SQL

YURIY MISCHERYAKOV

SQL

Languages

Command	Description
SELECT	Команда получения данных
INSERT, UPDATE, DELETE	Команды манипулирования данными (Data manipulation language, DML)
CREATE, ALTER, DROP, RENAME, TRUNCATE	Команды определения структуры данных (Data definition language , DDL)
COMMIT, ROLLBACK, SAVEPOINT	Команды управления транзакциями (Transaction control language, TCL)
GRANT, REVOKE	Команды управления доступом (Data control language , DCL)

DDL

CREATE

- CREATE {**DATABASE** | **SCHEMA**} [IF NOT EXISTS] *db_name* [*create_option*] ...
- CREATE [TEMPORARY] **TABLE** [IF NOT EXISTS] *tbl_name* (*create_definition*,...) [*table_options*] [*partition_options*]
- CREATE [UNIQUE | FULLTEXT | SPATIAL] **INDEX** *index_name* [*index_type*] ON *tbl_name* (*key_part*,...) [*index_option*] [*algorithm_option* | *lock_option*] ...
- CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}] [DEFINER = *user*] [SQL SECURITY { DEFINER | INVOKER }] **VIEW** *view_name* [(*column_list*)] AS *select_statement* [WITH [CASCADED | LOCAL] CHECK OPTION]
- CREATE [DEFINER = *user*] **TRIGGER** *trigger_name trigger_time trigger_event* ON *tbl_name* FOR EACH ROW [*trigger_order*] *trigger_body*
- CREATE [DEFINER = *user*] **PROCEDURE** *sp_name* ([*proc_parameter*[,...]]) [*characteristic* ...] *routine_body*
- CREATE [DEFINER = *user*] **FUNCTION** *sp_name* ([*func_parameter*[,...]]) RETURNS *type* [*characteristic* ...] *routine_body*

DDL

CREATE TABLE

- CREATE [TEMPORARY] TABLE [IF NOT EXISTS] *tbl_name* (*create_definition*,...) [*table_options*] [*partition_options*]

- *create_definition*: { *col_name column_definition* | {INDEX | KEY} [*index_name*] [*index_type*] (*key_part*,...) [*index_option*] ... | {FULLTEXT | SPATIAL} [INDEX | KEY] [*index_name*] (*key_part*,...) [*index_option*] ... | [CONSTRAINT [*symbol*]] PRIMARY KEY [*index_type*] (*key_part*,...) [*index_option*] ... | [CONSTRAINT [*symbol*]] UNIQUE [INDEX | KEY] [*index_name*] [*index_type*] (*key_part*,...) [*index_option*] ... | [CONSTRAINT [*symbol*]] FOREIGN KEY [*index_name*] (*col_name*,...) ***reference_definition*** | *check_constraint_definition* }

MySQL parses but ignores “inline REFERENCES specifications” (as defined in the SQL standard) where the references are defined as part of the column specification. MySQL accepts REFERENCES clauses only when specified as part of a separate FOREIGN KEY specification. For storage engines that do not support foreign keys (such as MyISAM), MySQL Server parses and ignores foreign key specifications.

See: FOREIGN KEY Constraint Differences

DDL

CONSTRAINT

- [CONSTRAINT [*symbol*]] FOREIGN KEY [*index_name*] (*col_name*, ...) REFERENCES *tbl_name* (*col_name*,...) [ON DELETE *reference_option*] [ON UPDATE *reference_option*]
  - *reference_option*: RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

- CHECK (*expr*)
- [CONSTRAINT [*symbol*]] CHECK (*expr*) [[NOT] ENFORCED]

```
CREATE TABLE t1 (
   CHECK (c1 <> c2),
   c1 INT CHECK (c1 > 10),
   c2 INT CONSTRAINT c2_positive CHECK (c2 > 0),
   c3 INT CHECK (c3 < 100),
   CONSTRAINT c1_nonzero CHECK (c1 <> 0),
   CHECK (c1 > c3)
);
```

DDL

DROP

- DROP {**DATABASE** | **SCHEMA**} [IF EXISTS] *db_name*
- DROP [TEMPORARY] **TABLE** [IF EXISTS] *tbl_name* [, *tbl_name*] ... [RESTRICT | CASCADE]
- DROP **INDEX** *index_name* ON *tbl_name* [*algorithm_option* | *lock_option*] ... *algorithm_option*: ALGORITHM [=] {DEFAULT | INPLACE | COPY} *lock_option*: LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
- DROP **VIEW** [IF EXISTS] *view_name* [, *view_name*] ... [RESTRICT | CASCADE]
- DROP **EVENT** [IF EXISTS] *event_name*
- DROP {**PROCEDURE** | **FUNCTION**} [IF EXISTS] *sp_name*
- DROP **TRIGGER** [IF EXISTS] [*schema_name*.]*trigger_name*

DDL

DROP TABLE

- DROP [TEMPORARY] **TABLE** [IF EXISTS] *tbl_name* [, *tbl_name*] ... [RESTRICT | CASCADE]

  - Without IF EXISTS, the statement fails with an error indicating which nonexisting tables it was unable to drop, and no changes are made.
  - With IF EXISTS, no error occurs for nonexisting tables. The statement drops all named tables that do exist, and generates a NOTE diagnostic for each nonexistent table.
  - The RESTRICT and CASCADE keywords do nothing. They are permitted to make porting easier from other database systems.

