MySQL

1. SQL Constraints
2. Alter Table
3. Group By
4. Order By
5. SQL Dates
6. SQL Views

SQL Constraint	Function
NOT NULL	It ensures that a column does not accept NULL values.
CHECK	It ensures that a column accepts values within the specified range of values.
UNIQUE	It ensures that a column does not accept duplicate values.
PRIMARY KEY	It uniquely identifies a row in the table. It is a combination of NOT NULL and UNIQUE constraints.
FOREIGN KEY	It is like a primary key constraint only. But it uniquely identifies a row in another table.
DEFAULT	It ensures that the column sets a default value for empty records.

1. NOT NULL CONSTRAINT

When a NOT NULL constraint is applied to a column, it ensures it will not accept NULL values. The syntax for the NOT NULL constraint in MYSQL is as follows :

Syntax:

```
CREATE  TABLE table_name(
column_name_1 datatype  NOT NULL,
column_name_2 datatype  NOT NULL,
.
.
column_name_n datatype  NOT NULL
);
```

The parameters used in the syntax are :

- **CREATE TABLE table_name:** This statement is used to create a new database table. Specify the name of the table.
- **column_name1, column_name2, ...:** Specify the name of the columns you want to create on the table.
- **datatype:** Specify the datatype of each column in the table.
- **[NULL | NOT NULL]:** Specify whether it can hold NULL values.

Example

SQL query to create NOT NULL constraint while creating a table.

Query:

```
CREATE TABLE students (
student_ID int NOT NULL,
student_Name varchar(255) NOT NULL,
class_name varchar(255) NOT NULL,
Age int
);
```

2. CHECK CONSTRAINT

When the CHECK constraint is applied to a column, it ensures it will not accept data values outside the specified range. The CHECK constraint can also be mentioned to ensure that data meets a specified condition like less than, equal to, not, etc.

Syntax:

```
CREATE  TABLE table_name(
column_name_1 datatype,
column_name_2 datatype
CHECK(condition_on_column_name_2),
.
.
column_name_n datatype
);
```

Parameters:

- **CREATE TABLE table_name:** This statement is used to create a new database table. Specify the name of the table.
- **column_name1, column_name2, ... :** Specify the name of the columns you want to create on the table.
- **datatype:** Specify the datatype of each column in the table.
- **CHECK(condition_on_column_name_2):** Specify the condition data should meet before entering a column.

Example

SQL query to create CHECK constraint while creating a table.

Query:

```
CREATE TABLE students (
student_ID int NOT NULL,
student_Name varchar(255) NOT NULL,
class_name varchar(255) NOT NULL,
Age int
CHECK(Age >9)
);
```

3. UNIQUE KEY CONSTRAINT

A unique key is a constraint in SQL that helps uniquely identify a record in the datatable. It can be considered somewhat similar to the Primary key as both guarantee a record's uniqueness. But unlike the primary key, a **unique key can accept NULL values** and be used on more than one column of the datatable.

Syntax:

```
CREATE TABLE table_name
(
Column_name1 datatype [NULL | NOT NULL] UNIQUE,
Column_name2 datatype [NULL | NOT NULL],
Column_name3 datatype [NULL | NOT NULL]
);
```

Parameters:

- **CREATE TABLE:** This statement is used to create a new database table.
- **Column_name1, Column_name2, Column_name3:** Mention the name of the columns you want to create on the table.
- **datatype:** Mention the datatype of each column in the table.
- **[NULL | NOT NULL]:** Mention whether it can hold NULL values.
- **UNIQUE:** Unique keyword written with the column name creates a unique key constraint. It ensures that there are no duplicate values in that particular column.

Example

SQL query to **create unique constraints** while creating a table.

Query:

```
CREATE TABLE students (
student_ID int UNIQUE,
student_Name varchar(255) NOT NULL,
class_name varchar(255) NOT NULL,
Age int
);
```

4. PRIMARY KEY CONSTRAINT

A primary key constraint uniquely identifies a record or row in the data table. It does not accept NULL values and accepts unique values.

