MYSQL

The MySQL SELECT Statement

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

SELECT Syntax

```
SELECT column1, column2, ...
FROM table_name;
```

SELECT roll, fname, lname from students;

SELECT roll, fname, lname, city, phone, gender from students;

SELECT * from students;

SELECT DISTINCT fname FROM students;

SELECT count(DISTINCT fname) from students;

The MySQL WHERE Clause

The `WHERE` clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

WHERE Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition
```

SELECT * from students WHERE roll = 5

SELECT * from students WHERE roll > 5;

SELECT * from students WHERE fname = 'kalpesh';

SELECT * from students WHERE not fname = 'kalpesh';

The MySQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

NOT Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

SELECT * from students WHERE fname = 'kalpesh' and roll = 10

SELECT * from students WHERE fname = 'kalpesh' and lname = 'chauhan';

SELECT * from students WHERE fname = 'kalpesh' and fname = 'harshil';

SELECT * from students WHERE fname = 'kalpesh' or fname = 'harshil';

SELECT * from students WHERE fname = 'kalpesh' or roll = 10

SELECT * from students WHERE not fname = 'kalpesh' or roll = 10;

The MySQL ORDER BY Keyword

The `ORDER BY` keyword is used to sort the result-set in ascending or descending order.

The `ORDER BY` keyword sorts the records in ascending order by default. To sort the records in descending order, use the `DESC` keyword.

SELECT * FROM `students` ORDER by fname

SELECT * FROM `students` ORDER by fname desc

ORDER BY Several Columns Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

SELECT * FROM `students` ORDER by fname, city;

SELECT * FROM `students` ORDER by fname, city desc;

The MySQL INSERT INTO Statement

The `INSERT INTO` statement is used to insert new records in a table.

INSERT INTO Syntax

It is possible to write the `INSERT INTO` statement in two ways:

1. Specify both the column names and the values to be inserted:

   INSERT INTO students (fname, lname, city, phone, gender) values ('sunny', 'sata', 'rajkot', '7894651320', 'male')

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the `INSERT INTO` syntax would be as follows:

INSERT INTO students values ('sunny', 'sata', 'rajkot', '7894651320', 'male')

Insert Data Only in Specified Columns

It is also possible to only insert data in specific columns.

INSERT INTO students (fname, lname) values ('sunny', 'sata')

What is a NULL Value?

A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

Note: A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

```
INSERT into students (fname, lname) values ('shreyas', 'riska');
```

How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the `IS NULL` and `IS NOT NULL` operators instead.

IS NULL Syntax

SELECT * FROM `students` where city is null

SELECT * FROM `students` where city is not null;

The MySQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement.
The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

update students set city = 'surat' // never do this without where clause.

update students set city = 'surat' where roll > 10;

update students set city = 'rajkot', phone = '9632574410' where roll > 10;

UPDATE Multiple Records

It is the WHERE clause that determines how many records will be updated.

The MySQL LIMIT Clause

The `LIMIT` clause is used to specify the number of records to return.

The `LIMIT` clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

LIMIT Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

SELECT * FROM `students` LImit 5

SELECT * FROM `students` WHERE city = 'rajkot' limit 3;

SELECT * FROM `students` LIMIT 5 OFFSET 5;

MySQL MIN() and MAX() Functions

The `MIN()` function returns the smallest value of the selected column.

The `MAX()` function returns the largest value of the selected column.

MIN() Syntax

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

SELECT min(roll) FROM students

SELECT max(roll) FROM students;

SELECT count(roll) FROM students;

SELECT sum(roll) FROM students;

SELECT avg(roll) FROM students;

The MySQL LIKE Operator

The `LIKE` operator is used in a `WHERE` clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the `LIKE` operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

The percent sign and the underscore can also be used in combinations!

LIKE Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

Tip: You can also combine any number of conditions using `AND` or `OR` operators.

Here are some examples showing different `LIKE` operators with '%' and '_' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

```
SELECT * from students WHERE fname like '%a'

SELECT * from students WHERE fname like '%a%';

SELECT * from students WHERE fname not like '%a%';