DML

INSERT

- INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE] [INTO] *tbl_name*
  [PARTITION (*partition_name* [, *partition_name*] ...)]
  [(*col_name* [, *col_name*] ...)]
  { { VALUES | VALUE} (*value_list*) [, (*value_list*)] ...
  | VALUES *row_constructor_list* }
  [AS *row_alias* [(*col_alias* [, *col_alias*] ...)]]
  [ON DUPLICATE KEY UPDATE *assignment_list*]

- INSERT INTO *tbl_name* (a,b,c) VALUES(1,2,3), (4,5,6), (7,8,9);

- LAST_INSERT_ID() – returns a BIGINT UNSIGNED (64-bit) value representing the first automatically generated value successfully inserted for an AUTO_INCREMENT column as a result of the most recently executed INSERT statement.

DML

DELETE

-
  ```
  DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
           FROM tbl_name [[AS] tbl_alias]
           [PARTITION (partition_name [, partition_name] ...)]
           [WHERE where_condition]
           [ORDER BY ...]
           [LIMIT row_count]
  ```

  -
    ```
    DELETE t1, t2 FROM t1
        INNER JOIN t2
        INNER JOIN t3
    WHERE t1.id=t2.id AND t2.id=t3.id;
    ```

DML

UPDATE

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
        SET assignment_list
        [WHERE where_condition]
        [ORDER BY ...]
        [LIMIT row_count]
```

```
UPDATE items, month
    SET items.price = month.price
    WHERE items.id = month.id;
```

DQL

SELECT

- SELECT [DISTINCT] *select_expr* [, *select_expr*] ...
  FROM *table_references*
  [WHERE *where_condition*]
  [GROUP BY {*col_name* | *expr* | *position*}, ...
  [HAVING *where_condition*]]
  [ORDER BY {*col_name* | *expr* | *position*}];

Select

MYSQL

```
• SELECT [ALL | DISTINCT | DISTINCTROW ] [HIGH_PRIORITY] [STRAIGHT_JOIN] [SQL_SMALL_RESULT]
    [SQL_BIG_RESULT] [SQL_BUFFER_RESULT] [SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr] ... [into_option]
    [FROM table_references [PARTITION partition_list]]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
        [HAVING where_condition]
        [WINDOW window_name AS (window_spec) [, window_name AS (window_spec)] ...]
    [ORDER BY {col_name | expr | position} [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}] [into_option]
    [FOR {UPDATE | SHARE} [OF tbl_name [, tbl_name] ...]
    [NOWAIT | SKIP LOCKED] | LOCK IN SHARE MODE] [into_option]
```

Select

- *
- column list
- AS
- LIMIT o, c;   LIMIT c OFFSET o
- ORDER BY
- WHERE
    - Operators
        - <, >, ANY…
        - LIKE; LIKE … ESCAPE …
- SUBQUERY
    - A subquery is a **SELECT** statement within another statement

Select

FILTERING

Operators	Example
=, >, <, >=, <=, <>, != , !> , !<	price <= 120 table1.Id = table2.Id
AND, OR, NOT	price <= 120 AND table1.Id = table2.id OR NOT old > 20
IS [NOT] NULL	price IS NULL
[NOT] BETWEEN	price BETWEEN 25 AND 50
[NOT] LIKE	name LIKE 'Pav%' email LIKE '%@.epam.com'
[NOT] IN	surname NOT IN ('Петров', 'Іванов', 'Скворцов')
[NOT] EXISTS	EXISTS (SELECT * FROM laptop WHERE l.model = p.model)
ALL, ANY, SOME	model = ANY (SELECT model FROM pc)

Operator Precedence

- INTERVAL
- BINARY, COLLATE
- !
- - (unary minus), ~ (unary bit inversion)
- ^
- *, /, DIV, %, MOD
- -, +
- <<, >>
- &
- |
- = (comparison), <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN, MEMBER OF
- BETWEEN, CASE, WHEN, THEN, ELSE
- NOT
- AND, &&
- XOR
- OR, ||
- = (assignment), :=

Operator precedences are shown in the list, from highest precedence to the lowest.
Operators that are shown together on a line have the same precedence.

