连接的行中选择行的顺序。选择的第一行的 ROWNUM 为 1，第二行的 ROWNUM 为 2，依此类推。 由于在 ORDER BY 子句之前把 ROWNUM 分配给了行，因此您不应使用 ROWNUM 来标识排序的行。 ROWNUM 主要用于限制查询所返回的行数，例如 WHERE ROWNUM < 10。

A.88 日期和时间字面值

表 82. 日期和时间字面值

日期/时间字面值	
<i>date_literal</i>	'YYYY-MM-DD'
<i>time_literal</i>	'HH:MM:SS'
<i>timestamp_literal</i>	'YYYY-MM-DD HH:MM:SS'

A.89 通配符

下列各项在某些表达式（例如 LIKE '<string>'）中可以用作通配符。

表 83. 通配符

字符	说明
_ (下划线)	下划线字符与任意单一字符匹配。例如，“J_NE”与“JANE”和“JUNE”匹配。
% (百分号)	百分号字符与任意一组 0 个或多个字符匹配。例如，“ED%”与“EDWARD”和“EDITOR”匹配。另一个示例，“%ED%”与“EDWARD”、“TEDDY”和“FRED”匹配。

A.89.1 使用 SQL 通配符

要执行精确匹配搜索，请指定字面值，例如：

```
SELECT * FROM table1 WHERE name = 'SMITH';
```

字符串 'SMITH' 是字面值。

要执行近似匹配搜索，请指定 SQL 通配符，它代表与另一个字符串近似的字符串。逻辑表达式（例如 WHERE 子句和 CHECK 约束中使用的逻辑表达式）可以使用“通配符”和关键字 LIKE 来匹配近似的字符串。

下划线字符（_）是与任何单一字符匹配的通配符。例如，以下查询：

```
SELECT * FROM table1 WHERE first_name LIKE 'J_NE';
```

将返回 JANE 和 JUNE（以及任何其他第一个字母为 J 并且最后两个字母为 NE 的 4 字符名称）。

百分号字符（%）是与 0 个或多个字符的任意匹配项匹配的通配符。例如，以下查询：

```
SELECT * FROM table1 WHERE first_name LIKE 'JOHN%';
```

可以返回 JOHN、JOHNNY 和 JOHNATHAN 等等。

% 通配符常用于字符串末尾，但也可以用于任何位置。例如，以下搜索模式：

```
LIKE '%JO%'
```

将返回名字的任何位置包含 JO 的所有人员，包括（但不限于）：

JOANNE、BILLY JO 和 LONG JOHN SILVER

在单一字符串中允许使用多个通配符。例如，字符串 J_V_ 与 JAVA 和 JIVE 以及任何其他以 J 开头并且将 V 作为第三个字符的 4 字符单词或名称匹配。注意，由于下划线（_）只与一个字符匹配，因此字符串 J_V_ 与长度超过 4 个字符的字符串 JOVIAL 不匹配。

A.89.2 作为字面值的通配符

在字符串的一个部分中使用字面值字符 %（百分比）或下划线（_）时，可以在同一字符串的另一部分中使用通配符。

要使用通配符作为字面值，请在通配符前加转义字符，并指定转义字符本身作为该查询的一部分。语法如下：

```
... LIKE <string> {ESCAPE <escape character>}
```

例如，以下表达式使用反斜杠字符（\）作为转义字符：

```
LIKE 'MY\_EXPRESSION_' ESCAPE '\';
```

前一表达式匹配该表达式：

- MY_EXPRESSION1
- MY_EXPRESSIONA
- MY_EXPRESSION_

前一表达式不匹配该表达式:

- MY#EXPRESSION1

切记: 依据 ANSI 标准, 必须使用单引号分隔字符串。

例如:

```
...LIKE 'J_N_'; -- 正确  
...LIKE "J_N_"; -- 错误
```

双引号用于定界标识, 而不用于数据。这不同于 C 语言和 Java 语言, 因为这两种语言使用双引号分隔如 *"C-language string"* 中的字符串, 使用单引号分隔如 *'C'* 中的字符。

附录 B. 函数

函数是由函数名称指示的一项操作，在函数后面跟着一对括号括住的自变量规范(可能没有自变量)。除了内置函数外，solidDB 支持用户定义的内部和外部存储函数。

内置函数在数据库引擎内提供。

使用 CREATE FUNCTION 语句将用户定义的函数注册到数据库中。用户定义的函数有两类：

- 用户定义的存储函数可用 solidDB 专有 SQL 过程语言编写。
- 用户定义的外部存储函数可用 C 编程语言编写。外部存储函数在运行时使用操作系统提供的标准动态库界面装入。通过编写 C 程序，外部存储过程可令您扩展服务器功能。

相关参考

第 157 页的 A.11, 『CREATE FUNCTION』

第 159 页的 A.12, 『CREATE FUNCTION (外部)』

B.1 字符串函数

表 84. 字符串函数

函数	用途
ASCII(<i>str</i>)	返回等同于字符串 <i>str</i> 的整数
CHAR(<i>code</i>)	返回等同于 <i>code</i> 的字符
CONCAT(<i>str1</i> , <i>str2</i>)	Concatenates <i>str2</i> to <i>str1</i>
<i>str1</i> { + } <i>str2</i>	将 <i>str2</i> 与 <i>str1</i> 并置。 例如： SELECT <i>str1</i> + <i>str2</i> , <i>col1</i> ... SELECT <i>str1</i> <i>str2</i> , <i>col1</i> ...
GET_UNIQUE_STRING(<i>str</i>)	此函数根据“前缀”（输入字符串，这可以是您选择的任何字符串）和序号（在内部创建和使用）生成唯一的字符串。即使输入为 NULL，此函数也仍然根据唯一的序号来返回字符串。
INSERT(<i>str1</i> , <i>start</i> , <i>length</i> , <i>str2</i>)	通过从 <i>str1</i> 中删除 <i>length</i> 个字符并插入 <i>str2</i> 来合并字符串。
LCASE(<i>str</i>)	将字符串 <i>str</i> 转换为小写。
LEFT(<i>str</i> , <i>count</i>)	返回字符串 <i>str</i> 的最左边 <i>count</i> 个字符。
LENGTH(<i>str</i>)	返回 <i>str</i> 中的字符数。

表 84. 字符串函数 (续)

函数	用途
LOCATE(<i>str1</i> , <i>str2</i> [, <i>start</i>])	返回 <i>str1</i> 在 <i>str2</i> 中的开始位置。如果指定了可选自变量 <i>start</i> , 那么将从 <i>start</i> 的值所指定的字符位置开始执行搜索。如果在 <i>string_exp2</i> 中找不到 <i>string_exp1</i> , 那么此函数将返回 0。对于返回值和输入参数 <i>start</i> , 字符串位置的编号从 1 (而不是 0) 开始。
LTRIM(<i>str</i>)	除去 <i>str</i> 的前导空格。
POSITION (<i>str1</i> IN <i>str2</i>)	返回 <i>str1</i> 在 <i>str2</i> 中的开始位置。
REPEAT(<i>str</i> , <i>count</i>)	返回 <i>str</i> 重复 <i>count</i> 次后的字符。
REPLACE(<i>str1</i> , <i>str2</i> , <i>str3</i>)	将 <i>str1</i> 中出现的 <i>str2</i> 替换为 <i>str3</i> 。
RIGHT(<i>str</i> , <i>count</i>)	返回字符串 <i>str</i> 的最右边 <i>count</i> 个字符。
RTRIM(<i>str</i>)	除去 <i>str</i> 中的尾部空格。
SOUNDEX(<i>str</i>)	计算 4 字符的探测法 (语音) 代码。
SPACE(<i>count</i>)	返回包含 <i>count</i> 个空格的字符串。
SUBSTRING(<i>str</i> , <i>start</i> , <i>length</i>)	根据 <i>str</i> 中从 <i>start</i> 开始的内容派生长度为 <i>length</i> 个字节的子串。 例如, 如果 <i>str</i> 为“First Second Third”, 那么 SUBSTRING(<i>str</i> , 7, 6) 将返回“Second”。 字符串位置从 1 (而不是 0) 开始编号。
TO_CHAR(<i>expression</i> , <i>format-string</i>)	返回已使用字符模板格式化的输入表达式字符说明。 TO_CHAR 标量函数是 VARCHAR_FORMAT 标量函数的同义词。
TRIM(<i>str</i>)	除去 <i>str</i> 中的前导空格和尾部空格。
UCASE(<i>str</i>)	将 <i>str</i> 转换为大写。
VARCHAR_FORMAT(<i>expression</i> , <i>format-string</i>)	返回已使用字符模板格式化的输入表达式字符说明。 VARCHAR_FORMAT 函数是 TO_CHAR 函数的同义词。

相关信息

第 284 页的 A.89, 『通配符』

B.2 数字函数

表 85. 数字函数

函数	用途
ABS(<i>numeric</i>)	<i>numeric</i> 的绝对值
ACOS(<i>float</i>)	<i>float</i> 的反余弦, 其中 <i>float</i> 以弧度表示
ASIN(<i>float</i>)	<i>float</i> 的正弦, 其中 <i>float</i> 以弧度表示
ATAN(<i>float</i>)	<i>float</i> 的正切, 其中 <i>float</i> 以弧度表示

表 85. 数字函数 (续)

函数	用途
ATAN2(float1, float2)	x 和 y 坐标的反正切, 这两个参数分别由 $float1$ 和 $float2$ 作为角度指定, 以弧度表示
CEILING(numeric)	大于或等于 $numeric$ 的最小整数
COS(float)	$float$ 的余弦, 其中 $float$ 以弧度表示
COT(float)	$float$ 的余切, 其中 $float$ 以弧度表示
DEGREES(numeric)	将 $numeric$ 弧度转换为度数
DIFFERENCE(str1, str2)	返回语音差别值: 0 - 4
EXP(float)	$float$ 的幂值
FLOOR(numeric)	小于或等于 $numeric$ 的最大整数
LOG(float)	$float$ 的自然对数
LOG10(float)	$float$ 的以 10 为底的对数
MOD(integer1, integer2)	$integer1$ 除以 $integer2$ 的模数
PI()	以浮点数表示的 π
POWER(numeric, integer)	$numeric$ 的值的 $integer$ 次幂
RADIANS(numeric)	从 $numeric$ 度转换为弧度
ROUND(numeric, integer)	$Numeric$ 四舍五入到 $integer$ 位
SIGN(numeric)	$numeric$ 的符号
SIN(float)	$float$ 的正弦, 其中 $float$ 以弧度表示
SQRT(float)	$float$ 的平方根
TAN(float)	$float$ 的正切, 其中 $float$ 以弧度表示
TRUNCATE(numeric, integer)	$Numeric$ 截断到 $integer$ 位

B.3 日期时间函数

表 86. 日期时间函数

函数	用途
ADD_MONTHS(<i>expression</i> , <i>numeric-expression</i>)	返回代表表达式加指定月份数目的日期时间值。 <i>expression</i> 指定开始日期。该表达式必须返回以下其中一种内置数据类型的值: DATE 或 TIMESTAMP。 <i>numeric-expression</i> 返回任何内置数字数据类型的值。如果该值不是 INTEGER 类型, 则在计算函数前, 会将它隐式转换成 INTEGER 类型。数字表达式指定月份数目, 并将其添加到由表达式指定的开始日期中。允许存在负数字值。
CURDATE() CURRENT_DATE CURRENT DATE	返回当前日期。
CURTIME() CURRENT_TIME CURRENT TIME	返回当前时间。
CURRENT_TIMESTAMP CURRENT TIMESTAMP	以时间戳记形式返回当前日期和时间。
DATE(<i>expression</i>)	返回来自值的日期。 自变量必须为 DATE、TIMESTAMP、一个小于或等于 3652059 的正数, 或有效的日期或时间戳记的字符串说明。 <ul style="list-style-type: none">• 如果自变量是 DATE、TIMESTAMP 或有效的日期或时间戳记的字符串说明, 结果则为值中的日期部分。• 如果自变量是一个长度为 7 的字符串, 它必须代表一个格式为 YYYYNNN 的有效日期, 其中 YYYY 是表示年份的数字, NNN 是表示该年某天的数字 (001 到 366 之间)。• 如果自变量是一个数字, 结果则为 1 月 1 日 (即 0001) 之后 n-1 天的日期, 其中 n 是该数的整数部分。
DAYNAME(<i>date</i>)	返回包含星期几的字符串。
DAYOFMONTH(<i>date</i>)	以整数形式返回一个月中的第几天 (介于 1 与 31 之间)。
DAYOFWEEK(<i>date</i>)	以整数形式返回一个星期中的第几天 (介于 1 与 7 之间, 其中 1 代表星期日)。
DAYOFYEAR(<i>date</i>)	以整数形式返回一年中的第几天 (介于 1 与 366 之间)。
EXTRACT (<i>date field</i> FROM <i>date_exp</i>)	抽取单一时间间隔或时间间隔字段并将其转换为数字。
HOUR(<i>time_exp</i>)	以整数形式返回小时 (介于 0 与 23 之间)。
MINUTE(<i>time_exp</i>)	以整数形式返回分钟 (介于 0 与 59 之间)。
MONTH(<i>date</i>)	以整数形式返回月份 (介于 1 与 12 之间)。

表 86. 日期时间函数 (续)

函数	用途
MONTHNAME(<i>date</i>)	以字符串形式返回月份名称。
NOW()	以时间戳记形式返回当前日期和时间。
QUARTER(<i>date</i>)	以整数形式返回季度（介于 1 与 4 之间）。
SECOND(<i>time_exp</i>)	以整数形式返回秒（介于 0 与 59 之间）。
SYSDATE	以时间戳记形式返回当前日期和时间。
TIMESTAMPADD(<i>interval</i> , <i>integer_exp</i> , <i>timestamp_exp</i>)	<p>通过对 <i>timestamp_exp</i> 加上 <i>integer_exp</i> 个 <i>interval</i> 类型的时间间隔来计算时间戳记。</p> <p>用于表示 TIMESTAMPADD 的有效 <i>interval</i> 值的关键字是:</p> <p>SQL_TSI_FRAC_SECOND</p> <p>SQL_TSI_SECOND</p> <p>SQL_TSI_MINUTE</p> <p>SQL_TSI_HOUR</p> <p>SQL_TSI_DAY</p> <p>SQL_TSI_WEEK</p> <p>SQL_TSI_MONTH</p> <p>SQL_TSI_QUARTER</p> <p>SQL_TSI_YEAR</p>
TIMESTAMPDIFF(<i>interval</i> , <i>timestamp-exp1</i> , <i>timestamp-exp2</i>)	<p>返回时间间隔中 <i>timestamp-exp2</i> 大于 <i>timestamp-exp1</i> 的整数。</p> <p>用于表示 TIMESTAMPDIFF 的有效 <i>interval</i> 值的关键字是:</p> <p>SQL_TSI_FRAC_SECOND</p> <p>SQL_TSI_SECOND</p> <p>SQL_TSI_MINUTE</p> <p>SQL_TSI_HOUR</p> <p>SQL_TSI_DAY</p> <p>SQL_TSI_WEEK</p> <p>SQL_TSI_MONTH</p> <p>SQL_TSI_QUARTER</p> <p>SQL_TSI_YEAR</p>

表 86. 日期时间函数 (续)

函数	用途
<p>TIMESTAMP_FORMAT(<i>input-string</i>, <i>format-string</i>)</p> <p>TO_DATE(<i>input-string</i>, <i>format-string</i>)</p> <p>TO_TIMESTAMP(<i>input-string</i>, <i>format-string</i>)</p>	<p>返回基于使用指定格式的输入字符串来解释的时间戳记。如果时间戳记表达式的时间戳记精确度小于该格式指定的精确度, 则会在该指定数字的右侧插入零。</p> <p>支持以下格式:</p> <ul style="list-style-type: none"> • YY 该年份的最后两位数 (00-99) • YYYY 4 位数的年份 (0000-9999) • RR 与 YY 遵循同一规律 • RRRR 与 YYYY 遵循同一规律 • MM 月份 (01-12) • DD 当月的某一天 (01-31) • HH24 24 小时格式的当日的某个小时 (00-24) • MI 分钟 (00-59) • SS 秒 (00-59) • FF 或 FF<i>n</i> 秒的小数部分 (微秒、毫秒、纳秒、皮秒) (0-999999999999)。 <p>数字 <i>n</i> 用于指定包含在返回值中数字的数目。 <i>n</i> 的有效值为 1-12 (不能以 0 开头)。</p> <p>指定 FF 相当于指定 FF6。</p> <p>TO_DATE、TIMESTAMP_FORMAT 和 TO_TIMESTAMP 函数是同义词。</p>
WEEK(<i>date</i>)	以整数形式返回一年中的第几个星期 (介于 1 与 52 之间)。
YEAR(<i>date</i>)	以整数形式返回年份。

B.4 系统函数

系统函数返回有关 solidDB 数据库的信息。

表 87. 系统函数

函数	用途
UIC()	返回与连接相关联的连接标识。
CURRENT_USERID ()	返回当前用户标识。
LOGIN_USERID ()	返回登录用户标识。
CURRENT_CATALOG ()	返回当前目录。
LOGIN_CATALOG ()	返回登录目录。
CURRENT_SCHEMA ()	返回当前模式。
LOGIN_SCHEMA ()	返回登录模式。

B.5 其他函数

表 88. 其他函数

函数	用途
BITAND(<i>integer1</i> , <i>integer2</i>) BIT_AND(<i>integer1</i> , <i>integer2</i>)	返回按位 AND 运算的结果。
BITANDNOT(<i>expression1</i> , <i>expression2</i>) BIT_ANDNOT(<i>expression1</i> , <i>expression2</i>)	清除位于第二个自变量的第一个自变量中的任何位。
BITNOT(<i>expression</i>)	执行按位 NOT 操作。
BITOR(<i>expression1</i> , <i>expression2</i>)	执行按位 OR 操作。
BITXOR(<i>expression1</i> , <i>expression2</i>)	执行按位互斥 OR 操作。
DATABASE_ENCRYPTION_LEVEL()	返回数据库的加密级别。 <ul style="list-style-type: none">• 0 - 未加密• 1 - 已加密, 密钥未受保护 (空密码)• 2 - 已加密, 密钥受单独启动密码保护• 3 - 已加密, 使用定制加密方法
IFNULL(<i>exp</i> , <i>value</i>)	如果 <i>exp</i> 为 NULL, 那么返回 <i>value</i> ; 否则, 返回 <i>exp</i> 。 (如果返回 <i>value</i> , 那么它会转换为 <i>exp</i> 的类型)
SLEEP(<i>milliseconds</i>)	只能从存储过程或触发器中调用 SLEEP 函数。此函数将导致该存储过程或触发器“休眠”(临时暂挂活动)指定的毫秒数。分辨率精确到约 1 秒 (即 1000 毫秒)。确切的休眠时间长度还取决于计算机处理其他进程和线程的繁忙程度。此值必须是字面值, 而不能是变量或表达式。

B.6 高级复制函数

本部分描述了特定于高级复制配置的内置函数

B.6.1 GET_PARAM() 函数

```
get_param('param_name')
```

用法

GET_PARAM() 函数检索放置在事务公告牌上、使用 PUT_PARAM() 函数或使用 SAVE PROPERTY 语句、SAVE DEFAULT PROPERTY 或 SET SYNC PARAMETER 参数。所检索的参数特定于目录; 并且每个目录都有一组不同的参数。

如果该参数不在公告牌上, 函数将返回该参数的 VARCHAR 值或 NULL。

由于 get_param() 是一个 SQL 函数, 因此它只能在过程中使用或者用作 SELECT 语句的组成部分。

参数名必须括在单引号中。

GET_PARAM() 函数只适用于高级复制配置; 可在主数据库和副本数据库上发出。

solidDB 系统参数

solidDB 系统参数分为下列类别:

- 只读系统参数, 这些参数由 solidDB 维护, 并且只能通过 GET_PARAM(*parameter_name*) 语法读取。

此类别中的参数的生命周期是一个事务, 即, 这些参数的值始终在该事务开始时进行初始化。

- 可更新系统参数, 这些参数可以由用户通过 PUT_PARAM(*parameter_name*, *value*) 进行设置和更新。可更新系统参数由 solidDB 使用。

与上一个类别相同, 这些参数的生命周期也是一个事务。

- 数据库目录级系统参数, 这些参数使用 SET SYNC PARAMETER *parameter_name value* 语法进行设置。

此类别中的参数是在更改或除去前保持有效的数据库目录级参数。它们是作为公告牌参数指定的。

返回值

表 89. GET_PARAM 返回值

错误代码	描述
13086	参数中的数据类型无效。

示例

```
SELECT put_param('myparam', '123abc');  
SELECT get_param('myparam');
```

相关参考

B.6.2, 『PUT_PARAM() 函数』

相关信息

第 253 页的 A.77, 『SAVE PROPERTY』

第 268 页的 『SET SYNC PARAMETER』

B.6.2 PUT_PARAM() 函数

PUT_PARAM(*param_name*, *param_value*)

您可以使用 PUT_PARAM() 函数将参数放入公告牌。如果该参数已存在, 那么新值将覆盖先前的值。

借助 solidDB 智能事务, 事务的 SQL 语句或过程可以使用参数公告牌来相互传递参数, 从而相互进行通信。公告牌是对事务的所有语句可视的参数存储器。

可以在主数据库和副本数据库中使用 PUT_PARAM() 函数, 以便将参数放入当前事务的参数公告牌。

参数特定于目录。不同的副本数据库目录和主数据库目录各有自己的一组公告牌参数, 并且相互不可视。

这些参数不会传播到主数据库。您可以使用 `SAVE PROPERTY` 语句将属性从副本数据库传播到主数据库。有关详细信息，请参阅第 253 页的 A.77，『`SAVE PROPERTY`』。

由于 `PUT_PARAM()` 是 SQL 函数，因此只能在过程或 SQL 语句中使用。

参数名和参数值的类型都是 `VARCHAR`。

“`PUT_PARAM()`”与“`SAVE PROPERTY property_name VALUE property_value;`”之间的区别

您通常在运行中的事务中使用 `PUT_PARAM()` 在过程之间传递参数。当该事务终止（落实或回滚）时，这些参数值将从公告牌中消失。

您通常在副本数据库中使用 `SAVE PROPERTY` 语句来设置整个事务的属性。这些属性可以在 `PROPAGATE TRANSACTIONS` 语句的 `WHERE` 子句中使用。在主数据库中执行事务时，在该事务开始时会把该事务的属性放入该事务的参数公告牌。因此，该事务的所有过程都可以使用 `GET_PARAM(param_name)` 函数来访问那些参数。

返回值

表 90. `PUT_PARAM()` 返回值

错误代码	描述
13086	参数中的数据类型无效。

执行成功后，`PUT_PARAM()` 将返回所指定的参数的新值。

示例

```
SELECT PUT_PARAM('myparam', '123abc');
```

相关参考

第 293 页的 B.6.1，『`GET_PARAM()` 函数』

相关信息

第 253 页的 A.77，『`SAVE PROPERTY`』

第 268 页的『`SET SYNC PARAMETER`』

B.7 触发器函数

系统支持的下列触发器堆栈函数对于分析和调试而言非常有用。

注：触发器堆栈是指那些已高速缓存的触发器，而无论它们是已执行还是被检测到即将执行。在应用程序中，可以像使用任何其他函数一样使用触发器堆栈函数。

这些函数包括：

- `TRIG_COUNT()`

此函数返回触发器堆栈中的触发器数目，其中包括当前触发器。返回值是整数。

- `TRIG_NAME(n)`

此函数返回触发器堆栈中的第 N 个触发器名。第一个触发器的位置或偏移为零。

- TRIG_SCHEMA(n)

