CHANGEPOND

MySQL DDL,DML

AGENDA – DDL

- MySQL CREATE TABLE syntax
- MySQL CREATE TABLE statement example
- MySQL ALTER TABLE To Change Table Structure
- Introduction to MySQL ADD COLUMN statement
- Introduction to MySQL RENAME TABLE statement
- MySQL DROP COLUMN
- MySQL DROP TABLE To Remove Existing Tables
- Creating Tables with existing Table's structure
- MySQL TRUNCATE TABLE

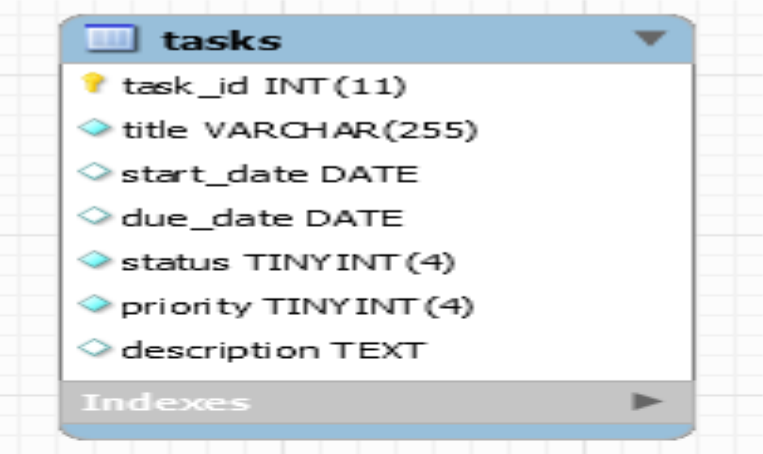
MySQL CREATE TABLE syntax

- First, you specify the name of the table that you want to create after the CREATE TABLE clause.
- The table name must be unique within a database.
- The IF NOT EXISTS is an optional clause that allows you to check if the table that you are creating already exists in the database.
- If this is the case, MySQL will ignore the whole statement and will not create any new table.
- It is highly recommended that you use IF NOT EXISTS in every CREATE TABLE statement to avoid an error of creating a new table that already exists.
- you can optionally specify the storage engine for the table in the ENGINE clause.
- You can use any storage engine such as InnoDB and MyISAM. If you don't explicitly declare the storage engine, MySQL will use InnoDB by default.

```
CREATE TABLE [IF NOT EXISTS] table_name(
    column_list
) ENGINE=storage_engine
```

```
column_name data_type(length) [NOT NULL] [DEFAULT value] [AUTO_INCREMENT]
```

MySQL CREATE TABLE statement example




```
1  CREATE TABLE IF NOT EXISTS tasks (
2      task_id INT AUTO_INCREMENT,
3      title VARCHAR(255) NOT NULL,
4      start_date DATE,
5      due_date DATE,
6      status TINYINT NOT NULL,
7      priority TINYINT NOT NULL,
8      description TEXT,
9      PRIMARY KEY (task_id)
10 )  ENGINE=INNODB;
```

MySQL ALTER TABLE To Change Table Structure

- The ALTER TABLE statement modifies the structure of an existing table.
- It allows you to add a column, drop a column, change the data type of column, add primary key, rename table, and many more.
- Changing columns using MySQL ALTER TABLE statement

```
1 ALTER TABLE tasks
2 CHANGE COLUMN task_id task_id INT(11) NOT NULL AUTO_INCREMENT;
```

- Using MySQL ALTER TABLE statement to add a new column into a table example

```
1 ALTER TABLE tasks
2 ADD COLUMN complete DECIMAL(2,1) NULL
3 AFTER description;
```

- Using MySQL ALTER TABLE to drop a column from a table example

```
1 ALTER TABLE tasks
2 DROP COLUMN description;
```

- Renaming tables using MySQL ALTER TABLE statement

```
1 ALTER TABLE tasks
2 RENAME TO work_items;
```

Introduction to MySQL ADD COLUMN statement

- To add a new column to an existing table, you use the ALTER TABLE ADD COLUMN statement

```
ALTER TABLE table
ADD [COLUMN] column_name column_definition [FIRST|AFTER existing_column];
```

