附录D SQL 语法参考

此附录介绍了MySQL提供的每条 SQL 语句。还介绍了在 SQL 代码中进行注释的语法。注释用来编写说明性的文本，而不对应实际操作。在 MySQL 中注释还可用来隐藏特定的关键字（MySQL 执行这些关键字，但其他数据库忽略它们）。请参阅“注释语法”一节。

MySQL 正处于发展中，经常会对其进行改进。因此应该经常参阅联机 MySQL 参考指南，看看又增加了什么新功能。此指南可在 http://www.mysql.com/ 处得到。

语法描述使用下列约定：

- 可选信息括在方括号（[ ]）中。如果某个列表括在方括号中，表示可以进行选择。
- 如果某个列表括在花括号（{ }）中，表示必须从中选择一项。
- 竖线（|）分开列表中的可选项。
- 省略号（...）表示该省略号前的项可以重复。

除非另有说明，否则这里给出的语句自 MySQL 3.22.0 版以来都有效。

D.1 SQL 语句

本节介绍每条 MySQL 的 SQL 语句的语法和含义。如果不具有使用某条语句的权限，则执行该语句将会失败。例如，如果没有访问数据库 db_name 的权限，执行 USE db_name 将会失败。

D.1.1 ALTER TABLE

ALTER [IGNORE] TABLE *tbl_name action _list*

ALTER TABLE 语句允许对表进行重命名或修改其结构。为了使用它，应该指定表名 *tbl_name*，然后给出要对表执行的一个或多个操作的说明。如果在新表的唯一键中出现重复值，IGNORE 关键字将起作用。这时如果不使用 IGNORE 关键字，则 ALTER TABLE 语句的作用将被取消。使用 IGNORE，唯一键值重复的行将被删除。

action_list 指定一个或多个操作，各操作之间用逗号分隔。各个操作依次执行。有如下的操作：

- ADD [COLUMN] *col_declaration* [FIRST | AFTER *col_name*] 给表增加一列。col_declaration 为列定义，它与 CREATE TABLE 语句中所用的格式相同。如果给出 FIRST 关键字则增加的列成为表中第一列，如果给出了 AFTER col_name，则增加的列放置在指定的列后。如果未指定增加列的位置，则此列成为表中最后一列。

```
ALTER TABLE member
    ADD member_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
ALTER TABLE member
    ADD member_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
    FIRST
ALTER TABLE member
    ADD member_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
    AFTER suffix
```

■ ADD INDEX [*index_name*] (*index_colums*) 给表增加索引。增加的索引基于 index_colums 中所指定的列，这些列都必须是表 tbl_name 中的列。如果指定了多列，列之间必须用逗号分隔。对于 CHAR 和 VARCHAR 列，可对列的前缀进行索引，即利用 col_name(n) 语法对列值的前 n 个字符进行索引。对于 BLOB 和 TEXT 列，必须指定前缀值，不允许对整个列进行索引。如果未指定索引名 index_name，那么系统会根据第一个索引列的名称自动选择一个索引名。

■ ADD PRIMARY KEY (*index_columns*) 在给定的列上增加一个主键。给此键的名称为 PRIMARY。如像在 ADD INDEX 子句中一样指定 index_columns。如果已经存在主键，则出错。

```
ALTER TABLE president ADD PRIMARY KEY (last_name, first_name)
```

■ ADD UNIQUE [*index_name*] (*index_columns*) 给表 tbl_name 增加一个唯一值索引。与 ADD INDEX 子句中一样指定 *index_name* 和 index_columns。

```
ALTER TABLE absence ADD UNIQUE id_date (student_id, date)
```

■ ALTER [COLUMN] *col_name* {SET DEFAULT *value* | DROP DEFAULT} 修改给定列的缺省值，或者对指定的值进行修改或者删除当前缺省值。在后一种情形中，将分配一个新的缺省值，如 CREATE TABLE 语句的相应条目所述。

```
ALTER TABLE event ALTER type SET DEFAULT "Q"
ALTER TABLE event ALTER type DROP DEFAULT
```

■ CHANGE [COLUMN] *col_name col_declaration* 更改列的名称和定义。col_name 为列的当前名称，*col_declaration*，为定义，该列的定义将更改成此定义。col_declaration 以与 CREATE TABLE 语句中相同的格式给出。注意该定义包括了新的列名，如果不想改动列名，可两次给出相同的列名。

```
ALTER TABLE student CHANGE name name VARCHAR(40)
ALTER TABLE student CHANGE name student_name CHAR(30) NOT NULL
```

■ DROP [COLUMN] *col_name* 从表中删除指定的列。如果要删除的列是索引的组成部分，则这个列也将从这些索引中删除。如果删除了一个索引中所有的列，则此索引也被删除。

```
ALTER TABLE president DROP suffix
```

■ DROP INDEX *index_name* 从表中删除指定的索引。

```
ALTER TABLE member DROP INDEX name
```

■ DROP PRIMARY KEY 从表中删除主键。如果表中没有创建 PRIMARY KEY 的唯一索引，但有一个或多个 UNIQUE 索引，则删除这些索引中的第一个索引。

```
ALTER TABLE president DROP PRIMARY KEY
```

■ MODIFY [COLUMN] *col_declaration* 更改列的定义。利用与 CREATE TABLE 语句条目中列描述相同的格式给出列定义 *col_declaration*。此定义以列名开始，它表示怎样对列进行修改。MODIFY 是在 MySQL 3.22.16 中引入的。

```
ALTER TABLE student MODIFY name VARCHAR(40) DEFAULT "" NOT NULL
```

■ RENAME [AS] *new_tbl_name* 重新将表 *tbl_name* 命名为 *new_tbl_name*。

```
ALTER TABLE president RENAME prez
```

■ *table_options* 指定表排序选项，此排序选项可在 CREATE TABLE 语句的 *table_options* 部分给出。

```
ALTER TABLE score TYPE = MYISAM CHECKSUM = 1
```

D.1.2 CREATE DATABASE

CREATE DATABASE *db_name*

创建一个具有指定名称的数据库。如果要创建的数据库已经存在，或者没有创建它的适当权限，则此语句失败。

D.1.3 CREATE FUNCTION

```
CREATE [AGGREGATE] FUNCTION function_name
    RETURNS {STRING | REAL | INTEGER}
    SONAME shared_library_name
```

定义一个装入 mysql 数据库的 function 表中的用户定义函数（UDF）。*function_name* 为一个名称，SQL 语句利用此名称引用函数。跟在 RETURNS 后的关键字指出此函数的返回值的类型。*shared_library_name* 为包含可执行代码或函数的文件路径名。

AGGREGATE 关键字指定此函数为一个类似 SUM() 或 MAX() 的聚合函数。AGGREGATE 是在 MySQL 3.23.5 中引入的。

CREATE FUNCTION 要求服务器设置为一个动态连接库（不能是静态库），因为 UDF 机制要求动态连接。关于编写用户定义函数的说明，请参阅 MySQL 参考指南。

D.1.4 CREATE INDEX

CREATE [UNIQUE] INDEX *index_name* ON *tbl_name (index_columns)*

给表 tbl_name 增加一个名为 index_name 的索引。此语句根据有无 UNIQUE 关键字，分别按 ALTER TABLE ADD INDEX 或 ALTER TABLE ADD UNIQUE 语句处理。详细内容请参阅 ALTER TABLE 的相应条目。CREATE INDEX 是在 MySQL 3.22 中引入的。

D.1.5 CREATE TABLE

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options]
    [[IGNORE | REPLACE] select_statement]

create_definition:
    {   col_declaration
      | PRIMARY KEY (index_columns)
      | KEY [index_name] (index_columns)
      | INDEX [index_name] (index_columns)
      | UNIQUE [INDEX] [index_name] (index_columns)
      | [CONSTRAINT symbol] FOREIGN KEY index_name (index_columns)
      | [reference_definition]
      | CHECK (expr) }

col_declaration:
    col_name col_type
        [NOT NULL | NULL] [DEFAULT default_value]
```

