



SQL Introduction

What is SQL



- > SQL stands for **Structured Query Language**.
- designed for managing data in a relational database management system (RDBMS)
- > It is pronounced as S-Q-L or sometime See-Qwell
- > SQL is a database language, it is used for database creation, deletion, fetching rows, and modifying rows, etc.
- > SQL is based on relational algebra and tuple relational calculus.

Why SQL is required



create new databases, tables and views

insert records in a database

update records in a database

delete records from a database

retrieve data from a database

SQL Syntax



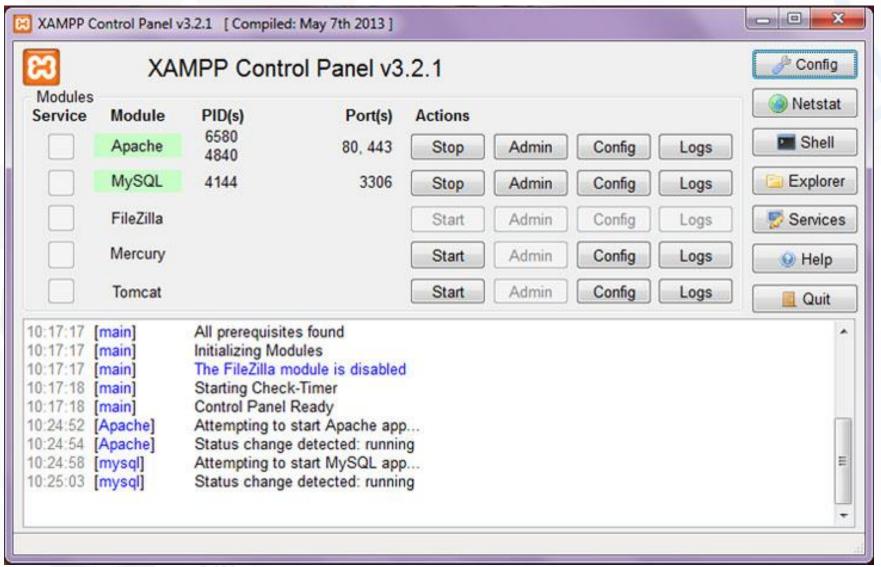
- > **SQL** is not case sensitive.
- > SQL keywords are written in uppercase.
- > We can place a single SQL statement on one or multiple text lines.
- > SQL statements are started with any of the SQL commands/keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP etc. and the statement ends with a semicolon(;).

FortuneCloud Important SQL Commands We Understand Technology CREATE DATABASE CREATE TABLE **INSERT INTO SELECT UPDATE** DELETE



Installation







Datatypes



String Data types

Numeric Data Types Date and time Data types

CHAR – 0 to 255 VARCHAR – 0 to 65535 TEXT -0 to 255 ENUM - 65535 BLOB - 65535 INT – 255 INTEGER FLOAT DOUBLE BOOL DATE
DATETIME
TIMESTAMP
TIME
YEAR