SELECT * from students WHERE fname like '_a%';
```

The MySQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

DELETE Syntax

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

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement.
The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

DELETE from students WHERE city = 'rajkot'

Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name;
```

The following SQL statement deletes all rows in the "Customers" table, without deleting the table:

Example

```
DELETE FROM Customers;
```

delete FROM `students`

Truncate table is used to delete all the data from table and also reset table structure.

truncate TABLE students

The MySQL IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

SELECT * FROM `students` WHERE fname = 'sunny' or fname = 'vivek' or fname = 'jigar'

SELECT * FROM `students` WHERE fname in ('jigar', 'vivek', 'sunny');

SELECT * FROM `students` WHERE fname not in ('jigar', 'vivek', 'sunny');

The MySQL BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

SELECT * FROM `students` WHERE roll BETWEEN 1 and 10

SELECT * FROM `students` WHERE roll not BETWEEN 1 and 10;

MySQL Aliases

Aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the AS keyword.

Alias Column Syntax

```
SELECT column_name AS alias_name
FROM table_name;
```

SELECT * FROM `students` WHERE fname BETWEEN 'ganga' and 'vivek'

select roll, concat_ws("_", fname, lname, city, phone, gender) as "Student Data" from students

SELECT COUNT(roll) from students

SELECT COUNT(roll) as "Total Students" from students;

```
SELECT fname as "First Name", lname as "Last Name" from students
```

MySQL Joining Tables

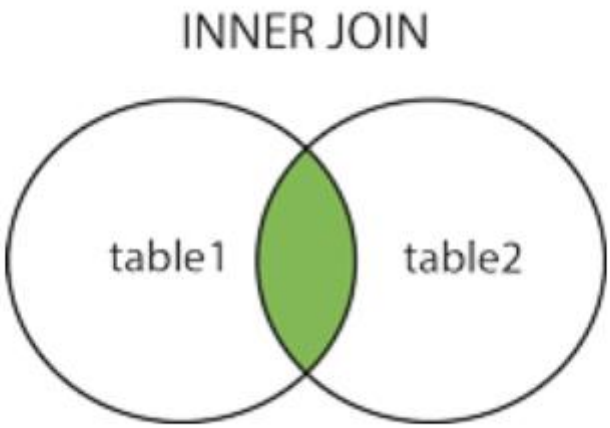
A `JOIN` clause is used to combine rows from two or more tables, based on a related column between them.

Supported Types of Joins in MySQL

- **INNER JOIN: Returns records that have matching values in both tables**
- **LEFT JOIN: Returns all records from the left table, and the matched records from the right table**
- **RIGHT JOIN: Returns all records from the right table, and the matched records from the left table**
- **CROSS JOIN: Returns all records from both tables**

The `INNER JOIN` keyword selects records that have matching values in both tables.

The `INNER JOIN` keyword selects records that have matching values in both tables.



INNER JOIN Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

INNER JOIN Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

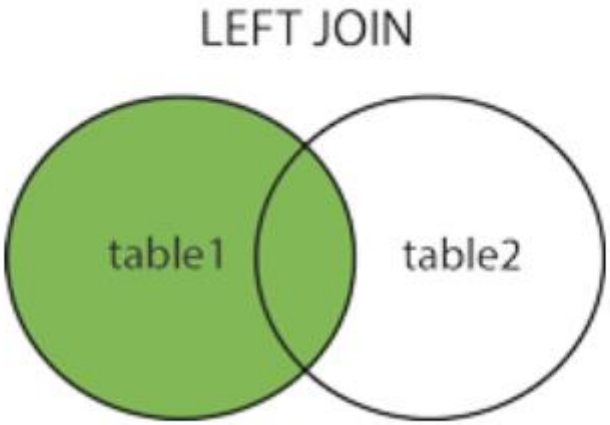
SELECT students.roll, students.fname, students.lname, students.city, students.phone, students.gender, result.marks, result.status from students INNER join result on students.roll = result.roll

SELECT s.roll, s.fname, s.lname, s.city, s.phone, s.gender, r.marks, r.status from students s INNER join result r on s.roll = r.roll;

SELECT s.roll, s.fname, s.lname, s.city, s.phone, s.gender, r.marks, r.status, a.presents, a.absents from students s INNER join result r on s.roll = r.roll inner join attendance a on s.roll = a.roll;

MySQL LEFT JOIN Keyword

The `LEFT JOIN` keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2).



