# SQL:
# Data Manipulation Language

(Part 3)

# Update Existing Row(s)

**Single-table syntax:**

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
    SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

**Multiple-table syntax:**

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
    SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
    [WHERE where_condition]
```

# Update Existing Row(s)

**Single-table syntax:**

- The `UPDATE` statement updates columns of existing rows in the named table with new values.
- The `SET` clause indicates which columns to modify and the values they should be given.
- Each value can be given as an expression, or the keyword `DEFAULT` to set a column explicitly to its default value.
- The `WHERE` clause, if given, specifies the conditions that identify which rows to update. With no `WHERE` clause, all rows are updated.
- If the `ORDER BY` clause is specified, the rows are updated in the order that is specified. The `LIMIT` clause places a limit on the number of rows that can be updated.

# Update Existing Row(s)

**Multiple-table syntax:**

- `UPDATE` updates rows in each table named in ***`table_references`*** that satisfy the conditions.
- Each matching row is updated once, even if it matches the conditions multiple times.
- For multiple-table syntax, `ORDER BY` and `LIMIT` cannot be used.

# Update Existing Row(s)

The `UPDATE` statement supports the following modifiers:

- With the `LOW_PRIORITY` keyword, execution of the `UPDATE` is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).
- With the `IGNORE` keyword, the update statement does not abort even if errors occur during the update. Rows for which duplicate-key conflicts occur on a unique key value are not updated. Rows updated to values that would cause data conversion errors are updated to the closest valid values instead.

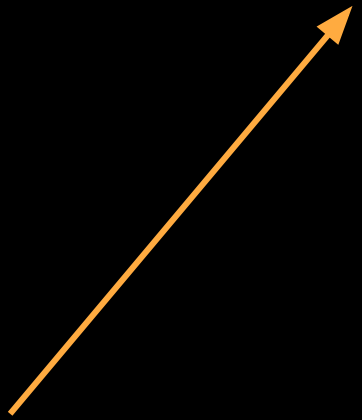
# Update Existing Row(s)

```
mysql> SELECT * FROM demo;
+------+
| num  |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)


mysql> UPDATE demo
    -> SET num = (num * 2)
    -> WHERE (num % 2) = 1;
```

# Update Existing Row(s)

```
mysql> SELECT * FROM demo;
+------+
| num  |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)
```



```
mysql> SELECT * FROM demo;
+------+
| num  |
+------+
|    2 |
|    2 |
|    6 |
+------+
3 rows in set (0.00 sec)
```

```
mysql> UPDATE demo
    -> SET num = (num * 2)
    -> WHERE (num % 2) = 1;
Query OK, 2 rows affected (0.04 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

# Update Existing Row(s)

```
mysql> SELECT * FROM demo2;
+------+
| num  |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

NOTE: demo2 has a PRIMARY KEY constraint on num.

```
mysql> UPDATE demo2
    -> SET    num = (num * 2)
    -> WHERE  (num % 2) = 1;
```

# Update Existing Row(s)

```
mysql> SELECT * FROM demo2;
+------+
| num  |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

NOTE: demo2 has a PRIMARY KEY constraint on num.

```
mysql> UPDATE demo2
    -> SET    num = (num * 2)
    -> WHERE  (num % 2) = 1;
ERROR 1062 (23000): Duplicate entry '2' for key 'PRIMARY'
```

# Update Existing Row(s)

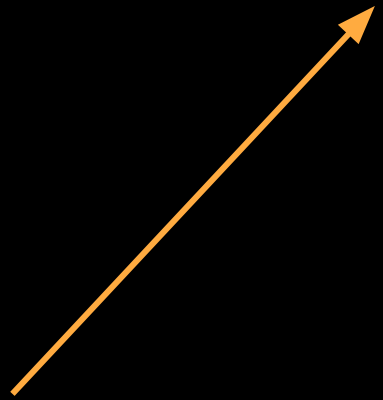
```
mysql> SELECT * FROM demo2;
+------+
| num  |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

NOTE: demo2 has a PRIMARY KEY constraint on num.