Operators

Comparison Operators



_	a=1	\sim
	()	
	<u> </u>	\sim

!= a!=b

> a>b

< a= a>=b

<= a<=b

!< a!<b

!> a!>b

Arithmetic Operators

+	a+b
-	a-b
*	a*b
/	a/b
%	a%b

Logical Operators



ALL	compare a value to all values
-----	-------------------------------

AND allows the existence of multiple conditions

ANY compare the value in list according to the condition

BETWEEN search for values, that are within a set of values

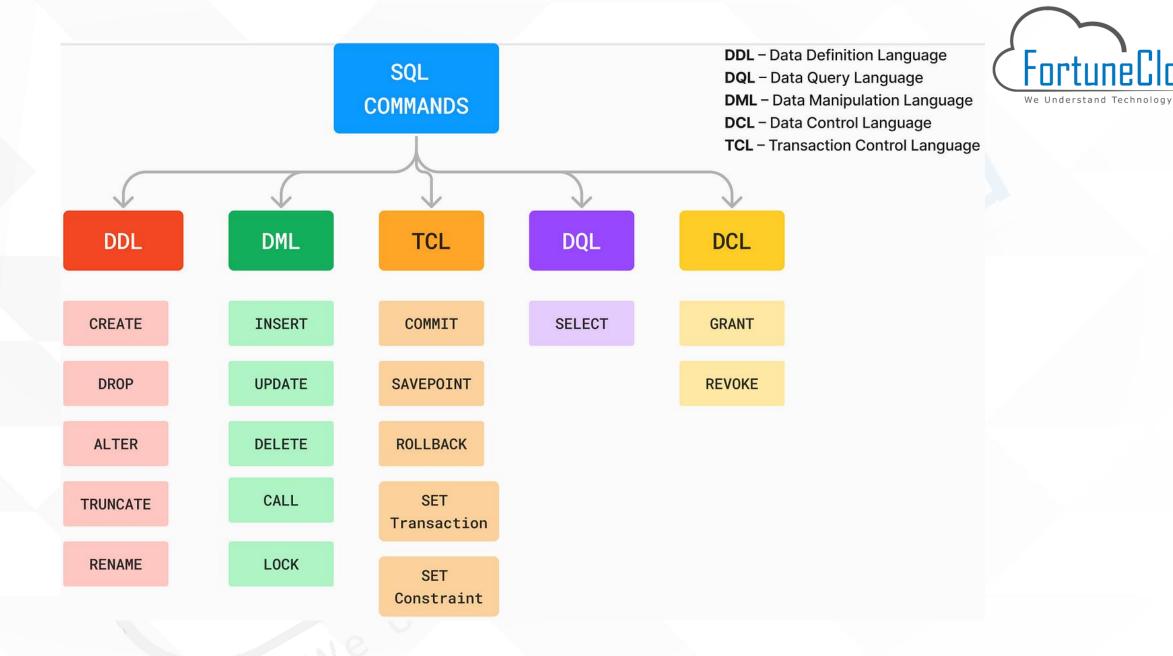
IN compare a value to that specified list value

NOT reverse the meaning of any logical operator

OR combine multiple conditions

EXISTS search for the presence of a row in a specified table

LIKE compare a value to similar values using wildcard operator





Database

CREATE DATABASE



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

Syntax

CREATE DATABASE database_name;

Eg. CREATE DATABASE mydb;

Database name



> The SQL **DROP DATABASE** statement is used to drop an existing database in SQL schema.

Syntax

DROP DATABASE DatabaseName;

Eg. DROP DATABASE mydb;



> SQL **RENAME DATABASE** is used when you need to change the name of your database.

Syntax

RENAME DATABASE OldDatabaseName **TO** NewDatabaseName;

Eg. RENAME DATABASE mydb TO mydb1;

Note: RENAME DATABASE Syntax was added in MySQL 5.1.7 but was found to be dangerous and was removed in MySQL 5.1.23.

SELECT DATABASE



When you have multiple databases in your SQL Schema, then before starting your operation, you would need to select a database

Syntax

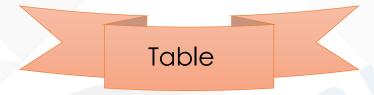
USE Databasename;

Eg. **USE** mydb;



Table





- > Table is a collection of data, organized in terms of rows and columns.
- > In DBMS term, table is known as relation and row as tuple.
- > A table has a specified number of columns, but can have any number of rows.
- > A table is also considered as a convenient representation of relations.

CREATE TABLE



- > SQL CREATE TABLE statement is used to create table in a database.
- If you want to create a table, you should name the table and define its column and each column's data type.

Syntax

```
CREATE TABLE table_name(column_name1 datatype, Column_name2 datatype,....);
```

Eg. CREATE TABLE student (id int (10), name varchar (20));

DROP TABLE



> A SQL **DROP TABLE** statement is used to delete a table definition and all data from a table.

Syntax

DROP TABLE table_name;

Eg. **DROP TABLE** student;

DELETE TABLE



The **DELETE** statement is used to delete rows from a table. If you want to remove a specific row from a table you should use **WHERE** condition.

Syntax

DELETE FROM table_name;

Eg. **DELETE FROM** student;

Syntax

DELETE FROM table_name [WHERE condition];

Eg. **DELETE FROM** student **WHERE** id=1;

RENAME TABLE



> SQL **RENAME TABLE** is used to change the name of a table.

Syntax

alter table old_table_name rename to new_table_name;

Eg. alter table student rename to student_new;

TRUNCATE TABLE



> A truncate SQL statement is used to remove all rows (complete data) from a table.

Syntax

TRUNCATE TABLE table_name;

Eg. TRUNCATE TABLE student;

COPY TABLE



Copy a SQL table into another table in the same SQL server database

Syntax

insert into destination_table select * from source_table;

Eg. Insert into student1 select * from student;

Note: SELECT INTO is totally different from INSERT INTO statement.

ALTER TABLE



- > ALTER TABLE statement is used to add, modify or delete columns in an existing table.
- > It is also used to rename a table.