Syntax:

```
CREATE  TABLE table_name(
column_name_1 datatype  NOT NULL,
column_name_2 datatype  NOT NULL,
.
.
column_name_n datatype  NOT NULL
PRIMARY KEY (column_name_1)
);
```

Parameters:

- **CREATE TABLE table_name:** This statement is used to create a new database table. Specify the name of the table.
- **column_name1, column_name2, ... :** Specify the name of the columns you want to create on the table.
- **datatype:** Specify the datatype of each column in the table.
- **PRIMARY KEY (column_name_1):** Specify the column name on which the primary key constraint has to be applied. It should be a NOT NULL column.

5. FOREIGN KEY CONSTRAINT

A foreign key constraint uniquely identifies a record or row in another data table. It refers to the primary key in the other table. So, it acts as a link between two tables.

Syntax:

```
CREATE  TABLE table_name_1(
column_name_1 datatype  NOT NULL,
column_name_2 datatype  NOT NULL,
.
.
column_name_n datatype  NOT NULL
PRIMARY KEY (column_name_1)
FOREIGN KEY (column_name_2) REFERENCES table_name_2(column_name_2)
);
```

1. **CREATE TABLE table_name_1:** This statement is used to create a new database table. Specify the name of the table.
2. **column_name1, column_name2, ... :** Specify the name of the columns you want to create on the table.
3. **datatype:** Specify the datatype of each column in the table.
4. **PRIMARY KEY (column_name_1):** Specify the column name on which the primary key constraint has to be applied. It should be a NOT NULL column.
5. **FOREIGN KEY (column_name_2):** Specify the column name on which foreign key constraint has to be applied.
6. **REFERENCES table_name_2(column_name_2):** Specify the table name and the column on which the foreign key in the first table is linked.

Example

SQL query to create foreign key constraints while creating a table.

Query:

```
CREATE TABLE Students(
Student_ID int NOT NULL,
Student_Name varchar(255) NOT NULL,
Class_Name varchar(255),
Age int,
PRIMARY KEY (Student_ID)
FOREIGN KEY (Class_Name) References classes(Class_Name)
);
```

6. DEFAULT CONSTRAINT

To ensure consistency in data, we sometimes require missing data or non-entered values to be set to a default value. We can use the DEFAULT constraint in such cases to ensure that the column is set to default.

Syntax:

```
CREATE  TABLE table_name(
column_name_1 datatype  NOT NULL,
column_name_2 datatype DEFAULT 'default_value',
.
.
);
```

Example

SQL query to create default constraints while creating a table.

Query:

```
CREATE TABLE Students(
Student_ID int NOT NULL,
Student_Name varchar(255) NOT NULL,
Class_Name varchar(255) DEFAULT 'IV'
);
```

Introduction to MySQL RENAME TABLE statement

Because business requirements change, we need to rename the current table to a new one to better reflect the new situation. MySQL provides us with a very useful statement that changes the name of one or more tables.

To change one or more tables, we use the `RENAME TABLE` statement as follows:

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

ALTER

```
CREATE TABLE my_table (
  id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  name VARCHAR(100) NOT NULL,
  PRIMARY KEY (id)
) AUTO_INCREMENT = 100;
```

or with ALTER TABLE statement

```
ALTER TABLE my_table AUTO_INCREMENT = 200;
```

Introduction to MySQL DROP COLUMN statement

In some situations, you want to remove one or more columns from a table. In such cases, you use the following `ALTER TABLE DROP COLUMN` statement:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Introduction to MySQL GROUP BY clause

The `GROUP BY` clause groups a set of rows into a set of summary rows by values of columns or expressions. The `GROUP BY` clause returns one row for each group. In other words, it reduces the number of rows in the result set.