```
        [AUTO_INCREMENT] [PRIMARY KEY] [reference_definition]

reference_definition:
    REFERENCES tbl_name [(index_columns)]
        [MATCH FULL | MATCH PARTIAL]
        [ON DELETE reference_option]
        [ON UPDATE reference_option]

reference_option:
    {RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT}
```

CREATE TABLE 语句创建一个名为 *tbl_name* 的新表。如果给出 TEMPORARY 关键字，则创建一个临时表，该表仅在当前客户机连接终止前存在，或者在发布 DROP TABLE 语句前存在。只有创建临时表的客户机才能看见此临时表。

通常，企图创建一个已经存在的表时将产生错误。但在两种情况下不会出错。第一种情况，如果给出 IF NOT EXISTS 子句，则不创建已经存在的这个表，但也不出错。第二种情况，给出 TEMPORARY，则原来名为 *tbl_name* 的表在临时表存在期间对客户机是隐藏的，这样也不会出错。但原来的表对其他客户机仍保持可见。在下一次客户机会话或对临时表明确发布 DROP TABLE 后，原来的表再次可见。

create_definition 清单指定希望创建的列和索引。如果通过后跟 SELECT 语句的方法创建此表，此清单可省略。*table_options* 子句允许指定表的各种属性。如果给出后面的 *select_statement*（以任意 SELECT 语句形式），则利用该 SELECT 语句返回的结果创建该表。后面的段落中将对这些子句进行更详细的介绍。

TEMPORARY 关键字、IF NOT EXISTS 子句、*table_options* 子句以及能够利用 SELECT 语句的结果创建表等都是在 MySQL 3.23 中引入的。

create_definition 可以是列定义、索引定义、FOREIGN KEY 子句、*reference_definition* 或 CHECK 子句。后三者在其数据库系统中是兼容的，但其余的不行。

列定义 *col_declaration* 以列名 *col_name* 和类型 *col_type* 开始，后面可跟几个可选的关键字。列类型可以是附录 B 中列出的任意类型。请参阅该附录中的类型专用属性。可跟在列类型之后的可选关键字如下：

- NULL 或 NOT NULL　指出该列是否可以包含 NULL 值。如果两者都不指出，缺省为 NULL。
- DEFAULT *default_value*　给出该列的缺省值。这种属性不能用于 BLOB 或 TEXT 类型。如果无缺省说明，自动分配一个缺省值。对于可取 NULL 值的列，缺省值为 NULL。对于不能为 NULL 的列，缺省值如下：
  - 对于数值列，除 AUTO_INCREMENT 列外，缺省值为 0。而对于 AUTO_INCREMENT 列，其缺省值为该列的序列中的下一个数。
  - 对于非 TIMESTAMP 的日期和时间类型，缺省值为该类型的“零”值（例如，DATE 类型的“零”值为“0000-00-00”）。对于 TIMESTAMP 类型，缺省值为当前日期和时间。
  - 对于非 ENUM 的串类型，缺省值为空串。对于 ENUM 类型，缺省值为第一个枚举元素。
- AUTO_INCREMENT　此关键字仅用于整数列类型。AUTO_INCREMENT 列在插入

NULL 时是很特殊的，这时插入的值实际为一个大于列中当前最大值的整数。 AUTO_INCREMENT 值缺省时从 1 开始，第一个值可用 UTO_INCREMENT 表选项确明指定。这样的列还必须指定为 UNIQUE 索引或 PRIMARY KEY，而且应该为 NOT NULL 列。每个表中至少有一个 AUTO_INCREMENT 列。

- PRIMARY KEY 指定该列为一个 PRIMARY KEY。
- UNIQUE 指定该列为 UNIQUE 索引列。自 MySQL 3.23 版以来都可指定此属性。

PRIMARY KEY、UNIQUE、INDEX 和 KEY 子句定义索引。 PRIMARY KEY 与 UNIQUE 指定必须包含唯一值的索引。 INDEX 和 KEY 意义相同，它们都指定可以包含重复值的索引。索引基于 *index_columns* 中所指定的列，其中的每个列都必须是 *tbl_name* 中的列。如果有多个列，必须用逗号将它们分隔。对于 CHAR 和 VARCHAR 列，可对列的前缀进行索引。如果未给出索引名 *index_name*，系统会根据第一个索引列的名称自动选一个。

在 MySQL 3.23 以前的版本中，索引列必须定义为 NOT NULL。自 MySQL 3.23 以来，只有 PRIMARY KEY 中的列必须定义为 NOT NULL。

自 MySQL 3.23 以来可使用 *table_options* 子句。如果给出此子句，它可以包含下面清单中的一个或多个选项。如果给出多个选项，不应用逗号分隔它们。除非另有说明，否则每个说明符都可用于所有表类型。

- AUTO_INCREMENT = *n* 为表生成的第一个 AUTO_INCREMENT 值（仅用于 MyISAM 表）。
- AVG_ROW_LENGTH = *n* 表的近似平均行长。MySQL 用此选项来确定数据文件的最大尺寸，只有具有 BLOB 或 TEXT 列的表需要此选项，具有这两种列的表可能会增大超过 4GB。
- CHECKSUM = {0 | 1} 如果设置为 1，MySQL 将对每个表行保持一个校验和。这样对于表的更新将会有某种轻微的损失，但改善了表的校验过程（仅用于 MyISAM 表）。
- COMMENT = “*string*” 表的注释。最大长度为 60 个字符。
- DELAY_KEY_WRITE = {0 | 1} 如果设置为 1，表的索引高速缓存只是偶尔刷新，而不是每个插入操作后都刷新（只用于 MyISAM 表）。
- MAX_ROWS = *n* 计划存储在表中的最多行数。
- MIN_ROWS = *n*

计划存储在表中的最少行数。

- PACK_KEYS = {0 | 1} 如果设置为 1，索引块以较平常高的百分比进行压缩。其一般效果是更新性能受损，但检索性能得以改善。如果设置为 0，索引块不进行特殊的压缩。在此情形，只有具有 8 个字符或更多字符的 CHAR 或 VARCHAR 键值进行压缩（只用于 MyISAM 和 ISAM 表）。
- PASSWORD = “*string*” 为加密表的描述文件指定一个口令。一般不起作用；只对具有一定支持协议的顾客可用。
- TYPE = {ISAM | MYISAM | HEAP} 指定表的存储格式。这些存储格式的特性在第 3 章的“表存储类型说明符”一节介绍。 MySQL 自版本 3.23 以来的缺省格式为 MyISAM。对 MySQL 3.23 以前的版本，CREATE TABLE 总是以 ISAM 格式创建表。

如果给出一个 *select_statement* 子句（如一条 SELECT 查询），该表利用查询结果的内容

进行创建。惟一索引中重复值的行根据指定的是 IGNORE 或 REPLACE，或者忽略或者替换现有行。如果两者都未指定，重复值时将出错，其余记录忽略。

下面创建一个具有三列的表。id 列为 PRIMARY KEY，其余两列组成一个索引：

```
CREATE TABLE customer
(
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    last_name CHAR(30) NOT NULL,
    first_name CHAR(20) NOT NULL,
    PRIMARY KEY (id),
    INDEX (last_name, first_name)
)
```

下面创建一个临时表并使其为一个 HEAP（内存中）表以加取得更快的速度：

```
CREATE TEMPORARY TABLE tmp_table
    (id MEDIUMINT NOT NULL UNIQUE, name CHAR(40))
    TYPE = HEAP
```

下面创建一个表作为另一表的拷贝：

```
CREATE TABLE prez_copy SELECT * FROM president
```

下面只用另一表的部分创建一个表：

```
CREATE TABLE prez_alive SELECT last_name, first_name, birth
    FROM president WHERE death IS NULL
```

如果指定所要创建的表的创建定义并利用后缀 SELECT 语句进行填充，则此定义在表的内容已经插入表中时才使用。例如，可以定义将某个选定的列做成一个 PRIMARY KEY：

```
CREATE TABLE new_tbl (PRIMARY KEY (a)) SELECT a, b, c FROM old_tbl
```

D.1.6 DELETE

```
DELETE [LOW_PRIORITY] FROM tbl_name
    [WHERE where_expr] [LIMIT n]
```

从表 *tbl_name* 中删除行。所删除的行是那些与表达式 *where_expr* 中给出条件相符的行。例：

```
DELETE FROM score WHERE event_id = 14
DELETE FROM member WHERE expiration < CURRENT_DATE
```

如果省略 WHERE 子句，表中所有记录都将被删除。

除非不给出 WHERE 子句，否则 DELETE 将返回所删除记录的数目。在不给出 WHERE 子句的这种特殊情况下，MySQL 将优化删除过程，它将从头开始建立数据和索引文件，而不是一行一行地删除记录。这样做极快，但不给出删除的记录计数，返回的计数值为 0。为了得出真实的删除记录数，应该给出能选中所有记录的 WHERE 子句。例如：