ALTER - Add Column



> Add columns in SQL table

Syntax

ALTER TABLE table_name ADD column_name column-definition;

Eg. alter table student add name varchar(20);

ALTER - Modify Column



Modify an existing column in SQL table

Syntax

ALTER TABLE table_name **MODIFY** column_name column_type;

Eg. alter table login MODIFY name int(10);

ALTER – Drop Column



> Drop an existing column in SQL table

Syntax

ALTER TABLE table_name DROP COLUMN column_name;

Eg. alter table login1 drop column name;

ALTER - Rename Column



> Rename an existing column in SQL table



ALTER TABLE table_name RENAME **COLUMN** old_name **to** new_name;

OR

ALTER TABLE login1 CHANGE column_name datatype;

Eg. alter table login1 RENAME COLUMN pass to pass1;

OR

alter table login1 change roll_no id int(20);



Select





- It is used to query the database and retrieve selected data that follow the conditions we want.
- > The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table.

Clauses in SELECT Statement

WHERE GROUP BY HAVING ORDER BY





SELECT * **FROM** table_name;

Eg. **SELECT** * **FROM** student;



SELECT column_name **FROM** table_name;

Eg. **SELECT** name **FROM** student;

OR

Fortune Cloud We Understand Technology

SELECT DISTINCT

> **SELECT UNIQUE** is an old syntax which was used in oracle description but later ANSI standard defines **DISTINCT** as the official keyword.

Syntax

SELECT UNIQUE column_name **FROM** table_name;

Eg. **SELECT UNIQUE** name **FROM** student;

OR

Syntax

SELECT DISTINCT column_name **FROM** table_name;

Eg. **SELECT DISTINCT** name **FROM** student;

SELECT COUNT



- > The **SQL COUNT()** function is used to return the number of rows in a query.
- > it is very useful to count the number of rows in a table having enormous data.

SELECT COUNT(column_name)

SELECT COUNT(*)

SELECT COUNT(DISTINCT column_name)

SELECT COUNT(column_name)



- > It will return the total number of names of student_table.
- > null fields will not be counted.

Syntax

SELECT COUNT (column_name) **FROM** table_name;

Eg. SELECT COUNT (name) FROM student;

SELECT COUNT(*)



> It is used to return the number of records in table.

Syntax

SELECT COUNT (*) **FROM** table_name;

Eg. **SELECT COUNT** (*) **FROM** student;

SELECT COUNT(DISTINCT column_name)



➤ It will return the total distinct names of student_table.

Syntax

SELECT COUNT (DISTINCT column_name) FROM table_name;

Eg. SELECT COUNT (DISTINCT name) FROM student;

SELECT TOP



- > It is used to select top data from a table.
- > The top clause specifies that how many rows are returned.

Syntax

SELECT * **FROM** table_name **LIMIT** count;

Eg. select * from student limit 2;

SELECT AS



- > SQL AS is used to assign temporarily a new name to a table column.
- Allows the developer to label results more accurately without permanently renaming table columns.

Syntax

SELECT column_name **AS** column_name **FROM** table_name;

Eg. select uername as 'name' from student;

SELECT IN



- > SQL IN is an operator used in a SQL query to help reduce the need to use multiple SQL "OR" conditions.
- > It minimizes the use of SQL OR operator.

Syntax

SELECT * **FROM** table_name **WHERE** column_name **IN** (val1,val2,....);

Eg. select * from student where name in('abc','lmn');

SELECT SUM



- ➤ It is also known as SQL SUM() function.
- > It is used in a SQL query to return summed value of an expression.

Syntax

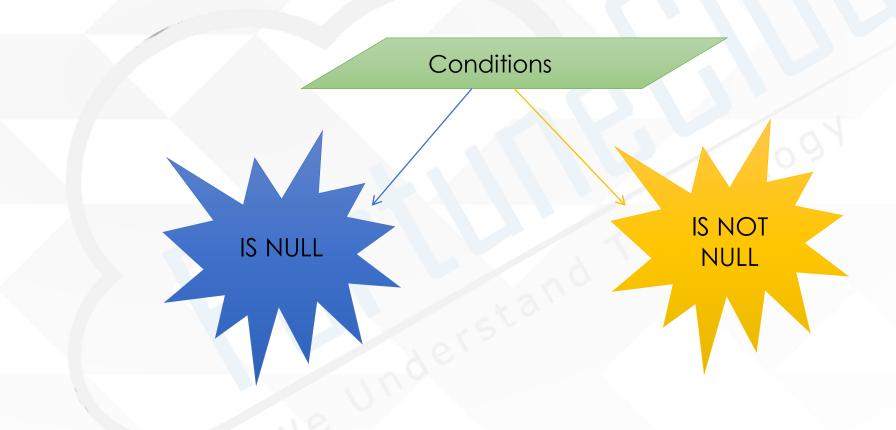
SELECT SUM (column_name) **AS** "expression_name" **FROM** table_name;

Eg. select sum(salary) as "total salary" from employee;

SELECT NULL



> Null values are used to represent missing unknown data.