此函数返回触发器堆栈中的第 N 个触发器模式名。第一个触发器的位置或偏移为零。返回值是字符串。

附录 C. 数据类型

本附录中的各个表按类别列示受支持的数据类型。

表 91. 数据类型表中的缩写

缩写	描述
DEFLEN	已定义的列长度；例如，对于 CHAR(24) 而言，精度和长度是 24
DEFPREC	已定义的精度；例如，对于 NUMERIC(10,3) 而言，精度是 10
DEFSCALE	已定义的标度；例如，对于 NUMERIC(10,3) 而言，精度是 3
MAXLEN	列的最大长度
不适用	不适用

数据类型表中的列值解析

大小（范围）

数字列的大小（范围）是指该列能够存储的最小值和最大值。

字符列的大小是指该数据类型的列中能够存储的最大数据长度。

精度

数字列的精度是指该列的数据类型所使用的最大位数。

非数字列的精度是指已定义的列长度。

标度

数字列的标度是指小数点右边的最大位数。对于适当的浮点数字列而言标度未定义，因为小数点右边的位数不固定。

长度

列的长度是将数据传送到它的缺省 C 类型时返回给应用程序的最大字节数。

对于字符数据而言，此长度不包括空终止字节。列的长度可能与数据源中存储该数据所需的字节数不同。

显示大小

列的显示大小是以字符格式显示数据所需的最大字节数。

C.1 字符数据类型

表 92. 字符数据类型

数据类型	大小	精度	标度	长度	显示大小
CHARACTER	2 G - 1*	DEFLEN	不适用	DEFLEN	DEFLEN
CHAR	(2147483647)				

表 92. 字符数据类型 (续)

数据类型	大小	精度	标度	长度	显示大小
WCHAR NATIONAL CHARACTER NATIONAL CHAR NCHAR	2 G - 1* (2147483647)	DEFLN	不适用	DEFLN	DEFLN
VARCHAR CHARACTER VARYING CHAR VARYING	2 G - 1** (2147483647)	DEFLN	不适用	DEFLN	DEFLN
WVARCHAR NATIONAL VARCHAR NCHAR VARYING NVARCHAR	2 G - 1** (2147483647)	DEFLN	不适用	DEFLN	DEFLN
LONG VARCHAR CHARACTER LARGE OBJECT CHAR LARGE OBJECT CLOB	2 G - 1 (2147483647)	MAXLEN	不适用	MAXLEN	MAXLEN
LONG VARCHAR LONG NATIONAL VARCHAR NCHAR LARGE OBJECT NCLOB	2 G - 1 (2147483647)	MAXLEN	不适用	MAXLEN	MAXLEN
* 缺省值是 1 ** 缺省值是 254					

C.2 数字数据类型

表 93. 数字数据类型

数据类型	大小	精度	标度	长度	显示大小
TINYINT	[-128, 255]	3	0	1 (字节)	4 (带符号) 3 (无符号)
SMALLINT	[-32768, 65535]	5	0	2 (字节)	6 (带符号) 5 (无符号)
INTEGER INT	$[-2^{31}, 2^{31} - 1]$	10	0	4 (字节)	11 (带符号) 10 (无符号)
BIGINT	$[-2^{63}, 2^{63} - 1]$	19	0	8 (字节)	20 (带符号)
REAL	正数: 1.175494351e-38 到 1.7014117e+38 负数: -1.7014117e+38 到 -1.175494351e-38 对于此数据类型, 还可以使用 零 (0) 值。	7	不适用	4 (字节)	13
FLOAT	正数: 2.2250738585072014e-308 到 8.98846567431157854e+307 负数: -8.98846567431157854e+307 到 -2.2250738585072014e-308 对于此数据类型, 还可以使用 零 (0) 值。	15	不适用	8 (字节)	22

表 93. 数字数据类型 (续)

数据类型	大小	精度	标度	长度	显示大小
DOUBLE PRECISION	正数: 2.2250738585072014e-308 到 8.98846567431157854e+307 负数: -8.98846567431157854e+307 到 -2.2250738585072014e-308 对于此数据类型, 还可以使用 零 (0) 值。	15	不适用	8 (字节)	22
DECIMAL*	±1.0e254	DEFPREC 最大值 52 缺省值 52	DEFSCALE 缺省值 0	2-27 (字节)	可变
NUMERIC	±1.0e254	DEFPREC 最大值 52 缺省值 52	DEFSCALE 缺省值 0	2-27 (字节)	可变

* 对于 DECIMAL, 如果既未指定精度也未指定标度, 那么值将表示成精度为 52 并且范围为 ±1.0e254 的精确十进制浮点数。

注:

虽然客户机程序可能把整数数据类型 (TINYINT、SMALLINT、INT 和 BIGINT) 解释成带符号或无符号, 但 solidDB 将它们作为带符号整数进行存储和排序。无法告知服务器将整数数据类型作为无符号整数进行排序。

注意:

BIGINT 包含大约 19 个有效位。这意味着, 将 **BIGINT** 存储到诸如 **FLOAT** (约 15 个有效位)、**SMALLFLOAT** (约 7 个有效位) 或 **DECIMAL** (约 16 个有效位) 的非整数数据类型时, 可能会丢失最不重要的有效位。

C.3 二进制数据类型

表 94. 二进制数据类型

数据类型	大小	精度	标度	长度	显示大小
BINARY	2 G*	DEFLEN	不适用	DEFLEN	DEFLEN x 2
VARBINARY	2 G**	DEFLEN	不适用	DEFLEN	DEFLEN x 2

表 94. 二进制数据类型 (续)

数据类型	大小	精度	标度	长度	显示大小
LONG VARBINARY BLOB	2 G	MAXLEN	不适用	MAXLEN	MAXLEN x 2
* 缺省值是 1 ** 缺省值是 254					

提示:

要将值插入到 BINARY、VARBINARY 和 LONG VARBINARY 字段中，可以将该值表达为十六进制值并使用 CAST 运算符，例如：

```
INSERT INTO table1 VALUES (CAST('FF00AA55' AS VARBINARY));
```

同样，可以在 WHERE 子句中使用 CAST() 表达式：

```
CREATE TABLE t1 (x VARBINARY);
INSERT INTO t1 (x) VALUES (CAST('000000A512' AS VARBINARY));
INSERT INTO t1 (x) VALUES (CAST('000000FF12' AS VARBINARY));
```

```
-- 要使用 LIKE 来比较 VARBINARY 值，请将
-- VARBINARY 强制转型为 VARCHAR。
SELECT * FROM t1 WHERE CAST(x AS VARCHAR) LIKE '000000A5%';
SELECT * FROM t1 WHERE CAST(x AS VARCHAR) LIKE '000000A5__';
```

```
-- 注：如果要使用 "=" 代替 "LIKE"，那么
-- 可以对任何一个操作数进行强制转型。
SELECT * FROM t1 WHERE CAST(x AS VARCHAR) = '000000A512';
SELECT * FROM t1 WHERE x = CAST('000000A512' AS VARBINARY);
```

警告：此类查询无法将索引搜索用于 LIKE 谓词，这在许多情况下将导致性能不佳。

C.4 日期数据类型

表 95. 日期数据类型

数据类型	大小	精度	标度	长度	显示大小
DATE	不适用	10*	不适用	6**	10*
* yyyy-mm-dd 格式中的字符数 ** DATE_STRUCT 结构的大小					

C.5 时间数据类型

表 96. 时间数据类型

数据类型	大小	精度	标度	长度	显示大小
TIME	不适用	8*	不适用	6**	8*

表 96. 时间数据类型 (续)

数据类型	大小	精度	标度	长度	显示大小
* hh:mm:ss 格式中的字符数					
** TIME_STRUCT 结构的大小					

C.6 时间戳记数据类型

表 97. 时间戳记数据类型

数据类型	大小	精度	标度	长度	显示大小
TIMESTAMP	不适用	19*	9	16**	19/29***
* “yyyy-mm-dd hh:mm:ss.ffffff”格式中的字符数					
** TIMESTAMP_STRUCT 结构的大小					
*** 大小是 29 (含小数部分)					

C.7 最小的可能非零数值

表 98. 最小的可能非零数值

数据类型	值
DOUBLE PRECISION	2.2250738585072014e-308
REAL	1.175494351e-38

C.8 BLOB 和 CLOB

solidDB 能够存储长度可达 2147483647 (2G - 1) 字节的二进制数据和字符数据。当此类数据超出特定长度时，可把该数据称为 BLOB (二进制大对象) 或 CLOB (字符大对象)，这取决于存储信息的数据类型。CLOB 只包含“纯文本”，并可以存储在下列任何数据类型中：

- CHAR 和 WCHAR
- VARCHAR 和 WVARCHAR
- LONG VARCHAR (映射到标准类型 CLOB)
- LONG WVARCHAR (映射到标准类型 NCLOB)

BLOB 能够存储任何可以表示为一系列字节的数据类型，例如数字化的图片、视频、音频以及格式化文本文档。它们还可以存储纯文本，但在 CLOB 中存储纯文本将更为灵活。

BLOB 可以存储在下列任何数据类型中：

- BINARY

- VARBINARY
- LONG VARBINARY (映射到标准类型 BLOB)

由于字符数据是一系列字节，因此字符数据可以存储在 BINARY 字段以及 CHAR 字段中。您可以将 CLOB 视为 BLOB 的子集。

提示: 使用术语 *BLOB* 来同时指代 CLOB 和 BLOB。

对于大多数非 BLOB 数据类型（例如整型、浮点型和日期型）而言，存在一组丰富的可以对该数据类型执行的有效操作。例如，对于 FLOAT 值，可以进行加、减、乘、除以及其他运算。由于 BLOB 是一系列字节，并且数据库服务器不知道该字节序列的“含义”（它不知道那些字节是表示电影、歌曲还是航天飞机的设计），因此，可以对 BLOB 执行的操作非常有限。

solidDB 允许您对 CLOB 执行某些字符串操作。例如，通过使用 LOCATE() 函数，可以在 CLOB 中搜索特定的子串（例如，人员的姓名）。由于此类操作需要大量服务器资源（内存和/或 CPU 时间），因此 solidDB 允许您限制所处理的 CLOB 字节数。例如，您可以指定执行字符串搜索时，只搜索每个 CLOB 的前 1 兆字节。有关更多信息，请参阅《IBM solidDB 管理员指南》中有关 **MaxBlobExpressionSize** 配置参数的描述。

虽然理论上可以将整个 BLOB 存储在典型表中，但如果该 BLOB 比较大，那么不将大部分或全部 BLOB 存储在表中有助于提高服务器性能。在 solidDB 中，如果 BLOB 的长度不超过 N 字节，那么该 BLOB 将存储在表中。如果该 BLOB 的长度超过 N 字节，那么前 N 字节将存储在表中，而其余的 BLOB 将作为物理数据库文件中的磁盘块存储在表外部。“N”的确切值部分取决于表的结构以及您创建数据库时指定的磁盘页面大小等等，但它始终至少是 256。256 字节或更短的数据始终存储在表中。

如果数据行大小大于数据库文件的磁盘块大小的 1/3，那么必须将其部分存储为 BLOB。

SYS_BLOBS 系统表用作物理数据库文件中所有 BLOB 数据的目录。一个 SYS_BLOB 条目可以容纳 50 个 BLOB 部件。如果 BLOB 大小超出 50 个部件，那么每个 BLOB 将需要多个 SYS_BLOB 条目。

以下查询返回数据库中 BLOB 总大小的估算值。

```
select sum(totalsize) from sys_blobs
```

估算值并不准确，这是因为仅在检查点维护信息。在两个空的检查点之后，此查询应该能够返回准确的响应。

附录 D. 保留字

本节包含多种 SQL 标准中的保留字，这些 SQL 标准包括：ODBC 3.0、X/Open and SQL Access Group SQL CAE 规范以及 Database Language - SQL: ANSI X3H2 (SQL-92)。某些字由 solidDB SQL 使用。本节还包含潜在的保留字；这些标记括在圆括号中。避免将这些关键字中的任何一个用于其他用途。

本节中的某些保留字也可以通过引在双引号 (") 中来用作标识（例如表名和列名）。可将双引号中的标识称为定界标识，它们遵循 ANSI 的 SQL 标准。在以下 SQL 语句示例中，将保留字 "NULL" 用作表名标识：

```
CREATE TABLE "NULL" (column_1 INTEGER);
```

注： solidDB SQL 允许将某些保留字用作标识，即使未将其引在双引号中亦如此。但是，您在任何要用作标识的保留字两旁使用双引号，这将提高可移植性。

表 99. 保留字列表

保留字	ODBC	X/Open SQL	ANSI SQL-92	solidDB SQL
ABSOLUTE	•		•	
ACTION	•		•	
ADA	•			
ADD	•	•	•	•
ADMIN				•
AFTER			(•)	•
ALIAS			(•)	
ALL	•	•	•	•
ALLOCATE	•	•	•	
ALTER	•	•	•	•
AND	•	•	•	•
ANY	•	•	•	•
APPEND				•
ARE	•		•	
AS	•	•	•	•
ASC	•	•	•	•
ASSERTION	•		•	

表 99. 保留字列表 (续)

保留字	ODBC	X/Open SQL	ANSI SQL-92	solidDB SQL
ASYNC			(*)	•
AT	•		•	
AUTHORIZATION	•		•	•
AVG	•	•	•	
BEFORE			(*)	•
BEGIN	•	•	•	•
BETWEEN	•	•	•	•
BINARY				•
BIT	•		•	
BIT_LENGTH	•		•	
BOOKMARK				•
BOOLEAN			(*)	
BOTH	•		•	
BREADTH			(*)	
BY	•	•	•	•
CALL			(*)	•
CASCADE	•	•	•	•
CASCADED*	•		•	•
CASE	•		•	•
CAST	•		•	•
CATALOG	•		•	•
CHAR	•	•	•	•
CHAR_LENGTH	•		•	
CHARACTER	•	•	•	•
CHARACTER_LENGTH	•		•	
CHECK	•	•	•	•
CLOSE	•	•	•	•

表 99. 保留字列表 (续)

保留字	ODBC	X/Open SQL	ANSI SQL-92	solidDB SQL
COALESCE	•			
COLLATE	•		•	
COLLATION	•		•	
COLUMN	•		•	•
COMMIT	•	•	•	•
COMMITBLOCK				•
COMMITTED				•
COMPLETION			(*)	
CONNECT	•	•	•	•
CONNECTION	•	•	•	
CONSTRAINT	•		•	•
CONSTRAINTS	•		•	
CONTINUE	•	•	•	
CONVERT	•		•	
CORRESPONDING	•		•	•
COUNT	•	•	•	
CREATE	•	•	•	•
CROSS	•		•	•
CURRENT	•	•		•
CURRENT_DATE	•		•	
CURRENT_TIME	•		•	
CURRENT_TIMESTAMP	•		•	
CURRENT_USER	•		•	
CURSOR	•	•	•	•
CYCLE			(*)	
DATA			(*)	•
DATE	•			

表 99. 保留字列表 (续)

保留字	ODBC	X/Open SQL	ANSI SQL-92	solidDB SQL
DAY	•			
DEALLOCATE				
DEC	•	•	•	•
DECIMAL	•	•	•	•
DECLARE	•	•	•	•
DEFAULT	•	•	•	•
DEFERRABLE	•		•	
DEFERRED	•		•	
DELETE	•	•	•	•
DENSE				•
DEPTH			(*)	
DESC	•	•	•	•
DESCRIBE	•	•	•	
DESCRIPTOR	•	•	•	
DIAGNOSTICS	•	•	•	
DICTIONARY			(*)	
DISCONNECT	•	•	•	
DISTINCT	•	•	•	•
DOMAIN	•		•	•
DOUBLE	•	•	•	•
DROP	•	•	•	•
EACH			(*)	
ELSE	•		•	•
ELSEIF			(*)	•
ENABLE				•
END	•	•	•	•
END-EXEC	•		•	

表 99. 保留字列表 (续)

保留字	ODBC	X/Open SQL	ANSI SQL-92	solidDB SQL
EQUALS			(*)	
ESCAPE	•		•	•
EVENT				•
EXCEPT	•		•	•
EXCEPTION	•			
EXEC	•	•	•	•
EXECUTE	•	•	•	•
EXISTS	•	•	•	•
EXPLAIN				•
EXPORT				•
EXTERNAL	•		•	•
EXTRACT	•		•	•
FALSE	•		•	
FETCH	•	•	•	•
FIRST	•		•	
FIXED				•
FLOAT	•	•	•	•
FOR	•	•	•	•
FOREIGN	•	•	•	•
FOREVER				•
FORTRAN	•			
FORWARD				•
FOUND	•	•	•	
FROM	•	•	•	•
FROMFIXED				•
FULL	•		•	•
GENERAL			(*)	

表 99. 保留字列表 (续)

保留字	ODBC	X/Open SQL	ANSI SQL-92	solidDB SQL
GET	•	•	•	•
GLOBAL	•		•	
GO	•		•	
GOTO	•	•	•	
GRANT	•	•	•	•
GROUP	•	•	•	•
HAVING	•	•	•	•
HINT				•
HOUR	•		•	
IDENTIFIED				•
IDENTITY	•		•	
IF			(•)	•
IGNORE	•		(•)	
IMMEDIATE	•	•	•	
IMPORT				•
IN	•	•	•	•
INCLUDE	•	•		
INDEX	•	•		•
INDICATOR	•		•	
INITIALLY	•		•	
INNER	•		•	•
INPUT	•		•	
INSENSITIVE	•		•	
INSERT	•	•	•	•
INT	•	•	•	•
INTEGER	•	•	•	•
INTERNAL				•

表 99. 保留字列表 (续)

保留字	ODBC	X/Open SQL	ANSI SQL-92	solidDB SQL
INTERSECT	•		•	•
INTERVAL	•		•	
INTO	•	•	•	•
IS	•	•	•	•
ISOLATION	•		•	•
JAVA				•
JOIN	•		•	•
KEY	•	•	•	•
LANGUAGE	•		•	
LAST	•		•	
LEADING	•		•	
LEAVE			(•)	•
LEFT	•		•	•
LESS			(•)	
LEVEL	•		•	•
LIKE	•	•	•	•
LIMIT			(•)	
LOCAL	•		•	•
LOCK				•
LONG				•
LOOP			(•)	•
LOWER	•		•	
MAINMEMORY				•
MASTER				•
MATCH	•		•	
MAX	•	•	•	
MERGE				•

表 99. 保留字列表 (续)

保留字	ODBC	X/Open SQL	ANSI SQL-92	solidDB SQL
MESSAGE				.
MIN	.	.	.	
MINUTE	.		.	
MODIFY			(*)	.
MODULE	.		.	
MONTH	.		.	
NAMES	.		.	
NATIONAL	.		.	
NATURAL	.		.	.
NCHAR	.		.	
NEW			(*)	.
NEXT	.		.	.
NO	.		.	.
NONE	.		(*)	
NOT
NULL
NULLIF	.		.	.
NUMERIC
OBJECT			(*)	
OCTET_LENGTH	.		.	
OF
OFF				
OID			(*)	
OLD			(*)	.
ON
ONLY	.		.	.
OPEN	.	.	.	

表 99. 保留字列表 (续)

保留字	ODBC	X/Open SQL	ANSI SQL-92	solidDB SQL
OPERATION			(*)	
OPERATORS			(*)	
OPTIMISTIC				•
OPTION	•			
OR				•
ORDER	•			
OTHERS				
OUTER				•
OUTPUT	•		•	
OVERLAPS	•		•	
PARAMETERS			(*)	
PARTIAL	•		•	
PASCAL	•			
PENDANT			(*)	
PESSIMISTIC				•
PLAN				•
PLI	•			
POSITION	•		•	
POST				•
PRECISION	•	•	•	•
PREORDER			(*)	
PREPARE	•			
PRESERVE	•			
PRIMARY	•	•	•	•
PRIOR	•		•	
PRIVATE			(*)	
PRIVILEGES	•		•	•

表 99. 保留字列表 (续)

保留字	ODBC	X/Open SQL	ANSI SQL-92	solidDB SQL
PROCEDURE	•		•	•
PROPAGATE				•
PROTECTED			(*)	
PUBLIC	•	•	•	•
PUBLICATION				•
READ			•	•
REAL		•	•	•
RECURSIVE			(*)	
REF			(*)	
REFERENCES	•	•	•	•
REFERENCING			(*)	•
REFRESH				•
REGISTER				•
RELATIVE	•		•	
RENAME				•
REPEATABLE				•
REPLACE			(*)	
REPLICA				•
REPLY				•
RESIGNAL			(*)	
RESTART				•
RESTRICT	•	•	•	•
RESULT				•
RETURN			(*)	•
RETURNS			(*)	•
REVERSE				•
REVOKE	•	•	•	•

表 99. 保留字列表 (续)

保留字	ODBC	X/Open SQL	ANSI SQL-92	solidDB SQL
RIGHT	•		•	•
ROLE			(*)	•
ROLLBACK	•	•	•	•
ROUTINE			(*)	
ROW			(*)	
ROWID				•
ROWNUM				•
ROWSPERMESAGE				•
ROWVER				•
ROWS	•		•	
SAVEPOINT			(*)	•
SCAN				•
SCHEMA	•		•	•
SCROLL	•		•	
SEARCH			(*)	
SECOND	•		•	
SECTION	•	•	•	
SELECT	•	•	•	•
SENSITIVE			(*)	
SEQUENCE			(*)	•
SERIALIZABLE				•
SESSION	•		•	
SESSION_USER	•		•	
SET	•	•	•	•
SIGNAL			(*)	
SIMILAR			(*)	
SIZE	•		•	

表 99. 保留字列表 (续)

保留字	ODBC	X/Open SQL	ANSI SQL-92	solidDB SQL
SMALLINT	•	•	•	•
SOME	•		•	•
SORT				•
SPACE	•			
SQL	•	•	•	•
SQLCA	•	•		
SQLCODE	•		•	
SQLERROR	•	•	•	•
SQLEXCEPTION			(*)	
SQLSTATE	•			
SQLWARNING	•		(*)	
START				•
STRUCTURE			(*)	
SUBSCRIBE				•
SUBSCRIPTION				•
SUBSTRING	•		•	
SUM	•	•	•	
SYNC_CONFIG				•
SYSTEM	•			
SYSTEM_USER			•	
TABLE	•	•	•	•
TEMPORARY	•		•	
TEST			(*)	
THEN	•		•	•
THERE			(*)	
TIME	•		•	•
TIMEOUT				•

表 99. 保留字列表 (续)

保留字	ODBC	X/Open SQL	ANSI SQL-92	solidDB SQL
TIMESTAMP	•		•	•
TIMEZONE_HOUR	•		•	
TIMEZONE_MINUTE	•		•	
TINYINT				•
TO	•	•	•	•
TRAILING			•	
TRANSACTION	•		•	•
TRANSACTIONS				•
TRANSLATE	•		•	
TRANSLATION	•		•	
TRIGGER			(*)	•
TRIM	•		•	
TRUE	•		•	
TRUNCATE				•
TYPE			(*)	
UNDER			(*)	
UNION	•	•	•	•
UNIQUE	•	•	•	•
UNKNOWN	•		•	
UNREGISTER				•
UPDATE	•	•	•	•
UPPER	•		•	
USAGE	•		•	
USER	•	•	•	•
USING	•	•	•	•
VALUE	•	•	•	•
VALUES	•	•	•	•

表 99. 保留字列表 (续)

保留字	ODBC	X/Open SQL	ANSI SQL-92	solidDB SQL
VARBINARY				•
VARCHAR	•	•	•	•
VARIABLE			(*)	
VARWCHAR				•
VARYING	•	•	•	
VIEW	•	•	•	•
VIRTUAL			(*)	
VISIBLE			(*)	
WAIT			(*)	•
WCHAR				•
WHEN	•		•	•
WHENEVER	•	•	•	
WHERE	•	•	•	•
WHILE			(*)	•
WITH	•	•	•	•
WITHOUT			(*)	
WORK	•	•	•	•
WRITE			•	•
WVARCHAR				•
YEAR	•		•	
ZONE			•	