```
mysql> UPDATE IGNORE demo2
    -> SET    num = (num * 2)
    -> WHERE  (num % 2) = 1;
```

# Update Existing Row(s)

```
mysql> SELECT * FROM demo2;
+------+
| num  |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)
```



```
mysql> SELECT * FROM demo2;
+------+
| num  |
+------+
|    1 |
|    2 |
|    6 |
+------+
3 rows in set (0.00 sec)
```

```
mysql> UPDATE IGNORE demo2
    -> SET    num = (num * 2)
    -> WHERE  (num % 2) = 1;
Query OK, 1 row affected, 1 warning (0.24 sec)
Rows matched: 2  Changed: 1  Warnings: 1
```

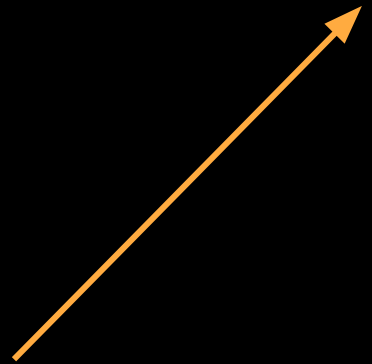
# Update Existing Row(s)

```
mysql> SELECT * FROM demo2;
+------+
| num  |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)

 mysql> UPDATE IGNORE demo2
     -> SET    num = (num * num)
     -> ORDER BY num DESC
     -> LIMIT 1;
```

# Update Existing Row(s)

```
mysql> SELECT * FROM demo2;
+------+
| num  |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)
```



```
mysql> SELECT * FROM demo2;
+------+
| num  |
+------+
|    1 |
|    2 |
|   36 |
+------+
3 rows in set (0.00 sec)
```

```
mysql> UPDATE IGNORE demo2
    -> SET    num = (num * num)
    -> ORDER BY num DESC
    -> LIMIT 1;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

# Insert Syntax (1)

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY]
[IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

# Insert Syntax (2)

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY]
[IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    SET col_name={expr | DEFAULT}, ...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

# Insert Syntax (3)

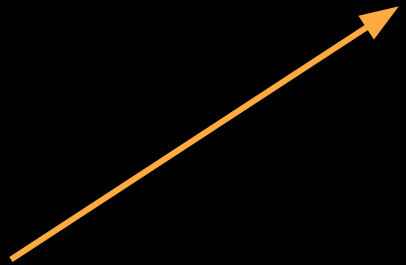
```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    [(col_name,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

# Insert Syntax

- `INSERT` inserts new rows into an existing table.
- The `INSERT...VALUES` and `INSERT...SET` forms of the statement insert rows based on explicitly specified values.
- The `INSERT...SELECT` form inserts rows selected from another table or tables.

# Insert New Row

```
mysql> SELECT * FROM demo;
+-----+
| num |
+-----+
|   2 |
|   2 |
|   6 |
+-----+
```



```
mysql> INSERT INTO demo
    -> VALUES (3),(4),(5);
```

```
mysql> SELECT * FROM demo;
+-----+
| num |
+-----+
|   2 |
|   2 |
|   6 |
|   3 |
|   4 |
|   5 |
+-----+
```

# Insert New Row

```
mysql> SELECT * FROM demo2;
+-----+
| num |
+-----+
|   1 |
|   2 |
|  36 |
+-----+
```

```
mysql> INSERT INTO demo2
    -> SET num = 0;
```

```
mysql> SELECT * FROM demo2;
+-----+
| num |
+-----+
|   0 |
|   1 |
|   2 |
|  36 |
+-----+
```

# Insert New Row

```
mysql> INSERT INTO demo
    ->   SELECT SUM(NUM)
    ->   FROM   demo;
```

```
mysql> SELECT * FROM demo;
+-----+
| num |
+-----+
|   2 |
|   2 |
|   6 |
|   3 |
|   4 |
|   5 |
|  22 |
+-----+
```

# Insert New Row