IS NULL



> Select records with null values.

Syntax

SELECT column_name **FROM** table_name **WHERE** column_name **IS NULL**;

Eg. select name from student where marks is null;

IS NOT NULL



> Select records with no null values.

Syntax

SELECT column_name **FROM** table_name **WHERE** column_name **IS NOT NULL**;

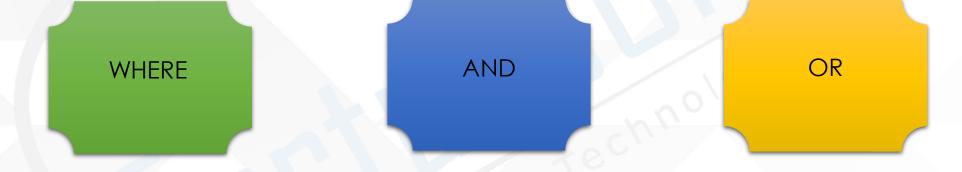
Eg. select name from student where marks is not null;



Clause



Clauses



WHERE CLAUSE



- The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables.
- You should use the WHERE clause to filter the records and fetching only the necessary records.

Syntax

SELECT * **FROM** table_name **WHERE** condition;

Eg. select * from student where id=1;

AND CLAUSE



> The SQL **AND** condition is used in SQL query to create two or more conditions to be met.

Syntax

SELECT * **FROM** table_name **WHERE** condition1 **AND** condition2;

Eg. select * from student where id=1 AND name='abc';



> The **OR** operator is used to combine multiple conditions in an SQL statement's WHERE clause.

Syntax

SELECT * **FROM** table_name **WHERE** condition1 **OR** condition2;

Eg. select * from student where id=1 OR id=5;



Order By Clause



- The SQL ORDER BY clause is used for sorting data in ascending and descending order based on one or more columns.
- > Databases sort query results in ascending order by default.

ASC DESC RANDOM LIMIT



- > It is used to sort data in ascending order.
- > SQL ORDER BY query takes ascending order by default.

Syntax

SELECT * **FROM** table_name **ORDER** BY column_name **ASC**;

Eg. select * from student ORDER BY id ASC;



> It is used to sort data in descending order.

Syntax

SELECT * **FROM** table_name **ORDER** BY column_name **DESC**;

Eg. select * from student ORDER BY id DESC;