```
DELETE FROM tbl_name WHERE 1 > 0
```

但这样做会有一定的性能损失。

指定 LOW_PRIORITY 使此语句被延迟至无客户机从表中读数据为止。LOW_PRIORITY 是在 MySQL 3.22.5 中引入的。

如果给出 LIMIT 子句，则值 n 规定了将要删除的最大行数。LIMIT 是在 MySQL 3.22.7 中引入的。

D.1.7 DESCRIBE

```
{DESCRIBE | DESC} tbl_name {col_name | pattern}
```

此语句提供与 SHOW COLUMNS 相同的信息。更详细的内容请参阅 SHOW 条目。

D.1.8 DROP DATABASE

```
DROP DATABASE [IF EXISTS] db_name
```

删除给定的数据库。在删除一个数据库后，它就永远没有了，因此要特别小心。如果要删除的数据库不存在，则此语句将失败（除非给出的 IF EXISTS）。在没有恰当的权限时，此语句也将失败。可利用 IF EXISTS 子句禁止在数据库不存在时通常将出现的错误消息。 IF EXISTS 子句是在 MySQL 3.22.2 中引入的。

一个数据库由数据目录下的一个目录表示。如果在该目录中放置了非表文件，这些文件不会被 DROP DATABASE 语句删除。在此情形下，数据库目录本身也不会被删除。

D.1.9 DROP FUNCTION

```
DROP FUNCTION function_name
```

删除以前用 CREATE FUNCTION 装入的用户定义函数。

D.1.10 DROP INDEX

```
DROP INDEX index_name ON tbl_name
```

从表 tbl_name 中删除索引 index_name。此语句处理为一条 ALTER TABLE DROP INDEX 语句。详细内容请参阅 ALTER TABLE 条目。DROP INDEX 是在 MySQL 3.22 中引入的。

D.1.11 DROP TABLE

```
DROP TABLE [IF EXISTS] tbl_name [, tbl_name] ...
```

从数据库中删除指定的表。如果给出 IF EXISTS 子句，则删除不存在的表不会出错。 IF EXISTS 是在 MySQL 3.22.2 中引入的。

D.1.12 EXPLAIN

```
EXPLAIN tbl_name
EXPLAIN select_statement
```

此语句的第一种形式等价于 SHOW COLUMNS FROM tbl_name。详细信息请参阅 SHOW 语句的说明。

EXPLAIN 语句的第二种形式提供了 MySQL 将怎样执行跟在 EXPLAIN 关键字后的 SELECT 语句的信息。例如：

```
EXPLAIN SELECT score.* FROM score, event
    WHERE score.event_id = event.event_id AND event.event_id = 14
```

EXPLAIN 的输出由包含以下各列的一个或多个行组成：

■ table 输出的行所涉及的表

- type MySQL 将执行的连接类型。可能的类型从最好到最差分别为：system、const、eq_ref、ref、range 和 ALL。更好的类型限止更严，这意味着 MySQL 在进行检索时必须考虑的行更少。
- possible_keys MySQL 在查找 rows 列中指定的表中的行时所考虑的索引。NULL 值表示未发现索引。
- key MySQL 查找表中行时将实际使用的索引。NULL 值表示将不使用索引。
- key_len 将使用多长的索引。如果 MySQL 将使用最左前缀索引，这个值可能小于完全索引行的长度。
- ref MySQL 将对此值比较索引值。词 const 或 '???' 表示该比较是针对常量进行的，一个列名表示一个列对列的比较。
- rows MySQL 为完成查询必须检查的表的行数估计。此列中值的积为必须要对所有表进行检查的行组合的总数估计。
- extra Only index 表示 MySQL 可以只利用索引检索表中的信息而不用查看数据文件。where used 表示使用 SELECT 语句的 WHERE 子句中的信息。

D.1.13 FLUSH

FLUSH *flush_option* [, *flush_option*] ...

刷新服务器所用的各种内部高速缓存。flush_option 可以是下列任意一项：

- HOSTS 刷新主机高速缓存。
- LOGS 通过关闭和重新打开刷新日志文件。
- PRIVILEGES 重新装载授权表。
- STATUS 重新初始化状态变量。
- TABLES 关闭高速缓存中任意打开的表。

FLUSH 语句是在 MySQL 3.22.9 中引入的。STATUS 刷新选项是在 MySQL 3.22.11 中引入的。

D.1.14 GRANT

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)] ] ...
    ON {*.* | * | db_name.* | db_name.tbl_name | tbl_name}
    TO user [, user ] ...
    [WITH GRANT OPTION]
```

GRANT 语句给一个或多个 MySQL 用户授予访问权限。priv_type 值指出所授予的权限。它由从下列清单中选出的用逗号隔开的权限类型组成：

权限说明符	该权限允许的操作
ALTER	更改表和索引
CREATE	创建数据库和表
DELETE	从表中删除记录
DROP	删除数据库和表
INDEX	建立或删除索引
INSERT	在表中插入新记录
REFERENCES	未使用

权限说明符	该权限允许的操作
SELECT	检索表中记录
UPDATE	修改表中已有记录
FILE	读写服务器上的文件
PROCESS	查看正在服务器中执行的线程的信息或删除线程
RELOAD	重新装载授权表或刷新日志、主机高速缓存或表高速缓存
SHUTDOWN	关闭服务器
ALL	全部权限，ALL PRIVILEGES 为其同义词
USAGE	一种特殊的“无权限”权限

您总是可以查找或删除自己所拥有的线程。PROCESS 权限允许查找或删除任意线程。

ON 子句指出所授的权限范围有多广，如下表所示：

权限说明符	权限适用的级别
.	全局权限：所有数据库，所有表
*	如果未选择缺省数据库，则为全局权限，否则为当前数据库的数据库级权限
*db_name.**	数据库级权限，指定数据库中所有表
db_name.tbl_name	表级权限，指定表中的所有列
tbl_name	表级权限，缺省数据库中指定表的所有列

如果在 ON 子句中指定了一个表，可通过在 *column_list* 子句中指定一个或多个用逗号分隔的列使权限为专门针对列的权限。

TO 子句指出一个或多个应授予权限的用户。每个用户由具有一个可选的 IDENTIFIED BY 子句的 *user_name@host_name* 中的说明符构成。user_name 可以是一个名字或空串（“”）；后者指定一个匿名用户。*host_name* 可给出作为 localhost、主机名、IP 地址或作为与域名匹配的模式或网络编号。模式字符为“%”和“-”，与用于 LIKE 操作符的意义相同。无主机名指定的 *user_name* 等价于 *user_name*@%。自 MySQL 3.23 以来，*host_name* 可以是一个用 *n.n.n.n./m* 表示法的 IP 号/网络掩码对，其中 n.n.n.n 表示 IP 地址，而 m 表示用于网络编号的位数目。

IDENTIFIED BY 子句，如果给出，将分配一个口令给用户。此口令应该以无格式文本指定，不使用 PASSWORD 函数，与为 SET PASSWORD 语句指定口令相反。如果用户已经存在而且指定了 IDENTIFIED BY 子句，则新的口令将替代旧口令。否则旧口令保持不变。

GRANT 语句是在 MySQL 3.22.11 中引入的。

下面的语句示范了使用 GRANT 语句的某些方法。其他例子请参阅第 11 章。

建立一个用户 paul，他能够从任何主机访问 samp_db 数据库中的所有表。下面的两个语句是等价的，因为在 user 标识符中缺少的主机名部分等价于“%”。

```
GRANT ALL ON samp_db.* TO paul IDENTIFIED BY "secret"
GRANT ALL ON samp_db.* TO paul@% IDENTIFIED BY "secret"
```

建立一个对 menagerie 数据库中的表具有只读权限的用户。该用户可以从 xyz.com 域中的任意主机进行连接：

```
GRANT SELECT ON menagerie.* TO lookonly@%.xyz.com
    IDENTIFIED BY "ragweed"
```