*CASCADED: 单词 CASCADED 保留在 solidDB 中; 然而, 该单词当前不用于任何的 solidDB SQL 语句中。

附录 E. 数据库系统表和系统视图

E.1 系统表

E.1.1 SQL_LANGUAGES

SQL_LANGUAGES 系统表列示 solidDB 支持的 SQL 标准和 SQL 方言。

表 100. SQL_LANGUAGES 系统表

列名	数据类型	描述
SOURCE	WVARCHAR	定义这个特定 SQL 版本的组织机构。
SOURCE_YEAR	WVARCHAR	相关标准的核准年份。
CONFORMANCE	WVARCHAR	与相关标准的一致性级别。
INTEGRITY	WVARCHAR	指示是否支持“完整性增强功能”。
IMPLEMENTATION	WVARCHAR	唯一地标识供应商的 SQL 语言；如果 SOURCE 为“ISO”，那么包含 NULL。
BINDING_STYLE	WVARCHAR	绑定样式 “DIRECT”、“*EMBED”或“MODULE”。
PROGRAMMING_LANG	WVARCHAR	所使用的主语言。

E.1.2 SYS_ATTAUTH

表 101. SYS_ATTAUTH

列名	数据类型	描述
REL_ID	INTEGER	表标识。
UR_ID	INTEGER	用户或角色的标识。
ATTR_ID	INTEGER	列标识。
PRIV	INTEGER	特权信息。
GRANT_ID	INTEGER	授权者标识。
GRANT_TIM	TIMESTAMP	授权时间。

E.1.3 SYS_AUDIT_TRAIL

SYS_AUDIT_TRAIL 系统表包含审计信息。启用审计时 (`Sql.AuditTrailEnabled=yes`)，关于数据库活动的信息将写入 SYS_AUDIT_TRAIL 系统表。具有管理员权限的用户可以使用常规 SQL 语法来查询 SYS_AUDIT_TRAIL 系统表。

表 102. SYS_AUDIT_TRAIL 表定义

列名	数据类型	描述
CREATIME	TIMESTAMP	落实已审计语句时的时间戳记。
LOGIN_USER	WVARCHAR	登录系统以执行并落实已审计数据的用户。
MACHINE_ID	WVARCHAR	说明执行已审计语句的位置。如果未设置 MACHINE_ID, 那么该值为 NULL。
APP_INFO	WVARCHAR	说明哪个应用程序执行了语句。如果未设置 APP_INFO, 那么该值为 NULL。
CURRENT_CATALOG	WVARCHAR	说明落实事务时使用的是哪个目录。
CURRENT_SCHEMA	WVARCHAR	说明落实事务时使用的是哪个模式。
TYPE	VARCHAR	说明已审计更改的是哪种类型的操作。可能的值描述如下。
SQLSTR	LONG WVARCHAR	说明所审计的已执行 SQL 字符串。

TYPE 的可能值包括:

```

STATUS
ALTER TABLE
ALTER USER (The password in the SQL statement is not recorded)
CREATE CATALOG
CREATE DOMAIN
CREATE EVENT
CREATE INDEX
CREATE PUBLICATION
CREATE ROLE
CREATE SCHEMA
CREATE SEQUENCE
CREATE TABLE
CREATE USER (The password in the SQL statement is not recorded)
CREATE VIEW
DELETE (Recorded if the DELETE statement affected the SYS_AUDIT_TRAIL system table)
DROP CATALOG
DROP DOMAIN
DROP EVENT
DROP INDEX
DROP PUBLICATION
DROP ROLE
DROP SCHEMA
DROP SEQUENCE
DROP TABLE
DROP USER
DROP VIEW
GRANT
GRANT ROLE
REVOKE
REVOKE ROLE
CREATE PROCEDURE
DROP PROCEDURE
CREATE TRIGGER
ALTER TRIGGER
DROP TRIGGER
    
```

E.1.4 SYS_BACKGROUNDJOB_INFO

如果 START AFTER COMMIT 语句的主体无法启动, 那么原因将记录在系统表 SYS_BACKGROUNDJOB_INFO 中。只有失败的 START AFTER COMMIT 语句才会记录到此表中。如果语句 (例如, 过程调用) 成功启动, 那么不会将任何信息存储到此系统表中。成功启动但未完成完成的语句也不会存储在此系统表中。

用户可以通过使用 SQL SELECT 语句或者通过调用系统过程 SYS_GETBACKGROUNDJOB_INFO 从 SYS_BACKGROUNDJOB_INFO 表中检索信息。有关更多详细信息，请参阅第 320 页的 E.1.4，『SYS_BACKGROUNDJOB_INFO』。

并且，当 START AFTER COMMIT 语句无法启动时，将发出系统定义的事件 SYS_EVENT_SACFAILED。有关更多详细信息，请参阅第 373 页的 H.1，『其他事件』中有关该事件的描述。应用程序可以等待此事件并使用作业标识从系统表 SYS_BACKGROUNDJOB_INFO 中检索错误消息。

您可以使用以下管理命令来清空系统表 SYS_BACKGROUNDJOB_INFO:

ADMIN COMMAND 'cleanbgjobinfo';

只有 DBA 才能执行 'cleanbgjobinfo' 命令。

表 103. SYS_BACKGROUNDJOB_INFO

列名	数据类型	描述
ID	INTEGER	作业标识。
STMT	WVARCHAR	未能执行的语句。
USER_ID	INTEGER	用户或角色的标识。
ERROR_CODE	INTEGER	尝试执行该语句时发生的错误。
ERROR_TEXT	WVARCHAR	有关错误的描述。

E.1.5 SYS_BLOBS

这个表包含有关存储到数据库中的 BLOB 的信息。并且，此表确保仅当以逻辑方式将 BLOB 保存多次后才以物理方式将其保存到磁盘。

表 104. SYS_BLOBS

列名	数据类型	描述
ID	BIGINT	BLOB 标识。
STARTPOS	BIGINT	相对于 BLOB 开头的字节偏移 - 页面的开始位置。
ENDSIZE	BIGINT	上一页末尾的字节偏移 +1。
TOTALSIZE	BIGINT	BLOB 的总大小。
REFCOUNT	INTEGER	引用次数，即，同一个 BLOB 的现有实例数。
COMPLETE	INTEGER	指示是否已准备好写 BLOB。
STARTCPNUM	INTEGER	指示在哪个检查点级别启动 BLOB 写操作。
NUMPAGES	INTEGER	BLOB 包含的页数。

表 104. SYS_BLOBS (续)

列名	数据类型	描述
P01_ADDR	INTEGER	第一页相对于 BLOB 开头的字节偏移。
P01_ENDSIZE	BIGINT	第一页的最后一个字节 +1。
P[02...50]_ADDR	INTEGER	第 [2...50] 页相对于 BLOB 开头的字节偏移。
P[02...50]_ENDSIZE	BIGINT	第 [2...50] +1 页的最后一个字节 +1。

E.1.6 SYS_CARDINAL

此表中的数据在每个检查点进行刷新，而在其他时候不会进行刷新。

表 105. SYS_CARDINAL

列名	数据类型	描述
REL_ID	INTEGER	SYS_TABLES 中的关系标识。
CARDIN	INTEGER	表中的行数。
SIZE	INTEGER	表中数据的大小。
LAST_UPD	TIMESTAMP	表中最近一次执行更新时的时间戳记。

E.1.7 SYS_CATALOGS

SYS_CATALOGS 列示可用的目录。

表 106. SYS_CATALOGS

列名	数据类型	描述
ID	INTEGER	目录标识。
NAME	WVARCHAR	目录名。
CREATIME	TIMESTAMP	创建日期和时间。
CREATOR	WVARCHAR	创建者的名称。

E.1.8 SYS_CHECKSTRINGS

SYS_CHECKSTRINGS 列示表的检查约束。

表 107. SYS_CHECKSTRINGS

列名	数据类型	描述
ID	INTEGER	引用 SYS_TABLES 的表标识。

表 107. SYS_CHECKSTRINGS (续)

列名	数据类型	描述
CONSTRAINT_NAME	WVARCHAR	检查约束的名称（在表中唯一）；或者，对于未命名的约束，则为空字符串（所有未命名的检查约束共用一个字符串。它们通过 AND 并置）。
CONSTRAINT	WVARCHAR	约束字符串本身。SQL 解释器对给定的表执行插入/更新时，将检查此约束字符串。

E.1.9 SYS_COLUMNS

此表列示所有系统表列。

不存在用于查看系统列的所有者或用户查看限制。这意味着，所有者可以查看他们在此表中创建的列以外的列，并且不具有访问权或具有特定访问权的用户也能够查看此表中的任何系统列。

表 108. SYS_COLUMNS

列名	数据类型	描述
ID	INTEGER	唯一的列标识。
REL_ID	INTEGER	SYS_TABLES 中的关系标识。
COLUMN_NAME	WVARCHAR	列的名称。
COLUMN_NUMBER	INTEGER	列在表中的编号（按创建顺序排列）。
DATA_TYPE	WVARCHAR	列的数据类型。
SQL_DATA_TYPE_NUM	SMALLINT	ODBC 相容数据类型编号。
DATA_TYPE_NUMBER	INTEGER	内部数据类型编号。
CHAR_MAX_LENGTH	INTEGER	CHAR 字段的最大长度。
NUMERIC_PRECISION	INTEGER	数字精度。
NUMERIC_PREC_RADIX	SMALLINT	数字精度基。
NUMERIC_SCALE	SMALLINT	数字标度。
NULLABLE	CHAR	是否允许 NULL 值（Yes 或 No）。
NULLABLE_ODBC	SMALLINT	（ODBC）是否允许 NULL 值（1 或 0）。
FORMAT	WVARCHAR	保留供将来使用。
DEFAULT_VAL	WVARCHAR	当前的缺省值（如果已设置的话）。
ATTR_TYPE	INTEGER	由用户定义（0）或内部（>0）。

表 108. SYS_COLUMNS (续)

列名	数据类型	描述
REMARKS	LONG WVARCHAR	保留供将来使用。

E.1.10 SYS_COLUMNS_AUX

将具有缺省值的列插入到已包含行的表时，不会对现有的行追加列缺省值。而是，将把列插入语句中定义的缺省值写入 SYS_COLUMNS_AUX 表。如果 SQL 查询面向先于该列插入到表中的行，那么将从 SYS_COLUMNS_AUX 表中读取列值，除非插入该列后已将该行的该列更改为新值。在 SYS_COLUMNS_AUX 表中，只保存原始缺省值。

表 109. SYS_COLUMNS_AUX

列名	数据类型	描述
ID	INTEGER	表标识。
ORIGINAL_DEFAULT	WVARCHAR	原始缺省值。

E.1.11 SYS_DL_REPLICA_CONFIG

这个表包含主服务器的无盘配置。此表仅通过 **solidsetup** 命令进行更新。用户不应该直接修改此表。直接修改此表会产生不良后果。

表 110. SYS_DL_REPLICA_CONFIG

列名	数据类型	描述
CFG_NAME	WVARCHAR (254) PRIMARY KEY NOT NULL	无盘副本服务器配置的名称。
INI_FILE	LONG WVARCHAR	副本服务器配置文件的名称。solid.ini 文件内容将作为 BLOB 插入到此列中。
LIC_FILE	LONG WVARCHAR	副本服务器许可证文件的名称。solid.lic 文件内容将作为 BLOB 插入到此列中。
SCHEMA_FILE	LONG WVARCHAR	副本服务器模式的名称。模式文件内容将作为 BLOB 插入到此列中。

E.1.12 SYS_DL_REPLICA_DEFAULT

这个表包含主服务器的无盘缺省配置。此表仅通过 **solidsetup** 命令进行更新。用户不应该直接修改此表。直接修改此表会产生不良后果。

表 111. SYS_DL_REPLICA_DEFAULT

列名	数据类型	描述
REPLICA_NAME	VARCHAR(254) NOT NULL PRIMARY KEY	副本服务器的名称。

表 111. SYS_DL_REPLICA_DEFAULT (续)

列名	数据类型	描述
INI_CFG	VARCHAR(254) REFERENCE SYS_DL_REPLICA_CONFIG(CFG_NAME)	副本服务器配置文件的名称。
LIC_CFG	VARCHAR(254) REFERENCE SYS_DL_REPLICA_CONFIG(CFG_NAME)	副本服务器许可证文件的名称。
SCHEMA_CFG	VARCHAR(254) REFERENCE SYS_DL_REPLICA_CONFIG(CFG_NAME)	副本服务器模式的名称。

E.1.13 SYS_EVENTS

表 112. SYS_EVENTS

列名	数据类型	描述
ID	INTEGER	唯一的事件标识。
EVENT_NAME	WVARCHAR	事件的名称。
EVENT_PARAMCOUNT	INTEGER	参数数目。
EVENT_PARAMTYPES	LONG VARBINARY	参数的类型。
EVENT_TEXT	WVARCHAR	事件的主体。
EVENT_SCHEMA	WVARCHAR	事件的所有者。
EVENT_CATALOG	WVARCHAR	事件的所有者。
CREATIME	TIMESTAMP	创建时间。
TYPE	INTEGER	保留供将来使用。

E.1.14 SYS_FEDT_DB_PARTITION

SYS_FEDT_DB_PARTITION 表描述可用的“事务日志阅读器”分区。

表 113. SYS_FEDT_DB_PARTITION 表定义

列名	数据类型	描述
PARTITION_ID	CHAR(30)	此列包含分区名。
描述	VARCHAR(255)	未使用
LOADACTIVE	INTEGER	未使用

E.1.15 SYS_FEDT_TABLE_PARTITION

SYS_FEDT_TABLE_PARTITION 表提供表与“日志阅读器”分区之间的映射。

表 114. SYS_FEDT_TABLE_PARTITION 表定义

列名	数据类型	描述
TABLE_CATALOG	VARCHAR(255)	此列指示目录名。 每个表都有相关的目录名。目录可在创建数据库时创建，也可以在以后使用 CREATE CATALOG 语句创建。目录名没有出厂值。
TABLE_SCHEMA	VARCHAR(255)	此列指示模式名。 每个表都有相关的模式名。缺省情况下，模式名为用户标识（用户名）。也可以使用 CREATE SCHEMA 语句来创建模式。
PARTITION_ID	CHAR(30)	此列指示分区名。 此列是 FEDT_TABLE_PARTITION 表的外键。
BE_TABLE_DATABASE	VARCHAR(255)	未使用
BE_TABLE_SCHEMA	VARCHAR(255)	未使用
BE_TABLE_NAME	CHAR(30)	未使用
RANGE_COL_NAME	CHAR(30)	未使用
LOWER_LIMIT	VARCHAR(255)	未使用
UPPER_LIMIT	VARCHAR(255)	未使用
SELECT_LIST	VARCHAR(255)	未使用
LOADORDER	INTEGER	未使用

E.1.16 SYS_FORKEYPARTS

表 115. SYS_FORKEYPARTS

列名	数据类型	描述
KEY_CATALOG	INTEGER	键的创建者名称或所有者。
ID	INTEGER	外键标识。
KEYP_NO	INTEGER	键部件号。
ATTR_NO	INTEGER	列号。
ATTR_ID	INTEGER	列标识。
ATTR_TYPE	INTEGER	列类型。
CONST_VALUE	VARBINARY	可能的内部常量值；否则为 NULL。

E.1.17 SYS_FORKEYS

表 116. SYS_FORKEYS

列名	数据类型	描述
ID	INTEGER	外键标识。
REF_REL_ID	INTEGER	所引用的表标识。
CREATE_REL_ID	INTEGER	创建者表标识。
REF_KEY_ID	INTEGER	所引用的键标识。
REF_TYPE	INTEGER	引用类型。
KEY_SCHEMA	WVARCHAR	创建者的名称。
KEY_CATALOG	WVARCHAR	键的创建者名称或所有者。
KEY_NREF	INTEGER	所引用的键部件的编号。

E.1.18 SYS_HOTSTANDBY

此系统表已被废弃。它仅用于 4.0 以前的版本。

E.1.19 SYS_INFO

表 117. SYS_INFO

列名	数据类型	描述
PROPERTY	WVARCHAR	属性的名称。
VALUE_STR	WVARCHAR	字符串形式的值。
VALUE_INT	INTEGER	整数形式的值。

E.1.20 SYS_KEYPARTS

表 118. SYS_KEYPARTS

列名	数据类型	描述
ID	INTEGER	此列是对 sys_keys.id 的外键引用，因此您可以确定键部件所属的键。
REL_ID	INTEGER	SYS_TABLES 中的关系标识。
KEYP_NO	INTEGER	键部件标识。
ATTR_ID	INTEGER	列标识。
ATTR_NO	INTEGER	列在表中的编号（按创建顺序排列）。

表 118. SYS_KEYPARTS (续)

列名	数据类型	描述
ATTR_TYPE	INTEGER	列的类型。
CONST_VALUE	VARBINARY	常量值或 NULL。
ASCENDING	CHAR	键是升序 (Yes) 还是降序 (No)。

E.1.21 SYS_KEYS

所有数据库表都必须有一个集群键。此键用于定义数据的物理排序顺序。它不会对容量产生影响。如果定义了主键，那么主键将作为集群键。如果未定义主键，那么将在 SYS_KEYS 中自动创建 key_name 为“\$CLUSTKEY_xxxxx”的条目。

如果存在表的主键定义，那么在 SYS_KEYS 中，将有一个 key_name 类似于“\$PRIMARYKEY_xxxx”的条目。key_primary 和 key_clustering 列的值均为 YES。

如果不存在表的主键定义，那么在 SYS_KEYS 中，将有一个 key_name 类似于“\$CLUSTKEY_xxxxx”的条目。key_primary 列的值为 NO，key_clustering 列的值为 YES。

表 119. SYS_KEYS

列名	数据类型	描述
ID	INTEGER	唯一的键标识。
REL_ID	INTEGER	SYS_TABLES 中的关系标识。
KEY_NAME	WVARCHAR	键的名称。
KEY_UNIQUE	CHAR	键是否唯一 (Yes 或 No)。
KEY_NONUNIQUE_ODBC	SMALLINT	(ODBC) 键是否不唯一 (1 或 0)。
KEY_CLUSTERING	CHAR	键是否集群键 (Yes 或 No)。
KEY_PRIMARY	CHAR	键是否主键 (Yes 或 No)。
KEY_PREJOINED	CHAR	保留供将来使用。
KEY_SCHEMA	WVARCHAR	键的所有者。
KEY_NREF	INTEGER	创建主键时，服务器将使用表的所有字段，即使用户指定了 N 个字段亦如此 (用户指定的 N 个字段将成为键的前 N 个字段)。KEY_NREF = N，即，用户指定的字段数。

E.1.22 SYS_LOGPOS

SYS_LOGPOS 表包含关于“事务日志阅读器”保持同步所需的日志间隔的信息。

每行表示一个日志间隔，用使用参数 `LogReader.SaveLogposInterval` 定义的近似粒度记录。当前时间间隔的大小达到 `SaveLogposInterval` 的值时，就会添加新行。日志间隔大小的总和约等于参数 `LogReader.MaxLogSize` 的值。所指示日志位置与最近记录的日志间隔头的距离大于 `LogReader.MaxLogSize` 时，将会自动删除日志间隔行。

`SYS_LOGPOS` 表仅由“日志阅读器”在执行 `SYS_LOG` 表查询时在内部使用；该表不包含设计使用“日志阅读器”的应用程序时要使用的任何信息。

如果查询请求在 `SYS_LOGPOS` 中表示的日志间隔之外的日志记录，查询将返回错误“日志溢出”。

表 120. `SYS_LOGPOS` 表定义

列名	数据类型	描述
ID	INTEGER	逻辑日志内部标识（日志间隔的标识）
LOG_POS	VARCHAR(255)	日志间隔头的物理位置
SIZE	INTEGER	与上一个日志间隔头位置的距离，以字节计（约等于使用 <code>LogReader.SaveLogposInterval</code> 参数定义的值）

E.1.23 `SYS_PROCEDURES`

此系统表列示过程。

特定的用户无法查看过程。所有者只能查看他们所创建的过程。用户只能查看他们有权执行的过程以查看过程定义。如果用户不具有访问权，那么他们无法查看所有过程。注意，执行访问权并未允许用户查看过程定义。对于 `DBA` 而言，没有任何限制。

表 121. `SYS_PROCEDURES`

列名	数据类型	描述
ID	INTEGER	唯一的过程标识。
PROCEDURE_NAME	WVARCHAR	过程名。
PROCEDURE_TEXT	LONG WVARCHAR	过程主体。
PROCEDURE_BIN	LONG VARBINARY	编译后的过程形式。
PROCEDURE_SCHEMA	WVARCHAR	包含 <code>PROCEDURE_NAME</code> 的模式的名称。
PROCEDURE_CATALOG	WVARCHAR	包含 <code>PROCEDURE_NAME</code> 的目录的名称。
CREATETIME	TIMESTAMP	创建时间。
TYPE	INTEGER	保留供将来使用。

E.1.24 SYS_PROCEDURE_COLUMNS

SYS_PROCEDURE_COLUMNS 定义输入参数和结果集列。

表 122. SYS_PROCEDURE_COLUMNS

列名	数据类型	描述
PROCEDURE_ID	INTEGER	过程标识。
COLUMN_NAME	WVARCHAR	过程列名。
COLUMN_TYPE	SMALLINT	过程列类型 (SQL_PARAM_INPUT 或 SQL_RESULT_COL)。
DATA_TYPE	SMALLINT	列的 SQL 数据类型。
TYPE_NAME	WVARCHAR	列的 SQL 数据类型名称。
COLUMN_SIZE	INTEGER	过程列的大小。
BUFFER_LENGTH	INTEGER	列大小 (以字节计)。
DECIMAL_DIGITS	SMALLINT	过程列的小数位数。
NUM_PREC_RADIX	SMALLINT	数字数据类型的基 (2 或 10, 不适用时为 NULL)。
NULLABLE	SMALLINT	过程列是否接受 NULL 值。
REMARKS	WVARCHAR	过程列的描述。
COLUMN_DEF	WVARCHAR	列的缺省值。始终为 NULL, 即, 未指定缺省值。
SQL_DATA_TYPE	SMALLINT	SQL 数据类型。
SQL_DATETIME_SUB	SMALLINT	日期时间的子类型代码。始终为 NULL。
CHAR_OCTET_LENGTH	INTEGER	字符或二进制数据类型列的最大长度 (以字节计)。
ORDINAL_POSITION	INTEGER	列的序数位置。
IS_NULLABLE	WVARCHAR	始终为“YES”。

E.1.25 SYS_PROPERTIES

此表仅供 HSB 内部使用。

表 123. SYS_PROPERTIES

列名	数据类型	描述
KEY	WVARCHAR	属性标识。
VALUE	WVARCHAR	属性的值。

表 126. SYS_SEQUENCES (续)

列名	数据类型	描述
ID	INTEGER	唯一标识。
DENSE	CHAR	紧密序列还是稀疏序列。
SEQUENCE_SCHEMA	WVARCHAR	包含 SEQUENCE_NAME 的模式名称。
SEQUENCE_CATALOG	WVARCHAR	包含 SEQUENCE_NAME 的目录名称。
CREATETIME	TIMESTAMP	创建时间。

E.1.29 SYS_SERVER

SYS_SERVER 表包含后端服务器登录数据，供 solidDB Universal Cache 中的 SQL 传递使用。

在 solidDB 和后端之间创建了第一个预定时，InfoSphere CDC for solidDB 实例会从后端 InfoSphere CDC 实例中检索后端数据服务器登录数据，然后将通过 CREATE REMOTE SERVER 语句将它存储在 solidDB 系统表 SYS_SERVER 中。

表 127. SYS_SERVER 表定义

列名	数据类型	描述
NAME	VARCHAR	后端服务器的名称
DRIVER	VARCHAR	未使用
CONNECT	VARCHAR	未使用
UID	VARCHAR	连接至后端服务器时要使用的用户名
PWD	VARCHAR	连接至后端服务器时要使用的密码 注：该密码数据已隐藏。