LIMIT

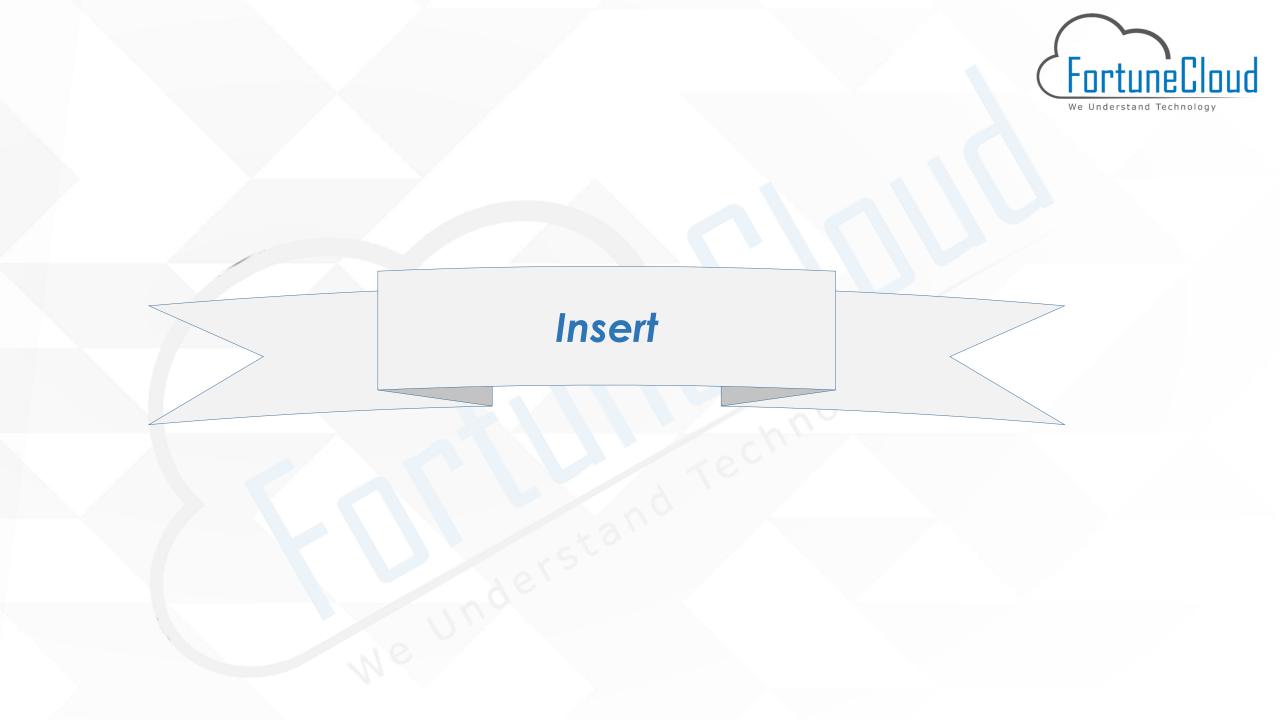


- If you want to fetch random rows from any of the databases you have to use RANDOM()
- > SQL LIMIT We can retrieve limited rows from the database.

Syntax

SELECT * FROM table_name ORDER BY RAND() LIMIT 1;

Eg. select * from student order by RAND() LIMIT 1;





> It is used to insert a single or a multiple records in a table.

By specifying column names

Insert

Without specifying column names

Without specifying column names



Syntax

INSERT INTO table_name **VALUES** (value1, value2, value3....);

Eg. INSERT INTO student VALUES (101, 'abcd');

By specifying column names

Syntax

INSERT INTO table_name (column_name1, column_name2)
VALUES (val1,val2,...);

Eg. INSERT INTO student (id, name) VALUES (101, 'abcd');



Update



- > **SQL UPDATE** statement is used to change the data of the records held by tables.
- > Which rows is to be update, it is decided by a condition.

Syntax

UPDATE table_name SET [column_name1= value1,... column_name
N = valueN] [WHERE condition]

Eg. UPDATE student SET name='Abc', address='Pune' WHERE id=1



Delete

Remove All Rows



Syntax

DELETE FROM table_name;

Eg. **DELETE FROM** student;

Remove All Rows With Condition

Syntax

DELETE FROM table_name WHERE condition;

Eg. **DELETE FROM** student **WHERE** id=101;



Joins



- JOIN means "to combine two or more tables".
- > The SQL JOIN clause takes records from two or more tables in a database and combines it together.

Inner Join

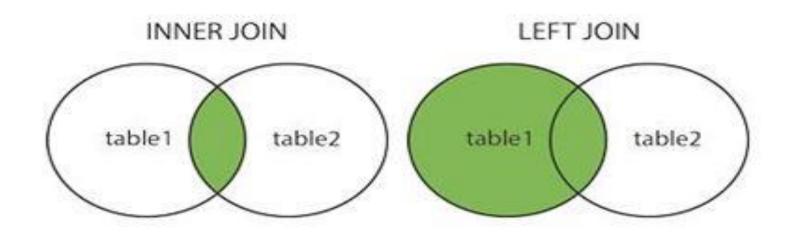
Left Outer Join

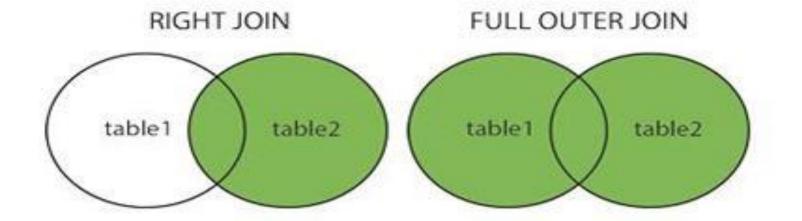
Right Outer Join

Full Outer Join

Cross Join





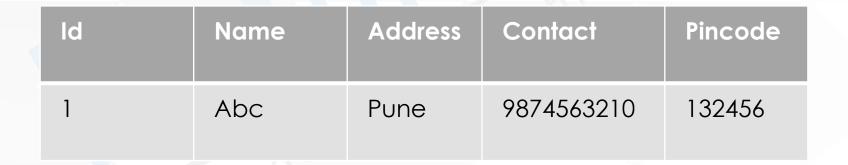




Id	Name	Address
1	Abc	Pune
2	Xyz	Mumbai



ld	Contact	Pincode
1	9874563210	132456
3	9874563210	789456



Inner Join



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

Syntax

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

Eg. **SELECT** * **FROM** student1 **INNER JOIN** student2 **ON** student1.id=student2.id;

Left Outer Join



- > The SQL left join returns all the values from the left table and it also includes matching values from right table
- > if there are no matching join value it returns NULL.