建立对 samp_db 数据库中的 member 表具有全部权限的一个用户。该用户可从一个专门

的主机上进行连接：

```
GRANT ALL ON samp_db.member TO member_mgr@boa.snake.net
    IDENTIFIED BY "doughnut"
```

建立一个超级用户，他可以做任何事情，包括给其他用户授权，但他必须从本地主机进行连接：

```
GRANT ALL ON *.* TO superduper@localhost IDENTIFIED BY "homer"
    WITH GRANT OPTION
```

建立 menagerie 数据库的一个匿名用户，他能从本地主机不用口令进行连接：

```
GRANT ALL ON menagerie.* TO ""@localhost
```

D.1.15 INSERT

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE] [INTO]
    tbl_name [(column_list)] VALUES (expr [, expr] ...) [, (...)] ...
INSERT [LOW_PRIORITY | DELAYED] [IGNORE] [INTO]
    tbl_name [(column_list)] SELECT ...
INSERT [LOW_PRIORITY | DELAYED] [IGNORE] [INTO]
    tbl_name SET col_name=expr [, col_name=expr] ...
```

在一个已有的表 *tbl_name* 中插入行，返回所插入的行数。MySQL 3.22.5 以前的版本需要 INTO 关键字。

LOW_PRIORITY 使语句延迟至无客户机从表中读取数据时执行。LOW_PRIORITY 是在 MySQL 3.22.5 中引入的。

RELAYED 为以后的插入将行放入队列中，且此语句立即返回，以便客户机可以继续执行而不用等待。但在此情形下，LAST_INSERT_TO( ) 将不返回表中 AUTO_INCREMENT 列的 AUTO_INCREMENT 值。DELAYED 插入是在 MySQL 3.22.15 中引入的。

如果指定 IGNORE，现有行中唯一键值重复的行被丢弃。如果出现重复值而未指定 IGNORE，将出错且不再有更多的行被插入。IGNORE 是在 MySQL 3.22.10 中引入的。

INSERT 语句的第一种形式需要一个 VALUES( ) 列表，它给出所有要插入的值。如果不给出 *column_list* ，VALUES( ) 列表必须对表中每列指定一个值。如果给出由一个或多个以逗号分隔的列构成的 *column_list*，则必须在 VALUES( ) 列表中给每列指定一个值。未在列表中指定的列被设置为它们的缺省值。自 MySQL 3.22.5 以来，可以给出多个值的列表，这允许在单个 INSERT 语句中插入多个行。自 MySQL 3.23.3 以来，VALUES( ) 列表可以为空，这将插入一个每列都设置为缺省值的行。

INSERT 语句的第二种形式根据 SELECT 语句检索记录并将它们插入 *tbl_name* 中。SELECT 语句必须选择 tbl_name 表的所有列，或者如果给出了列的列表，将选择 *column_list* 中所指定的所有列。如果给出列的列表，在该列表中未指定的任何列将被设置为缺省值。

自 MySQL 3.22.10 以来可使用 INSERT 的第三种形式，这种形式将由相应表达式给出值赋予 SET 子句中指定的列。未指定的列设置为缺省值。

```
INSERT INTO absence (student_id, date) VALUES(14,"1999-11-03"),(34,NOW())
INSERT INTO absence SET student_id = 14, date = "1999-11-03"
INSERT INTO absence SET student_id = 34, date = NOW()
INSERT INTO score (student_id, score, event_id)
    SELECT student_id, 100 AS score, 15 AS event_id FROM student
```

D.1.16 KILL

```
KILL thread_id
```

取消具有给定 thread_id 的服务器线程。要取消线程，除非取消的线程是您自己所拥有的，否则必须具有 PROCESS 权限。

mysqladmin kill 命令完成相同的操作，但在命令行上允许指定多个线程的 ID 值。而 KILL 语句只允许指定一个 ID。

引语句是在 MySQL 3.22.9 中引入的。

D.1.17 LOAD DATA

```
LOAD DATA [LOW_PRIORITY] [LOCAL] INFILE 'file_name'
    [IGNORE | REPLACE]
    INTO TABLE tbl_name
    import_options
    [IGNORE n LINES]
    [(column_list)]
```

LOAD DATA 从文件 *file_name* 中读取记录并将它们成批装入表 *tbl_name* 中。这样比 INSERT 快。

LOW_PRIORITY 使本语句延迟至无客户机从表中读取数据为止。LOW_PRIORITY 是在 MySQL 3.23 中引入的。

一般情况下，此文件是由服务器在服务器主机上直接读入的。在此情形下，必须具有 FILE 权限而且相应文件必须或者位于缺省数据库的数据库目录中，或者为完全可读的。如果给出 LOCAL，则客户机在客户机主机上读取该文件并将其内容通过网络发送到服务器。在此情形下，不需要 FILE 权限。LOCAL 是在 MySQL 3.22.15 中开始起作用的。

如果不给出 LOCAL，服务器按如下方法对文件定位：

- 如果 *file_name* 为绝对路径，服务器从根目录开始查找该文件。
- 如果 *file_name* 为一个相对路径，则其解释要根据该名称是否包含单一成分来进行。如果是包含单一成分，服务器将在缺省数据库的数据库目录中查找该文件。如果该文件名包含多个成分，服务器将在服务器的数据目录中从头开始查找该文件。如果给出 LOCAL，则文件名如下解释：
  - 如果 *file_name* 为绝对路径，则客户机从根目录开始查找文件。
  - 如果 *file_name* 为一个相对路径，则客户机从当前目录开始查找文件。

在惟一索引上重复值的行根据给出的是 IGNORE 或 REPLACE，或者忽略或者替换现有行。如果两者都未指定，将出错，并忽略其余记录。

import_options 子句指出数据的格式。子句中可用的选项也适用于 SELECT ... INTO OUTFILE 语句。*import_options* 的语法为：

```
[FIELDS
    [TERMINATED BY 'string']
    [OPTIONALLY] ENCLOSED BY 'char']
    [ESCAPED BY 'char' ]]
[LINES TERMINATED BY 'string']
```

string 和 *char* 值可能包括下列转义序列以给出特殊字符：

序列	说　明
\0	ASCII 0
\b	退格
\n	换行（不走纸）
\r	回车换行
\s	空格
\t	Tab（制表符）
\'	单引号
\"	双引号
\\	反斜杠

自 MySQL 3.22.10 以来，还可使用十六进制常量表示任意字符。例如，LINES TERMINATED BY 0x02 表示该行由 Ctrl-B（ASCII 2）字符结束。

如果给出 FIELDS，则至少应该给出 TERMINATED BY、ENCLOSED BY 或 ESCAPED BY 子句中的某一条，并且给出不止一个子句，必须按这里的顺序给出。如果 FIELDS 和 LINES 都给出，FIELDS 必须在 LINE 之前。FIELDS 子句的成分如下使用：

- TERMINATED BY 给出限定行中值的字符。
- ENCLOSED BY指定引用字符，如果给出它，则将从字段值的最后去掉。不管是否给出OPTIONALLY，都会出现这种情况。对于输出 (SELECT...INTO OUTFILE), ENCLOSED BY 字符用来括起输出行中的字段值。如果给出 OPTIONALLY，则只将 CHAR 和 VARCHAR 列的值括起来。

为了在输入字段值内包含一个 ENCLOSED BY 字符，应该双写它或在其前面放置一个 ESCAPED BY 字符。否则，它将被理解为表示字段结束。对于输出，字段值中的 ENCLOSED BY 字符跟在一个 ESCAPED BY 字符之后。

- ESCAPED BY 字符用来指出特殊字符的转义。在下面的例子中，假定转义字符为反斜杠（“ \ ”）。对于输入，“ \N ”（反斜杠 N）解释为空格，而“ \0 ”（反斜杠 ASCII “ 0 ”）解释为一个零值的字节。对于其他字符，转义字符被去掉，按字面使用所跟的字符。例如，“ \" ” 解释为一个双引号，即使字段值是用双引号括起来的也是如此。

对于输出，用转义字符来将 NULL 和零值字节编码为“ \N ”“ \0 ”。此外，ESCAPED BY 和 ENCLOSED BY 字符跟在转义字符之后，就像字段和行结束串的第一个字符一样。如果 ESCAPED BY 字符为空（ESCAPED BY ' '），则不做转义工作。为了给出“ \ ”的转义字符，可双写它（ESCAPED BY '\\'）。

LINES TERMINATED BY 值指出满足表示行结束的字符。

如果既不给出 FIELDS 又不给出 LINES，则将它们缺省指定为如下所示：