- First, you specify the table name after the ALTER TABLE clause.
- Second, you put the new column and its definition after the ADD COLUMN clause. Note that COLUMN keyword is optional so you can omit it.
- Third, MySQL allows you to add the new column as the first column of the table by specifying the FIRST keyword
- It also allows you to add the new column after an existing column using the AFTER existing_column clause.
- If you don't explicitly specify the position of the new column, MySQL will add it as the last column.

```
ALTER TABLE table
ADD [COLUMN] column_name_1 column_1_definition [FIRST|AFTER existing_column],
ADD [COLUMN] column_name_2 column_2_definition [FIRST|AFTER existing_column],
...;
```

```
CREATE TABLE IF NOT EXISTS vendors (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255)
);
```

```
ALTER TABLE vendors
ADD COLUMN phone VARCHAR(15) AFTER name;
```

```
ALTER TABLE vendors
ADD COLUMN email VARCHAR(100) NOT NULL,
ADD COLUMN hourly_rate decimal(10,2) NOT NULL;
```

Introduction to MySQL RENAME TABLE statement

- MySQL provides us with a very useful statement that changes the name of one or more tables.

```
RENAME TABLE old_table_name TO new_table_name;
```

- In addition to the tables, we can use the RENAME TABLE statement to rename views.
- Renaming multiple tables

```
RENAME TABLE old_table_name_1 TO new_table_name_2,
             old_table_name_2 TO new_table_name_2,..
```

```
RENAME TABLE depts TO departments,
             people TO employees;
```

- Renaming tables using ALTER TABLE statement

```
ALTER TABLE old_table_name
RENAME TO new_table_name;
```

MySQL DROP COLUMN

- Introduction to MySQL DROP COLUMN statement

```
ALTER TABLE table
DROP COLUMN column;
```

```
ALTER TABLE posts
DROP COLUMN excerpt;
```

- To remove multiple columns from a table at the same time, you use the following syntax:

```
ALTER TABLE table
DROP COLUMN column_1,
DROP COLUMN column_2,
…;
```

```
ALTER TABLE posts
DROP COLUMN created_at,
DROP COLUMN updated_at;
```

MySQL DROP TABLE To Remove Existing Tables

- MySQL DROP TABLE statement syntax

```
DROP [TEMPORARY] TABLE [IF EXISTS] table_name [, table_name] ...
[RESTRICT | CASCADE]
```

```
DROP TABLE IF EXISTS tasks, nonexistent_table;
```

- If you check the database, you will see that the tasks table was removed
- You can check the NOTE, which is generated by MySQL because of the non-existing table, by using the SHOW WARNING statement as follows:

```
SHOW WARNINGS;
```

Level	Code	Message
Note	1051	Unknown table 'nonexistent_table'

MySQL DROP TABLE BASED ON A PATTERN

- Suppose you have a lot of tables whose names start with test in your database and you want to save time by removing all of them using a single DROP TABLE statement.
- Unfortunately, MySQL does not provide the DROP TABLE LIKE statement that can remove tables based on pattern matching like the following:

```
1 DROP TABLE LIKE '%pattern%'
```

Creating Tables with existing Table's structure

```
1  CREATE TABLE IF NOT EXISTS test1(
2    id int(11) NOT NULL AUTO_INCREMENT,
3    PRIMARY KEY(id)
4  );
5
6  CREATE TABLE IF NOT EXISTS test2 LIKE test1;
7  CREATE TABLE IF NOT EXISTS test3 LIKE test1;
8  CREATE TABLE IF NOT EXISTS test4 LIKE test1;
```

MySQL TRUNCATE TABLE

- The MySQL TRUNCATE TABLE statement allows you to delete all data in a table.
- Therefore, in terms of functionality, The TRUNCATE TABLE statement is like a DELETE statement without a WHERE clause.
- However, in some cases, the MySQL TRUNCATE TABLE statement is more efficient than the DELETE statement.

```
1 TRUNCATE TABLE table_name;
```

- f the table has any foreign key constraints, the TRUNCATE TABLE statement deletes rows one by one. If the foreign key constraint has DELETE CASCADE action, the corresponding rows in the child tables are also deleted.
- If the foreign key constraint does not specify the DELETE CASCADE action, the TRUNCATE TABLE deletes rows one by one, and it will stop and issue an error when it encounters a row that is referenced by a row in a child table.
- If the table does not have any foreign key constraint, the TRUNCATE TABLE statement drops the table and recreates a new empty one with the same structure, which is faster and more efficient than using the DELETE statement especially for big tables.
- Notice that the TRUNCATE TABLE statement resets auto increment value to zero if the table has an AUTO_INCREMENT column.