Syntax

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

Eg. **SELECT** * **FROM** student1 **LEFT JOIN** student2 **ON** student1.id=student2.id;

Right Outer Join



- > The SQL right join returns all the values from the rows of right table.
- > It also includes the matched values from left table
- if there is no matching in both tables, it returns NULL.

Syntax

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

Eg. **SELECT** * **FROM** student1 **RIGHT JOIN** student2 **ON** student1.id=student2.id;

Full Outer Join



- The SQL full join is the result of combination of both left and right outer join
- > It puts NULL on the place of matches not found.
- it is known as SQL FULL JOIN.

Syntax

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

Eg.

SELECT * FROM student1 LEFT JOIN student2 ON student1.id=student2.id UNION SELECT * FROM student1 RIGHT JOIN student2 ON student1.id=student2.id;

Cross Join



Each row of first table is combined with each row from the second table, known as Cartesian join or cross join.

1st Syntax

SELECT * **FROM** table1 **CROSS JOIN** table2;

Eg. SELECT * FROM student1 CROSS JOIN student2;

2nd Syntax

SELECT * **FROM** table1, table2;

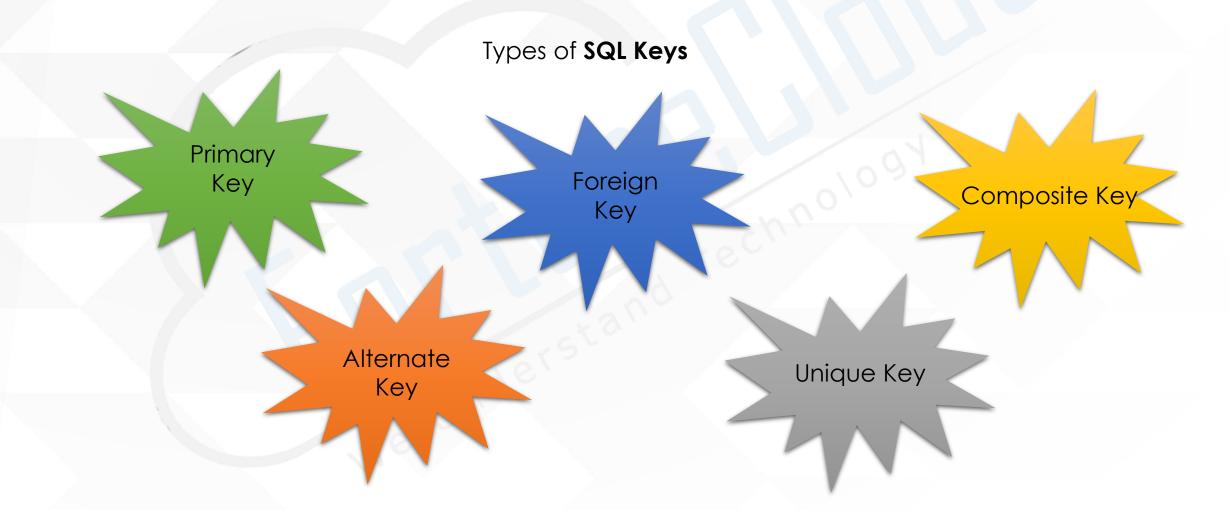
Eg. SELECT * FROM student1, student2;



Keys



- > A key is a single or combination of multiple fields in a table.
- > Keys are also used to create a relationship among different database tables.



Primary Key



- pk- that uniquely identifies each row in the table.
- Primary key always has unique data.
- > A primary key cannot have null value.

Syntax

create table table_name(column_name datatype(size),
PRIMARY KEY(column_name));

Eg. create table student(id int, name varchar(20), PRIMARY KEY(id));

Foreign Key



- > Establish a link between two tables.
- > Foreign key in one table used to point primary key in another table.

Syntax

create table second_table_name(column_name datatype(size),
PRIMARY KEY(column_name),FOREIGN KEY(column_name)
REFERENCES first_table_name(first_table_column_name));

Eg. create table student2(s_id int, primary key(s_id), foreign key(s_id) references student1 (id));

Composite Key



A primary key that is made by the combination of more than one attribute is known as a composite key.

