9 Table Data Manipulation

Objectives

After completing this lesson, you will be able to:

- Delete and modify table data
- Add new data to a table by using the `INSERT` and `REPLACE` statements
- Modify data in a table by using the `UPDATE` statement
- Remove table rows by using the `DELETE` statement

9 - 2

Manipulation of Table Row Data

- Data in a database is dynamic:
  - Row data can be inserted, updated, replaced, or deleted.
  - Care must be taken to prevent data loss or damage.
- Take these precautions:
  - Grant change access only to users that need it.
  - Keep regular backups.
  - Make additional backups as required.
  - Set the "safe updates" option.
  - Always think in terms of rows (the entire record is affected).
  - Test the scope of a change first with a `SELECT ... WHERE` statement.

9 - 3

Data manipulation can be unsafe. Take the following precautions with your data:

- Do not grant users (including yourself) more permissions than they need. For example, if you are running some queries on a database for the finance department, do not use the MySQL `root` account. Instead, create a user who has permission to run only the required `SELECT` queries and then log in as that user.
- Keep daily backups.
- Create backups before you make any major changes or use any unfamiliar features.
- The safe updates option is useful for beginners. Enable it with the command-line client option `--safe-updates`, or `SET SQL_SAFE_UPDATES=1` within the mysql client program. This option stops you from running a `DELETE` statement without a `WHERE` clause. If you do not set this option and execute a `DELETE` statement without a `WHERE` clause, it will delete every record in the table.
- Test statements that will affect your data on a copy of the table before running them on the real table.
- Use a `SELECT` with the same `WHERE` clause to make sure you retrieve the right records before you change them with a `DELETE` or `UPDATE` statement.

INSERT Statement

- Populates a table with row data
- Syntax:

```
INSERT INTO table_name (<column_list>)
  VALUES (<value_list>)
```

- Example (inserts a single row):

```
INSERT INTO CountryLanguage (CountryCode,
  Language, Percentage)
  VALUES ('ALB', 'Chinese', 20);
```

  - Adds the following record:

```
+-------------+------------+------------+------------+
| CountryCode | Language   | IsOfficial | Percentage |
+-------------+------------+------------+------------+
...
| ALB         | Chinese    | F          | 20.0       |
...
```

9 - 4

The statement syntax to insert a single record is `INSERT INTO`, followed by the table name, the columns affected, and the values to be inserted into those columns. The number of columns and values must be the same.

The example does not provide an entry for the `IsOfficial` column, so it is given the default value for that column, which is `F` (false).

Using INSERT for Multiple Rows

- Specify values for each row.
- Example:

```
INSERT INTO CountryLanguage (CountryCode, Language)
  VALUES ('GRL', 'MySQL'),     ← 1st row
    ('FJI', 'MySQL'),          ← 2nd row
    ('BEL', 'MySQL');          ← 3rd row
```

  - Content of the new row:

```
+-------------+----------+------------+------------+
| CountryCode | Language | IsOfficial | Percentage |
+-------------+----------+------------+------------+
| BEL         | MySQL    | F          |        0.0 |
| FJI         | MySQL    | F          |        0.0 |
| GRL         | MySQL    | F          |        0.0 |
+-------------+----------+------------+------------+
```

9 - 5

The statement in the example adds new `CountryLanguage` records for Greenland, Fiji, and Belgium. It assigns a country code and “MySQL” as the language for each. Because it does not assign values for the `Percentage` or `IsOfficial` columns, they receive the column defaults.