```
FIELDS
    TERMINATED BY '\t'
    ENCLOSED BY ''
    ESCAPED BY '\\'
LINES TERMINATED BY '\n'
```

换句话说，行中的字段是用制表符限定的，无需括起来，反斜杠作为转义字符对待，而且行以换行符终止。

如果 FIELDS 子句的 TERMINATED BY 和 ENCLOSED BY 值皆为空，则使用定长的行

格式，字段之间不用定界符。列值利用列的显示宽度读取（或输出写）。例如，VARCHAR(15) 和 MEDIUMINT(5) 分别按 15 个字符和 5 个字符读取。NULL 值按空格串写入。

如果给出 IGNORE n LINES 子句，则输入的前 n 行被丢掉。例如，如果数据文件具有一个列标题行，不希望将其输入数据库表中，可使用 IGNORE 1 LINES 将其去掉：

```
LOAD DATA LOCAL INFILE "my_tbl.txt" INTO TABLE my_tbl IGNORE 1 LINES
```

如果不给出 *column_list*，则假定输入行包含表中每列的值。如果给出由一个或多个逗号分隔的列名清单，则输入行应该包含每个指定列的值。清单中没有的列设置为它们的缺省值。如果某个输入行中的值达不到所期望的值的数目，缺少的值的列将设置为其缺省值。

如果有一个在 Windows 中建立的用制表符限定的文本文件，可利用缺省的列分隔符，但行大概是用回车/换行对结束的。为了加载文件，应该给出不同的行结束符（’ \r ’ 表示一个回车换行，而 ‘ \n ’ 表示换行）：

```
LOAD DATA LOCAL INFILE "my_tbl.txt" INTO TABLE my_tbl
    LINES TERMINATED BY "\r\n"
```

不幸的是，对于在 Windows 中建立的文件，如果建立该数据文件的程序使用了古怪的 MS-DOS 约定，在文件的结尾利用 Ctrl-Z 字符表示文件结束，则可能会在数据库中以不正确的记录作为结束记录。这种情况下，或者不用这样的程序来写入该文件，或装载文件后将不正确的记录删掉。

逗号分隔值（CSV）的文件在字段间有逗号，而字段可能用双引号括起来。假定行在结尾处有换行符，LOAD DATA 语句按如下装载一个文件：

```
LOAD DATA LOCAL INFILE "my_tbl.txt" INTO TABLE my_tbl
    FIELDS TERMINATED BY "," ENCLOSED BY "\""
```

下面的语句示出怎样读取一个字段由 Ctrl-A（ASCII 1）字符分隔，行用 Ctrl-B（ASCII 2）字符结束的文件。

```
LOAD DATA LOCAL INFILE "my_tbl.txt" INTO TABLE my_tbl
    FIELDS TERMINATED BY 0x01 LINES TERMINATED BY 0x02
```

D.1.18 LOCK TABLES

```
LOCK TABLES lock_list
```

获得在 lock_list 中指定的表上的一个锁，如果有必要，则一直等待获得所有锁。 LOCK TABLE 为 LOCK TABLES 的同义词。

lock_list 由一个或多个逗号分隔的项构成，每个项的语法如下：

```
tbl_name [AS alias_name] {READ | [LOW_PRIORITY] WRITE}
```

READ 表示一个读锁，这个锁锁定其他希望写表的客户机，但允许其他客户机读取该表。WRITE 表示一个写锁，它阻塞所有其他客户机，不管它们是读表或还是写表。如果此请求正等待另一读此表的客户机，则 LOW_PRIORITY WRITE 锁请求允许其他读取程序读此表。这个锁直到无其他读取程序时才能获得。

LOCK TABLES 允许指定一个别名，以便在后面的查询中能利用此别名锁住准备引用的表。如果在某个查询中多次使用一个表，那么必须对表的每个实例获得一个锁，按需要锁定

别名。

LOCK TABLES 释放当前占用的任何锁。因而，为了锁定多个表，必须利用单一的 LOCK TABLES 语句锁定它们。一个客户机占用的任何锁，当此客户机终止时都将被自动释放。

LOW_PRIORITY 是在 MySQL 3.22.5 中引入的。

```
LOCK TABLES student READ, score WRITE, event READ
LOCK TABLE member READ
```

D.1.19 OPTIMIZE TABLE

```
OPTIMIZE TABLE tbl_name
```

释放表中浪费的空间。DELETE、REPLACE 与 UPDATE 语句可能会在表中产生无用的空间，特别是那种具有可变长行的表更是如此。OPTIMIZE TABLE 语句消除浪费的空间，得到一个所占空间更小的表。这条语句是在 MySQL 3.22.7 中引入的。其他表维护选项安排在 MySQL 3.23 系列中增加，因此对于最新增加的选项应该查看联机 MySQL 参考指南。

D.1.20 REPLACE

```
REPLACE [LOW_PRIORITY | DELAYED] [INTO]
    tbl_name [(column_list)] VALUES (expr [, expr] ...) [, (...)] ...
REPLACE [LOW_PRIORITY | DELAYED] [INTO]
    tbl_name [(column_list)] SELECT ...
REPLACE [LOW_PRIORITY | DELAYED] [INTO]
    tbl_name SET col_name=expr [, col_name=expr] ...
```

REPLACE 语句类似于 INSERT，但有例外，在 INSERT 语句中，如果要插入的行具有一个与表中已有行重复的 UNIQUE 索引值，则在插入新行前将删除旧行。而 REPLACE 语句不这样，所以 REPLACE 语句中没有 IGNORE 子句选择。

D.1.21 REVOKE

```
REVOKE priv_type [(column_list)] [, priv_type [(column_list)] ...]
    ON {*.* | * | db_name.* | db_name.tbl_name | tbl_name}
    FROM user [, user ] ...
```

此语句取消指定用户的权限。*priv_type*、*column_list* 和 *user* 子句按与 GRANT 语句相同的方式指定。ON 子句中也允许相同的说明符。

REVOKE 并不从 user 授权表中删除用户。这表示相应的用户仍然可以连接到 MySQL 服务器。为了删除所有用户，必须从授权表中手工删除用户的条目。

REVOKE 语句是在 MySQL 3.22.11 中引入的。

下面的 REVOKE 语句取消 superduper@localhost 的所有权限：

```
REVOKE ALL ON *.* FROM superduper@localhost
```

取消允许 member_mgr 用户修改 samp_db 数据库中 member 表的权限：

```
REVOKE INSERT,DELETE,UPDATE ON samp_db.member FROM member_mgr@boa.snake.net
```

取消本地主机上的匿名用户对 menagerie 数据库中单个表拥有的所有权限：

```
REVOKE ALL ON menagerie.pet FROM ""@localhost
```

D.1.22 SELECT

```
SELECT
    [select_options]
    select_list
    [INTO OUTFILE 'file_name' export_options]
    [FROM tbl_list
    [WHERE where_expr]
    [GROUP BY column_list]
    [HAVING where_expr]
    [ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC] , ...]
    [LIMIT [offset,] rows]
    [PROCEDURE procedure_name] ]
```

此语句从指定的表中检索行。实际上，除了 SELECT 关键字和 select_list 子句以外，本语句中所有成分都是可选的，这样考虑到了只是进行简单表达式求值的语句。例如：