AGENDA – DML

- Introduction to the MySQL INSERT statement
- Inserting rows using default values
- Inserting dates into the table
- MySQL INSERT INTO SELECT
- MySQL INSERT ON DUPLICATE KEY UPDATE
- MySQL INSERT ON DUPLICATE KEY UPDATE example
- MySQL INSERT IGNORE Statement
- MySQL UPDATE
- MySQL UPDATE JOIN
- MySQL DELETE
- MySQL DELETE and LIMIT clause
- MySQL DELETE JOIN
- MySQL DELETE JOIN with INNER JOIN example
- MySQL DELETE JOIN with LEFT JOIN
- MySQL REPLACE

Introduction to the MySQL INSERT statement

```
1 INSERT INTO table(c1, c2, ...)
2 VALUES (v1, v2, ...);
```

To add multiple rows into a table using a single INSERT statement, you use the following syntax:

```
1 INSERT INTO table(c1, c2, ...)
2 VALUES
3    (v11, v12, ...),
4    (v21, v22, ...),
5     ...
6    (vnn, vn2, ...);
```

INSERTING ROWS USING DEFAULT VALUES

The following example demonstrates how the second way:

```
1 INSERT INTO
2  tasks(title,priority)
3 VALUES
4  ('Understanding DEFAULT keyword in INSERT statement',DEFAULT);
```

In this example, we specified the priority column and the `DEFAULT` keyword.

Because the default value for the `priority` column is 3 as declared in the table definition:

```
1 priority TINYINT NOT NULL DEFAULT 3
```

MySQL uses the number 3 to insert into the `priority` column.

INSERTING DATES INTO THE TABLE

To insert a literal date value into a column, you use the following format:

```
'YYYY-MM-DD'
```

In this format:

- YYYY represents a four-digit year e.g., 2018.
- MM represents a two-digit month e.g., 01, 02, and 12.
- DD represents a two-digit day e.g., 01, 02, 30.

The following example adds a new row to the `tasks` table with the start and due date values:

```
INSERT INTO tasks(title, start_date, due_date)
VALUES('Insert date into table','2018-01-09','2018-09-15');
```

The following picture shows the contents of the `tasks` table after the insert:

task_id	title	start_date	due_date	priority	description
1	Learn MySQL INSERT Statement	NULL	NULL	1	NULL
2	Understanding DEFAULT keyword in INSERT statement	NULL	NULL	3	NULL
3	Insert date into table	2018-01-09	2018-09-15	3	NULL

MySQL INSERT INTO SELECT

- Besides using row values in the VALUES clause, you can use the result of a SELECT statement as the data source for the INSERT statement.

```
INSERT INTO table_name(column_list)
SELECT
   select_list
FROM
   another_table;
```

```
INSERT INTO suppliers (
     supplierName,
     phone,
     addressLine1,
     addressLine2,
     city,
     state,
     postalCode,
     country,
     customerNumber
)
SELECT
     customerName,
     phone,
     addressLine1,
     addressLine2,
     city,
     state ,
     postalCode,
     country,
     customerNumber
FROM
     customers
WHERE
     country = 'USA' AND
     state = 'CA';
```

MySQL INSERT ON DUPLICATE KEY UPDATE

- When you insert a new row into a table if the row causes a duplicate in UNIQUE index or PRIMARY KEY , MySQL will issue an error.

- However, if you specify the ON DUPLICATE KEY UPDATE option in the INSERT statement, MySQL will update the existing row with the new values instead.

The syntax of `INSERT ON DUPLICATE KEY UPDATE` statement is as follows:

```
1 INSERT INTO table (column_list)
2 VALUES (value_list)
3 ON DUPLICATE KEY UPDATE
4    c1 = v1,
5    c2 = v2,
6    ...;
```

The only addition to the `INSERT` statement is the `ON DUPLICATE KEY UPDATE` clause where you specify a list of column-value-pair assignments in case of duplicate.