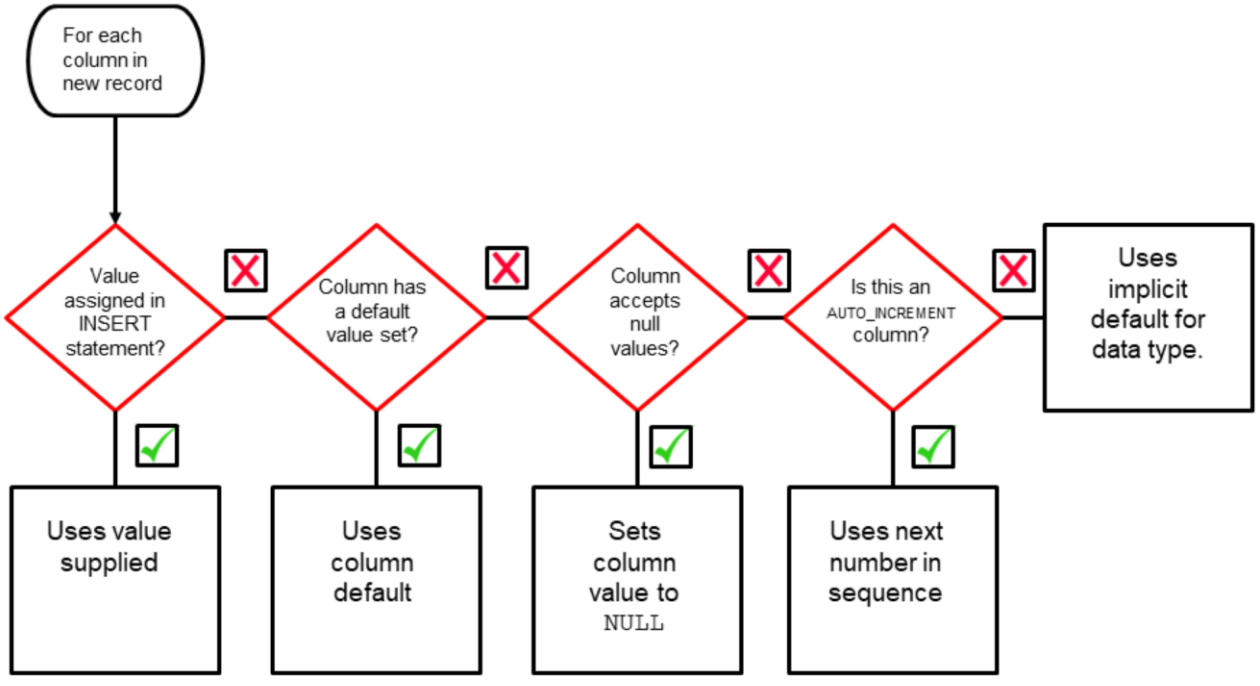
The row data for each of the three rows is enclosed in parentheses.

The following query lists the new records:

```
SELECT * FROM CountryLanguage
WHERE CountryCode IN ('GRL', 'FJI', 'BEL')
AND LANGUAGE = 'MySQL';
```

Note: You must enclose column values for string and temporal data types in single quotation marks.

Column Value Assignment with `INSERT`



9 - 6

REPLACE Statement

- Use in tables with primary key or unique constraint.
  - It replaces the existing row with new values.
  - In tables without a primary key or unique constraint, it acts in the same way as an **INSERT** statement.
- **REPLACE** is a MySQL extension to the SQL standard.
- General syntax:

```
REPLACE INTO table_name (<column_list>)
  VALUES (<value_list>)
```

- Example:

```
REPLACE INTO CountryLanguage (CountryCode,
  Language, Percentage)
  VALUES ('ALB', 'Albaniana', 78.1);
```

9-7

Unless the table has a `PRIMARY KEY` or a `UNIQUE` constraint, using a `REPLACE` statement makes no sense. It is equivalent to `INSERT` because MySQL needs an index to determine if the new row duplicates an existing one.

You supply column values in the `REPLACE` statement. Like `INSERT`, any missing columns are set to their default values. You cannot refer to values from the current row and use them in the new row.

In the example in the slide, you use the `REPLACE` statement to overwrite the existing Albaniana record with a new one. The new record compensates for the addition of Chinese in an earlier step because it has a different `Percentage` value for Albaniana.

The results of this change are as follows:

```
mysql> SELECT * FROM CountryLanguage WHERE CountryCode = 'ALB';
+-------------+-----------+------------+------------+
| CountryCode | Language  | IsOfficial | Percentage |
+-------------+-----------+------------+------------+
| ALB         | Albaniana | F          | 78.1       |
| ALB         | Chinese   | F          | 20.0       |
| ALB         | Greek     | F          | 1.8        |
| ALB         | Macedonian| F          | 0.1        |
```

```
+-------------+-----------+------------+------------+
```

REPLACE Results

Example:

```
Query OK, 2 rows affected (0.06 sec)
```

- Returns a count to indicate the number of rows affected
- Count is the sum of the rows deleted and inserted:
  - Count of 1: One row was inserted. No rows were deleted.
  - Count greater than 1: One or more old rows were deleted before the new row was inserted.