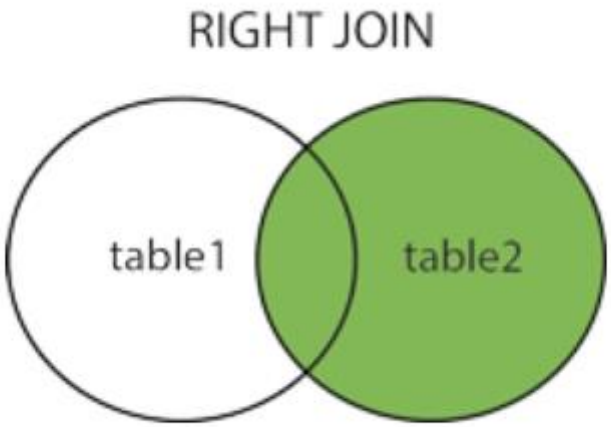
LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

SELECT s.roll, s.fname, s.lname, s.city, s.phone, s.gender, r.marks, r.status from students s left join result r on s.roll = r.roll;

MySQL RIGHT JOIN Keyword

The `RIGHT JOIN` keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1).



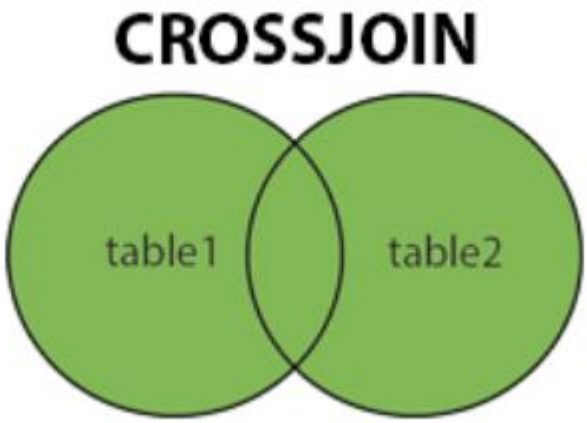
RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

```
SELECT s.roll, s.fname, s.lname, s.city, s.phone, s.gender, r.marks, r.status from students s cross join result
```

SQL CROSS JOIN Keyword

The `CROSS JOIN` keyword returns all records from both tables (table1 and table2).



CROSS JOIN Syntax

```
SELECT column_name(s)
FROM table1
CROSS JOIN table2;
```

Note: `CROSS JOIN` can potentially return very large result-sets!

The MySQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

UNION Syntax

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

UNION ALL Syntax

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

Note: The column names in the result-set are usually equal to the column names in the first SELECT statement.

SELECT * FROM students

union

SELECT * FROM backup_students

SELECT * FROM students

union all

SELECT * FROM backup_students;

The MySQL GROUP BY Statement

The `GROUP BY` statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The `GROUP BY` statement is often used with aggregate functions (`COUNT()`, `MAX()`, `MIN()`, `SUM()`, `AVG()`) to group the result-set by one or more columns.

SELECT gender, COUNT(*) FROM students GROUP by gender

SELECT city, count(*) from students GROUP by city

SELECT city, count(*) from students GROUP by city ORDER by count(*);

SELECT city, count(*) from students where gender = 'male' GROUP by city ORDER by count(*);

The MySQL HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

HAVING Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

SELECT city, count(*) from students where gender = 'male' GROUP by city having COUNT(*) >= 3 ORDER by count(*)

The MySQL EXISTS Operator

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

EXISTS Syntax

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

SELECT * from students where EXISTS (SELECT roll FROM attendance WHERE presents >= 100);

SELECT * from students where EXISTS (SELECT roll FROM attendance WHERE presents >= 500);

The MySQL ANY and ALL Operators

The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

The ANY Operator

The ANY operator:

- returns a boolean value as a result
- returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

ANY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

Note: The *operator* must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

The ALL Operator

The ALL operator:

- returns a boolean value as a result
- returns TRUE if ALL of the subquery values meet the condition
- is used with SELECT, WHERE and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