MySQL INSERT ON DUPLICATE KEY UPDATE EXAMPLE

	id	name
▶	1	Router F1
	2	Switch 1
	3	Switch 2

Now, we have three rows in the `devices` table.

After that, insert one more row into the `devices` table.

```
1  INSERT INTO
2     devices(name)
3  VALUES
4     ('Printer')
5  ON DUPLICATE KEY UPDATE name = 'Printer';
```

	id	name
▶	1	Router F1
	2	Switch 1
	3	Switch 2
	4	Printer

Because there is no duplicate, MySQL inserts a new row into the `devices` table. The statement above has the same effect as the following statement:

```
1  INSERT INTO devices(name)
2  VALUES ('Printer');
```

Finally, insert a row with a duplicate value in the `id` column.

```
1  INSERT INTO devices(id,name)
2  VALUES
3     (4,'Printer')
4  ON DUPLICATE KEY UPDATE name = 'Central Printer';
```

MySQL issues the following message:

```
1  2 row(s) affected
```

Because a row with id 4 already exists in the `devices` table, the statement updates the name from `Printer` to `Central Printer`.

	id	name
▶	1	Router F1
	2	Switch 1
	3	Switch 2
	4	Central Printer

Changep

MySQL INSERT IGNORE Statement

- When you use the INSERT statement to add multiple rows to a table and if an error occurs during the processing, MySQL terminates the statement and returns an error. As the result, no rows are inserted into the table.
- However, if you use the INSERT IGNORE statement, the rows with invalid data that cause the error are ignored and the rows with valid data are inserted into the table.

```
INSERT IGNORE INTO table(column_list)
VALUES( value_list),
      ( value_list),
      ...
```

```
INSERT INTO subscribers(email)
VALUES('john.doe@gmail.com'),
      ('jane.smith@ibm.com');
```

It returns an error.

```
Error Code: 1062. Duplicate entry 'john.doe@gmail.com' for key 'email'
```

As indicated in the error message, the email `john.doe@gmail.com` violates the `UNIQUE` constraint.

However, if you use the `INSERT IGNORE` statement instead.

```
INSERT IGNORE INTO subscribers(email)
VALUES('john.doe@gmail.com'),
      ('jane.smith@ibm.com');
```

MySQL returned a message indicating that one row was inserted and the other row was ignored.

```
1 row(s) affected, 1 warning(s): 1062 Duplicate entry 'john.doe@gmail.com' for key 'emai
l' Records: 2  Duplicates: 1  Warnings: 1
```

To find the detail of the warning, you can use the `SHOW WARNINGS` command as shown below:

```
SHOW WARNINGS;
```

Level	Code	Message
Warning	1062	Duplicate entry 'john.doe@gmail.com' for key 'email'

MySQL UPDATE

- You use the UPDATE statement to update existing data in a table. you can also use the UPDATE statement to change column values of a single row, a group of rows, or all rows in a table.

```
UPDATE [LOW_PRIORITY] [IGNORE] table_name
SET
    column_name1 = expr1,
    column_name2 = expr2,
    ...
[WHERE
    condition];
```

- MySQL supports two modifiers in the UPDATE statement.
- The LOW_PRIORITY modifier instructs the UPDATE statement to delay the update until there is no connection reading data from the table. The LOW_PRIORITY takes effect for the storage engines that use table-level locking only, for example, MyISAM, MERGE, MEMORY.
- The IGNORE modifier enables the UPDATE statement to continue updating rows even if errors occurred. The rows that cause errors such as duplicate-key conflicts are not updated.

```
UPDATE employees
SET
    email = 'mary.patterson@classicmodelcars.com'
WHERE
    employeeNumber = 1056;
```

MySQL UPDATE JOIN

- You often use joins to query rows from a table that have (in the case of INNER JOIN) or may not have (in the case of LEFT JOIN) matching rows in another table.
- In MySQL, you can use the JOIN clauses in the UPDATE statement to perform the cross-table update.

```
1 UPDATE T1, T2,
2 [INNER JOIN | LEFT JOIN] T1 ON T1.C1 = T2. C1
3 SET T1.C2 = T2.C2,
4     T2.C3 = expr
5 WHERE condition
```