Pattern Matching

- MySQL provides standard SQL pattern matching as well as a form of pattern matching based on extended regular expressions similar to those used by Unix utilities such as vi, grep, and sed.
- LIKE
  - SQL pattern matching enables you to use ‘**_**’ to match any single character and ‘**%**’ to match an arbitrary number of characters (including zero characters).
  - In MySQL, SQL patterns are case-insensitive by default.
  - Do not use ‘**=**’ or ‘**<>**’ when you use SQL patterns. Use the **LIKE** or **NOT LIKE** comparison operators instead.
- REGEX
  - **REGEXP_LIKE()** function (or the **REGEXP** or **RLIKE** operators, which are synonyms for **REGEXP_LIKE()**)

https://dev.mysql.com/doc/refman/8.0/en/pattern-matching.html

Select

JOIN CLAUSE

- FROM <table 1>
  {[INNER] | {LEFT | RIGHT | FULL} [OUTER] | CROSS}
  JOIN <table 2> [ON <condition>]

    - [INNER] JOIN - internal join;
    - LEFT [OUTER] JOIN - left external join;
    - RIGHT [OUTER] JOIN - right external join;
    - FULL [OUTER] JOIN - full external join;
    - CROSS JOIN - cross join;

Select

JOIN CLAUSE

- *joined_table*: {
    table_reference [INNER | CROSS] JOIN *table_factor* [*join_specification*]
  | *table_reference* STRAIGHT_JOIN *table_factor*
  | *table_reference* STRAIGHT_JOIN *table_factor* ON *search_condition*
  | *table_reference* {LEFT|RIGHT} [OUTER] JOIN *table_reference join_specification*
  | *table_reference* NATURAL [{LEFT|RIGHT} [OUTER]] JOIN *table_factor*
  }
- *join_specification*: {
    ON *search_condition*
  | USING (*join_column_list*)
  }

Select

UNION

- Combines the result from multiple SELECT statements into a single result set.

- SELECT ...
    UNION [ALL | DISTINCT] SELECT ...
    [UNION [ALL | DISTINCT] SELECT ...]

Aggregate Functions

MYSQL

Name	Description
AVG()	Return the average value of the argument
BIT_AND()	Return bitwise AND
BIT_OR()	Return bitwise OR
BIT_XOR()	Return bitwise XOR
COUNT()	Return a count of the number of rows returned
COUNT(DISTINCT)	Return the count of a number of different values
GROUP_CONCAT()	Return a concatenated string
JSON_ARRAYAGG()	Return result set as a single JSON array
JSON_OBJECTAGG()	Return result set as a single JSON object

Aggregate Functions

Name	Description
MAX()	Return the maximum value
MIN()	Return the minimum value
STD()	Return the population standard deviation
STDDEV()	Return the population standard deviation
STDDEV_POP()	Return the population standard deviation
STDDEV_SAMP()	Return the sample standard deviation
SUM()	Return the sum
VAR_POP()	Return the population standard variance
VAR_SAMP()	Return the sample variance
VARIANCE()	Return the population standard variance

Stored objects

OBJECTS

Trigger

Procedure

Function

View

Event

ADVANTAGES

- Implementation in the form of database objects.
- Encapsulation.
- Providing protection.
- Reduction of network traffic.
- Ensuring business rules.

Trigger

- CREATE
  [DEFINER = { *<ім'я користувача>* | CURRENT_USER }]
  TRIGGER *<ім'я тригера>*
  { BEFORE | AFTER }
  { INSERT | UPDATE | DELETE }
  ON *<ім'я таблиці>* FOR EACH ROW
  [{ FOLLOWS | PRECEDES }]
  { < *SQL-код* > }

Procedure

- CREATE
  [DEFINER = { *<ім’я користувача>* | CURRENT_USER }]
  PROCEDURE *<ім’я процедури>*
  ([[IN | OUT | INOUT] *<ім’я параметра>* *<тип даних>* [, …]])
  [[NOT] DETERMINISTIC]
  [SQL SECURITY { DEFINER | INVOKER }]
  < SQL-код >

Function

- CREATE
  [DEFINER = { <*ім'я користувача*> | CURRENT_USER }]
  FUNCTION <*ім'я користувацької функції*>
  ([<*ім'я параметра*> <*тип даних*> [, ...]])
  RETURNS <*тип даних*>
  [[NOT] DETERMINISTIC]
  [SQL SECURITY { DEFINER | INVOKER }]
  < *Блок SQL-коду* ***з*** RETURN >

Variables

- SET @value = (SELECT MAX(price) FROM product) ;
- SELECT @value3 := MAX(price) FROM product;

- Local variables can present in stored objects only

    -
      ```
      DECLARE var1 int ;
      SET var1 = 10 ;

      DECLARE var2 varchar(100) ;
      SELECT t.name INTO var2
      FROM user_details AS t WHERE t.id = p_param1;
      ```

Flow Control Statements

- CASE Statement
- IF Statement
- ITERATE Statement
- LEAVE Statement
- LOOP Statement
- REPEAT Statement
- RETURN Statement
- WHILE Statement