ALL Syntax With SELECT

```
SELECT ALL column_name(s)
FROM table_name
WHERE condition;
```

SELECT * from students WHERE roll = any (SELECT roll FROM result WHERE status = 'fail');

SELECT * from students where roll = 1

SELECT * from students where roll = any (SELECT roll from result where status = 'pass');

SELECT * from students where roll = all (SELECT roll from result where status = 'pass');

SELECT * from students where roll = all (SELECT roll from result where marks > 100);

SELECT roll, fname, lname from students where roll = all (SELECT roll FROM result WHERE marks = 123);

SELECT roll, fname, lname from students where roll = all (SELECT roll FROM result WHERE marks >= 123);

```
SELECT * from students WHERE roll > all( select roll from result where marks<125);
```

The MySQL INSERT INTO SELECT Statement

The `INSERT INTO SELECT` statement copies data from one table and inserts it into another table.

The `INSERT INTO SELECT` statement requires that the data types in source and target tables matches.

Note: The existing records in the target table are unaffected.

INSERT INTO SELECT Syntax

Copy all columns from one table to another table:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

Copy only some columns from one table into another table:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

TRUNCATE TABLE backup_students

INSERT INTO backup_students SELECT * FROM students WHERE city = 'rajkot'

The MySQL CASE Statement

The CASE statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

CASE Syntax

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

SELECT roll, fname, lname, city, CASE

WHEN city = 'Rajkot' THEN 'Home Town'

WHEN city = 'Ahamdabad' THEN 'Around 200 Kms.'

WHEN city = 'Baroda' THEN 'Around 300 Kms.'

ELSE 'Remote City More than 500 Kms.'

end as 'Location', phone, gender from students

MySQL IFNULL() and COALESCE() Functions

Look at the following "Products" table:

P_Id	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder
1	Jarlsberg	10.45	16	15
2	Mascarpone	32.56	23	
3	Gorgonzola	15.67	9	20

Suppose that the "UnitsOnOrder" column is optional, and may contain NULL values.

Look at the following SELECT statement:

```
SELECT ProductName, UnitPrice * (UnitsInStock + UnitsOnOrder)
FROM Products;
```

SELECT roll, presents, absents, (presents + absents) as 'Total Days' from attendance;

SELECT roll, presents, absents, (ifnull(presents, 0) + ifnull(absents,0)) as 'Total Days' from attendance;

SELECT roll, presents, absents, (COALESCE(presents, 0) + COALESCE(absents,0)) as 'Total Days' from attendance;

MySQL Comments

Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.

Single Line Comments

Single line comments start with --.

Any text between -- and the end of the line will be ignored (will not be executed).

The following example uses a single-line comment as an explanation:

```
-- Select all:
SELECT * FROM Customers;
```

The following example uses a single-line comment to ignore the end of a line:

Example

```
SELECT * FROM Customers -- WHERE City='Berlin';
```

SELECT roll, presents, absents, (COALESCE(presents, 0) + COALESCE(absents,0)) as 'Total Days' from attendance; -- get total number of working days

SELECT roll, /*presents, absents,*/ (COALESCE(presents, 0) + COALESCE(absents,0)) as 'Total Days' from attendance; -- get total number of working days

The MySQL CREATE DATABASE Statement

The `CREATE DATABASE` statement is used to create a new SQL database.

Syntax

```
CREATE DATABASE databasename;
```

```
Create database jigar
```

show DATABASEs;

The MySQL DROP DATABASE Statement

The `DROP DATABASE` statement is used to drop an existing SQL database.

Syntax

```
DROP DATABASE databasename;
```

drop DATABASE jigar

The MySQL CREATE TABLE Statement

The `CREATE TABLE` statement is used to create a new table in a database.

Syntax

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

create TABLE person (personid int, fname varchar(20), lname varchar(30), city varchar(50), email varchar(100), salary decimal, dob date, created_at timestamp)

Create Table Using Another Table

A copy of an existing table can also be created using `CREATE TABLE`.

The new table gets the same column definitions. All columns or specific columns can be selected.