- First, specify the main table (T1) and the table that you want the main table to join to (T2) after the UPDATE clause. Notice that you must specify at least one table after the UPDATE clause. The data in the table that is not specified after the UPDATE clause will not be updated.
- Next, specify a kind of join you want to use i.e., either INNER JOIN or LEFT JOIN and a join predicate. The JOIN clause must appear right after the UPDATE clause.
- Then, assign new values to the columns in T1 and/or T2 tables that you want to update.
- After that, specify a condition in the WHERE clause to limit rows to rows for updating.

MySQL UPDATE JOIN – EXAMPLE

```
1 UPDATE employees
2         INNER JOIN
3     merits ON employees.performance = merits.performance
4 SET
5     salary = salary + salary * percentage;
```

```
1 UPDATE employees
2         LEFT JOIN
3     merits ON employees.performance = merits.performance
4 SET
5     salary = salary + salary * 0.015
6 WHERE
7     merits.percentage IS NULL;
```

MySQL DELETE

- To delete data from a table, you use the MySQL DELETE statement. The following illustrates the syntax of the DELETE statement:

```
DELETE FROM table_name
WHERE condition;
```

Suppose you want to delete employees whose the `officeNumber` are4, you use the `DELETE` statement with the `WHERE` clause as shown in the following query:

```
DELETE FROM employees
WHERE
    officeCode = 4;
```

To delete all rows from the `employees` table, you use the `DELETE` statement without the `WHERE` clause as follows:

```
DELETE FROM employees;
```

MySQL DELETE and LIMIT clause

- If you want to limit the number of rows to be deleted, you use the LIMIT clause as follows

```
DELETE FROM table
LIMIT row_count;
```

Note that the order of rows in a table is unspecified, therefore, when you use the `LIMIT` clause, you should always use the `ORDER BY` clause.

```
DELETE FROM table_name
ORDER BY c1, c2, ...
LIMIT row_count;
```

For example, the following statement sorts customers by customer's names alphabetically and deletes the first 10 customers:

```
DELETE FROM customers
ORDER BY customerName
LIMIT 10;
```

Similarly, the following `DELETE` statement selects customers in `France`, sorts them by credit limit in from low to high, and deletes the first 5 customers:

```
DELETE FROM customers
WHERE country = 'France'
ORDER BY creditLimit
LIMIT 5;
```

MySQL DELETE JOIN

- MySQL also allows you to use the INNER JOIN clause in the DELETE statement to delete rows from a table and the matching rows in another table.

- For example, to delete rows from both T1 and T2 tables that meet a specified condition, you use the following statement:

```
1 DELETE T1, T2
2 FROM T1
3 INNER JOIN T2 ON T1.key = T2.key
4 WHERE condition;
```

Notice that you put table names `T1` and `T2` between the `DELETE` and `FROM` keywords. If you omit `T1` table, the `DELETE` statement only deletes rows in `T2` table. Similarly, if you omit`T2` table, the `DELETE` statement will delete only rows in `T1` table.

The expression `T1.key = T2.key` specifies the condition for matching rows between `T1` and`T2` tables that will be deleted.

The condition in the `WHERE` clause determine rows in the `T1` and `T2` that will be deleted.

MySQL DELETE JOIN WITH INNER JOIN EXAMPLE

```sql
DROP TABLE IF EXISTS t1, t2;

CREATE TABLE t1 (
    id INT PRIMARY KEY AUTO_INCREMENT
);

CREATE TABLE t2 (
    id VARCHAR(20) PRIMARY KEY,
    ref INT NOT NULL
);

INSERT INTO t1 VALUES (1),(2),(3);

INSERT INTO t2(id,ref) VALUES('A',1),('B',2),('C',3);
```

id
1
2
3

id	Ref
A	1
B	2
C	3

```sql
DELETE
      t1 , t2
FROM
      t1
      INNER JOIN t2
            ON t2.ref = t1.id
WHERE
      t1.id = 1;
```

MySQL DELETE JOIN WITH LEFT JOIN

- We often use the LEFT JOIN clause in the SELECT statement to find rows in the left table that have or don't have matching rows in the right table.