Syntax

create table table_name(column_name1 datatype(size),
column_name2 datatype(size), PRIMARY KEY(column_name1,
column_name2));

Eg. create table student(id int, name varchar(20), PRIMARY KEY(id,name));

Alternate Key



Eg. Id, Roll_no, Email are qualified to become a primary key. But since Id is the primary key, Roll No, Email becomes the alternative key.

ld	Roll_no	Email
101	1	abc@gmail.com

Unique Key



- > set of one or more than one fields/columns of a table that uniquely identify a record in a database table.
- > it is little like primary key but it can accept only one null value.
- > it cannot have duplicate values.
- There may be many unique key constraints for one table, but only one PRIMARY KEY constraint for one table.

Syntax

create table table_name(column_name1 datatype(size),
column_name2 datatype(size), UNIQUE(column_name1));

Eg. create table student(id int NOT NULL, name varchar(20), UNIQUE(id));





- primary key cannot be null
- Primary key is always unique
- Primary key uniquely identify a record in the table.
- There is only one primary key in the table

- Foreign key can be null
- Foreign key can be duplicate
- Foreign key is a field in a table that is primary key in another table.
- We can have more than one foreign key in the table.

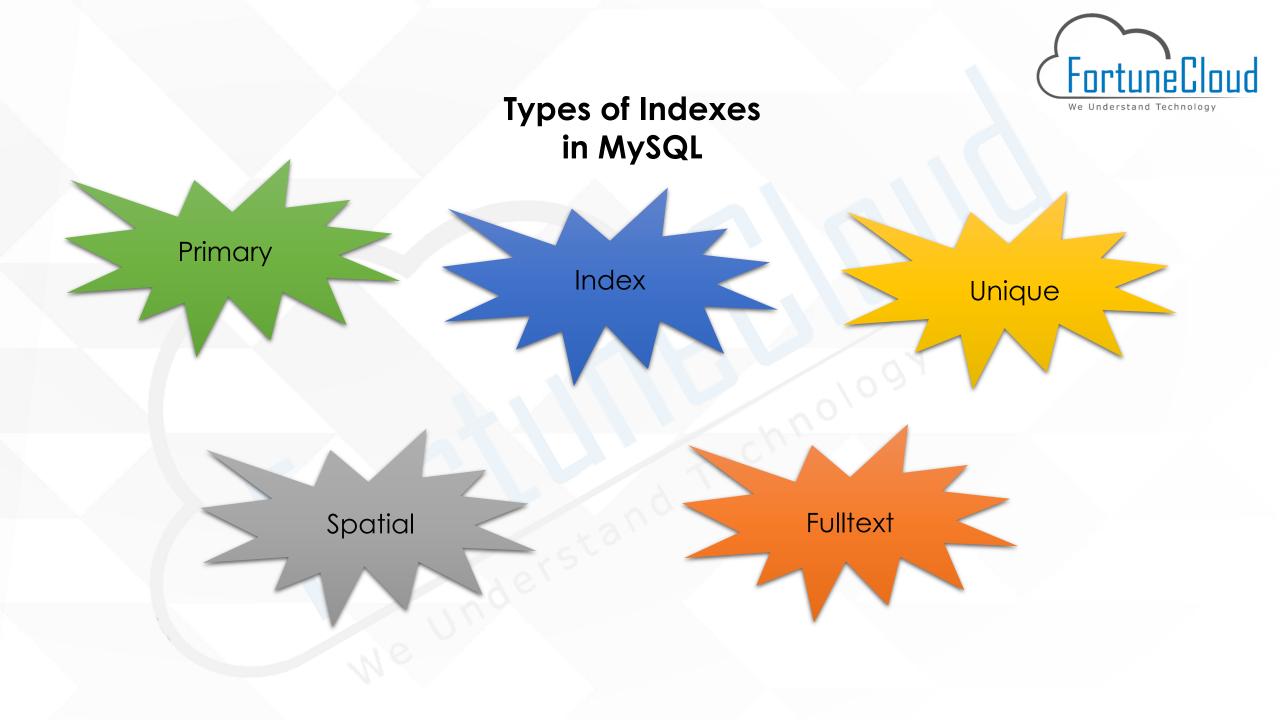


Indexes

Introduction



- > A SQL index is used to retrieve data from a database very fast.
- Indexing a table or view is, without a doubt, one of the best ways to improve the performance of queries and applications.
- > SQL indexes are primarily a performance tool, so they really apply if a database gets large.
- > It can reduce disk I/O(input/output) by using a rapid path access method to locate data quickly.