```
SELECT "one plus one =", 1+1
```

如果给出 *select_options* 子句，则它可能包含下列选项：

- ALL

  DISTINCT

  DISTINCTROW

  这几个关键字控制重复行是否返回， ALL 使所有行返回。这是缺省的， DISTINCT 和 DISTINCTROW 指出重复行应从结果中删除。

- HIGH_PRIORITY 如果一般情况下必须等待，指定 HIGH_PRIORITY 将给语句一个更高的优先权。如果其他语句，如 INSERT 或 UPDATE 由于另外的语句正在读 SELECT 语句中给出的表而进行等待，这时 HIGH_PRIORITY 将使 SELECT 语句具有高于这些写语句的优先权。这只在您知道 SELECT 语句将快速执行且必须立即进行的情况下才这样做，因为这样做将把写语句放在后面执行。 HIGH_PRIORITY 是在 MySQL 3.22.9 中引入的。

- SQL_BIG_RESULT

  SQL_SMALL_RESULT

  这些关键字指定结果集大或小，它向优化程序提供了可用来更有效地处理查询的信息。 SQL_SMALL_RESULT 是在 MySQL 3.22.12 中引入的。而 SQL_BIG_RESULT 是在 MySQL 3.23 中引入的。

- STRAIGHT_JOIN 强制表以 FROM 子句中指定的顺序进行连接。如果认为优化程序不能做出最好选择时，可使用这个关键字。

select_list 子句指定要返回的输出列。列之间应该用逗号分隔。这里的列可能涉及表列或表达式。任何列都可以用 AS alias_name 语法指定一个别名。然后此别名成为输出中的列名，而且还可以在 GROUP BY、ORDER BY 和 HAVING 子句中引用。但不能在 WHERE 子句中引用别名。

特殊的符号 * 表示“ FROM 子句中指定的表中所有列”，而 tbl_name.* 表示“给定表中所有的列”。

SELECT 语句的结果可利用 INTO ' file_name' OUTFILE 子句写入一个文件中。您必须具有 FILE 权限，并且该文件必须不是已存在的文件。此文件名用与 LOAD DATA 读非 LOCAL

文件时所用的相同规则来解释。 export_options 子句的语法与 LOAD DATA 语句的 import_options 子句的相同。更详细的内容请参阅 LOAD DATA 条目。

FROM 子句指定一个或多个表，必须从这些表中选择行。使用 SELECT 语句时，MySQL 支持下列 JOIN 类型：

```
tbl_list:
    tbl_name
    tbl_list, tbl_name
    tbl_list [CROSS] JOIN tbl_name
    tbl_list INNER JOIN tbl_name
    tbl_list STRAIGHT_JOIN tbl_name
    tbl_list LEFT [OUTER] JOIN tbl_name ON conditional_expr
    tbl_list LEFT [OUTER] JOIN tbl_name USING (column_list)
    tbl_list NATURAL LEFT [OUTER] JOIN tbl_name
    { oj tbl_list LEFT OUTER JOIN tbl_name ON conditional_expr }
```

连接类型从指定的表中按下述描述选择行，虽然实际返回到客户机的行可能会受 WHERE、HAVING 或 LIMIT 子句所限定。

对于由其自身给出的表，SELECT 从此表中检索行。

如果指定多个表并用逗号将它们隔开， SELECT 返回这些表中行的所有可能组合。使用 JOIN、CROSS JOIN 或 INNER JOIN 等价于使用逗号。STRAIGHT_JOIN 也类似，但强制优化程序按表给出的顺序对表进行连接。如果认为优化程序不能做出最好的选择时，可以使用它。

LEFT JOIN 从连接的表中检索行，但强制对左边表中的每一行生成一行，即使右边表中没有匹配的行也是如此。在无匹配的行时，右边表中的列作为空值返回。根据 ON conditional_expr 子句或 USING（column_list） 子句中给定的条件确定匹配的行。conditional_expr 是一种可用于 WHERE 子句的表达式形式。column_list 由一个或多个逗号分隔的列名构成，其中每一列都必须是出现在两个连接表中的列。 LEFT OUTER JOIN 等价于 LEFT JOIN。其语法也同样以 oj 开始（这是为与 ODBC 兼容而引入的）。

NATURAL LEFT JOIN 等价于 LEFT JOIN USING (column_list)，其中 column_list 给出两个表共有的每一列。

表可在 FROM 子句中用 tbl_name alias_name 或 tbl_name AS alias_name 语法指派别名。SELECT 语句的任何地方的引用一个别名化的列必须利用别名来进行。

WHERE 子句给出一个用于从 FROM 子句指定的表中选择的行的表达式。不能满足表达式所给出条件的行将被丢弃。结果还可以进一步由 HAVING 和 LIMIT 子句进行限制。

GROUP BY column_list 子句根据列表中给出的列对结果集进行分组。如果在 select_list 子句中给出诸如 COUNT( ) 或 MAX( ) 一类的汇总函数时，使用这个子句。列可用列名或别名或在 select_list 内用位置引用。列的位置从 1 开始编号。

HAVING 子句在行已经满足了 WHERE 子句中指定的条件后，给出一个用来限制行的辅助表达式。不满足 HAVING 条件的行被丢弃。对于涉及不能在 WHERE 子句中测试的汇总函数的表达式，HAVING 子句是很有用的。但如果条件既在 WHERE 子句中又在 HAVING 子句中都是合法的，最好将其放在 WHERE 子句内，它在 WHERE 子句中容易为优化程序所分析。

ORDER BY 确定怎样对结果进行排序。与 GROUP BY 一样，列可由列名或别名引用，

或者按列选择列表内的位置进行引用。输出列缺省时按升序排序。为了明确地指定列的排序次序，可在列指示符后跟 ASC（升序）或 DESC（降序）。自 MySQL 3.23.2 以来，也可以指定一个表达式作为排序列。例如，ORDER BY RAND( ) 以随机次序返回行。

LIMIT 子句可用来从 SELECT 语句的结果中选择部分行。LIMIT n 返回前 n 行。LIMIT m, n 返回从第 m 行开始的 n 行。对于 LIMIT，行从 0 而不是从 1 开始编号。

PROCEDURE 命名一个过程，结果集在返回给客户之前先送给此过程。自 MySQL 3.23 以来，可利用 PROCEDURE ANALYSE( ) 获得列选择表中给出的列中数据的特征。

下面的语句演示了某些使用 SELECT 语句的方法。更多的例子请参阅第 1 章。

选择表的所有内容：

```
SELECT * FROM president
```

选择所有内容，但按名字进行排序：

```
SELECT * FROM president ORDER BY last_name, first_name
```

选择在“1900-01-01”出生或之后出生的总统的记录：

```
SELECT * FROM president WHERE birth >= "1900-01-01"
```

选择在“1900-01-01”出生或之后出生的总统的记录，但按出生的顺序进行排序：

```
SELECT * FROM president WHERE birth >= "1900-01-01" ORDER BY birth
```

确定 member 表中的行给出哪些州：

```
SELECT DISTINCT state FROM member;
```

从 member 表中选择行并将列作为逗号分隔的值写入一个文件：

```
SELECT * INTO OUTFILE "/tmp/member.txt"
    FIELDS TERMINATED BY "," FROM member
```

选择某个特定学分事件的前五个学分：

```
SELECT * FROM score WHERE event_id = 9 ORDER BY score DESC LIMIT 5
```

D.1.23 SET

```
SET [OPTION] option_setting, ...
```

SET 语句用来指定各种选项。最好是省去词 OPTION，因为它在 MySQL 的未来版本中将被删除。

option_setting 值可为下列清单中的任一项：

- CHARACTER SET {*charset_name* | DEFAULT} 指定客户机所用的字符集。客户机接收和发送的串用这个字符集进行映射。当前可用的惟一字符集是 cp1251_koi8。字符集名 DEFAULT 恢复缺省字符集。

```
SET CHARACTER SET cp1251_koi8
SET CHARACTER SET DEFAULT
```

- INSERT_ID = *n* 指定下一条 INSERT 语句插入一行时，AUTO_INCREMENT 列所用的值。

```
SET INSERT_ID = 1973
```

- LAST_INSERT_ID = *n* 指定 LAST_INSERT_ID( ) 返回的值。此选项用于更新日志的处理。

```
SET LAST_INSERT_ID = 48731
```

■ PASSWORD [FOR *user*] = PASSWORD(" *password* ") 在没有 FOR 子句时，设置当前用户的口令为 “ password ”。在有 FOR 子句时，设置给定用户的口令。为了能够设置其他用户的口令，必须具有修改 *mysql* 数据库的权限。 *user* 是以 *user_name@host_name* 形式，利用与GRANT 语句所接受的 *user_name* 和 *host_name* 相同类型的值给出的。

```
SET PASSWORD = PASSWORD("secret")
SET PASSWORD FOR paul = PASSWORD("secret")
SET PASSWORD FOR paul@localhost = PASSWORD("secret")
SET PASSWORD FOR bill@%.bigcorp.com = PASSWORD("old-sneep")
```

■ SQL_AUTO_IS_NULL = {0 | 1} 如果此选项设置为 1，最后插入的包含一个 AUTO_INCREMENT 值的行可利用 WHERE *auto_inc_col* IS NULL 形式的 WHERE 子句来选择。如 Access 这样的 ODBC 程序使用此选项。此选项的缺省设置为 1，它是在 MySQL 3.23.5 中引入的。