If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

Syntax

```
CREATE TABLE new_table_name AS
    SELECT column1, column2,...
    FROM existing_table_name
    WHERE ....;
```

create TABLE backup_person as SELECT * FROM person

The MySQL DROP TABLE Statement

The `DROP TABLE` statement is used to drop an existing table in a database.

Syntax

```
DROP TABLE table_name;
```

Note: Be careful before dropping a table. Deleting a table will result in loss of complete information stored in the table!

MySQL ALTER TABLE Statement

The `ALTER TABLE` statement is used to add, delete, or modify columns in an existing table.

The `ALTER TABLE` statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype;
```

The following SQL adds an "Email" column to the "Customers" table:

ALTER TABLE person ADD Email varchar(255);

ALTER TABLE person MODIFY COLUMN lname varchar(20)

alter TABLE person drop COLUMN Email

alter TABLE person add COLUMN email varchar(20) AFTER lname

MySQL Constraints

SQL constraints are used to specify rules for data in a table.

Create Constraints

Constraints can be specified when the table is created with the `CREATE TABLE` statement, or after the table is created with the `ALTER TABLE` statement.

Syntax

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

MySQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- `NOT NULL` - Ensures that a column cannot have a NULL value
- `UNIQUE` - Ensures that all values in a column are different
- `PRIMARY KEY` - A combination of a `NOT NULL` and `UNIQUE`. Uniquely identifies each row in a table
- `FOREIGN KEY` - Prevents actions that would destroy links between tables
- `CHECK` - Ensures that the values in a column satisfies a specific condition
- `DEFAULT` - Sets a default value for a column if no value is specified
- `CREATE INDEX` - Used to create and retrieve data from the database very quickly

MySQL NOT NULL Constraint

By default, a column can hold NULL values.

The `NOT NULL` constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

NOT NULL on CREATE TABLE

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

create TABLE person (personid int, fname varchar(20) **not null**, lname varchar(20))

INSERT INTO `person` (`personid`, `fname`, `lname`) VALUES (NULL, 'Kairvi', NULL);

NOT NULL on ALTER TABLE

To create a `NOT NULL` constraint on the "Age" column when the "Persons" table is already created, use the following SQL:

Example

```
ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

alter TABLE person MODIFY COLUMN lname varchar(20) **not null**

MySQL UNIQUE Constraint

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

create TABLE person (personid int UNIQUE, fname varchar(20), lname varchar(20), city varchar(20), age int)

INSERT INTO `person` (`personid`, `fname`, `lname`, `city`, `age`) VALUES ('1', 'Sunny', 'Sata', 'Jetpur', '19');

INSERT INTO `person` (`personid`, `fname`, `lname`, `city`, `age`) VALUES ('1', 'SUNNY', 'Sata', 'Jetpur', '19');

INSERT INTO `person` (`fname`, `lname`, `city`, `age`) VALUES ('SUNNY', 'Sata', 'Jetpur', '19');

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

ALTER table person add CONSTRAINT unq_fn_ln UNIQUE(fname, lname)

INSERT INTO `person` (`personid`, `fname`, `lname`, `city`, `age`) VALUES (NULL, 'Sunny', 'Satta', NULL, NULL);

INSERT INTO `person` (`fname`, `lname`, `city`, `age`) VALUES ('SonNY', 'Sata', 'Jetpur', '19');

alter table person DROP CONSTRAINT unq_fn_ln

UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    UNIQUE (ID)
);
```

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

UNIQUE Constraint on ALTER TABLE

To create a UNIQUE constraint on the "ID" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons
ADD UNIQUE (ID);
```

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

```
ALTER TABLE Persons
ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
```

DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

```
ALTER TABLE Persons
DROP INDEX UC_Person;
```

MySQL PRIMARY KEY Constraint

The `PRIMARY KEY` constraint uniquely identifies each record in a table.