E.1.30 SYS_SYNC_REPLICA_PROPERTIES

表 128. SYS_SYNC_REPLICA_PROPERTIES

列名	数据类型	描述
ID	INTEGER	副本服务器标识。
NAME	VARCHAR	属性名。
VALUE	VARCHAR	属性值。

主键基于 ID 和 NAME 字段。

E.1.31 SYS_SYNONYM

表 129. SYS_SYNONYM

列名	数据类型	描述
TARGET_ID	INTEGER	保留供将来使用。
SYNON	INTEGER	保留供将来使用。

E.1.32 SYS_TABLEMODES

表 130. SYS_TABLEMODES

列名	数据类型	描述
ID	INTEGER	关系标识。
MODE	WVARCHAR	并行控制方式（允许的值：OPTIMISTIC、PESSIMISTIC、MAINMEMORY 或 MAINMEMORY PESSIMISTIC）。
MODIFY_TIME	TIMESTAMP	上次修改时间。
MODIFY_USER	WVARCHAR	上次执行修改的用户。

SYS_TABLEMODES 只显示已显式设置方式的表的方式。SYS_TABLEMODES 不显示仍处于缺省方式的表的方式。除非已设置 solid.ini 配置参数 **General.Pessimistic** 为“是”，否则 D 表的缺省方式是“乐观”。M 表始终是悲观表。

要列示已将方式显式设置为乐观或悲观的表的名称和方式，请执行以下命令：

```
SELECT SYS_TABLEMODES.ID, SYS_TABLEMODES.MODE, SYS_TABLES.TABLE_NAME
FROM SYS_TABLEMODES, SYS_TABLES
WHERE SYS_TABLEMODES.ID = SYS_TABLES.ID
AND SYS_TABLES.TABLE_NAME = '<table_name>';
```

输出将类似于：

```
      ID TABLE_NAME  MODE
-----
10054 TABLE2      OPTIMISTIC
10056 TABLE3      PESSIMISTIC
```

有关设置并行控制方式的更多信息，请参阅第 110 页的『将并行（锁定）方式设置为乐观或悲观』。

E.1.33 SYS_TABLES

此表列示所有系统表。

不存在与查看系统表相关的限制；这意味着，即使不具有访问权的用户也可以查看系统表。但是，特定的用户无法查看用户表信息。所有者只能查看他们创建的用户表，用户只能查看他们对其具有 INSERT、UPDATE、DELETE 或 SELECT 访问权的表。用户如果不具有访问权，那么无法查看任何用户表。对于 DBA 而言，没有任何限制。

表 131. SYS_TABLES

列名	数据类型	描述
ID	INTEGER	唯一的表标识。
TABLE_NAME	WVARCHAR	表的名称。
TABLE_TYPE	WVARCHAR	表的类型 (BASE TABLE 或 VIEW)。
TABLE_SCHEMA	WVARCHAR	包含 TABLE_NAME 的模式名称。
TABLE_CATALOG	WVARCHAR	包含 TABLE_NAME 的目录名称。
CREATIME	TIMESTAMP	表的创建时间。
CHECKSTRING	LONG WVARCHAR	为表定义的可能检查选项。
REMARKS	LONG WVARCHAR	保留供将来使用。

E.1.34 SYS_TRIGGERS

此系统表列示触发器。

特定的用户无法查看触发器。所有者只能查看他们所创建的那些触发器。常规用户无法查看触发器。对于 DBA 而言，没有任何限制。

表 132. SYS_TRIGGERS

列名	数据类型	描述
ID	INTEGER	唯一的表标识。
TRIGGER_NAME	WVARCHAR	触发器名称。
TRIGGER_TEXT	LONG WVARCHAR	触发器主体。
TRIGGER_BIN	LONG VARBINARY	编译后的触发器形式。
TRIGGER_SCHEMA	WVARCHAR	包含 TRIGGER_NAME 的模式名称。
TRIGGER_CATALOG	WVARCHAR	包含 TRIGGER_NAME 的目录名称。
TRIGGER_ENABLED	CHAR	如果已启用触发器，那么包含“YES”，否则包含“NO”。
CREATIME	TIMESTAMP	触发器的创建时间。
TYPE	INTEGER	保留供将来使用。
REL_ID	INTEGER	关系标识。

E.1.35 SYS_TYPES

表 133. SYS_TYPES

列名	数据类型	描述
TYPE_NAME	WVARCHAR	数据类型的名称。
DATA_TYPE	SMALLINT	(ODBC) 数据类型编号。
PRECISION	INTEGER	(ODBC) 数据类型的精度。
LITERAL_PREFIX	WVARCHAR	(ODBC) 字面值的可能前缀。
LITERAL_SUFFIX	WVARCHAR	(ODBC) 字面值的可能后缀。
CREATE_PARAMS	WVARCHAR	(ODBC) 创建该数据类型的列时所需的参数。
NULLABLE	SMALLINT	(ODBC) 该数据类型能否包含 NULL 值。
CASE_SENSITIVE	SMALLINT	(ODBC) 该数据类型是否区分大小写。
SEARCHABLE	SMALLINT	(ODBC) 受支持的搜索操作。
UNSIGNED_ATTRIBUTE	SMALLINT	(ODBC) 该数据类型是否不带符号。
MONEY	SMALLINT	(ODBC) 该数据类型是否货币数据类型。
AUTO_INCREMENT	SMALLINT	(ODBC) 该数据类型是否自动递增。
LOCAL_TYPE_NAME	WVARCHAR	(ODBC) 该数据类型是否具有另一实现定义的名称。
MINIMUM_SCALE	SMALLINT	(ODBC) 该数据类型的最小标度。
MAXIMUM_SCALE	SMALLINT	(ODBC) 该数据类型的最大标度。

E.1.36 SYS_UROLE

SYS_UROLE 包含用户到角色的映射。

表 134. SYS_UROLE

列名	数据类型	描述
U_ID	INTEGER	用户标识。
R_ID	INTEGER	角色标识。

E.1.37 SYS_USERS

SYS_USERS 列示有关用户和角色的信息。

表 135. SYS_USERS

列名	数据类型	描述
ID	INTEGER	用户或角色的标识。
NAME	WVARCHAR	用户或角色名称。
TYPE	WVARCHAR	用户类型，即 USER 或 ROLE。
PRIV	INTEGER	特权信息。
PASSW	VARBINARY	加密格式的密码。
PRIORITY	INTEGER	保留供将来使用。
PRIVATE	INTEGER	指定该用户是专用用户还是公用用户。
LOGIN_CATALOG	WVARCHAR	保留供将来使用。

E.1.38 SYS_VIEWS

表 136. SYS_VIEWS

列名	数据类型	描述
V_ID	INTEGER	此视图的唯一标识。
TEXT	LONG WVARCHAR	视图定义。
CHECKSTRING	LONG WVARCHAR	为视图定义的可能检查选项。
REMARKS	LONG WVARCHAR	保留供将来使用。

E.2 用于执行数据同步的系统表

solidDB 包含多个用于实现同步功能的系统表。通常，这些表仅供内部使用。但是，在开发新应用程序以及对其进行故障诊断时，您可能需要知道这些表的内容。

注意，下面的表名按字母顺序排列。

E.2.1 SYS_BULLETIN_BOARD

这个表包含在此数据库目录中执行事务时参数公告牌始终包含的持久参数。

表 137. SYS_BULLETIN_BOARD

列名	描述
PARAM_NAME	持久参数的名称。

表 137. SYS_BULLETIN_BOARD (续)

列名	描述
PARAM_VALUE	参数的值。
PARAM_CATALOG	定义主/副本目录。

E.2.2 SYS_PUBLICATION_ARGS

这个表包含此主数据库中的发布输入自变量。

表 138. SYS_PUBLICATION_ARGS

列名	描述
PUBL_ID	发布的内部标识。
ARG_NUMBER	自变量的序号。
NAME	自变量的名称。
TYPE	自变量的类型。
LENGTH_OR_PRECISION	自变量的长度或精度。
SCALE	自变量的标度。

E.2.3 SYS_PUBLICATION_REPLICA_ARGS

这个表包含副本数据库中发布自变量的定义。

表 139. SYS_PUBLICATION_REPLICA_ARGS

列名	描述
MASTER_ID	主数据库的内部标识，将根据这个数据库来刷新数据。
PUBL_ID	发布的内部标识。
ARG_NUMBER	自变量的序号。
NAME	自变量的名称。
LENGTH_OR_PRECISION	自变量的长度或精度。
SCALE	自变量的标度。

E.2.4 SYS_PUBLICATION_REPLICA_STMTARGS

这个表包含副本数据库中发布自变量与语句之间的映射。

表 140. SYS_PUBLICATION_REPLICA_STMTARGS

列名	描述
MASTER_ID	主数据库的内部标识, 将根据这个数据库来刷新数据。
PUBL_ID	发布的内部标识。
STMT_NUMBER	语句的序号。
STMT_ARG_NUMBER	语句自变量的序号。
PUBL_ARG_NUMBER	发布自变量的序号。

E.2.5 SYS_PUBLICATION_REPLICA_STMTS

这个表包含副本数据库中发布语句的定义。

表 141. SYS_PUBLICATION_REPLICA_STMTS

列名	描述
MASTER_ID	主数据库的内部标识, 将根据这个数据库来刷新数据。
PUBL_ID	发布的内部标识。
STMT_NUMBER	语句的序号。
REPLICA_CATALOG	副本数据库中目标目录的名称。
REPLICA_SCHEMA	副本数据库中目标模式的名称。
REPLICA_TABLE	副本数据库中目标表的名称。
TABLE_ALIAS	目标表的别名。
REPLICA_FROM_STR	字符串形式的 SQL FROM tables。
WHERE_STR	字符串形式的 SQL WHERE arguments。
LEVEL	此 SQL 语句在此发布层次结构中所处的层。

E.2.6 SYS_PUBLICATION_STMTARGS

这个表包含主数据库中发布自变量与语句之间的映射。

表 142. SYS_PUBLICATION_STMTARGS

列名	描述
PUBL_ID	发布的内部标识。
STMT_NUMBER	语句的序号。

表 142. SYS_PUBLICATION_STMTARGS (续)

列名	描述
STMT_ARG_NUMBER	语句自变量的序号。
PUBL_ARG_NUMBER	发布自变量的序号。

E.2.7 SYS_PUBLICATION_STMTS

这个表包含主数据库中的发布语句。

表 143. SYS_PUBLICATION_STMTS

列名	描述
PUBL_ID	发布的内部标识。
MASTER_SCHEMA	主数据库中发布模式的名称。
MASTER_TABLE	主数据库中表的名称。
REPLICA_SCHEMA	副本数据库中模式的名称。
REPLICA_TABLE	副本数据库中表的名称。
TABLE_ALIAS	目标表的别名。
MASTER_SELECT_STR	字符串形式的 SQL SELECT INTO columns。
REPLICA_SELECT_STR	字符串形式的 SQL SELECT INTO columns。
MASTER_FROM_STR	字符串形式的 SQL SELECT FROM tables。
REPLICA_FROM_STR	字符串形式的 SQL SELECT FROM tables。
WHERE_STR	字符串形式的 SQL WHERE arguments。
DELETEFLAG_STR	供内部使用。
LEVEL	此 SQL 语句在此发布层次结构中所处的层。

E.2.8 SYS_PUBLICATIONS

这个表包含此主数据库中定义的发布。

表 144. SYS_PUBLICATIONS

列名	描述
ID	发布的内部标识。
NAME	发布的名称。
CREATOR	此发布的创建者的用户标识。

表 144. SYS_PUBLICATIONS (续)

列名	描述
CREATETIME	此发布的创建日期和时间。
ARGCOUNT	此发布的输入自变量数目。
STMTCOUNT	此发布中包含的语句数目。
TIMEOUT	不适用。
TEXT	CREATE PUBLICATION 语句的内容。
PUBL_CATALOG	定义主目录。

E.2.9 SYS_PUBLICATIONS_REPLICA

这个表包含此副本数据库中正在使用的发布。

表 145. SYS_PUBLICATIONS_REPLICA

列名	描述
MASTER_ID	主数据库的内部标识，将根据这个数据库来刷新数据。
ID	发布的内部标识。
NAME	发布的名称。
CREATOR	此发布的创建者的用户标识。
ARGCOUNT	此发布的输入自变量数目。
STMTCOUNT	此发布所包含的语句数目。

E.2.10 SYS_SYNC_BOOKMARKS

这个表包含主数据库中正在使用的书签。

表 146. SYS_SYNC_BOOKMARKS

列名	描述
BM_ID	书签的内部标识。
BM_CATALOG	保留供将来使用。
BM_NAME	书签的名称。
BM_VERSION	主数据库中书签的内部版本信息。
BM_CREATOR	书签的创建者的用户标识。
BM_CREATETIME	书签的创建时间。

E.2.11 SYS_SYNC_HISTORY_COLUMNS

如果对一个表打开同步历史记录，那么可以对所有列都打开该历史记录，也可以只对部分列打开该历史记录。如果对部分列打开该历史记录，那么 SYS_SYNC_HISTORY_COLUMNS 表将记录您正在保留哪些列的同步历史记录信息。对于每个已保留同步历史记录的列，SYS_SYNC_HISTORY_COLUMNS 都包含一行。

表 147. SYS_SYNC_HISTORY_COLUMNS

列名	描述
REL_ID	要保留其同步历史记录的表的标识。
COLUMN_NUMBER	该表中已保留同步历史记录的列的序号。（例如，如果保留该表中第二列的同步历史记录，那么此字段将包含数字 2。）

E.2.12 SYS_SYNC_INFO

这个表包含同步信息，其中，每个节点各占一行。

表 148. SYS_SYNC_INFO

列名	描述
NODE_NAME	主控节点或副本节点。
NODE_CATALOG	节点所属的目录。
IS_MASTER	如果为 YES，那么此节点是主控节点。
IS_REPLICA	如果为 YES，那么此节点是副本节点。
CREATIME	节点的创建日期和时间。
CREATOR	节点创建者的用户名。

E.2.13 SYS_SYNC_MASTER_MSGINFO

这个表包含关于主数据库中当前活动消息的信息。

这个表中的数据用于控制副本数据库与主数据库之间的同步过程。这个表还包含对于故障诊断而言特别有用的信息。如果主数据库中消息的执行由于出错而停止，那么您可以查询这个表以获取问题原因以及引起错误的事务和语句。

表 149. SYS_SYNC_MASTER_MSGINFO

列名	描述
状态	消息的当前状态。可能的值如下所示： <ul style="list-style-type: none"> • 0 = DELETED - 不适用（内部非持久状态）。 • 1 = ERROR - 消息处理期间发生错误；错误原因已记录在该行的错误列中。 • 10 = RECEIVED - 主数据库已接收到来自副本数据库的消息。 • 11 = SAVED - 已将消息保存在主数据库中，并且正在对其进行处理。 • 12 = READY - 主数据库已处理消息。 • 13 = SENT - 不适用（内部非持久状态）。
REPLICA_ID	从中发送消息的副本数据库的标识。
MASTER_ID	要将消息发送到的主数据库的标识。
MSG_ID	消息的内部标识。
MSG_NAME	用户对消息指定的名称。
MSG_TIME	消息的创建时间。
MSG_BYTE_COUNT	消息的大小（以字节计）。
CREATE_UID	创建消息的用户的标识。
FORWARD_UID	转发消息的用户的标识。
ERROR_CODE	导致消息执行终止的错误的代码。您可以根据 TRX_ID 和 STMT_ID 信息来确定引起错误的事务和语句。
ERROR_TEXT	导致消息执行终止的错误的描述。
TRX_ID	引起错误的事务的序号。
STMT_ID	事务中引起错误的语句的序号。
ORD_ID_COUNT	不适用（仅供内部使用）。
ORD_ID	不适用（仅供内部使用）。
FLAGS	NULL 或 0 = 正常消息。 1 = 消息在应答被发送到副本数据库时被删除。
FAILED_MSG_ID	这是作为主键组成部分的 INTEGER 列。对于正常消息而言，值为零。如果 LOG_ERRORS 选项为 ON 并且存在任何错误，那么值为 msg_id。

E.2.14 SYS_SYNC_MASTER_RECEIVED_BLOB_REFS

接收到的 BLOB 存储在主数据库中的这个表中。此实现确保仅当以逻辑方式将 BLOB 保存多次后才以物理方式将其保存到磁盘。

表 150. SYS_SYNC_MASTER_RECEIVED_BLOB_REFS

列名	描述
REPLICA_ID	接收到的消息所来自于的副本数据库的内部标识。
MSG_ID	消息的内部标识。
BLOB_NUM	用于标识 BLOB 的编号。
DATA	对 BLOB 的引用。

E.2.15 SYS_SYNC_MASTER_RECEIVED_MSGPARTS

这个表包含主数据库中已从副本数据库接收到但尚未在主数据库中进行处理的消息部件。

表 151. SYS_SYNC_MASTER_RECEIVED_MSGPARTS

列名	描述
REPLICA_ID	接收到的消息所来自于的副本数据库的内部标识。
MSG_ID	消息的内部标识。
PART_NUMBER	消息部件的序号。
DATA_LENGTH	消息部件中数据的长度。
DATA	消息部件的数据。

E.2.16 SYS_SYNC_MASTER_RECEIVED_MSGS

这个表包含主数据库中已从副本数据库接收到但尚未在主数据库中进行处理的消息。

表 152. SYS_SYNC_MASTER_RECEIVED_MSGS

列名	描述
REPLICA_ID	接收到的消息所来自于的副本数据库的内部标识。
MSG_ID	消息的内部标识。
CREATIME	消息的创建时间。
CREATOR	创建消息的用户的用户标识。

E.2.17 SYS_SYNC_MASTER_STORED_BLOB_REFS

要发送的 BLOB 存储在主数据库中的这个表中。此实现确保仅当以逻辑方式将 BLOB 保存多次后才以物理方式将其保存到磁盘。

表 153. SYS_SYNC_MASTER_STORED_BLOB_REFS

列名	描述
REPLICA_ID	消息将被发送到的副本数据库的内部标识。
MSG_ID	消息的内部标识。
BLOB_NUM	用于标识 BLOB 的编号。
DATA	对 BLOB 的引用。

E.2.18 SYS_SYNC_MASTER_STORED_MSGPARTS

这个表包含已在主数据库中创建但尚未发送到副本数据库的消息部件结果集。

表 154. SYS_SYNC_MASTER_STORED_MSGPARTS

列名	描述
REPLICA_ID	消息将被发送到的副本数据库的内部标识。
MSG_ID	消息的内部标识。
ORDER_ID	结果集的序号。
RESULT_SET_ID	结果集的内部标识。
RESULT_SET_TYPE	结果集的类型。
PART_NUMBER	结果集中消息部件的序号。
DATA_LENGTH	结果集中的消息部件中数据的长度。
DATA	消息部件的数据。

E.2.19 SYS_SYNC_MASTER_STORED_MSGS

这个表包含已在主数据库中创建但尚未发送到副本数据库的消息。

表 155. SYS_SYNC_MASTER_STORED_MSGS

列名	描述
REPLICA_ID	消息将被发送到的副本数据库的内部标识。
MSG_ID	消息的内部标识。
CREATIME	消息的创建时间。
CREATOR	创建消息的用户的用户标识。

E.2.20 SYS_SYNC_MASTER_SUBSC_REQ

这个表包含已请求并正在等待在主数据库中执行的预订的列表。

表 156. SYS_SYNC_MASTER_SUBSC_REQ

列名	描述
REPLICA_ID	接收到的语句来自于的副本数据库的内部标识。
MSG_ID	接收到的语句所在消息的内部标识。
ORD_ID	预订的序号。
TRX_ID	预订所属的事务的内部标识。
STMT_ID	预订中的语句的内部标识。
REQUEST_ID	不适用。
PUBL_ID	所预订/刷新的发布的内部标识。
VERSION	主数据库中预订的内部版本信息。
REPLICA_VERSION	副本数据库中预订的内部版本信息。
FULLSUBSC	指示该预订是完全预订还是递增预订。

E.2.21 SYS_SYNC_MASTER_VERSIONS

这个表包含主数据库对副本数据库进行的预订的列表。

表 157. SYS_SYNC_MASTER_VERSIONS

列名	描述
REPLICA_ID	副本数据库的内部标识。
REQUEST_ID	预订的序号。
VERS_TIME	预订的创建时间。
PUBL_ID	发布的标识。
TABNAME	发布的表的名称。
TABSCHEMA	表的模式的名称。
PARAM_CRC	不适用（仅供内部使用）。
PARAM	二进制格式的发布参数。
VERSION	已从副本数据库获取的数据的版本。

E.2.22 SYS_SYNC_MASTERS

这个表包含副本数据库所访问的主数据库的列表。

表 158. SYS_SYNC_MASTERS

列名	描述
NAME	对主数据库指定的名称。
ID	主数据库的内部标识。
REMOTE_NAME	不适用。
REPLICA_NAME	对副本数据库指定的名称。
REPLICA_ID	副本数据库的超大字符集标识。
REPLICA_CATALOG	定义向此主数据库注册的副本目录。
CONNECT	主数据库的连接字符串。
CREATOR	将数据库设置为主数据库的用户的标识。
ISDEFAULT	保留供将来使用。

E.2.23 SYS_SYNC_RECEIVED_BLOB_ARGS

这个表在主数据库中。抽取来自副本数据库的消息时，BLOB 参数将保存在这个表中。这些行仅存在到消息中的事务执行完毕为止。

表 159. SYS_SYNC_RECEIVED_BLOB_ARGS

列名	描述
REPLICA	接收到的 BLOB 参数来自于的副本数据库的内部标识。
MSG	消息的内部标识。
ORD_ID	BLOB 部件的序号。
TRX_ID	用于标识事务的事务标识。
ID	用户的内部标识。
ARGNO	参数的编号。
ARG_VALUE	二进制格式的参数值。

E.2.24 SYS_SYNC_RECEIVED_STMTS

这个表包含已在主数据库中接收到的所传播语句。

表 160. SYS_SYNC_RECEIVED_STMTS

列名	描述
REPLICA	接收到的语句来自于的副本数据库的内部标识。
MSG	接收到的语句所在消息的内部标识。
ORD_ID	不适用。
TXN_ID	语句所属的事务的内部标识。
ID	语句在事务中的序号。
CLASS	常量的类型。
STRING	字符串形式的 SQL 语句。
ARG_COUNT	与语句绑定的参数的数目。
ARG_TYPES	与语句绑定的参数的类型。
ARG_VALUES	二进制格式的参数值。
USER_ID	保存语句的用户的标识。
REQUEST_ID	不适用。
FLAGS	此列指示错误处理方式（例如，IGNORE_ERRORS 和 LOG_ERRORS 等等）。
ERRCODE	如果在主数据库中执行语句时失败，那么此列包含错误代码。
ERR_STR	如果在主数据库中执行语句时失败，那么此列包含有关所发生的错误的描述。

E.2.25 SYS_SYNC_REPLICA_MSGINFO

这个表包含关于副本数据库中当前活动消息的信息。

这个表中的数据用于控制副本数据库与主数据库之间的同步过程。这个表还包含对于故障诊断而言特别有用的信息。如果副本数据库中消息的执行由于出错而停止，那么您可以查询这个表以获取问题原因以及引起错误的事务和语句。