```
SET SQL_AUTO_IS_NULL = 0
```

■ SQL_BIG_SELECTS = {0 | 1} 如果此选项设置为 1，则允许返回多于 max_join_size 行的 SELECT 语句。否则，查询将被中止。此选项的缺省设置为 1。

```
SET SQL_BIG_SELECTS = 0
```

■ SQL_BIG_TABLES = {0 | 1} 如果此选项设置为 1，则所有临时表都存放在磁盘上而不是存入在内存中。性能降低，但需要大临时表的 SELECT 语句将不会出现 “ table full（表满）” 错误。此选项的缺省设置为 0（将临时表装在内存中）。此选项一般对于 MySQL 3.23 及以上的版本不需要。

```
SET SQL_BIG_TABLES = 1
```

■ SQL_LOG_OFF = {0 | 1} 如果此选项设置为 1，当前客户机的需求不记录在通常的日志文件中。如果设置为 0，则允许客户机记录日志。如果要此语句起作用，客户机必须具有 PROCESS 权限。对更新日志的记录不受影响。

```
SET SQL_LOG_OFF = 1
```

■ SQL_LOG_UPDATE = {0 | 1} 除影响更新日志的记录和不影响常规日志外，此选项类似于 SQL_LOG_OFF（包括要求 PROCESS 权限）。

SQL_LOG_UPDATE 是在 MySQL 3.22.5 中引入的。

```
SET SQL_LOG_UPDATE = 1
```

■ SQL_LOW_PRIORITY_UPDATES = {0 | 1} 如果此选项设置为 1，修改表内容的语句（DELETE、INSERT、REPLACE、UPDATE）将等待直到无 SELECT 语句活动时为止，或等待直到该表挂起为止。在其他语句活动中到达的 SELECT 语句立即开始执行而不用等待较低优先级的修改语句。

SQL_LOW_PRIORITY_UPDATES 是在 MySQL 3.22.5 中引入的。

```
SET SQL_LOW_PRIORITY_UPDATES = 0
```

■ SQL_SELECT_LIMIT = {*n* | DEFAULT} 指定从 SELECT 语句返回记录的最多个数。在一条语句中直接给出 LIMIT 子句将优于此选项。此选项的缺省值为 “ 不限制 ”。如果曾经做过更改，则设置为 DEFAULT 将恢复缺省设置。

```
SET SQL_SELECT_LIMIT = 100000
SET SQL_SELECT_LIMIT = DEFAULT
```

■ SQL_WARNINGS = {1 | 0} 如果设置为 1，MySQL 即使对单行插入也报告警告计数。一般，只对插入多行的 INSERT 语句报告警告计数。此选项的缺省设置为 0。SQL_WARNINGS 是在 MySQL 3.22.11 中引入的。

```
SET SQL_WARNINGS = 1
```

■ TIMESTAMP = {*timestamp_value* | DEFAULT} 指定一个 TIMESTAMP 值。此选项用于更新日志处理。

```
SET TIMESTAMP = DEFAULT
```

D.1.24 SHOW

```
SHOW COLUMNS FROM tbl_name [FROM db_name] [LIKE pattern]
SHOW DATABASES [LIKE pattern]
SHOW GRANTS FOR user_name
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW PROCESSLIST
SHOW STATUS
SHOW TABLE STATUS [FROM db_name] [LIKE pattern]
SHOW TABLES [FROM db_name] [LIKE pattern]
SHOW VARIABLES [LIKE pattern]
```

SHOW 的各种形式提供了关于数据库、表、列及索引的信息，或提供了关于服务器运行的信息。有几种形式具有一个可选的 FROM db_name 子句，允许指定要显示信息的数据库。如果不给出该子句，则针对的是缺省数据库。有的形式允许可选的 LIKE pattern 子句以限定与模式相匹配的值的输出。这里的模式应该为一个串或包含“ % ”或“ _ ”字符的 SQL 模式。

1. SHOW COLUMNS

SHOW COLUMNS 语句列出给定表的列。SHOW FIELDS 为 SHOW COLUMNS 的同义词。

```
SHOW COLUMNS FROM president
SHOW FIELDS FROM president
SHOW COLUMNS FROM president FROM samp_db
SHOW COLUMNS FROM tables_priv FROM mysql LIKE "%priv"
```

SHOW COLUMN 的输出包含下面各列：

■ Field 表列名。

■ Type 列类型

■ Null 列是否可包含 NULL。此列或者为 YES，或者为空白（无）。

■ Key 列是否进行了索引。

■ Default 列的缺省值。

■ Extra 关于列的额外信息。

■ Privileges 您对列所具有的权限。只在 MYSL 3.23 以后才显示此信息。

2. SHOW DATABASES

SHOW DATABASES 语句列出服务器主机上可用的数据库。

```
SHOW DATABASES
SHOW DATABASES LIKE "test%"
```

3. SHOW GRANTS

SHOW GRANTS 语句显示特定用户的权限信息，它应该以 *user_name$host_name* 的形式给出，利用与 GRANT 语句接受的 *user_name* 和 *host_name* 相同类型的值。SHOW GRANTS 是在 MySQL 3.23.4 中引入的。

```
SHOW GRANTS FOR root@localhost
SHOW GRANTS FOR ""@pit-viper.snake.net
```

4. SHOW INDEX

SHOW INDEX 语句显示表索引的有关信息。SHOW KEYS 为 SHOW INDEX 的同义词。

```
SHOW INDEX FROM score
SHOW KEYS FROM score
SHOW INDEX FROM samp_db.score
SHOW INDEX FROM score FROM samp_db
```

SHOW INDEX 的输出包含下面的列：

- Table 包含相应索引的表名。
- Non_unique 如果索引可以包含重复值则此值为 1，否则为 0。
- Key_name 索引名。
- Seq_in_index 索引中列的数目。列从 1 开始编号。
- Column_name 列名。
- Collation 列在索引中的排序顺序。此值可能是 A（升序）、D（降序）或 NULL（未排序）。降序键至今尚不能使用，但以后会实现。
- Cardinality 索引中唯一值的数目。这个值通过运行具有 --analyze 选项的 myisamchk 或 isamchk 来更新。
- Sub_part 如果仅对列的一个前缀进行索引，则为此前缀的长度。如果对整个列都进行了索引，此列值为 NULL。
- Packed 键怎样压缩，如果未压缩，则此列的值为 NULL。此列仅在 MySQL 3.23 或以上的版本中显示。
- Comment 为关于索引的内部注释所保留。此列仅在 MySQL 3.23或以上的版本中显示。

5. SHOW PROCESSLIST

SHOW PROCESSLIST 显示服务器中执行的线程的信息。其输出包含以下的列：

- Id 客户机线程 ID 号。
- User 与线程有关的客户机名。
- Host 客户机从其进行连接的主机。
- db 线程的缺省数据库
- Command 线程执行的命令。
- Time 当前在线程中执行的命令所用的时间量，以秒计。
- State 在处理一条 SQL 语句时有关 MySQL 状态的信息。这个值对于报告 MySQL 的问题，或通过邮件清单提供一个关于为什么线程在某种状态下呆了很长时间的问题时，很有用。
- Info 正在执行的查询。

6. SHOW STATUS

SHOW STATUS 语句显示服务器的状态变量及其值：

- Aborted_clients 由于客户机未正常关闭连接而中止的客户机连接数目。
- Aborted_connects 企图连接到服务器的失败次数。
- Connections 企图连接到服务器的次数（包括成功的和失败的）。如果这个数相当大，可能的话应该考虑在您的客户机中使用永久连接。
- Created_tmp_tables 处理查询时创建的临时表的数目。
- Delayed_errors 处理 INSERT DELAYED 行时出现的错误数。
- Delayed_insert_threads 当前 INSERT DELAYED 处理程序的数目。
- Delayed_writes 已写入的 INSERT DELAYED 行的数目。
- Flush_commands 已执行的 FLUSH 命令数。
- Handler_delete 从表中删除一行的请求数目。
- Handler_read_first 从表中读取第一行的请求数目。
- Handler_read_key 基于索引值读取一行的请求数目。
- Handler_read_next 按索引顺序读取下一行的请求数目。
- Handler_read_rnd 基于行的位置读取一行的请求数目。如果这个数目比所有其他报告的数目大得多，表示或者执行了许多需要全表扫描的查询，或者未恰当地使用索引。
- Handler_update 更新表中一行的请求数目。
- Handler_write 在表中插入一行的请求数目。
- Key_blocks_used 索引高速缓存中使用的块数。
- Key_read_requests 从索引高速缓存中读一个块的请求数目。
- Key_reads 从磁盘物理读索引块的数目。
- Key_write_requests 写一个块到索引高速缓存的请求数目。
- Key_writes 物理写索引块到磁盘的数目。
- Max_used_connections 曾经同时打开的连接的最大数目。
- Not_flushed_delayed_rows 等待 INSERT DELATED 查询写入的行数。
- Not_flushed_key_blocks 已经修改但未进行磁盘刷新的键高速缓存中的块数。
- Open_files 打开的文件的数目。
- Open_streams 打开的流的数目。
- Open_tables 打开的表的数目。
- Opened_tables 曾经打开过的表的数目。如果这个数太大，可能需要增加表高速缓存的尺寸。
- Questions 服务器曾经接到过的查询数目（包括成功的和未成功的查询）。
- Slow_queries 执行时间比 long_query_time 长的查询数目。
- Threads_connected 当前打开的连接数目。
- Threads_running 未睡眠的线程数目。
- Uptime 服务器启动以来的秒数。

7. SHOW TABLE STATUS

SHOW TABLE STATUS 语句显示数据库中表的描述性信息。SHOW TABLE STATUS 是在 MySQL 3.23 中引入的。