Primary keys must contain **UNIQUE** values, and **cannot contain NULL** values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

`

PRIMARY KEY on CREATE TABLE

The following SQL creates a `PRIMARY KEY` on the "ID" column when the "Persons" table is created:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
```

---

create TABLE persons (id int PRIMARY key, fname varchar(20), lname varchar(20), age int)

INSERT into persons (id, fname, lname, age) values (1, 'kairvi', 'parsana', 19)

INSERT into persons (id, fname, lname, age) values (1, 'kairvi', 'patel', 19)

---

create TABLE persons (id int, fname varchar(20), lname varchar(20), age int)

alter table persons add CONSTRAINT pri_key PRIMARY key(id)

INSERT into persons (id, fname, lname, age) values (1, 'kairvi', 'parsana', 19)

INSERT into persons (id, fname, lname, age) values (1, 'kairvi', 'patel', 19)

alter TABLE persons drop PRIMARY key

MySQL FOREIGN KEY Constraint

The `FOREIGN KEY` constraint is used to prevent actions that would destroy links between tables.

A `FOREIGN KEY` is a field (or collection of fields) in one table, that refers to the `PRIMARY KEY` in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Look at the following two tables:

Persons Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the `PRIMARY KEY` in the "Persons" table.

The "PersonID" column in the "Orders" table is a `FOREIGN KEY` in the "Orders" table.

The `FOREIGN KEY` constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

create table fees (feesid int AUTO_INCREMENT PRIMARY key, roll int, paymentdate date, amount decimal(10,2), FOREIGN key(roll) REFERENCES students(roll))

INSERT INTO `fees` (`feesid`, `roll`, `paymentdate`, `amount`) VALUES (NULL, '6', '2023-06-14', '8500');

INSERT INTO `fees` (`feesid`, `roll`, `paymentdate`, `amount`) VALUES (NULL, '6', '2023-06-14', '8500');

---

alter TABLE attendance add CONSTRAINT fk_roll FOREIGN key (roll) REFERENCES students(roll)

alter TABLE attendance drop CONSTRAINT fk_roll

MySQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

ALTER table attendance add CONSTRAINT check_presents CHECK (presents >= 0)

alter TABLE attendance drop CONSTRAINT check_presents

MySQL DEFAULT Constraint

The DEFAULT constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

DEFAULT on CREATE TABLE

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);
```

The DEFAULT constraint can also be used to insert system values, by using functions like CURRENT_DATE():

```
CREATE TABLE Orders (
    ID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderDate date DEFAULT CURRENT_DATE()
);
```

alter TABLE students add COLUMN state varchar(20) DEFAULT 'gujarat' AFTER city

alter TABLE students add COLUMN created_at timestamp DEFAULT CURRENT_TIMESTAMP

INSERT INTO `students` (`roll`, `fname`, `lname`, `city`, `state`, `phone`, `gender`, `created_at`) VALUES (NULL, 'vivek', 'gorasiya', 'Rajkot', 'gujarat', '7894561230', 'Male', current_timestamp());

DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:

```
ALTER TABLE Persons
ALTER City DROP DEFAULT;
```

alter TABLE students ALTER state DROP DEFAULT

MySQL CREATE INDEX Statement

The `CREATE INDEX` statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

CREATE index fnameindex on students(fname)

DROP INDEX Statement

The `DROP INDEX` statement is used to delete an index in a table.

```
ALTER TABLE table_name
DROP INDEX index_name;
```

What is an AUTO INCREMENT Field?

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

MySQL AUTO_INCREMENT Keyword

MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

The following SQL statement defines the "Personid" column to be an auto-increment primary key field in the "Persons" table:

```
CREATE TABLE Persons (
    Personid int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (Personid)
);
```

To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:

```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

ALTER TABLE students AUTO_INCREMENT = 501

MySQL Dates

The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.

As long as your data contains only the date portion, your queries will work as expected. However, if a time portion is involved, it gets more complicated.

MySQL 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

Note: The date data type are set for a column when you create a new table in your database!

SELECT * from fees WHERE paymentdate = '2023-06-01'

SELECT paymentdate, sum(amount) from fees GROUP by paymentdate

SELECT paymentdate, sum(amount) from fees where paymentdate BETWEEN '2023-06-01' and '2023-06-10' GROUP by paymentdate ;

SELECT * FROM `students` where created_at = '2023-06-02'

SELECT * FROM `students` where created_at like '2023-06-02%';

SELECT * FROM `students` where date(created_at) = '2023-06-02';