表 161. SYS_SYNC_REPLICA_MSGINFO

列名	描述
状态	<p>消息的当前状态。可能的值如下所示：</p> <ul style="list-style-type: none"> • 0 = DELETED - 不适用（内部非持久状态）。 • 1 = ERROR - 消息处理期间发生内部错误；错误原因已记录在该行的错误列中。 • 20 = R_INIT - 不适用（内部非持久状态）。 • 21 = R_INITEND - 不适用（内部非持久状态）。 • 22 = R_SAVED - 副本数据库已保存外发消息。 • 23 = R_SENT - 副本数据库已将消息发送到主数据库。 • 24 = R_RECEIVED - 副本数据库已从主数据库接收到应答消息。 • 25 = R_EXECUTE - 副本数据库中的应答消息已准备好执行。 • 26 = R_EXECUTE_NOTIFYMASTER - 副本数据库已接收到应答，但尚未向主数据库确认该应答。
MASTER_ID	要将消息发送到的主数据库的标识。
MASTER_NAME	要将消息发送到的主数据库的名称。
MSG_ID	消息的内部标识。
MSG_NAME	用户对消息指定的名称。
MSG_TIME	消息的创建时间。
MSG_BYTE_COUNT	消息的大小（以字节计）。
CREATE_UID	创建消息的用户的标识。
FORWARD_UID	发送消息的用户的标识。
ERROR_CODE	导致消息执行终止的错误的代码。
ERROR_TEXT	导致消息执行终止的错误的描述。
FLAGS	<p>NULL 或 0 = 正常消息。</p> <p>1 = 消息在从主数据库接收到应答时被删除。</p> <p>3 = 消息是注册消息。</p>

E.2.26 SYS_SYNC_REPLICA_RECEIVED_BLOB_REFS

接收到的 BLOB 存储在这个表中。此实现确保仅当以逻辑方式将 BLOB 保存多次后才以物理方式将其保存到磁盘。

表 162. SYS_SYNC_REPLICA_RECEIVED_BLOB_REFS

列名	描述
MASTER_ID	接收到的消息所来自于的主数据库的内部标识。

表 162. SYS_SYNC_REPLICA_RECEIVED_BLOB_REFS (续)

列名	描述
MSG_ID	消息的内部标识。
BLOB_NUM	用于标识 BLOB 的编号。
DATA	对 BLOB 的引用。

E.2.27 SYS_SYNC_REPLICA_RECEIVED_MSGPARTS

这个表包含副本数据库中已从主数据库接收到但尚未在副本数据库中进行处理的应答消息部件。

表 163. SYS_SYNC_REPLICA_RECEIVED_MSGPARTS

列名	描述
MASTER_ID	接收到的消息所来自于的主数据库的内部标识。
MSG_ID	消息的内部标识。
PART_NUMBER	消息部件的序号。
DATA_LENGTH	消息部件中数据的长度。
RESULT_SET_TYPE	结果集的类型。
DATA	消息部件的数据。

E.2.28 SYS_SYNC_REPLICA_RECEIVED_MSGS

这个表包含副本数据库中已从主数据库接收到但尚未在副本数据库中进行处理的应答消息。

表 164. SYS_SYNC_REPLICA_RECEIVED_MSGS

列名	描述
MASTER_ID	接收到的消息所来自于的主数据库的内部标识。
MSG_ID	消息的内部标识。
CREATIME	消息的创建时间。
CREATOR	创建消息的用户的用户标识。

E.2.29 SYS_SYNC_REPLICA_STORED_BLOB_REFS

流消息中的 BLOB 存储在这个表中。此实现确保仅当以逻辑方式将 BLOB 保存多次后才以物理方式将其保存到磁盘。

表 165. SYS_SYNC_REPLICA_STORED_BLOB_REFS

列名	描述
MASTER_ID	接收到的消息所来自于的主数据库的内部标识。
MSG_ID	消息的内部标识。
BLOB_NUM	用于标识 BLOB 的编号。
DATA	对 BLOB 的引用。

E.2.30 SYS_SYNC_REPLICA_STORED_MSGS

这个表包含已在副本数据库中创建但尚未发送到主数据库的消息。

表 166. SYS_SYNC_REPLICA_STORED_MSGS

列名	描述
MASTER_ID	消息将被发送到的主数据库的内部标识。
MSG_ID	消息的内部标识。
CREATIME	消息的创建时间。
CREATOR	创建消息的用户的用户标识。

E.2.31 SYS_SYNC_REPLICA_STORED_MSGPARTS

这个表包含已在副本数据库中创建但尚未发送到主数据库的消息部件。

表 167. SYS_SYNC_REPLICA_STORED_MSGPARTS

列名	描述
MASTER_ID	消息将被发送到的主数据库的内部标识。
MSG_ID	消息的内部标识。
PART_NUMBER	消息部件的序号。
DATA_LENGTH	消息部件中数据的长度。
DATA	消息部件的数据。

E.2.32 SYS_SYNC_REPLICA_VERSIONS

这个表包含主数据库对此副本数据库进行的预订的列表。

表 168. SYS_SYNC_REPLICA_VERSIONS

列名	描述
BOOKMARK_ID	预订中的书签的内部标识。

表 168. SYS_SYNC_REPLICA_VERSIONS (续)

列名	描述
REQUEST_ID	预订中的发布请求的内部标识。
VERS_TIME	预订的创建时间。
PUBL_ID	所预订的发布的标识。
MASTER_ID	从中预订发布的主数据库的标识。
PARAM_CRC	仅供内部使用。
PARAM	预订的参数。
VERSION	所预订的发布在主数据库中的版本号。
LOCAL_VERSION	所预订的发布在副本数据库中的版本号。
PUBL_NAME	发布的名称。
REPLY_ID	发布应答的标识。

E.2.33 SYS_SYNC_REPLICAS

这个表包含向主数据库注册的副本数据库的列表。

表 169. SYS_SYNC_REPLICAS

列名	描述
NAME	对副本数据库指定的名称。
ID	副本数据库的内部标识。
MASTER_NAME	不适用。
MASTER_CATALOG	定义副本数据库注册到的目录。
CONNECT	此列包含副本数据库的连接字符串（例如，“tcp MyWorkstation 1315”）。

E.2.34 SYS_SYNC_SAVED_BLOB_ARGS

如果用户将带有 BLOB 参数的事务保存到副本数据库，那么将在 SYS_SYNC_SAVED_BLOB_ARGS 表中保存对该 BLOB 的引用。此引用指向 SYS_SYNC_REPLICA_STORED_BLOB_REFS 表。这些行仅存在到要发送的消息准备完毕为止。

表 170. SYS_SYNC_SAVED_BLOB_ARGS

列名	描述
MASTER	参数要发送到的主数据库的标识。

表 170. SYS_SYNC_SAVED_BLOB_ARGS (续)

列名	描述
TRX_ID	用于标识事务的事务标识。
ID	用户的内部标识。
ARGNO	参数的编号。
ARG_VALUE	二进制格式的参数值。

E.2.35 SYS_SYNC_SAVED_STMTS

这个表包含已保存到副本数据库以便将来进行传播的语句。

表 171. SYS_SYNC_SAVED_STMTS

列名	描述
MASTER	语句将会传播到的主数据库的内部标识。
TRX_ID	语句所属的事务的内部标识。
ID	语句在事务中的序号。
CLASS	常量的类型。
STRING	字符串形式的 SQL 语句。
ARG_COUNT	与语句绑定的参数的数目。
ARG_TYPES	与语句绑定的参数的类型。
ARG_VALUES	二进制格式的参数值。
USER_ID	保存语句的用户的标识。
REQUEST_ID	不适用。
FLAGS	此列指示错误处理方式（例如，IGNORE_ERRORS 和 LOG_ERRORS 等等）。

E.2.36 SYS_SYNC_TRX_PROPERTIES

保存事务时，可以对它们指定属性。以后，可以通过这些属性来选择要传播的事务。这些属性将保存在 SYS_SYNC_TRX_PROPERTIES 表中。

表 172. SYS_SYNC_TRX_PROPERTIES

列名	描述
TRX_ID	用于标识事务的事务标识。
NAME	事务属性名（例如 COLOR）。

表 172. SYS_SYNC_TRX_PROPERTIES (续)

列名	描述
VALUE_STR	事务属性值（例如 RED）。

E.2.37 SYS_SYNC_USERMAPS

这个表将副本数据库用户标识映射到 SYS_SYNC_USERS 表中的主数据库用户。

表 173. SYS_SYNC_USERMAPS

列名	描述
REPLICA_UID	映射到主数据库用户的副本数据库用户标识。
MASTER_ID	主数据库标识。
REPLICA_USERNAME	副本数据库用户名。
MASTER_USERNAME	主数据库用户名。
PASSW	经过加密的主数据库用户名密码。

E.2.38 SYS_SYNC_USERS

这个表包含有权访问副本数据库的同步功能的用户列表。这些功能包括保存事务以及创建同步消息。

在副本数据库中，这个表的数据是使用以下命令通过一条消息从主数据库下载的：

```
MESSAGE unique-message-name APPEND SYNC_CONFIG
['sync-config-arg']
```

表 174. SYS_SYNC_USERS

列名	描述
MASTER_ID	主数据库的内部标识。
ID	用户的内部标识。
NAME	用户名。
PASSW	经过加密的用户密码。

E.3 系统视图

solidDB 支持 X/Open SQL 标准所指定的视图。

E.3.1 COLUMNS

COLUMNS 系统视图用于标识可供当前用户访问的列。

表 175. COLUMNS

列名	数据类型	描述
TABLE_CATALOG	WVARCHAR	包含 TABLE_NAME 的目录的名称。
TABLE_SCHEMA	WVARCHAR	包含 TABLE_NAME 的模式名称。
TABLE_NAME	WVARCHAR	表或视图的名称。
COLUMN_NAME	WVARCHAR	所指定表或视图的列的名称。
DATA_TYPE	WVARCHAR	列的数据类型。
SQL_DATA_TYPE_NUM	SMALLINT	ODBC 相容数据类型编号。
CHAR_MAX_LENGTH	INTEGER	字符数据类型列的最大长度；对于所有其他列，包含 NULL。
NUMERIC_PRECISION	INTEGER	如果 DATA_TYPE 是适当的数字数据类型，那么包含列的尾数精度的位数，计量单位由 NUMERIC_PREC_RADIX 指示；对于其他数字类型，包含该列中允许的总小数位数；对于字符数据类型，包含 NULL。
NUMERIC_PREC_RADIX	SMALLINT	如果 DATA_TYPE 是其中一种适当的数字数据类型，那么包含数字精度的基；否则包含 NULL。
NUMERIC_SCALE	SMALLINT	小数点右边的总有效位数；对于 INTEGER 和 SMALLINT，包含 0；对于其他数据类型，包含 NULL。
NULLABLE	CHAR	如果已知该列不可为空，那么包含“NO”；否则包含“YES”。
NULLABLE_ODBC	SMALLINT	(ODBC) 如果已知该列不可为空，那么包含“0”；否则包含“1”。
REMARKS	LONG WVARCHAR	保留供将来使用。

E.3.2 SERVER_INFO

SERVER_INFO 系统视图提供当前数据库系统或服务器的属性。

表 176. SERVER_INFO

列名	数据类型	描述
SERVER_ATTRIBUTE	WVARCHAR	标识服务器的属性。
ATTRIBUTE_VALUE	WVARCHAR	该属性的值。

E.3.3 TABLES

TABLES 系统视图用于标识与当前用户相关联的表。

表 177. TABLES

列名	数据类型	描述
TABLE_CATALOG	WVARCHAR	包含 TABLE_NAME 的目录的名称。
TABLE_SCHEMA	WVARCHAR	包含 TABLE_NAME 的模式名称。
TABLE_NAME	WVARCHAR	表或视图的名称。
TABLE_TYPE	WVARCHAR	表的类型。
REMARKS	LONG WVARCHAR	保留供将来使用。

E.3.4 USERS

USERS 系统视图用于标识用户和角色。

表 178. USERS

列名	数据类型	描述
ID	INTEGER	用户或角色的标识。
NAME	WVARCHAR	用户或角色名称。
TYPE	WVARCHAR	用户类型，即 USER 或 ROLE。
PRIV	INTEGER	特权信息。
PRIORITY	INTEGER	保留供将来使用。
PRIVATE	INTEGER	指定该用户是专用用户还是公用用户。

E.4 与同步相关的视图

solidDB 提供了 4 个视图来显示关于主数据库与副本数据库之间的同步消息的信息。其中一对视图（SYNC_FAILED_MESSAGES 和 SYNC_FAILED_MASTER_MESSAGES）显示失败的消息。另一对视图（SYNC_ACTIVE_MESSAGES 和 SYNC_ACTIVE_MASTER_MESSAGES）显示活动消息。

E.4.1 SYNC_FAILED_MESSAGES

这个表在主数据库中，它包含关于从副本数据库接收到的消息的信息。您可以使用一个简单的视图来查看有关失败的消息的所有必需信息：

```
SELECT * FROM SYNC_FAILED_MESSAGES.
```

这将返回下列各列：

表 179. SYNC_FAILED_MESSAGES

列名	数据类型	描述
REPLICA_NAME	WVARCHAR	从中发送消息的副本数据库的给定节点名。
MESSAGE_NAME	WVARCHAR	用户对消息指定的名称。
TRANSACTION_ID	BINARY	失败的副本数据库事务的内部标识。
STATEMENT_ID	INTEGER	语句在事务中的序号。
STATEMENT_STRING	WVARCHAR	字符串形式的 SQL 语句。
ERROR_CODE	INTEGER	导致消息执行终止的错误的代码。
ERROR_MESSAGE	VARCHAR	有关错误的描述。

所有用户都有权访问此视图；不需要特定的特权。

E.4.2 SYNC_FAILED_MASTER_MESSAGES

这个表在副本数据库中，它包含关于发送到主数据库的消息的信息。您可以使用一个简单的视图来查看有关失败的消息的所有必需信息：

```
SELECT * FROM SYNC_FAILED_MASTER_MESSAGES.
```

这将返回下列各列：

表 180. SYNC_FAILED_MASTER_MESSAGES

列名	数据类型	描述
MASTER_NAME	WVARCHAR	主数据库的给定节点名。
MESSAGE_NAME	WVARCHAR	用户对消息指定的名称。
ERROR_CODE	INTEGER	导致消息执行终止的错误的代码。
ERROR_MESSAGE	VARCHAR	有关错误的描述。

所有用户都有权访问此视图；不需要特定的特权。

E.4.3 SYNC_ACTIVE_MESSAGES

这个表在主数据库中，它包含关于从副本数据库接收到的消息的信息。这将返回下列各列：

表 181. SYNC_ACTIVE_MESSAGES

列名	数据类型	描述
REPLICA_NAME	WVARCHAR	副本数据库的给定节点名。
MESSAGE_NAME	WVARCHAR	用户对消息指定的名称。

表 181. SYNC_ACTIVE_MESSAGES (续)

列名	数据类型	描述
MESSAGE STATE	VARCHAR	字符串形式的消息的当前状态。请参阅系统表 SYS_SYNC_MASTER_MSGINFO 中的详细信息。

所有用户都有权访问此视图；不需要特定的特权。

E.4.4 SYNC_ACTIVE_MASTER_MESSAGES

这个表在副本数据库中，它包含关于发送到主数据库的消息的信息。您可以使用一个简单的视图来查看有关失败的消息的所有必需信息：

```
SELECT * FROM SYNC_FAILED_MASTER_MESSAGES.
```

这将返回下列各列：

表 182. SYNC_ACTIVE_MASTER_MESSAGES

列名	数据类型	描述
MASTER_NAME	WVARCHAR	主数据库的给定节点名。
MESSAGE_NAME	WVARCHAR	用户对消息指定的名称。
MESSAGE STATE	VARCHAR	字符串形式的消息的当前状态。请参阅系统表 SYS_SYNC_REPLICA_MSGINFO 中的详细信息。

所有用户都有权访问此视图；不需要特定的特权。

附录 F. 数据库虚拟表

F.1 SYS_LOG

SYS_LOG 表是一个虚拟表：当“日志阅读器”接收到针对 SYS_LOG 表的 SQL 请求时，将会从内部日志结构动态生成相应的结果集。

表 183. SYS_LOG 表定义

列名	数据类型	描述
RECID	INTEGER	记录标识 有关详细描述，请参阅第 360 页的『记录标识描述』一节。
RECNAME	VARCHAR	日志记录的描述性名称。这是 RECID 字段的更符合人们阅读习惯的格式。
TRXID	INTEGER	事务标识
STMTTRXID	INTEGER	事务中的语句标识
RELID	INTEGER	表标识，与 SYS_TABLES 表中存储的一样。
FLAGS	INTEGER	标志字段，此字段有下列含义： <ul style="list-style-type: none">• 0 NOP 无数据可用，在此情况下，所有其他列值均为 NULL。• 位 0 设置 DATA 数据可用。• 位 1 设置 SHUTDOWN 服务器关闭已开始。• 位 6 设置 CAPTURE_OFF 如果在日志记录中设置为 DBE_LOGREADER_LOG_REC_TRX_START，那么不应传播此事务中的操作。
LOGADDR	VARBINARY	这是当前日志记录的日志地址。 LOGADDR 可用作从 SYS_LOG 读取的起始点。 日志地址的长度是 20 字节。 可以使用二进制比较来对日志地址进行比较，以查看哪个日志地址在前面。
DATA	VARBINARY	表中的实际用户数据。 有关详细说明，请参阅下面的第 360 页的『行数据的 DATA 列格式』和第 361 页的『DDL 数据的 DATA 列格式』章节。
TEXTDATA	LONG VARCHAR	未使用，始终为 NULL。（用作 DATA 的用户可阅读文本格式。）

记录标识描述

表 184. 记录标识描述

RECNAME	RECID	描述
DBE_LOGREADER_LOG_REC_EMPTY	0	空记录，无数据可用。
DBE_LOGREADER_LOG_REC_INSERT	1	行插入。
DBE_LOGREADER_LOG_REC_UPDATE	3	后映像更新。如果主键已更改，那么不会将它显示为更新。而会将它显示为插入和更新。
DBE_LOGREADER_LOG_REC_DELETE_FULL	4	行删除，同时所有列值存储到日志中。
DBE_LOGREADER_LOG_REC_UPDATE_BEFOREIMAGE	5	更新的前映像。如果主键已更改，那么不会将它显示为更新。而会将它显示为插入和更新。
DBE_LOGREADER_LOG_REC_TRX_START	7	事务开始。请参阅 FLAGS 字段以了解如何检测未传播的删除。
DBE_LOGREADER_LOG_REC_COMMIT	12	事务落实。
DBE_LOGREADER_LOG_REC_DDL	13	DDL 操作。
DBE_LOGREADER_LOG_REC_SQL	6	DDL 操作的 SQL 字符串。

行数据的 DATA 列格式

本节描述 RECNAME 字段值如下的行的 DATA 列格式：

- DBE_LOGREADER_LOG_REC_INSERT
- DBE_LOGREADER_LOG_REC_UPDATE
- DBE_LOGREADER_LOG_REC_UPDATE_BEFOREIMAGE
- DBE_LOGREADER_LOG_REC_DELETE_FULL

格式化 DATA 列，以便对应于表中的每个用户列，该 DATA 列包含二进制格式的 <length><data> 对。列的排序顺序为在表中定义这些列的顺序。

整数值的格式也称为“从最有意义的字节（MSB）开始”格式。举例来说，这意味着四字节十六进制整数 0x12345678 中的字节将存储为字节 12 34 56 78。

在“从 MSB 开始”格式中，<length> 字段始终是四字节整数。

NULL 值存储成 <length> 字段的 -1。

在“从 MSB 开始”格式中，BLOB 值存储成 <length> 字段为 -2，后跟 <data> 字段中的 8 字节整数。

这 8 字节整数是唯一的 BLOB 标识。每当存储新的 BLOB 值或更改旧值时，都会为它分配唯一的 BLOB 标识。

表 185. 行数据的 DATA 类型描述

Name	描述	SQL 数据类型
BINARY UNICODE CHAR	存储为原始数据字节，与存储到列中的格式相同。	CHAR VARCHAR LONG VARCHAR WCHAR WVARCHAR LONG WVARCHAR BINARY VARBINARY LONG VARBINARY
DOUBLE FLOAT	采用“从 MSB 开始”格式存储为 8 字节 IEEE 浮点数。	FLOAT REAL DOUBLE PRECISION
INTEGER	采用“从 MSB 开始”格式存储为 4 字节整数。	BIT TINYINT
BIGINT	采用“从 MSB 开始”格式存储为 8 字节整数。	BIGINT
DATE	存储为 11 字节原始数据格式。各字节有下列含义： <ul style="list-style-type: none"> • 0-1 年，2 字节，采用“从 MSB 开始”格式 • 2 月，1 字节 • 3 日，1 字节 • 4 时，1 字节 • 5 分，1 字节 • 6 秒，1 字节 • 7 微秒，4 字节，采用“从 MSB 开始”格式 	DATE TIME TIMESTAMP
DFLOAT	字符串格式的十进制浮点。	NUMERIC DECIMAL

DDL 数据的 DATA 列格式

对于 RECNAME 字段含有值 DBE_LOGREADER_LOG_REC_DDL 的行，DATA 列格式如下所示：

- 格式化 DATA 列，以便它包含 <length><logrecid><length><info> 值。

对于 RECNAME 字段含有值 DBE_LOGREADER_LOG_REC_SQL 的行，DATA 列格式如下所示：

- 格式化 DATA 列，以便它包含 <length><info> 值。<info> 字段的值为 SQL 字符串。

整数值的格式也称为“从最有意义的字节（MSB）开始”格式。这意味着四字节十六进制整数 0x12345678 中的字节将存储为字节 12 34 56 78。

在“从 MSB 开始”格式中，<length> 字段始终是四字节整数。

下表描述了可能的 <logrecid> 字段描述和 <info> 值。

表 186. DDL 数据的 DATA 类型描述

LOGRECID	描述	INFO
45	创建表。	标准表名
17	删除表。	表名。
47	重命名表。	标准表名。
22	改变表	表名。
73	截断表。	表名。
16	创建索引。	索引名。
18	删除索引。	索引名。
46	创建视图。	标准视图名。
20	删除视图。	视图名。
28	创建序列。	序列名。
30	删除序列。	序列名。
27	创建计数器。	计数器名。
29	删除计数器。	计数器名。

附录 G. 系统存储过程

系统存储过程是指由 solidDB 提供的、用于简化任务（例如管理同步）的存储过程。

G.1 与同步相关的存储过程

同步关联存储过程可以用于简化例程同步任务。要执行同步系统过程，您必须具有管理员或同步管理员访问权。

G.1.1 SYNC_SETUP_CATALOG

SYNC_SETUP_CATALOG 系统存储过程将创建一个目录，对其指定节点名，并将该目录的角色设置为主控目录和/或副本目录。

```
CALL SYNC_SETUP_CATALOG (  
    catalog_name,  -- WVARCHAR  
    node_name,    -- WVARCHAR  
    is_master,    -- INTEGER  
    is_replica    -- INTEGER  
)
```

执行位置：主数据库或副本数据库。

您必须有管理员或同步管理员访问权限来执行该存储过程。

如果 *catalog_name* 参数为 NULL，那么将对当前目录指定所给定的角色名和角色。

对于 *is_master* 和 *is_replica*，值 0 表示“No”，任何其他值都表示“Yes”。在这两个参数中，至少一个参数应该具有非零值。由于单一目录可以同时作为副本目录和主控目录，因此可以同时将 *is_master* 和 *is_replica* 设置为非零值。

表 187. SYNC_SETUP_CATALOG 错误代码

原因码	文本	描述
13047	无权执行操作。	
13110	不允许 NULL 值。	只有目录名才能为 NULL；所有其他参数都必须是非 NULL 值。
13133	不是此产品的有效许可证。	
25031	事务处于活动状态，操作失败。	用户已进行一些更改，但尚未落实那些更改。
25052	未能将节点名设置为 <i>node_name</i> 。	<i>node_name</i> 无效。
25059	执行注册后，不能更改节点名。	目录已具有名称，并且有一个或多个副本数据库。

G.1.2 SYNC_REGISTER_REPLICA

SYNC_REGISTER_REPLICA 系统存储过程将创建一个新目录并向指定的主数据库注册该副本数据库。

```
CALL SYNC_REGISTER_REPLICA (  
    replica_node_name,    -- WVARCHAR  
    replica_catalog_name, -- WVARCHAR  
    master_network_name,  -- WVARCHAR  
    master_node_name,     -- WVARCHAR  
    user_id,              -- WVARCHAR  
    password               -- WVARCHAR  
)
```