9 - 8

A single row can replace more than one old row if the table contains multiple unique indexes and the new row duplicates values for different old rows in different unique indexes.

REPLACE Algorithm

MySQL performs the following steps for **REPLACE**:

1. It tries to insert the new row into the table.
2. If insertion fails because a duplicate-key error occurs (for a primary key or unique constraint), it deletes the conflicting row.
3. It tries again to insert the new row into the table.

9 - 9

Caution: If a table has more than one constraint (unique or primary key), replacing one row can delete several rows, if there are conflicts on more than one constraint.

The main difference between `REPLACE` and `INSERT` is that `INSERT` only adds new data, but `REPLACE` can also change existing data.

To use `REPLACE`, you must have `INSERT` and `DELETE` privileges on the table.

UPDATE Statement

- Modifies contents of existing rows
- Is used with the **SET** clause to assign new values
- General syntax:

```
UPDATE table_name SET column=expression
[,column=expression,...]
[WHERE condition] [other_clauses]
```

- Example:

```
mysql> UPDATE Country
    -> SET Population = Population * 2,
    ->     Region = 'Dolphin Country'
    -> WHERE Code = 'SWE';
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

9 - 10

The slide example shows an `UPDATE` of the `Country` table. The `Population` column value is doubled and the `Region` changed for Sweden (`Code = SWE`).

The results show that only one row was affected.

- Matched: One row was retrieved for the `WHERE` clause.
- Changed: One row's values were updated.

UPDATE Statement Ordering

- There is no guarantee about the order in which rows are updated. This can result in errors:

```
mysql> UPDATE City SET ID = ID+1;
ERROR 1062 (23000): Duplicate entry '2' for key 'PRIMARY'
```

- Use **ORDER BY** to control the order of updates:

```
mysql> UPDATE City SET ID = ID+1
    -> ORDER BY ID DESC;
Query OK, 4079 rows affected (0.13 sec)
Rows matched: 4079 Changed: 4079 Warnings: 0
```

  - Content of the new row:

```
+----+---------------+-------------+---------------+------------+
| ID | Name          | CountryCode | District      | Population |
+----+---------------+-------------+---------------+------------+
|  2 | Kabul         | AFG         | Kabol         |    1780000 |
|  3 | Qandahar      | AFG         | Qandahar      |     237500 |
|  4 | Herat         | AFG         | Herat         |     186800 |
...
```

9 - 11

Use an `ORDER BY` clause to force row updates to occur in a particular order.

The results shown in the slide are a result of the following query:

```
mysql> SELECT * FROM City ORDER BY ID;
+------+---------------+-------------+---------------+------------+
| ID   | Name          | CountryCode | District      | Population |
+------+---------------+-------------+---------------+------------+
|  2   | Kabul         | AFG         | Kabol         |    1780000 |
|  3   | Qandahar      | AFG         | Qandahar      |     237500 |
|  4   | Herat         | AFG         | Herat         |     186800 |
|  5   | Mazar-e-Sharif | AFG        | Balkh         |     127800 |
|  6   | Amsterdam     | NLD         | Noord-Holland |     731200 |
...
| 4079 | Nablus        | PSE         | Nablus        |     100231 |
| 4080 | Rafah         | PSE         | Rafah         |      92020 |
+------+---------------+-------------+---------------+------------+
4079 rows in set (0.02 sec)
```

Note: Because of the update, the first `ID` is now `2`.

UPDATE with LIMIT

- Use **LIMIT** to control the number of rows updated:

```
mysql> UPDATE City SET ID = ID-1 LIMIT 1;
```

  - The content of the row:

```
+----+---------------+-------------+--------------+------------+
| ID | Name          | CountryCode | District     | Population |
+----+---------------+-------------+--------------+------------+
|  1 | Kabul         | AFG         | Kabol        |    1780000 |
|  3 | Qandahar      | AFG         | Qandahar     |     237500 |
|  4 | Herat         | AFG         | Herat        |     186800 |
...
```

    - Note that only the first `ID` was affected by the update.

- **UPDATE** can include **ORDER BY** used with **LIMIT**.

9 - 12

The slide example demonstrates using a `LIMIT` of 1 to affect the first record in the table and leave the rest unchanged.

Note: It is not a good practice to update a primary key value.

When `UPDATE` Has No Effect

`UPDATE` has no effect when:

- No rows are matched:
  - None match the **`WHERE`** clause.
  - The table is empty.
- No column values are changed:
  - The new value is the same as the existing one.
  - No rows in the table contain the specified key values.

9 - 13

Difference Between UPDATE and REPLACE

When handling rows with unique key values, **UPDATE** and **REPLACE** are not equivalent:

UPDATE	REPLACE
Does nothing if there is no existing row with specified key values	Adds a record if there is no existing row with specified key values
Can change some column values in an existing row and leave others unchanged	Replaces the entire row*

*If you want to keep the same values for any columns, you need to supply those values in the **REPLACE** statement.

9 - 14

DELETE Statement

- Removes entire rows from a table
- General syntax:

```
DELETE FROM table_name
[WHERE condition]
[ORDER BY ...]
[LIMIT row_count]
```

  - Use **WHERE** to specify which rows to remove.
- Use the table name without a `WHERE` clause to remove all rows in the table:

```
DELETE FROM my_table
```

- Use `DELETE` with extreme caution; it cannot be reversed.

9 - 15

The `DELETE` statement returns the number of rows affected.

DELETE with ORDER BY and LIMIT

- Control the order and number of records deleted by using **ORDER BY** and **LIMIT**:
- Example (using `ORDER BY ... DESC`):

```
DELETE FROM CountryLanguage
WHERE Language = 'MySQL'
ORDER BY CountryCode
DESC LIMIT 1;
```

  - Affects the following row:

```
+-------------+----------+------------+------------+
| CountryCode | Language | IsOfficial | Percentage |
+-------------+----------+------------+------------+
| BEL         | MySQL    | F          |        0.0 |
| FJI         | MySQL    | F          |        0.0 |
| GRL         | MySQL    | F          |        0.0 |
+-------------+----------+------------+------------+
```

Removed →

9 - 16

`LIMIT` restricts the number of records that will be deleted. They are deleted from the top of the result set, so use `ORDER BY` to ensure that they are in the correct order.

The `DELETE` statement in the slide retrieves all records from the `CountryLanguage` table where the `Language` is "MySQL," in descending order of `CountryCode`. It then deletes the first record.

A `SELECT` statement listing the remaining records in the default sort order confirms the deletion:

```
mysql> SELECT * FROM CountryLanguage WHERE Language = 'MySQL';
+-------------+----------+------------+------------+
| CountryCode | Language | IsOfficial | Percentage |
+-------------+----------+------------+------------+
| BEL         | MySQL    | F          | 0.0        |
| FJI         | MySQL    | F          | 0.0        |
+-------------+----------+------------+------------+
3 rows in set (0.01 sec)
```

Note: If you do not use the `ORDER BY` clause in conjunction with `LIMIT`, MySQL cannot guarantee which one of the items will be deleted.

DELETE with ORDER BY and LIMIT

- Example (using `ORDER BY ... ASC`):

```
DELETE FROM CountryLanguage
WHERE Language = 'MySQL'
ORDER BY CountryCode
ASC LIMIT 1;
```

  - Affects the following row:

```
+-------------+----------+------------+------------+
| CountryCode | Language | IsOfficial | Percentage |
+-------------+----------+------------+------------+
| BEL         | MySQL    | F          |        0.0 |
| FJI         | MySQL    | F          |        0.0 |
| GRL         | MySQL    | F          |        0.0 |
+-------------+----------+------------+------------+
```

Removed →

9 - 17

In this example, records are retrieved in ascending order of `CountryCode` and the first one is deleted.