The `GROUP BY` clause is an optional clause of the `SELECT` statement. The following illustrates the `GROUP BY` clause syntax:

```
SELECT
    c1, c2,..., cn, aggregate_function(ci)
FROM
    table
WHERE
    where_conditions
GROUP BY c1 , c2,...,cn;
```

Introduction to the MySQL ORDER BY clause

When you use the `SELECT` statement to query data from a table, the order of rows in the result set is unspecified. To sort the rows in the result set, you add the `ORDER BY` clause to the `SELECT` statement.

The following illustrates the syntax of the `ORDER BY` clause:

```
SELECT
   select_list
FROM
   table_name
ORDER BY
   column1 [ASC|DESC],
   column2 [ASC|DESC],
   ...;
```

SQL Date Data Types

MySQL comes with the following data types for storing a date or a date/time value in the database:

- `DATE` - format YYYY-MM-DD
- `DATETIME` - format: YYYY-MM-DD HH:MI:SS
- `TIMESTAMP` - format: YYYY-MM-DD HH:MI:SS
- `YEAR` - format YYYY or YY

First, create a table named people with birth date column with `DATE` data type.

```
CREATE TABLE people (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    birth_date DATE NOT NULL
);
```

Next, insert a row into the `people` table.

```
INSERT INTO people(first_name,last_name,birth_date)
VALUES('John','Doe','1990-09-01');
```

SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the `CREATE VIEW` statement.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

MySQL DATE functions

MySQL provides many useful date functions that allow you to manipulate date effectively.

To get the current date and time, you use `NOW()` function.

```
SELECT NOW();
```

```
+---------------------+
| NOW()               |
+---------------------+
| 2017-05-13 07:59:38 |
+---------------------+
1 row in set (0.02 sec)
```

To get only date part of a `DATETIME` value, you use the `DATE()` function.

```
SELECT DATE(NOW());
```

```
+-------------+
| DATE(NOW()) |
+-------------+
| 2015-07-13  |
+-------------+
1 row in set (0.01 sec)
```

To get the current system date, you use `CURDATE()` function as follows:

```
SELECT CURDATE();
```

```
+------------+
| CURDATE()  |
+------------+
| 2015-07-13 |
+------------+
1 row in set (0.02 sec)
```

To format a date value, you use `DATE_FORMAT` function. The following statement formats th as `mm/dd/yyyy` using the date format pattern `%m/%d/%Y` :

```
SELECT DATE_FORMAT(CURDATE(), '%m/%d/%Y') today;
```

```
+------------+
| today      |
+------------+
| 07/13/2015 |
+------------+
```

To calculate the number of days between two date values, you use the `DATEDIFF` function

```
SELECT DATEDIFF('2015-11-04','2014-11-04') days;
```

```
+------+
| days |
+------+
|  365 |
+------+
1 row in set (0.02 sec)
```

To add a number of days, weeks, months, years, etc., to a date value, you use the `DATE_ADD`

```
SELECT
    '2015-01-01' start,
    DATE_ADD('2015-01-01', INTERVAL 1 DAY) 'one day later',
    DATE_ADD('2015-01-01', INTERVAL 1 WEEK) 'one week later',
    DATE_ADD('2015-01-01', INTERVAL 1 MONTH) 'one month later',
    DATE_ADD('2015-01-01', INTERVAL 1 YEAR) 'one year later';
```

start	one day later	one week later	one month later	one year later
2015-01-01	2015-01-02	2015-01-08	2015-02-01	2016-01-01

Similarly, you can subtract an interval from a date using the `DATE_SUB` function:

```
SELECT
    '2015-01-01' start,
    DATE_SUB('2015-01-01', INTERVAL 1 DAY) 'one day before',
    DATE_SUB('2015-01-01', INTERVAL 1 WEEK) 'one week before',
    DATE_SUB('2015-01-01', INTERVAL 1 MONTH) 'one month before',
    DATE_SUB('2015-01-01', INTERVAL 1 YEAR) 'one year before';
```

start	one day before	one week before	one month before	one year before
2015-01-01	2014-12-31	2014-12-25	2014-12-01	2014-01-01