执行位置：副本数据库。

您必须有管理员或同步管理员访问权限来执行该存储过程。

master_network_name 是主数据库服务器的连接字符串。

如果指定的目录不存在，将自动创建该目录。

如果该 *replica_catalog_name* 是 NULL，那么使用当前目录。并且，*master_node_name* 可以是 NULL。所有其他参数都不能为 NULL。

如果注册失败，那么主控端和副本端都将复位到原始状态。如果任何参数的值不合法，那么将返回错误。

如果存在任何包含已修改的数据的已打开事务，将返回错误。

此系统过程不返回结果集。

表 188. SYNC_REGISTER_REPLICA 错误代码

原因码	文本	描述
13047	无权执行操作。	
13110	不允许 NULL 值。	只有目录名和主数据库节点名才能为 NULL；所有其他参数都必须是非 NULL 值。
13133	不是此产品的有效许可证。	
21xxx	通信错误	无法连接到主数据库。有关 21xxx 错误的更多详细信息，请参阅《IBM solidDB 管理员指南》中的 solidDB 通信错误。
25005	消息已处于活动状态。	
25031	事务处于活动状态，操作失败。	用户已进行一些更改，但尚未落实那些更改。
25035	消息在使用中。	
25051	找到未完成的消息。	
25052	未能将节点名设置为 <i>node_name</i> 。	<i>node_name</i> 无效。

表 188. SYNC_REGISTER_REPLICA 错误代码 (续)

原因码	文本	描述
25056	不允许自动落实。	必须在自动落实功能处于关闭状态的情况下运行此存储过程。
25057	副本数据库已向主数据库注册。	
25059	执行注册后，不能更改节点名。	

G.1.3 SYNC_UNREGISTER_REPLICA

SYNC_UNREGISTER_REPLICA 系统存储过程从主数据库中注销指定的副本目录，并可以选择删除副本目录。

```
CALL SYNC_UNREGISTER_REPLICA (
    replica_catalog_name,  -- WVARCHAR
    drop_catalog,         -- INTEGER
    force                  -- INTEGER
)
```

执行位置：副本数据库。

您必须有管理员或同步管理员访问权限来执行该存储过程。

如果 *drop_catalog* 参数具有非零值，那么删除该副本目录。在系统的两端，将删除此副本数据库的任何处于挂起状态的消息。

如果该 *replica_catalog_name* 是 NULL，那么使用当前目录。

如果 *force* 是非零值，那么主数据库将接受注销，即使主数据库中存在此副本数据库的消息亦如此。在那种情况下，将删除那些消息。

如果用户有任何未落实的更改（即，打开的事务），那么此调用将失败并返回错误。

此系统过程不返回结果集。

表 189. SYNC_UNREGISTER_REPLICA 错误代码

原因码	文本	描述
13047	无权执行操作。	
13110	不允许 NULL 值。	如果 <i>drop_catalog</i> 为非零值，那么目录名不能为 NULL 值。
13133	不是此产品的有效许可证。	
21xxx	通信错误	无法连接到主数据库。有关 21xxx 错误的更多详细信息，请参阅《IBM solidDB 管理员指南》中的 <i>solidDB</i> 通信错误。
25005	消息已处于活动状态。	
25019	此数据库不是副本数据库。	

表 189. SYNC_UNREGISTER_REPLICA 错误代码 (续)

原因码	文本	描述
25020	此数据库不是主数据库。	
25023	未注册副本数据库。	
25031	事务处于活动状态，操作失败。	用户已进行一些更改，但尚未落实那些更改。
25035	消息在使用中。	
25051	找到未完成的消息。	
25056	不允许自动落实。	必须在自动落实功能处于关闭状态的情况下运行此存储过程。
25079	目录是主数据库并存在已注册的副本数据库。未删除目录。	
25093	该副本数据库的主数据库中存在失败操作。	用户试图删除注册到主数据库中的副本数据库目录时，返回该消息。

G.1.4 SYNC_REGISTER_PUBLICATION

SYNC_REGISTER_PUBLICATION 系统存储过程注册来自主数据库的发布。

```
CALL SYNC_REGISTER_PUBLICATION (
    replica_catalog_name, -- WVARCHAR
    publication_name      -- WVARCHAR
)
```

执行位置：副本数据库。

您必须有管理员或同步管理员访问权限来执行该存储过程。

如果 *replica_catalog_name* 是 NULL，那么使用当前目录。

如果有未落实的更改，此调用将失败并返回错误。

此系统过程不返回结果集。

表 190. SYNC_REGISTER_PUBLICATION 错误代码

原因码	文本	描述
13047	无权执行操作。	
13110	不允许 NULL 值。	只有目录名才能为 NULL；所有其他参数都必须是非 NULL 值。
13133	不是此产品的有效许可证。	
21xxx	通信错误	无法连接到主数据库。有关 21xxx 错误的更多详细信息，请参阅《IBM solidDB 管理员指南》中的 <i>solidDB</i> 通信错误。
25005	消息已处于活动状态。	

表 190. SYNC_REGISTER_PUBLICATION 错误代码 (续)

原因码	文本	描述
25010	找不到发布。	
25019	此数据库不是副本数据库。	
25020	此数据库不是主数据库。	
25023	未注册副本数据库。	
25035	消息在使用中。	
25056	不允许自动落实。	必须在自动落实功能处于关闭状态的情况下运行此存储过程。
25072	已向发布注册。	

G.1.5 SYNC_UNREGISTER_PUBLICATION

SYNC_UNREGISTER_PUBLICATION 系统存储过程用于注销发布。

```
CALL SYNC_UNREGISTER_PUBLICATION (
    replica_catalog_name, -- WVARCHAR
    publication_name,    -- WVARCHAR
    drop_data            -- INTEGER
)
```

执行位置: 副本数据库。

如果 *drop_data* 标志设置为非零值, 将自动删除所有对该发布的预订。

如果 *replica_catalog_name* 是 NULL, 那么使用当前目录。

如果有未落实的更改, 此调用将失败并返回错误。

此系统过程不返回结果集。

表 191. SYNC_UNREGISTER_PUBLICATION 错误代码

原因码	文本	描述
13047	无权执行操作。	
13110	不允许 NULL 值。	只有目录名才能为 NULL; 所有其他参数都必须是非 NULL 值。
13133	不是此产品的有效许可证。	
21xxx	通信错误	无法连接到主数据库。有关 21xxx 错误的更多详细信息, 请参阅《IBM solidDB 管理员指南》中的 <i>solidDB</i> 通信错误。
25005	消息已处于活动状态。	
25010	找不到发布。	

表 191. SYNC_UNREGISTER_PUBLICATION 错误代码 (续)

原因码	文本	描述
25019	此数据库不是副本数据库。	
25020	此数据库不是主数据库。	
25023	未注册副本数据库。	
25031	事务处于活动状态，操作失败。	用户已进行一些更改，但尚未落实那些更改。
25035	消息在使用中。	
25056	不允许自动落实。	必须在自动落实功能处于关闭状态的情况下运行此存储过程。
25071	尚未向发布注册。	

G.1.6 SYNC_SHOW_SUBSCRIPTIONS

SYNC_SHOW_SUBSCRIPTIONS 系统存储过程显示主数据库中某个发布的哪些预订（字符串形式的发布名称和参数）处于活动状态。

```
CREATE PROCEDURE SYNC_SHOW_SUBSCRIPTIONS (
  publication_name  -- WVARCHAR
)
```

执行位置：副本数据库。

您必须有管理员或同步管理员访问权限来执行该存储过程。

提示： 主控目录和副本目录提供相同功能。在副本目录中使用 SYNC_SHOW_SUBSCRIPTIONS，在主目录中使用 SYNC_SHOW_REPLICA_SUBSCRIPTIONS。

此过程调用的结果集是：

表 192. CREATE PROCEDURE SYNC_SHOW_SUBSCRIPTIONS 结果集

列名	数据类型	描述
SUBSCRIPTION	WVARCHAR	字符串形式的发布名和参数。
SUBSCRIPTION_TIME	TIMESTAMP	上次预订时间。

表 193. SYNC_SHOW_SUBSCRIPTIONS 错误代码

原因码	文本	描述
13047	无权执行操作。	
13133	不是此产品的有效许可证。	
25009	找不到副本数据库。	

表 193. SYNC_SHOW_SUBSCRIPTIONS 错误代码 (续)

原因码	文本	描述
25010	找不到发布。	
25019	此数据库不是副本数据库。	
25020	此数据库不是主数据库。	
25023	未注册副本数据库。	
25071	尚未向发布注册。	

相关参考

G.1.7, 『 SYNC_SHOW_REPLICA_SUBSCRIPTIONS 』

SYNC_SHOW_REPLICA_SUBSCRIPTIONS 系统存储过程显示指定的副本数据库中某个发布的哪些预订（字符串形式的发布名称和参数）处于活动状态。

G.1.7 SYNC_SHOW_REPLICA_SUBSCRIPTIONS

SYNC_SHOW_REPLICA_SUBSCRIPTIONS 系统存储过程显示指定的副本数据库中某个发布的哪些预订（字符串形式的发布名称和参数）处于活动状态。

```
CREATE PROCEDURE SYNC_SHOW_REPLICA_SUBSCRIPTIONS (
    replica_name,          -- WVARCHAR
    publication_name      -- WVARCHAR
)
```

执行位置：主数据库。

您必须有管理员或同步管理员访问权限来执行该存储过程。

提示： 主控目录和副本目录提供相同功能。在副本目录中使用 SYNC_SHOW_SUBSCRIPTIONS，在主目录中使用 SYNC_SHOW_REPLICA_SUBSCRIPTIONS。

如果发布名为 NULL，那么将列示对所有发布的预订。

此过程调用的结果集是：

表 194. SYNC_SHOW_REPLICA_SUBSCRIPTIONS 结果集

列名	数据类型	描述
REPLICA_NAME	WVARCHAR	副本数据库名称。
SUBSCRIPTION	WVARCHAR	字符串形式的发布名和参数。
SUBSCRIPTION_TIME	TIMESTAMP	上次预订时间。

表 195. SYNC_SHOW_REPLICA_SUBSCRIPTIONS 错误代码

原因码	文本	描述
13047	无权执行操作。	

表 195. SYNC_SHOW_REPLICA_SUBSCRIPTIONS 错误代码 (续)

原因码	文本	描述
13133	不是此产品的有效许可证。	
25009	找不到副本数据库。	
25010	找不到发布。	
25019	此数据库不是副本数据库。	
25020	此数据库不是主数据库。	
25023	未注册副本数据库。	
25071	尚未向发布注册。	

相关参考

第 368 页的 G.1.6, 『 SYNC_SHOW_SUBSCRIPTIONS 』

SYNC_SHOW_SUBSCRIPTIONS 系统存储过程显示主数据库中某个发布的哪些预订 (字符串形式的发布名称和参数) 处于活动状态。

G.1.8 SYNC_DELETE_MESSAGES

SYNC_DELETE_MESSAGES 系统存储过程删除副本数据库中的挂起的消息。如果副本应用程序创建了消息并且未正确地检查和处理消息, 那么将有若干挂起的消息。

```
CALL SYNC_DELETE_MESSAGES (
    replica_catalog_name, -- WVARCHAR
)
```

执行位置: 副本数据库。

您必须有管理员或同步管理员访问权限来执行该存储过程。

如果 *replica_catalog_name* 是 NULL, 那么使用当前目录。

此过程不返回结果集。

表 196. SYNC_DELETE_MESSAGES 错误代码

原因码	文本	描述
13047	无权执行操作。	
13133	不是此产品的有效许可证。	
25005	消息已处于活动状态。	
25009	找不到副本数据库。	
25019	此数据库不是副本数据库。	
25020	此数据库不是主数据库。	
25035	消息在使用中。	

相关参考

G.1.9, 『 SYNC_DELETE_REPLICA_MESSAGES 』

SYNC_DELETE_REPLICA_MESSAGES 系统存储过程用于从主数据库中删除所指定副本数据库的消息。如果副本应用程序创建了消息并且未正确地检查和处理消息，那么将有若干挂起的消息。

G.1.9 SYNC_DELETE_REPLICA_MESSAGES

SYNC_DELETE_REPLICA_MESSAGES 系统存储过程用于从主数据库中删除所指定副本数据库的消息。如果副本应用程序创建了消息并且未正确地检查和处理消息，那么将有若干挂起的消息。

```
CALL SYNC_DELETE_REPLICA_MESSAGES(  
    master_catalog_name -- WVARCHAR,  
    replica_name        -- WVARCHAR  
)
```

执行位置: 主数据库。

您必须有管理员或同步管理员访问权限来执行该存储过程。

master_catalog_name 参数指定主数据库中的目录，将在该目录中搜索所指定副本数据库的消息。如果 *master_catalog_name* 设置为 NULL，那么使用当前目录。

此过程不返回结果集。

表 197. SYNC_DELETE_REPLICA_MESSAGES 错误代码

原因码	文本	描述
13047	无权执行操作。	
13133	不是此产品的有效许可证。	
25005	消息已处于活动状态。	
25009	找不到副本数据库。	
25019	此数据库不是副本数据库。	
25020	此数据库不是主数据库。	
25035	消息在使用中。	

相关参考

第 370 页的 G.1.8, 『 SYNC_DELETE_MESSAGES 』

SYNC_DELETE_MESSAGES 系统存储过程删除副本数据库中的挂起的消息。如果副本应用程序创建了消息并且未正确地检查和处理消息, 那么将有若干挂起的消息。

G.2 其他存储过程

G.2.1 SYS_GETBACKGROUNDJOB_INFO

`SYS_GETBACKGROUNDJOB_INFO` 系统存储过程检索来自 `SYS_BACKGROUNDJOB_INFO` 表的信息。

```
CREATE PROCEDURE SYS_GETBACKGROUNDJOB_INFO(  
    jobid INTEGER)  
RETURNS(  
    ID INTEGER,  
    STMT WVARCHAR,  
    USER_ID INTEGER,  
    ERROR_CODE INTEGER,  
    ERROR_TEXT INTEGER)
```

`SYS_GETBACKGROUNDJOB_INFO` 过程会返回与给定的 *jobid* 相匹配的行。*jobid* 是已执行的 `START AFTER COMMIT` 语句的作业标识。执行 `START AFTER COMMIT` 语句时, 服务器将返回此作业标识。

附录 H. 系统事件

本节描述 solidDB 系统事件。这些事件随 solidDB 提供，旨在允许程序在某些操作发生时接收到通知。您可以使用这些事件来监视活动（例如主数据库与副本数据库之间的同步）的进度。

系统事件与用户定义的事件所遵循的规则大体相同。

这些是预定义的事件，您不能创建这些事件。此外，您不可以发布系统事件；只可以等待系统事件。

大部分系统事件有 5 个相同参数：

- **ename**: 事件名称。
- **postsrvtime**: 服务器发出事件时的时间。
- **uid**: 用户标识（如果适用）。
- **numdatainfo**: 其他数字数据 - 确切含义取决于事件。例如，当备份启动和备份完成时，都将发出事件 `SYS_EVENT_BACKUP`。numdatainfo 参数中的值指示情况 - 即，备份是刚刚启动还是刚刚完成。如果没有任何数字数据，那么此参数可能为 `NULL`。
- **textdata**: 其他文本数据 - 确切含义取决于事件。如果没有任何数字数据，那么此参数可能为 `NULL`。

注：

- 有关与 HotStandby 相关的事件的描述，请参阅《*IBM solidDB 高可用性用户指南*》。
- 有关与高级复制功能相关的事件的描述，请参阅 *IBM solidDB Advanced Replication User Guide*。

H.1 其他事件

其他事件与服务器的内部调度和内务处理密切相关，例如备份、检查点及合并。

尽管您不可能发出这些事件，但是您可能间接引起这些事件，例如，请求备份或在高级复制设置上打开“维护方式”就会导致其发生。如有必要，您可以监视这些事件。

表 198. 其他事件

EVENT NAME	EVENT DESCRIPTION	PARAMETERS
SYS_EVENT_BACKUP	系统已启动或完成了备份操作。“state”参数（NUMDATAINFO）指示： 0: 备份已完成。 1: 备份已启动。 注意，当服务器启动或完成备份之后，它还将发出第二个事件（SYS_EVENT_MESSAGES）。	ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 TEXTDATA WVARCHAR

表 198. 其他事件 (续)

EVENT NAME	EVENT DESCRIPTION	PARAMETERS
SYS_EVENT_BACKUPREQ	<p>已请求备份操作（但仍未启动）。</p> <p>如果用户应用程序回调函数返回非零值，则不执行备份。</p> <p>仅当用户使用共享内存访问或链接库访问时，用户才能捕获到此事件。</p> <p>不使用参数。</p>	<p>ENAME WVARCHAR、</p> <p>POSTSRVTIME TIMESTAMP、</p> <p>UID INTEGER、</p> <p>NUMDATAINFO INTEGER 和</p> <p>TEXTDATA WVARCHAR</p>
SYS_EVENT_CHECKPOINT	<p>系统已启动或完成了检查点操作。</p> <p>如果系统启动了检查点，那么“state”参数（NUMDATAINFO）是 1，而 message（TEXTDATA）参数是“started”。</p> <p>如果系统完成检查点那么“state”参数（NUMDATAINFO）为 0，并且 message（TEXTDATA）参数为“completed”。</p>	<p>ENAME WVARCHAR、</p> <p>POSTSRVTIME TIMESTAMP、</p> <p>UID INTEGER、</p> <p>NUMDATAINFO INTEGER 和</p> <p>TEXTDATA WVARCHAR</p>
SYS_EVENT_CHECKPOINTREQ	<p>已启动执行但尚未启动检查点操作。每当完成一些日志编写时通常将执行检查点。</p> <p>如果用户应用程序回调函数返回非零值，那么将不执行合并。</p> <p>仅当用户使用共享内存访问或链接库访问时，用户才能捕获到此事件。</p> <p>不使用参数。</p>	<p>ENAME WVARCHAR、</p> <p>POSTSRVTIME TIMESTAMP、</p> <p>UID INTEGER、</p> <p>NUMDATAINFO INTEGER 和</p> <p>TEXTDATA WVARCHAR</p>
SYS_EVENT_ERROR	<p>发生某类服务器错误。message 参数（TEXTDATA）包含错误文本。有关可能导致发出此事件的服务器错误列表，请参阅第 379 页的 H.2，『导致 SYS_EVENT_ERROR 的错误』。</p>	<p>ENAME WVARCHAR、</p> <p>POSTSRVTIME TIMESTAMP、</p> <p>UID INTEGER、</p> <p>NUMDATAINFO INTEGER 和</p> <p>TEXTDATA WVARCHAR</p>
SYS_EVENT_IDLE	<p>系统空闲。</p> <p>一些任务具有“空闲”优先级，仅当系统没有运行任何其他任务时才运行它们。因为优先级极低的任务可能在“空闲”系统中运行，所以系统不一定处于不执行任何操作的真正空闲状态。</p> <p>仅当用户使用共享内存访问或链接库访问时，用户才能捕获到此事件。</p> <p>不使用参数。</p>	<p>ENAME WVARCHAR、</p> <p>POSTSRVTIME TIMESTAMP、</p> <p>UID INTEGER、</p> <p>NUMDATAINFO INTEGER 和</p> <p>TEXTDATA WVARCHAR</p>

表 198. 其他事件 (续)

EVENT NAME	EVENT DESCRIPTION	PARAMETERS
SYS_EVENT_IMDB_MEMORY	<p>系统遇到与内存数据库内存限制相关的事件。</p> <p>NUMDATAINFO 参数指示当前的内存分配，以千字节为单位。</p> <p>TEXTDATA 参数可以具有以下各值：</p> <ul style="list-style-type: none"> • IMDB_LIMIT_ABOVE - 可用虚拟内存的量大于使用 MME.ImdbMemoryLimit 参数指定的限制 • IMDB_LIMIT_BELOW - 可用虚拟内存的量低于使用 MME.ImdbMemoryLimit 参数指定的限制 • IMDB_LOW_LEVEL_ABOVE - 可用虚拟内存的量大于使用 MME.ImdbMemoryLowPercentage 参数指定的限制 • IMDB_LOW_LEVEL_BELOW - 可用虚拟内存的量低于使用 MME.ImdbMemoryLowPercentage 参数指定的限制 • IMDB_WARNING_LEVEL_ABOVE - 可用虚拟内存的量大于使用 MME.ImdbMemoryLowPercentage 参数指定的限制 • IMDB_WARNING_LEVEL_BELOW - 可用虚拟内存的量低于使用 MME.ImdbMemoryLowPercentage 参数指定的限制 	<p>ENAME WVARCHAR,</p> <p>POSTSRVTIME TIMESTAMP,</p> <p>UID INTEGER,</p> <p>NUMDATAINFO INTEGER 和</p> <p>TEXTDATA WVARCHAR</p>
SYS_EVENT_ILL_LOGIN	<p>尝试进行非法的登录。用户名 (TEXTDATA) 和用户标识 (NUMDATAINFO) 指示尝试登录的用户。</p>	<p>ENAME WVARCHAR,</p> <p>POSTSRVTIME TIMESTAMP,</p> <p>UID INTEGER,</p> <p>NUMDATAINFO INTEGER 和</p> <p>TEXTDATA WVARCHAR</p>
SYNC_MAINTENANCEMODE_BEGIN	<p>当同步方式从 NORMAL 更改为 MAINTENANCE 时，服务器将发送此系统事件。node_name 是启动维护方式的节点的名称。</p> <p>单个 solidDB 服务器可以有多个“节点” (目录)。有关同步方式的更多详细信息，请参阅第 265 页的『SET SYNC MODE』。</p>	<p>node_name WVARCHAR,</p>

表 198. 其他事件 (续)

EVENT NAME	EVENT DESCRIPTION	PARAMETERS
SYNC_MAINTENANCEMODE_END	<p>当同步方式由 MAINTENANCE 更改为 NORMAL 时，服务器将发送此系统事件。node_name 是启动维护方式的节点的名称。</p> <p>单个 solidDB 服务器可以有多个“节点”（目录）。有关同步方式的更多详细信息，请参阅第 265 页的『SET SYNC MODE』。</p>	node_name WVARCHAR
SYS_EVENT_MERGE	<p>发生与“合并”操作（将数据从 Bonsai Tree 合并到主存储器树）相关联的事件。参数 STATE (NUMDATAINFO) 给出了更多的详细信息：</p> <p>0: 停止合并</p> <p>1: 启动合并</p> <p>2: 正在进行合并</p> <p>3: 加速合并。</p>	ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 TEXTDATA WVARCHAR
SYS_EVENT_MERGREQ	<p>已启动执行但尚未启动合并操作。</p> <p>如果用户应用程序回调函数返回非零值，则将不执行合并。</p> <p>仅当用户使用共享内存访问或链接库访问时，用户才能捕获到此事件。</p> <p>不使用参数。</p>	ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 TEXTDATA WVARCHAR
SYS_EVENT_MESSAGES	<p>当服务器收到登录 solerror.out 或 solmsg.out 的消息（错误消息或警告消息）时，将发出此事件。在这种情况下，TEXTDATA 包含消息文本和 NUMDATAINFO 代码。如果要写入的消息是一个错误，将同时发出 SYS_EVENT_ERROR 和 SYS_EVENT_MESSAGES。如果消息仅是警告，只发出 SYS_EVENT_MESSAGES。有关可能导致 SYS_EVENT_MESSAGES 的警告的列表，请参阅第 380 页的 H.3，『导致 SYS_EVENT_MESSAGES 的条件或警告』。</p>	ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 MESSAGE WVARCHAR
SYS_EVENT_NOTIFY	<p>用 admin command 'notify' 发送事件。</p>	ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 TEXTDATA WVARCHAR

表 198. 其他事件 (续)