- We can also use the LEFT JOIN clause in the DELETE statement to delete rows in a table (left table) that does not have matching rows in another table (right table).

We can use DELETE statement with LEFT JOIN clause to clean up our customers master data. The following statement removes customers who have not placed any order:

```
1  DELETE customers
2  FROM customers
3           LEFT JOIN
4      orders ON customers.customerNumber = orders.customerNumber
5  WHERE
6      orderNumber IS NULL;
```

We can verify the delete by finding whether customers who do not have any order exists using the following query:

```
1  SELECT
2      c.customerNumber,
3      c.customerName,
4      orderNumber
5  FROM
6      customers c
7          LEFT JOIN
8      orders o ON c.customerNumber = o.customerNumber
9  WHERE
10     orderNumber IS NULL;
```

The query returned an empty result set which is what we expected.

MySQL REPLACE

- The MySQL REPLACE statement is a MySQL extension to the standard SQL. The MySQL REPLACE statement works as follows:
- If the new row already does not exist, the MySQL REPLACE statement inserts a new row.
- If the new row already exist, the REPLACE statement deletes the old row first and then inserts a new row. In some cases, the REPLACE statement updates the existing row only.
- To determine whether the new row already exists in the table, MySQL uses PRIMARY KEY or UNIQUE KEY index. If the table does not have one of these indexes, the REPLACE statement is equivalent to the INSERT statement.

MySQL REPLACE – EXAMPLE

First, create a new table named cities as follows:

```
CREATE TABLE cities (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50),
    population INT NOT NULL
);
```

Next, insert some rows into the cities table:

```
INSERT INTO cities(name,population)
VALUES('New York',8008278),
   ('Los Angeles',3694825),
   ('San Diego',1223405);
```

We query data from the cities table to verify the insert operation.

```
SELECT
    *
FROM
    cities;
```

id	name	population
1	New York	8008278
2	Los Angeles	3694825
3	San Diego	1223405

We have three cities in the cities table.

After that, use the REPLACE statement to update the population of the Los Angeles city to 3696820.

```
REPLACE INTO cities(id,population)
VALUES(2,3696820);
```

Finally, query the data of the cities table again to verify the replacement.

```
SELECT
    *
FROM
    cities;
```

id	name	population
1	New York	1008256
2	Los Angeles	3696820
3	San Diego	1223405

MySQL REPLACE – EXAMPLE

- The REPLACE statement first inserts the new row into the cities table with the information provided by the column list. The insertion fails because the row with id 2 already exists in the cities table, therefore, MySQL raises a duplicate-key error.
- The REPLACE statement then updates the row that has the key specified in the value of the id column. In the normal process, it would delete the old row with conflict id first and then inserts a new row.
- We know that the REPLACE statement did not delete the old row and inserted the new row because the value of the id column is 2 instead of 4.

MySQL REPLACE and INSERT

- The first form of the REPLACE statement is similar to the INSERT statement except the keyword INSERT is replaced by the REPLACE keyword as follows:

```
1 REPLACE INTO table_name(column_list)
2 VALUES(value_list);
```

For example, if you want to insert a new row into the cities table, you use the following query:

```
1 REPLACE INTO cities(name,population)
2 VALUES('Phoenix',1321523);
```

MySQL REPLACE and UPDATE

- The second form of REPLACE statement is similar to the UPDATE statement as follows:

```
REPLACE INTO table
SET column1 = value1,
    column2 = value2;
```

Notice that there is no WHERE clause in the REPLACE statement.

For example, if you want to update the population of the Phoenix city to 1768980, you use the REPLACE statement as follows:

```
REPLACE INTO cities
SET id = 4,
    name = 'Phoenix',
    population = 1768980;
```

Unlike the UPDATE statement, if you don't specify the value for the column in the SET clause, the REPLACE statement will use the default value of that column.

```
SELECT
    *
FROM
    cities;
```

id	name	population
1	New York	1008256
2	NULL	3696820
3	San Diego	1223405
4	Phoenix	1768980

QUERIES??

THANK YOU!!!