```
SHOW TABLE STATUS
SHOW TABLE STATUS FROM samp_db
SHOW TABLE STATUS FROM mysql LIKE "%priv"
```

SHOW TABLE STATUS 的输出包括以下各列：

- Name 表名。
- Type 表的类型，可能是 NISAM（ISAM）、MyISAM 或 HEAP。
- Row_format 行的存储格式，可以是 Fixed（定长行）、Dynamic（行可变长）或 Compressed。
- Rows 表中行的数目。
- Avg_row_length 表行所用字节的平均数目。
- Data_length 表数据文件的实际字节数。
- Max_data_length 表数据文件可使用的最大字节数。
- Index_length 索引文件的实际字节数。
- Data_free 数据文件中未用的字节数。如果这个数非常大，应该对相应的表发布一条 OPTIMIZE TABLE 语句。
- Auto_increment AUTO_INCREMENT 列将产生的下一个值。
- Create_time 表创建的时间。
- Update_time 表最近被修改的时间。
- Check_time 最近用 myisamchk 检查或修复表的时间。如果该表从未进行过检查或修复，则此列的值为 NULL。
- Create_options 在创建该表的 CREATE TABLE 语句的 table_options 子句中给出的其他选项。
- Comment 创建表时给出的注释文本。

8. SHOW TABLES

SHOW TABLES 语句显示数据库中表的清单。

```
SHOW TABLES
SHOW TABLES FROM samp_db
SHOW TABLES FROM mysql LIKE "%priv"
```

9. SHOW VARIABLES

SHOW VARIABLES 语句显示服务器变量及其值的一个清单。这些变量在附录 E 的 mysqld 条目中描述。

```
SHOW VARIABLES
SHOW VARIABLES LIKE "%thread%"
```

D.1.25 UNLOCK TABLES

```
UNLOCK TABLES
```

此语句释放当前客户机所占用的所有表锁。UNLOCK TABLE 为 UNLOCK TABLES 的同义词。

D.1.26 UPDATE

```
UPDATE [LOW_PRIORITY] tbl_name
```

```
SET col_name=expr [, col_name=expr ] ...
[WHERE where_expr] [LIMIT n]
```

修改表 *tbl_name* 中已有行的内容。被修改的行为那些由 WHERE 子句中给出的表达式选中的行。对于那些选中的行，SET 子句中给出的每列都被设置为相应表达式的值。

```
UPDATE member SET expiration = NULL, phone = "197-602-4832"
    WHERE member_id = 14
```

如果未给出 WHERE 子句，将更新表中所有行。

UPDATE 返回更新的行数。但是除非确实对行中某些列的值进行了更改，否则该行并不认为已进行了更新。设置某列的值为原来该列已经含有的值对行不产生影响。如果应用程序确实需要知道有多少行与 WHERE 子句相符而不管 UPDATE 是否实际修改了值，那么应该在建立与服务器的连接时，指定 CLIENT_FOUND_ROWS 标志。请参阅附录 F 中 mysql_real_connect( ) 函数的相应条目。

LOW_PRIORITY 使该语句延迟至无客户读取表时才进行更新。LOW_PRIORITY 是在 MySQL 3.22.5 中引入的。

如果给出 LIMIT 子句，值 n 将指定更新的行的最大数目。LIMIT 是在 MySQL 3.23.3 中引入的。

D.1.27 USE

```
USE db_name
```

选择 *db_name* 使其成为当前数据库（即不包含明确的数据库名的表引用中的缺省数据库）。如果数据库不存在或没有访问它的权限，则 USE 语句将会失败。

D.2 注释语法

本节介绍如何在 SQL 代码中撰写注释。并且指出 mysql 客户机程序在注释解释方面存在的缺陷。在利用 mysql 批量方式执行的查询文件中，常常使用注释，因此在编写这样的查询文件时要特别注意下列规定。

MySQL 服务器理解三种类型的注释：

- 从“ # ”号开始到行结束的任何字符都作为一个注释处理。这个语法与大多数外壳程序和 Perl 中的相同。
- “ /* ”与“ */ ”之间的任何字符都作为一个注释处理。这种形式的注释可跨多行。这个语法与 C 语言中的相同。
- 自 MySQL 3.23.3 以来，可以“ -- ”（两个短划线和一个空格）开始一个注释，从这两个短划线到行结束的任何字符都作为一个注释处理。这样的注释风格（除了在两个短划线后不要空格）为某些其他数据库所用。MySQL 需要用空格作为区分字符，以使诸如 value1 - value2（其中 value2 是负的）这样的表达式不被作为注释处理。

在执行查询时，服务器忽略所有的注释。但有一个例外，那就是以“ /*！”开始的 C 风格的注释应该特殊处理。这种注释的文本中应该包含 SQL 关键字，这些关键字将被 MySQL 当作注释所在行的语句成分进行处理。例如，服务器认为下面两行是等价的：

```
INSERT LOW_PRIORITY INTO my_tbl SET ...
INSERT /*! LOW_PRIORITY */ INTO my_tbl SET ...
```

这种注释是为 MySQL 的特殊扩展与关键字而设计的。MySQL 能识别这些关键字，而其他 SQL 服务器会忽略它们。这使编写能利用 MySQL 特殊功能而又能为其他数据库系统所接受的查询更为容易。“ /*! ”注释风格是在 MySQL 3.22.7 中引入的。

自 MySQL 3.22.26 以来，可在“ /*! ”之后跟一个版本号告诉 MySQL，除非服务器的版本号不小于该版本号，否则忽略该注释。下面 UPDATE 语句中的注释除非服务器的版本号为 3.23.3 或以上，否则将忽略：

```
UPDATE my_table SET my_col = 100 WHERE my_col < 100 /*!32303 LIMIT 100 */
```

mysql 客户机在理解注释方面的限制较 MySQL 服务器更多。如果某些结构以 C 风格注释出现，将会产生混淆，因为它使用的分析程序较服务器的简单。例如，出现在 C 风格注释内的引用字符将被视为一个串，并继续查找该串的结束符直到找出一个匹配的引号为止。下面的语句说明了这种情况：

```
mysql> SELECT /* I have no quote */ 1;
+---+
| 1 |
+---+
| 1 |
+---+
mysql> SELECT /* I've got a quote */ 1;
    '>
```

mysql 分析第一条语句，不出问题，然后将它送服务器执行，并显示另一个“ mysql> ”提示符。第二条语句含有一个有未匹配引号的注释。结果， mysql 进入串分析模式。在键入回车键以后，它仍然处于串分析模式中。从它显示“ '> ”提示符就可以知道这一点。为了解决这个问题，键入一个匹配的引号后跟一个“ \c ”命令以结束此查询。关于 mysql 提示符的意义的详细信息，请参阅附录 E。