EVENT NAME	EVENT DESCRIPTION	PARAMETERS
SYS_EVENT_PARAMETER	<p>如果用 ADMIN COMMAND 'parameter...' 命令更改配置参数, 则会发布该事件;</p> <p>参数 MESSAGE (TEXTDATA) 包含该段的段名和参数名。</p>	<p>ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 TEXTDATA WVARCHAR</p>
SYS_EVENT_PROCESS_MEMORY	<p>系统遇到与进程大小内存限制相关的事件。</p> <p>NUMDATAINFO 参数指示当前的内存分配, 以千字节为单位。</p> <p>TEXTDATA 参数可以具有以下各值:</p> <ul style="list-style-type: none"> • PROCESS_LIMIT_ABOVE - 可用虚拟内存的量大于使用 Srv.ProcessMemoryLimit 参数指定的限制 • PROCESS_LIMIT_BELOW - 可用虚拟内存的量低于使用 Srv.ProcessMemoryLimit 参数指定的限制 • PROCESS_LOW_LEVEL_ABOVE - 可用虚拟内存的量大于使用 Srv.ProcessMemoryLowPercentage 参数指定的限制 • PROCESS_LOW_LEVEL_BELOW - 可用虚拟内存的量低于使用 Srv.ProcessMemoryLowPercentage 参数指定的限制 • PROCESS_WARNING_LEVEL_ABOVE - 可用虚拟内存的量大于使用 Srv.ProcessMemoryWarningPercentage 参数指定的限制 • PROCESS_WARNING_LEVEL_BELOW - 可用虚拟内存的量低于使用 Srv.ProcessMemoryWarningPercentage 参数指定的限制 	<p>ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 TEXTDATA WVARCHAR</p>
SYS_EVENT_ROWS2MERGE	<p>此事件指示需要有需要从 Bonsai Tree 合并到主存储器树的行。行参数 (NUMDATAINFO) 指示出了 Bonsai 树中不需合并的行数。</p>	<p>ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 TEXTDATA WVARCHAR</p>

表 198. 其他事件 (续)

EVENT NAME	EVENT DESCRIPTION	PARAMETERS
SYS_EVENT_SACFAILED	<p>当 START AFTER COMMIT (SAC) 失败时，将发出此事件。应用程序可能等待此事件并使用作业标识（在 NUMDATAINFO 字段中）来检索系统表 SYS_BACKGROUNDJOB_INFO 中的错误消息。（NUMDATAINFO 中的作业标识与执行 START AFTER COMMIT 语句时返回的作业标识相匹配。）</p>	<p>ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 TEXTDATA WVARCHAR</p>
SYS_EVENT_SHUTDOWNREQ	<p>已接收到关闭请求。如果用户应用程序回调函数返回非零值，那么将不执行关闭。</p> <p>仅当用户使用共享内存访问或链接库访问时，用户才能捕获到此事件。</p> <p>不使用参数。</p>	<p>ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 TEXTDATA WVARCHAR</p>
SYS_EVENT_STATE_MONITOR	<p>监视设置更改时，将发出此事件。</p> <p>State (NUMDATAINFO) 是下列其中某项:</p> <p>0: 监视关闭。</p> <p>1: 监视打开。</p> <p>UID 是监视打开或关闭的用户的用户标识。</p>	<p>ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 TEXTDATA WVARCHAR</p>
SYS_EVENT_STATE_OPEN	<p>当数据库的“state”更改时将发出此事件。参数 STATE (NUMDATAINFO) 指示新状态:</p> <p>0: 已关闭。不允许新的连接。</p> <p>1: 已打开。允许新的连接。</p>	<p>ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 TEXTDATA WVARCHAR</p>
SYS_EVENT_STATE_SHUTDOWN	<p>当服务器关闭时将发出此事件。NUMDATAINFO 和 TEXTDATA 参数不提供任何信息。</p>	<p>ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 TEXTDATA WVARCHAR</p>
SYS_EVENT_STATE_TRACE	<p>用 ADMIN COMMAND 'trace' 命令打开或关闭服务器跟踪。</p> <p>参数 STATE (NUMDATAINFO) 指示新的跟踪状态:</p> <p>0: 跟踪关闭。</p> <p>1: 跟踪打开。</p>	<p>ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 TEXTDATA WVARCHAR</p>

表 198. 其他事件 (续)

EVENT NAME	EVENT DESCRIPTION	PARAMETERS
SYS_EVENT_TMCMD	执行“AT”命令（即定时命令）时将发出此事件。message 参数（TEXTDATA）包含该命令。	ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 TEXTDATA WVARCHAR
SYS_EVENT_TRX_TIMEOUT	不使用该事件。	ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 TEXTDATA WVARCHAR
SYS_EVENT_USERS	参数 REASON (NUMDATAINFO) 包含事件的原因: 0: 已连接用户。 1: 已断开与用户的连接。 2: 与用户断开连接异常。 4: 因为超时而断开与用户的连接。	ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER 和 TEXTDATA WVARCHAR

H.2 导致 SYS_EVENT_ERROR 的错误

本节列示将会导致服务器发出 SYS_EVENT_ERROR 事件的错误。

错误代码列中的编号与 solidDB 错误代码编号相符，这些编号列示在《《IBM solidDB 管理员指南》》内的错误代码中。这些值在 NUMDATAINFO 事件参数中传递。

表 199. 导致 SYS_EVENT_ERROR 的错误

错误代码	错误描述
30104	关闭中止；因为用户回调拒绝关闭。
30208	合并未启动；因为用户回调拒绝合并。
30284	检查点未启动；因为用户回调拒绝检查点。
30302	备份启动失败。正在关闭。
30302	备份启动失败。备份已处于活动状态。
30303	备份中止。

表 199. 导致 SYS_EVENT_ERROR 的错误 (续)

错误代码	错误描述
30304	备份失败。 <i>error_description</i>
30305	备份未启动; 因为用户回调拒绝备份操作。
30306	备份未启动; 备份在无盘服务器上不受支持。
30307	备份未启动, 索引检查失败。 错误已写入文件 <i>ssdebug.log</i> 。
30360	AT 命令失败。 <原因>
30403	写日志文件失败。
30454	未能保存配置文件 <i>file_name</i>
30573	网络备份失败。 <i>原因</i>
30640	<服务器 RPC 错误消息>

H.3 导致 SYS_EVENT_MESSAGES 的条件或警告

本节列示将会导致服务器发出 SYS_EVENT_MESSAGES 事件的警告消息。

错误代码列中的编号与 solidDB 错误代码编号相符, 这些编号列示在《《IBM solidDB 管理员指南》》内的错误代码中。这些值在 NUMDATAINFO 事件参数中传递。

表 200. 导致 SYS_EVENT_MESSAGES 的警告

错误代码	错误描述
30010	用户 <i>username</i> 未能连接, 版本不匹配。 客户机版本为 <i>version</i> , 服务器版本为 <i>version</i> 。
30011	用户 <i>username</i> 未能连接, 整理版本不匹配。
30012	用户 <i>username</i> 未能连接, 已连接的客户机过多。
30020	服务器处于致命状态, 不允许建立新连接。
30020	用户“ <i>username</i> ”未能连接, 因为数据库字符集为 UTF-8, 客户机不支持此字符集。
30282	由于正在关闭, 因此未启动检查点创建操作。
30283	由于已禁止创建检查点, 因此未启动检查点创建操作。
30300	备份成功完成。 服务器还将在启动或完成备份时发出第二个事件 (SYS_EVENT_BACKUP)。

表 200. 导致 *SYS_EVENT_MESSAGES* 的警告 (续)

错误代码	错误描述
30301	<p>已开始备份到 <i>directory_path</i>。</p> <p>服务器还将在启动或完成备份时发出第二个事件 (SYS_EVENT_BACKUP)。</p>
30359	<p>在执行 AT 命令期间，服务器检测到时间不一致。如果已更改系统时间，那么请重新启动服务器。</p>
30361	<p>已忽略不合法的 AT 命令 <i>command</i>。</p>
30362	<p>已忽略不合法的立即 AT 命令 <i>command</i>。</p>
30405	<p>无法打开消息日志文件 <i>file_name</i></p>
30800	<p>无法为外部排序器保留所请求的 <i>number</i> 个内存块。</p> <p>只有 <i>number</i> 个可用的内存块。SQL: <i>SQL_statement</i></p>
30801	<p>无法为外部排序器保留所请求的 <number> 个内存块。</p> <p>只有 <i>number</i> 个可用的内存块。</p>

索引

[B]

- 百分号字符 284
- 保存
 - 消息 234
- 悲观锁定 104
- 备份
 - SYS_EVENT_BACKUP 373
- 被引用表 93
- 比较运算符 27
- 变量
 - 触发器 59
 - 在存储过程中进行赋值 25
 - SQLERRNUM 38
 - SQLERROR 38
 - SQLERROR OF cursorname 39
 - SQLERRSTR 38
 - SQLROWCOUNT 38
 - SQLSUCCESS 38
- 标量函数 25
 - 描述 26, 280
- 表 1, 7
 - 别名 11
 - 除去 89
 - 创建 89
 - 管理 88
 - 将列添加到 90
 - 删除列 90
 - 在更改后落实工作 90
- 表达式 280
 - 存储过程 27
- 表锁定 106
- 并行控制
 - 悲观 104
 - 方式
 - 显示 333
 - MAINMEMORY 333
 - MAINMEMORY PESSIMISTIC 333
 - OPTIMISTIC 333
 - PESSIMISTIC 333
 - PESSIMISTIC 与 OPTIMISTIC 104
- 混合 110
- 乐观 104
- 设置 110
- 用途 103

并行（锁定）方式

- 乐观或悲观 110

并置型索引 128

[C]

参数

- 触发器 59
- 从公告牌中检索 293
- 定义永久数据库级 268
- 放入公告牌 294
- 可更新 293
- 删除 268
- 数据库级 293
- 只读 293
- EnableHints 130
- GET_PARAM() 293
- get_param() 293
- MaxStartStatements 123
- PUT_PARAM() 294
- put_param() 293
- SimpleSQLOpt 126

参数方式 164

- 输出参数 164
- 输入参数 164
- 输入/输出参数 164

参数公告牌 293

- 定义数据库级参数 268
- 描述 294
- 智能事务 254

超时

- 为应答消息设置 237

重复插入 241

重新执行

- 消息 235

触发器

- 参数和变量 59
- 参数设置 69
- 创建 58
- 错误处理 66
- 代码示例 69
- 递归触发器 67
- 跟踪工具 121
- 工作方式 58
- 过程 59
- 获取信息 68
- 嵌套的触发器 67
- 设置缺省值或派生的列 59
- 使用 57
- 事务 61
- 特权和安全性 67
- 用于执行分析和调试的函数 295
- 注释和限制 196

传播事务

- 设置缺省属性 254

- 传播事务 (续)
 - 设置优先级 254
 - 用法 228
 - 终止的消息 243
 - SAVE 命令 252
- 创建
 - 发布 180
- 存储过程
 - 查看过程堆栈 42
 - 触发器 59
 - 错误处理 38
 - 定位型更新和删除 20
 - 对变量赋值 25
 - 跟踪工具 121
 - 过程主体 25
 - 描述 21
 - 嵌套过程 40
 - 缺省游标管理 41
 - 缺省值 22
 - 声明局部变量 25
 - 使用参数 22
 - 使用事件 73
 - 使用 SQL 34, 42
 - 事务 41
 - 输出参数 22
 - 输入参数 22
 - 输入/输出参数 22
 - 特权 42
 - 退出 33
 - 循环 31
 - 游标 42
 - 游标中的参数标记 36
 - 远程 43
 - 自动落实 164
 - CREATE PROCEDURE 语句 22
- 存储过程中的循环 31
- 错误
 - 致命错误, 同步错误 243
 - DBMS 233, 241
 - SYS_EVENT_ERROR 379
- 错误处理
 - 存储过程 38

[D]

- 大型副本数据库
 - 创建 210
- 调整
 - SQL 语句 125
 - SQL 语句和应用程序 125
- 对性能不佳问题进行诊断
 - 解决方案 131
 - 诊断 131
 - 症状 131
- 多列索引 128

[E]

- 二进制数据 300

[F]

- 发布
 - 撤销访问权 251
 - 创建 180
 - 删除 202
 - 授予访问权 215
 - 刷新 228
- 发送
 - 消息 237
- 访问权
 - 发布 215, 251
 - 注册用户 269
- 辅键
 - 索引 92
- 副本数据库
 - 保存事务 252
 - 根据发布执行刷新 228
 - 删除 203
 - 删除消息 231
 - 设置参数 268
 - 属性 253
 - 向主数据库请求应答消息 242
 - 注册 228, 267, 269
 - 注销 228
- 副本数据库属性名 46

[G]

- 更新锁定 107
- 公告牌参数 293
- 共享锁定 107
- 关系数据库 1
- 过程
 - 存储过程 22

[H]

- 函数
 - 标量 25, 26
 - 触发器 295
 - 其他 293
 - 在存储过程中查看堆栈 42
 - AVG 281
 - COUNT 281
 - MAX 281
 - MIN 281
 - SET_PARAM() 293
 - SQL 函数 294
 - SUM 281
- 行 1, 7

行值构造器 18
候选键 93
互斥锁定 107
恢复
 和事务日志记录 5
 DBMS 级错误 233, 241

[J]

集合论 9
集群 91
 键 328
检查点
 SYS_EVENT_CHECKPOINT 374
 'makecp' 命令 137
降序 92
角色
 PUBLIC 85, 184
 SYS_ADMIN_ROLE 85, 184
 SYS_CONSOLE_ROLE 85, 184
 SYS_SYNC_ADMIN_ROLE 85, 184
 SYS_SYNC_REGISTER_ROLE 85, 184
 _SYSTEM 85, 184
节点
 设置 267
结束消息 234

[K]

客户机/服务器体系结构
 多用户能力 4
 描述 4
控制结构, 存储过程 29

[L]

乐观锁定 104
连接字符串
 更改为主数据库名 264
列 1, 7
 从表中删除 90
 对表添加 90
列示用户 143
零长度字符串 33
逻辑数据库 154
逻辑条件
 描述 29
逻辑运算符
 描述 28
 AND 28
 IS NULL 29
 NOT 28, 32
 OR 28
落实读 257

落实工作
 在更改表之后 90
 在更改用户和角色之后 87
落实块
 定义刷新大小 237, 242

[M]

密码
 更改 85
 输入 85
模式
 创建 101
 描述 99, 184
 删除 101
目录
 创建 100, 154
 描述 99
 删除 100

[P]

配置
 同步 268
批处理插入和更新 129
评估应用程序性能 125

[Q]

全表扫描 127

[R]

日期和时间字面值 284
日期时间函数 290

[S]

删除
 发布 202
 副本数据库 203
 失败的消息 233
 书签 187, 206
 消息 231
 预订 205
 主数据库 201
升序 92
事件
 代码示例 73
 等待 128
 使用 73
 ADMIN EVENT 命令 144
事务 19
 保存 252

- 事务 (续)
 - 保存缺省属性 254
 - 传播 228
 - 存储过程 41
 - 定义 103
 - 可读可写 103
 - 描述 4, 108
 - 日志
 - 概述 5
 - 设置传播优先级 254
 - 设置要传播的缺省属性 254
 - 使用触发器 61
 - 事务日志 5
 - 指定属性 253
 - 只读 103
 - 智能事务 228, 252
 - COMMIT WORK 4
 - ROLLBACK 4
- 事务公告牌 293
- 事务耐久性级别
 - 设置 113
 - 提高性能 113
 - 选择 112
- 书签
 - 删除 187, 206
- 数据
 - 从文件导入 221
 - 导出到文件 210
 - 在存储过程中返回 33
- 数据管理
 - solidDB SQL 103
- 数据库
 - 表 1, 7
 - 创建时间 136
 - 关系 1
 - 行 1, 7
 - 可用空间 136
 - 列 1, 7
- 数据库对象 98
- 数据库中的可用空间 136
- 数据类型 8, 280
 - SQL 83
- 属性
 - 保存缺省事务传播条件 254
 - 保存为缺省值 254
 - 指定 253
- 数字函数 288
- 刷新
 - 处理副本数据库中的故障 243
 - 处理主数据库中的故障 243
 - 发布 228
- 锁定
 - 悲观 104
 - 持续时间 108
 - 方式
 - EXCLUSIVE 107

- 锁定 (续)
 - 方式 (续)
 - SHARED 107
 - UPDATE 107
 - 方式, 显示 333
 - 更新 107
 - 共享 107
 - 互斥 107
 - 乐观 104
 - 描述 110
 - EXCLUSIVE LOCK 107
 - SHARED LOCK 107
 - UPDATE LOCK 107
- 锁定方式
 - 混合 110
- 索引 126
 - 并置型 128
 - 创建 91
 - 唯一索引 91
 - 多列 128
 - 辅键索引 92
 - 管理 90
 - 列 128
 - 删除 91
 - 外键 94
 - 主键索引 91

[T]

- 特权
 - 存储过程 42
 - 管理 84
- 通配符 284
- 同步
 - 消息 294
- 同步拉取通知 46
 - 示例 53
- 推送同步 46
 - 示例 53

[W]

- 外键 93, 94, 188
- 维护方式 265
- 唯一约束 90
- 伪列 284

[X]

- 系统表 319
 - 查看 88
 - 描述 88
 - 授予访问权 88
- 系统参数 293
- 系统函数 292

- 系统视图 353
- 下划线 284
- 消息
 - 保存 234
 - 重新执行 235
 - 错误消息, 失败的消息, 应答消息 233, 241
 - 发送 237
 - 结束 234
 - 开始 230
 - 删除 231
 - 向主数据库请求应答 242
 - 执行 235
- 性能 125
 - 单表 SQL 查询 126
 - 观察 115
 - 使用索引来提高 126
 - 索引 126
 - 诊断问题 131
- 序列 72

[Y]

- 延迟型过程调用 46
- 引用表 93
- 引用动作
 - 级联 96
 - 设置缺省值 96
 - 设置 NULL 96
 - 无动作 96
 - 限制 96
- 引用完整性 93, 188
 - 动态约束管理 97
 - 瞬态表 188
 - 约束 96
- 应答消息
 - 设置超时 237
 - 向主数据库请求 242
- 用户
 - 创建 85
 - 删除 85
- 用户和角色
 - 在更改后落实工作 87
- 用户角色 84
 - 保留的角色名 84
 - 撤销特权 86
 - 撤销用户的角色 86
 - 创建 86
 - 对用户指定角色 86
 - 更改密码 85
 - 管理员 85, 87, 184
 - 将特权授予 86
 - 删除 86
 - 系统控制台角色 85, 184
- 用户名
 - 保留的名称 84
- 用户特权 84

- 用户特权 (续)
 - 撤销 86
 - 授予 86
 - 授予管理员特权 87
- 用于存储过程和触发器的跟踪工具 121
- 优化
 - 批处理插入和更新 129
- 优化器提示
 - 使用 130
- 游标
 - 参数标记 36
 - 存储过程 42
 - 处理 34
 - 访存 36
 - 关闭 36
 - 删除 36
 - 执行 35
 - 准备 35
 - 存储过程中的缺省管理 41
- 预订
 - 导出 210
 - 导入 221
 - 定义落实块 242
 - 删除 205
- 元数据
 - 导出 210
- 远程存储过程 43
- 约束
 - 外键 188

[Z]

- 增量发布
 - 指定 147
- 执行
 - 失败的消息 241
 - 消息 235
- 致命错误 (高级复制) 243
- 智能事务
 - 参数公告牌 254
 - 使用已保存的属性 253
- 主键 90, 93
 - 索引 91
- 主数据库
 - 参数 268
 - 撤销对发布的访问权 251
 - 更改网络名 264
 - 将事务传播到 228
 - 请求应答消息 242
 - 删除 201
 - 设置节点名 267
 - 授予对发布的访问权 215
 - 属性 253
 - 用户信息 228
 - MESSAGE APPEND 语句 228

- 主数据库用户
 - 下载列表 228
- 注册
 - 副本数据库 228
 - 设置副本数据库节点名 267
- 注册数据库
 - 注册用户 269
- 注销
 - 副本数据库 228
- 转义序列 fn 27
- 转义字符 285
- 自动落实 164
- 字符串
 - 零长度 33
- 字符串函数 287
- 字符数据类型 297, 299

A

- ABS (函数) 288
- ACCESS RIGHTS (远程存储过程) 44
- ACOS (函数) 288
- ADD CONSTRAINT 97
- ADD_MONTHS (函数) 290
- ADMIN COMMAND
 - 备份 134
 - 传递状态 138
 - 命令 133
 - 消息 137
 - abort 134
 - assertexit 134
 - backgroundjob 134
 - backuplist 134
 - checkpointing 134
 - cleanbgjobinfo 135
 - close 135
 - describe 135
 - errorcode 135
 - errorexit 135
 - filespec 135
 - help 135
 - hotstandby 135
 - indexusage 136
 - info 136
 - logreader 137
 - makecp 137
 - memory 137
 - monitor 137
 - netbackup 137
 - netbackuplist 137
 - netstat 137
 - notify 137
 - open 138
 - parameter 138
 - perfmon 139
 - perfmon diff 139

ADMIN COMMAND (续)

- pid 139
- proctrace 139
- protocols 140
- runmerge 140
- save parameters 140
- shutdown 140
- sqlist 140
- startmerge 140
- status 140
- throwout 140
- tid 140
- trace 141
- userid 141
- userlist 142, 143
- usertrace 143
- version 144

ADMIN EVENT 语句 144

ALL (关键字)

PROPAGATE TRANSACTIONS 228

ALTER REMOTE SERVER 语句 145

ALTER TABLE 语句 145

ALTER TABLE SET HISTORY COLUMNS 146

ALTER TABLE SET NOSYNCHISTORY 语句
描述 147

ALTER TABLE SET SYNCHISTORY
描述 147

ALTER TRIGGER 语句 149

ALTER USER 语句 149

AND (运算符) 28, 279

APPEND (关键字) 228

AS 16

SELECT 语句 16

ASCII (函数) 287

ASIN (函数) 288

ATAN (函数) 288

ATAN2 (函数) 289

AVG (函数) 281

B

bcktime ADMIN COMMAND 136

BEGIN 163

BIGINT 数据类型 299

BINARY 数据类型

CAST 函数 301

BITANDNOT() 函数 293

BIT_AND 函数 (按位 AND 运算符) 293

BLOB (二进制大对象) 13, 302

CAST 函数 301

BLOB 和 CLOB 302

C

CALL 语句 151
 调用过程 22
 使用 EXECDIRECT 的示例 174
CASCADE 89, 199, 204
 REVOKE 语句中关键字 250
CASCADED (保留字) 305
CASE 16, 281
CAST (函数) 16, 281
 二进制值 301
CEILING (函数) 289
CHAR (函数) 287
CHAR 数据类型 297
CHAR LARGE OBJECT 数据类型 298
CHAR VARYING 数据类型 298
CHARACTER 数据类型 297
CHARACTER LARGE OBJECT 数据类型 298
CHARACTER VARYING 数据类型 298
CHECK 97
CLOB 数据类型 298, 302
COALESCE 281
COLUMNS 系统视图 354
COMMIT WORK 语句 4, 19, 154
 存储过程 41
COMMITBLOCK (关键字)
 DROP SUBSCRIPTION 205
 MESSAGE FORWARD 237
 MESSAGE GET REPLY 242
 REFRESH 246
CONCAT (function) 287
CONCURRENCY 103
ConnectStrForMaster (参数) 152
CONVERTORSTOUNIONS 262
CONVERT_CHAR 281
CONVERT_DATE 281
CONVERT_DECIMAL 281
CONVERT_DOUBLE 281
CONVERT_FLOAT 281
CONVERT_INTEGER 281
CONVERT_LONGVARCHAR 281
CONVERT_NUMERIC 281
CONVERT_REAL 281
CONVERT_SMALLINT 281
CONVERT_TIME 281
CONVERT_TIMESTAMP 281
CONVERT_TINYINT 281
CONVERT_VARCHAR 281
COS (函数) 289
COT (函数) 289
COUNT (函数) 281
cptime ADMIN COMMAND 136
CREATE CATALOG 语句 100, 154
CREATE EVENT 语句 73, 156
CREATE INDEX 语句 162
CREATE PROCEDURE (外部) 语句 175