```
mysql> INSERT INTO demo2
    ->   SELECT *
    ->   FROM   demo
    ->   WHERE demo.num > (
    ->     SELECT AVG(demo2.num)
    ->     FROM demo2
    ->   );
```

```
mysql> SELECT * FROM demo2;
+-----+
| num |
+-----+
|   0 |
|   1 |
|   2 |
|  22 |
|  36 |
+-----+
```

# Delete Row

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

# Delete Row(s)

The `DELETE` statement deletes rows from ***`tbl_name`*** and returns the number of deleted rows.

# Delete Row(s)

**Main Clauses**

- The conditions in the optional `WHERE` clause identify which rows to delete.
- With no `WHERE` clause, all rows are deleted.
- ***`where_condition`*** is an expression that evaluates to true for each row to be deleted.

# Delete Row(s)

**Main Clauses**

- If the `ORDER BY` clause is specified, the rows are deleted in the order that is specified. `ORDER BY` also helps to delete rows in an order required to avoid referential integrity violations.
- The `LIMIT` clause places a limit on the number of rows that can be deleted.

# Delete Row(s)

```
mysql> SELECT * FROM demo;
+-----+
| num |
+-----+
|   2 |
|   2 |
|   6 |
|   3 |
|   4 |
|   5 |
|  22 |
+-----+
```

```
mysql> DELETE FROM demo
    -> WHERE num < (
    ->   SELECT AVG(num)
    ->   FROM demo
    -> );
```

# Delete Row(s)

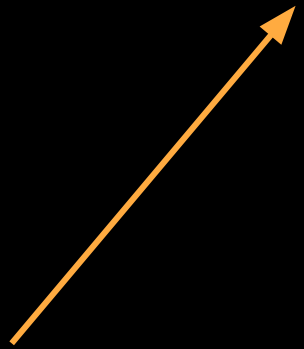
```
mysql> SELECT * FROM demo;
+-----+
| num |
+-----+
|   2 |
|   2 |
|   6 |
|   3 |
|   4 |
|   5 |
|  22 |
+-----+
```

```
mysql> DELETE FROM demo
    -> WHERE num < (
    ->   SELECT AVG(num)
    ->   FROM demo
    -> );
ERROR 1093 (HY000): You can't specify target
table 'demo' for update in FROM clause
```

# Delete Row(s)

```
mysql> SELECT * FROM demo;
+-----+
| num |
+-----+
|   2 |
|   2 |
|   6 |
|   3 |
|   4 |
|   5 |
|  22 |
+-----+
```

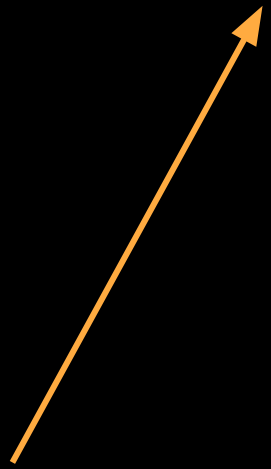
```
mysql> DELETE FROM demo
    ->   WHERE num < (
    ->     SELECT AVG(num)
    ->     FROM demo2
    ->   );
Query OK, 6 rows affected (0.05 sec)
```



```
mysql> SELECT * FROM demo;
+------+
| num  |
+------+
|   22 |
+------+
1 row in set (0.00 sec)
```

# Delete Row(s)

```
mysql> SELECT * FROM demo2;
+------+
| num  |
+------+
|    0 |
|    1 |
|    2 |
|   22 |
|   36 |
+------+
5 rows in set (0.00 sec)
```



```
mysql> SELECT * FROM demo;
+------+
| num  |
+------+
|    2 |
|   22 |
|   36 |
+------+
1 row in set (0.00 sec)
```

```
mysql> DELETE FROM demo2
    ->   ORDER BY num
    ->   LIMIT 2;
Query OK, 2 rows affected (0.07 sec)
```

# Truncate

```
TRUNCATE TABLE table;
```

Logical equivalent of `DELETE FROM table;`

But typically doesn’t generate journal entries for each row being deleted; very efficient; can’t be rolled back