If you want to get the day, month, quarter, and year of a date value, you can use the corresponding function `DAY`, `MONTH`, `QUARTER`, and `YEAR` as follows:

```
SELECT DAY('2000-12-31') day,
       MONTH('2000-12-31') month,
       QUARTER('2000-12-31') quarter,
       YEAR('2000-12-31') year;
```

```
+------+-------+---------+------+
| day  | month | quarter | year |
+------+-------+---------+------+
|   31 |    12 |       4 | 2000 |
+------+-------+---------+------+
1 row in set (0.00 sec)
```

To get the week information week related functions. For example, `WEEK` function returns the week number, `WEEKDAY` function returns the weekday index, and `WEEKOFYEAR` function returns the calendar week.

```
SELECT
    WEEKDAY('2000-12-31') weekday,
    WEEK('2000-12-31') week,
    WEEKOFYEAR('2000-12-31') weekofyear;
```

```
+---------+------+------------+
| weekday | week | weekofyear |
+---------+------+------------+
|       6 |   53 |         52 |
+---------+------+------------+
1 row in set (0.04 sec)
```

1.Display Year and total number of payments done in that year

2.Display year ,total number of bills , sum of amount, average of amount against each year

1.Display month ,total number of bills , sum of amount, average of amount against each month

1.Modify Above query to show *Year-Month* in place of *Year*

Introduction to the MySQL Subquery

A MySQL subquery is a query nested within another query such as `SELECT`, `INSERT`, `UPDATE` or `DELETE`. Also, a subquery can be nested within another subquery.

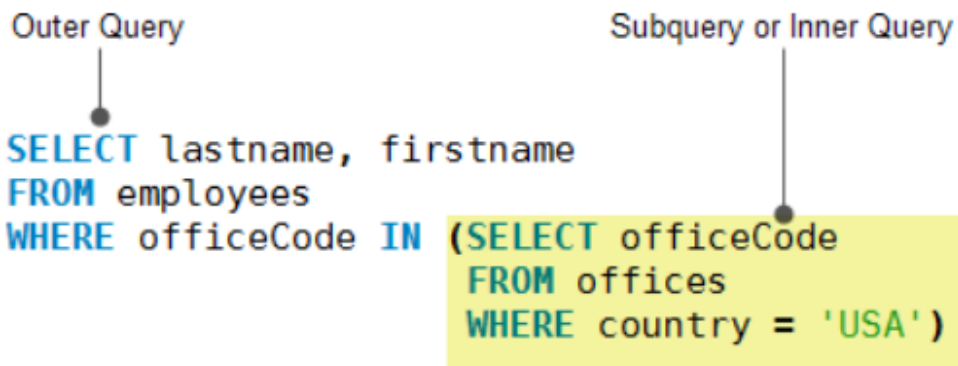
A MySQL subquery is called an inner query while the query that contains the subquery is called an outer query. A subquery can be used anywhere that expression is used and must be closed in parentheses.

For example, the following query uses a subquery to return the employees who work in the offices located in the USA.

```
SELECT
    lastName, firstName
FROM
    employees
WHERE
    officeCode IN (SELECT
            officeCode
        FROM
            offices
        WHERE
            country = 'USA');
```

In this example:

- The subquery returns all *office codes* of the offices located in the USA.
- The outer query selects the last name and first name of employees who work in the offices whose office codes are in the result set returned by the subquery.



MySQL subquery with comparison operators

You can use comparison operators e.g., =, >, < to compare a single value returned by the subquery with the expression in the `WHERE` clause.

For example, the following query returns the customer who has the highest payment.

```
SELECT
    customerNumber,
    checkNumber,
    amount
FROM
    payments
WHERE
    amount = (SELECT MAX(amount) FROM payments);
```

For example, you can find customers whose payments are greater than the average payment using a subquery:

```
SELECT
    customerNumber,
    checkNumber,
    amount
FROM
    payments
WHERE
    amount > (SELECT
            AVG(amount)
        FROM
            payments);
```