CREATE PROCEDURE 语句 163
 参数部分 22
 声明部分 25
CREATE PUBLICATION 语句
 描述 180
CREATE ROLE 语句 184
CREATE SCHEMA 语句 184
CREATE SEQUENCE 语句 72
CREATE SYNC BOOKMARK 语句
 描述 187
CREATE TABLE 语句 188
CREATE TRIGGER 语句 191
CREATE USER 语句 197
CREATE VIEW 语句 197
CREATE [OR REPLACE] REMOTE SERVER 语句 183
CURDATE (函数) 290
CURRENT_CATALOG (系统函数) 292
CURRENT_SCHEMA (系统函数) 292
CURRENT_USERID (系统函数) 292
CURSORNAME 163, 171
 用法示例 171, 173
CURTIME (函数) 290

D

DATE (函数) 290
DATE 数据类型
 描述 301
DAYNAME (函数) 290
DAYOFMONTH (函数) 290
DAYOFWEEK (函数) 290
DAYOFYEAR (函数) 290
dbconfigsize ADMIN COMMAND 136
dbcreatetime ADMIN COMMAND 136
dbfreesize ADMIN COMMAND 136
dbpagesize ADMIN COMMAND 136
dbsize ADMIN COMMAND 136
DECIMAL 数据类型 300
DEFAULT 43
DEGREES (函数) 289
DELETE 语句 197
DESCRIBE 语句 198
DIFFERENCE (函数) 289
DOUBLE 数据类型 300
DOUBLE PRECISION 数据类型 302
DROP BOOKMARK 语句 187
DROP CATALOG 语句 199
DROP CONSTRAINT 语句 97
DROP EVENT 语句 73, 200
DROP INDEX 语句 200
DROP MASTER 语句 201
DROP PROCEDURE 语句 201
DROP PUBLICATION 语句 202
DROP PUBLICATION REGISTRATION 语句 202
DROP REMOTE SERVER 语句 203
DROP REPLICA 语句 203

DROP ROLE 语句 204
DROP SCHEMA 语句 204
DROP SEQUENCE 语句 205
DROP SUBSCRIPTION 语句 205
DROP SYNC BOOKMARK 语句 206
DROP TABLE 语句 207
DROP TRIGGER 语句 208
DROP USER 语句 209
DROP VIEW 语句 209

E

EnableHints (参数) 130
END 163
END LOOP 170
EVENT
 等待事件 163
 对事件注册 163
 对事件注销 163
 发出事件 163
 删除事件 200
EXCLUSIVE (锁定方式) 107
EXECDIRECT 171
 用法示例 174
 在 VARCHAR 变量中使用 SQL 语句 174
EXP (函数) 289
EXPLAIN PLAN FOR 语句 116, 131, 209
EXPORT SUBSCRIPTION
 描述 210
EXTRACT FROM 290

F

FLOAT 数据类型 299
FLOOR (函数) 289
fn
 在 {fn func_name} 中的用法 26, 34
FOR EACH REPLICA 46
FOREIGN KEY (约束) 98
FULL (关键字) 228

G

GET_PARAM()
 描述 293
GET_UNIQUE_STRING 171, 287
 用法示例 173, 174
GRANT 语句 214
GRANT EXECUTE ON 语句 42
GRANT PASSTHROUGH 语句 214
GRANT REFRESH ON
 描述 215

H

HINT 语句 215
HOUR (函数) 290

I

IF 语句
 描述 29
IFNULL (系统函数) 293
IF-THEN 构造
 描述 29
IF-THEN-ELSE 构造
 描述 29
IF-THEN-ELSEIF 构造
 描述 30
imdbsize ADMIN COMMAND 136
IMPORT 语句
 描述 221
INSERT (字符串函数) 287
INSERT 语句 223
 多行 223
 使用缺省值 223
INT 数据类型 299
INTEGER 数据类型 299
IS NULL (运算符)
 描述 29

L

LCASE (函数) 287
LEFT (函数) 287
LENGTH (函数) 287
LIKE 279, 283, 285
 使用 START AFTER COMMIT 274
LIST 语句 224
LOCATE (函数) 288
LOCK TABLE 语句 225
Lock timeout
 设置 111
 设置, 乐观表 112
LOG (函数) 289
LOG10 (函数) 289
LOGIN_CATALOG (系统函数) 292
LOGIN_SCHEMA (系统函数) 292
LOGIN_USERID (系统函数) 292
logsize ADMIN COMMAND 136
LONG NATIONAL VARCHAR 数据类型 298
LONG VARBINARY 数据类型 301
 使用 CAST 来输入值 301
LONG VARCHAR 数据类型 298
LONG WVARCHAR 数据类型 298
LOOP 170
LTRIM (函数) 288

M

MAINTENANCE
 SET SYNC MODE MAINTENANCE 265
MAX (函数) 281
MaxStartStatements (参数) 123
maxusers ADMIN COMMAND 136
memtotal ADMIN COMMAND 136
MESSAGE APPEND PROPAGATE TRANSACTIONS
 描述 228
MESSAGE APPEND PROPAGATE WHERE
 使用属性 254
MESSAGE APPEND REFRESH 228
 描述 228
MESSAGE APPEND REGISTER PUBLICATION
 描述 228
MESSAGE APPEND REGISTER REPLICA
 描述 228
MESSAGE APPEND SUBSCRIBE 228
MESSAGE APPEND SYNC_CONFIG
 描述 228
MESSAGE APPEND UNREGISTER PUBLICATION
 描述 228
MESSAGE APPEND UNREGISTER REPLICA
 描述 228
MESSAGE BEGIN
 语句 230
MESSAGE DELETE
 描述 231
MESSAGE END
 描述 234
MESSAGE EXECUTE
 描述 235
MESSAGE FORWARD
 描述 237
MESSAGE FROM REPLICA DELETE 241
 描述 233
MESSAGE FROM REPLICA EXECUTE
 描述 241
MESSAGE FROM REPLICA RESTART 242
MESSAGE GET REPLY
 描述 242
MIN (函数) 281
MINUTE (函数) 290
MOD (函数) 289
monitorstate ADMIN COMMAND 136
MONTH (函数) 290
MONTHNAME (函数) 291

N

name ADMIN COMMAND 136
NATIONAL CHAR 数据类型 298
NATIONAL CHARACTER 数据类型 298
NATIONAL VARCHAR 数据类型 298
NCHAR 数据类型 298

NCHAR LARGE OBJECT 数据类型 298
NCHAR VARYING 数据类型 298
NCLOB 数据类型 298
node-def 43
NONUNIQUE 46
NORMAL
 SET SYNC MODE NORMAL 265
NOT (运算符) 28, 279
NOT NULL 16
NOTUNIQUE 274
NOW (函数) 291
NULL 14
NULLIF 281
nulls
 处理 33
numcursors ADMIN COMMAND 136
NUMERIC 数据类型 300
numlocks ADMIN COMMAND 136
nummerges ADMIN COMMAND 136
numtransactions ADMIN COMMAND 136
numusers ADMIN COMMAND 136
NVARCHAR 数据类型 298

O

openstate ADMIN COMMAND 136
OR (运算符) 28, 279

P

PI (函数) 289
POSITION (函数) 288
POWER (函数) 289
PRECISION 数据类型 300
primarystarttime ADMIN COMMAND 136
processsize ADMIN COMMAND 136
proctrace 121
PROC_COUNT 函数
 存储过程堆栈 42
PROC_NAME (N) 函数
 存储过程堆栈 42
PROC_SCHEMA (N) 函数
 存储过程 42
psize ADMIN COMMAND 136
PUT_PARAM()
 描述 294

Q

QUARTER (函数) 291

R

RADIANS (函数) 289
REAL 数据类型 299, 302

REFERENCES (关键字) 188, 214, 250
REFRESH 语句 246
 定义落实块 237
REGISTER EVENT 语句 249
REPEAT (函数) 288
REPEATABLE READ 257
REPLACE (函数) 288
RESTRICT 关键字 89, 199, 204
 REVOKE 语句 250
RETURN 关键字 33
REVOKE (撤销角色或用户的特权) 语句 250
REVOKE (撤销用户的角色) 语句 250
REVOKE PASSTHROUGH 语句 250
REVOKE REFRESH 251
REVOKE REFRESH ON
 描述 251
REVOKE SUBSCRIBE 251
RIGHT (函数) 288
ROLLBACK (语句) 4
 存储过程 41
ROLLBACK WORK 语句 252
ROUND (函数) 289
ROWID 126
ROWNUM 126, 284, 315
RTRIM (函数) 288
RVC (行值构造器) 18

S

SAVE
 描述 252
SAVE DEFAULT PROPAGATE PROPERTY WHERE
 描述 254
SAVE DEFAULT PROPERTY
 描述 254
SAVE PROPERTY 语句 253
SECOND 291
secondarystarttime ADMIN COMMAND 136
SELECT 语句 255
SERIALIZABLE 257
sernum ADMIN COMMAND 136
SERVER_INFO 系统视图 354
SET
 SET 与 SET TRANSACTION 之间的差别 258, 271
SET 与 SET TRANSACTION 258, 271
SET 语句 257
 在存储过程中 25
SET CATALOG 语句 99
SET CATALOG catalog_name 257
SET DELETE CAPTURE 语句 259
SET DURABILITY 113, 257
SET HISTORY COLUMNS
 描述 147
SET IDLE TIMEOUT 257
SET ISOLATION LEVEL 257
SET LOCK TIMEOUT 257
SET NOSYNCHISTORY
 描述 147
SET OPTIMISTIC LOCK TIMEOUT 257
SET PASSTHROUGH 传递 260
SET READ-ONLY 257
SET READ-WRITE 257
SET SAFENESS 257
SET SCHEMA 257
SET SCHEMA 语句 99, 261
SET SCHEMA USER 语句 261
SET SEQUENCE 72
SET SQL 语句 262
SET STATEMENT MAXTIME 257
SET SYNC CONNECT 152
 描述 264
SET SYNC MODE 语句 265
SET SYNC NODE
 描述 267
SET SYNC PARAMETER
 描述 268
SET SYNC USER IDENTIFIED BY
 描述 269
SET SYNCHISTORY 146
 描述 147
SET TRANSACTION
 SET 与 SET TRANSACTION 之间的差别 258, 271
SET TRANSACTION 语句 271
SET TRANSACTION DELETE CAPTURE 语句 272
SET TRANSACTION DURABILITY 113
SET TRANSACTION PASSTHROUGH 语句 273
SET TRANSACTION WRITE 271
SET WRITE 257
SHARED (锁定方式) 107
SIGN (函数) 289
SimpleSQLOpt (参数) 126
SIN 289
SLEEP 293
SMALLINT 数据类型 299
solidDB
 数据管理 103
solidDB SQL
 函数 83
 扩展 83
 数据管理 103
 数据类型 83
 用于进行数据库管理 83
solidDB SQL 语法
 一致性 83
soltrace.out 121
SOUNDEX (函数) 288
SPACE (函数) 288
space ADMIN COMMAND 136
SQL
 入门 7
 在存储过程中使用 42
 子查询 11

SQL 函数

- GET_PARAM() 293
- PUT_PARAM() 294

SQL 脚本 84

- sample.sql 88
- users.sql 84

SQL 通配符 285

SQL 优化 126

SQL 语句

- 调整 125
- 使用 83
- 示例 88
- 有关管理数据库对象的示例 100
- 有关管理索引的示例 91
- 有关管理用户、角色和用户特权的示例 85

SQLERRNUM (变量)

- 错误代码 38

SQLERROR (变量)

- 错误字符串 38

SQLERROR OF cursorname (变量) 39

SQLERRSTR (变量)

- 错误字符串 38

SQLROWCOUNT (变量)

- 行数 38

SQLSUCCESS (变量)

- 存储过程 38

SQL-92 83

SQL-99 83

SQL_LANGUAGES 系统表 319

SQL_TSI_DAY 291

SQL_TSI_FRAC_SECOND 291

SQL_TSI_HOUR 291

SQL_TSI_MINUTE 291

SQL_TSI_MONTH 291

SQL_TSI_QUARTER 291

SQL_TSI_SECOND 291

SQL_TSI_WEEK 291

SQL_TSI_YEAR 291

SQRT (函数) 289

SSC_TASK_BACKGROUND 122

START AFTER COMMIT 语句 274

- 调整性能 122
- 分析故障 123

SUBSCRIBERefresh 228

SUBSTRING (函数) 288

SUM (函数) 281

SYNCHISTORY 146

SYNC_CONFIG 228

SYNC_DELETE_MESSAGES 存储过程 370

SYNC_DELETE_REPLICA_MESSAGES 存储过程 371

SYNC_MAINTENANCEMODE_BEGIN (事件) 265, 375

SYNC_MAINTENANCEMODE_END (事件) 265, 376

SYNC_REGISTER_PUBLICATION 存储过程 366

SYNC_REGISTER_REPLICA 存储过程 364

SYNC_SETUP_CATALOG 存储过程 363

SYNC_SHOW_REPLICA_SUBSCRIPTIONS 存储过程 369

SYNC_SHOW_SUBSCRIPTIONS 存储过程 368

SYNC_UNREGISTER_PUBLICATION 存储过程 367

SYNC_UNREGISTER_REPLICA 存储过程 365

SYS_ATTAUTH 系统表 319

SYS_BACKGROUNDJOB_INFO 123

SYS_BACKGROUNDJOB_INFO 系统表 320

SYS_BLOBS 系统表 321

SYS_BULLETIN_BOARD 系统表 336

SYS_CARDINAL 系统表 322

SYS_CATALOGS 系统表 322

SYS_CHECKSTRINGS 系统表 322

SYS_COLUMNS 系统表 323

SYS_COLUMNS_AUX 系统表 324

SYS_DL_REPLICA_CONFIG 系统表 324

SYS_DL_REPLICA_DEFAULT 系统表 324

SYS_EVENTS 系统表 325

SYS_EVENT_BACKUP 373

SYS_EVENT_BACKUPREQ 374

SYS_EVENT_CHECKPOINT (事件) 374

SYS_EVENT_CHECKPOINTREQ 374

SYS_EVENT_ERROR 374, 379, 380

SYS_EVENT_IDLE 374

SYS_EVENT_ILL_LOGIN 375

SYS_EVENT_IMDB_MEMORY 375

SYS_EVENT_MERGE 376

SYS_EVENT_MERGEREQ 376

SYS_EVENT_MESSAGES 376

SYS_EVENT_NOTIFY 376

SYS_EVENT_PARAMETER 377

SYS_EVENT_PROCESS_MEMORY 377

SYS_EVENT_ROWS2MERGE 377

SYS_EVENT_SACFAILED 123, 378

SYS_EVENT_SHUTDOWNREQ 378

SYS_EVENT_STATE_MONITOR 378

SYS_EVENT_STATE_OPEN 378

SYS_EVENT_STATE_SHUTDOWN 378

SYS_EVENT_STATE_TRACE 378

SYS_EVENT_TMCMD 379

SYS_EVENT_TRX_TIMEOUT 379

SYS_EVENT_USERS 379

SYS_FORKEYPARTS 系统表 326

SYS_FORKEYS 系统表 327

SYS_GETBACKGROUNDJOB_INFO 123

SYS_GETBACKGROUNDJOB_INFO 存储过程 372

SYS_HOTSTANDBY 系统表 327

SYS_KEYPARTS 系统表 327

SYS_KEYS 系统表 328

SYS_PROCEDURES 系统表 329

SYS_PROCEDURE_COLUMNS 系统表 330

SYS_PROPERTIES 系统表 330

SYS_PUBLICATIONS 系统表 339

SYS_PUBLICATIONS_REPLICA 系统表 340

SYS_PUBLICATION_ARGS 系统表 337

SYS_PUBLICATION_REPLICA_ARGS 系统表 337

SYS_PUBLICATION_REPLICA_STMTARGS 系统表 338

SYS_PUBLICATION_REPLICA_STMTS 系统表 338

SYS_PUBLICATION_STMTARGS 系统表 338
SYS_PUBLICATION_STMTS 系统表 339
SYS_RELAUTH 系统表 331
SYS_SCHEMAS 系统表 331
SYS_SEQUENCES 系统表 331
SYS_SYNC_BOOKMARKS 系统表 340
SYS_SYNC_HISTORY_COLUMNS 系统表 341
SYS_SYNC_INFO 系统表 341
SYS_SYNC_MASTERS 系统表 346
SYS_SYNC_MASTER_MSGINFO 系统表 341
SYS_SYNC_MASTER_RECEIVED_BLOB_REFS 系统表 342
SYS_SYNC_MASTER_RECEIVED_MSGPARTS 系统表 343
SYS_SYNC_MASTER_RECEIVED_MSGS 系统表 343
SYS_SYNC_MASTER_STORED_BLOB_REFS 系统表 343
SYS_SYNC_MASTER_STORED_MSGPARTS 系统表 344
SYS_SYNC_MASTER_STORED_MSGS 系统表 344
SYS_SYNC_MASTER_SUBSC_REQ 系统表 345
SYS_SYNC_MASTER_VERSIONS 系统表 345
SYS_SYNC_RECEIVED_BLOB_ARGS 系统表 346
SYS_SYNC_RECEIVED_STMTS 系统表 347
SYS_SYNC_REPLICAS 系统表 351
SYS_SYNC_REPLICA_MSGINFO 系统表 347
SYS_SYNC_REPLICA_PROPERTIES 系统表 332
SYS_SYNC_REPLICA_RECEIVED_BLOB_REFS 系统表 348
SYS_SYNC_REPLICA_RECEIVED_MSGPARTS 系统表 349
SYS_SYNC_REPLICA_RECEIVED_MSGS 系统表 349
SYS_SYNC_REPLICA_STORED_BLOB_REFS 系统表 349
SYS_SYNC_REPLICA_STORED_MSGPARTS 系统表 350
SYS_SYNC_REPLICA_STORED_MSGS 系统表 350
SYS_SYNC_REPLICA_VERSIONS 系统表 350
SYS_SYNC_SAVED_BLOB_ARGS 系统表 351
SYS_SYNC_SAVED_STMTS 系统表 352
SYS_SYNC_TRX_PROPERTIES 系统表 352
SYS_SYNC_USERMAPS 系统表 353
SYS_SYNC_USERS 系统表 353
SYS_SYNONYM 系统表 333
SYS_TABLEMODES 系统表 333
SYS_TABLES 系统表 333
SYS_TRIGGERS 系统表 334
SYS_TYPES 系统表 335
SYS_URole 系统表 335
SYS_USERS 系统表 336
SYS_VIEWS 系统表 336

T

TABLES 系统视图 355
TAN 289
THEN
 CASE 语句中的关键字 281
TIME 数据类型 301
TIMEOUT (关键字)
 MESSAGE FORWARD 237
 MESSAGE GET REPLY 237
TIMESTAMP 数据类型 302
TIMESTAMPADD 291

TIMESTAMPDIFF 291
TIMESTAMP_FORMAT (函数) 292
TINYINT 数据类型 299
TO (关键字)
 MESSAGE FORWARD 237
TO_CHAR (函数) 288
TO_DATE (函数) 292
tracestate ADMIN COMMAND 136
TRIM (函数) 288
TRUNCATE (函数) 289
TRUNCATE TABLE 语句 276

U

UCASE (函数) 288
UIC (系统函数) 292
UNIQUE 46, 98, 274
UNLOCK TABLE 语句 276
UPDATE (锁定方式) 107
UPDATE (已搜索) 语句 278
uptime ADMIN COMMAND 136
userlist ADMIN COMMAND 142, 143
USERS 系统视图 355
usertrace 121

V

VARBINARY 数据类型 300
VARCHAR 数据类型 298
VARCHAR_FORMAT (function) 288

W

WCHAR 数据类型 298
WEEK (函数) 292
WHEN
 事件规范中的关键字 156
 在 case_specification 中 281
WHERE (关键字)
 PROPAGATE TRANSACTIONS 228
WHILE-LOOP 语句
 描述 31
WRITETRACE 121
WVARCHAR 数据类型 298

Y

YEAR (函数) 292

[特别字符]

* (星号) 280
+ (加) 280, 287
- (减) 280

/ (斜杠) 280
= (等于) 282
>> (大于) 282
>>= (大于或等于) 282
< (小于) 282
<= (小于或等于) 282
<> (不等于) 282
|| (并置运算符) 287
% 284
_ (下划线) 284
“SQL 信息”工具 115

声明

© Copyright Oy International Business Machines Ab 1993, 2011.

All rights reserved.

除非经过 International Business Machines Ab 书面授权，否则不能以任何方式使用本产品中的任何部分。

本产品受美国专利 6144941, 7136912, 6970876, 7139775, 6978396, 7266702, 7406489, 7502796, 和 7587429 保护。

为此产品指定的美国出口管制分类编号是 ECCN=5D992b。

本信息是为在美国提供的产品和服务编写的。

IBM 可能在其他国家或地区不提供本文中讨论的产品、服务或功能特性。有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务，则由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可。您可以用书面方式将许可查询寄往：

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

有关双字节 (DBCS) 信息的许可证查询，请联系您所在国家的 IBM 知识产权部门，或将查询以书面的形式发送至：

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区：INTERNATIONAL BUSINESS MACHINES CORPORATION“按现状”提供本出版物，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的有关非侵权、适销和适用于某种特定用途的保证。某些国家或地区在某些交易中不允许免除明示或暗含的保证。因此本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本资料的新版本中。IBM 可以随时对本资料中描述的产品和/或程序进行改进和/或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。那些 Web 站点中的资料不是 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的：(i) 允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及 (ii) 允许对已经交换的信息进行相互使用，请与下列地址联系：

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

只要遵守适当的条件和条款，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际软件许可协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。因此，在其他操作环境中获得的数据可能会有明显的不同。有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。此外，有些测量是通过推算而估计的，实际结果可能会有差异。本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的资料中获取。IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

本信息包含在日常业务操作中使用的数据和报告的示例。为了尽可能完整地说明这些示例，示例中可能会包括个人、公司、品牌和产品的名称。所有这些名字都是虚构的，若现实生活中实际业务企业使用的名字和地址与此相似，纯属巧合。

版权许可：

本信息包括源语言形式的样本应用程序，这些样本说明不同操作平台上的编程方法。如果是为按照在编写样本程序的操作平台上的应用程序编程接口 (API) 进行应用程序的开发、使用、经销或分发为目的，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。这些示例并未在所有条件下作全面测试。因此，IBM 不能担保或暗示这些程序的可靠性、可维护性或功能。样本程序按原样提供，而没有任何类型的保证。IBM 对使用样本程序过程中出现的任何损害不提供任何保障。

凡这些实例程序的每份拷贝或其任何部分或任何衍生产品，都必须包括如下版权声明：

©（贵公司的名称）（年）。此部分代码是根据 IBM Corp. 公司的样本程序衍生出来的。

© Copyright IBM Corp.（输入年份）。All rights reserved.

如果你要查看此信息软拷贝，那么可能不显示相片和颜色说明。

商标

IBM、IBM 徽标、ibm.com[®]、Solid、solidDB、InfoSphere、DB2、Informix[®] 和 WebSphere[®] 是 International Business Machines Corp. 在全球许多管辖区域内注册的商标或注册商标。其他产品和服务名称可能是 IBM 或其他公司的商标。在 Web 上的『版权和商标信息』（www.ibm.com/legal/copytrade.shtml）处提供了 IBM 商标的最新列表。

Java 和所有基于 Java 的商标和徽标是 Oracle 和/或其附属机构的商标或注册商标。

Linux 是 Linus Torvalds 在美国和/或其他国家或地区的注册商标。

Microsoft 和 Windows 是 Microsoft Corporation 在美国和/或其他国家或地区的注册商标。

UNIX 是 The Open Group 在美国和其他国家或地区的注册商标。

其他产品和服务可能是 IBM 或其他公司的商标。



Printed in China

S151-1699-00