Primary Index



- > It has an associated index, for fast query performance.
- Query performance benefits from the NOT NULL optimization, because it cannot include any NULL values.
- > The table data is physically organized to do ultra-fast lookups and sorts based on the primary key column.
- You might create a separate column with auto-increment values to use as the primary key.



create table student(id int(255) AUTO_INCREMENT, name varchar(50), PRIMARY KEY(id));

Index



- In this type of Index named as simple, regular or normal, the specified column values do not require to be unique and as well as can be NULL also.
- > They are supplemented basically to aid the database searching process for records quicker.



Syntax:

CREATE INDEX Index_Name ON Table_Name (Column_Name);

E.g.

CREATE INDEX sort_name **ON** student(name ASC);

Unique Index



- > Unique is a type of MySQL Index which specifies that all values of the tables columns, when implemented, have to be distinct.
- > This Unique index is useful to avoid the duplicity of values in the columns.
- > This index is the best way to maintain the data integrity of the SQL tables.
- > It does not allow multiple values to enter into the table



Syntax:

CREATE UNIQUE INDEX Index_Name ON Table_Name (Column_Name);

E.g.

CREATE UNIQUE INDEX unique_contact ON student(contact);

Spatial Index



- > The Spatial indexes are a new index type in MySQL and not broadly used.
- MySQL allows to create Spatial indexes on geometry-valued columns with NOT NULL constraint. The spatial index creates an R-tree index.
- > Spatial data, also known as geospatial data or geographic information, is data that identifies the geographic location of features and boundaries on Earth.
- We usually use spatial data to store coordinates, topology, or other data that can be mapped.



CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL INDEX(g));

INSERT INTO geom VALUES (ST_GeomFromText('POINT(1 1)')), (ST_GeomFromText('LINESTRING(2 1, 6 6)')), (ST_GeomFromText('POLYGON((0 5, 2 5, 2 7, 0 7, 0 5))'));

ST_GeomFromText:

Converts a Well-Known Text (WKT) string into its corresponding GEOMETRY object.

Fulltext



- This type of index is implemented for full-text searches as the name itself implies it.
- In MySQL, the full-text index is always named FULLTEXT. We can define the full-text index whose column data type is CHAR, VARCHAR, and TEXT.



CREATE TABLE books(title VARCHAR(220), content TEXT, FULLTEXT(title, content));

SELECT * **FROM** books **WHERE MATCH** (title, content) **AGAINST** ('Introduction');



PL/SQL



- > PI/SQL stands for "Procedural Language extension of SQL" that is used in Oracle.
- > pl/sql is a block structured language that can have multiple blocks in it.
- A line of PL/SQL text contains groups of characters known as lexical units. It can be classified as follows:

DELIMITER //
BEGIN
END //
DELIMITER ;



Data types



Each value in PL/SQL such as a constant, variable and parameter has a data type that determines the storage format, valid values, and allowed operations.

PL/SQL has two kinds of data types: scalar and composite. The scalar types are types that store single values such as number, Boolean, character, and datetime whereas the composite types are types that store multiple values, for example, record and collection.

PL/SQL divides the scalar data types into four families:

- 1. Number
- 2. Boolean
- 3. Character
- 4. Datetime



Operators



Operators

Arithmetic operators + - * /

Relational operators = != > < >= <=

Comparison LIKE BETWEEN IN IS NULL operators

Logical operators and or not

Control Statements

If then else



```
DELIMITER //
CREATE FUNCTION check_if_else(val INT)
RETURNS varchar(20)
```

BEGIN

```
DECLARE income_level varchar(20);
 IF val = 10 THEN
   SET income_level = 'value is 10';
 ELSEIF val =20 THEN
   SET income_level = 'value is 20';
 ELSE
   SET income_level = 'Incorrect value';
 END IF;
 RETURN income_level;
END; //
DELIMITER;
```

select check_if_else(10);

Control Statements

DELIMITER;

Case



```
DELIMITER //
CREATE FUNCTION check_case(val INT)
RETURNS varchar(20)
BEGIN
 DECLARE income_level varchar(20);
 CASE val
   WHEN 1000 THEN
    SET income_level = 'Low Income';
   WHEN 5000 THEN
    SET income_level = 'Avg Income';
   ELSE
    SET income_level = 'High Income';
 END CASE;
 RETURN income_level;
END; //
```

select check_case(1000);

Leave

Loop



the LEAVE statement is used when you want to exit a block of code identified by a label_name

```
DELIMITER //
CREATE FUNCTION cal_income (val INT)
RETURNS INT
BEGIN
 DECLARE income INT;
 SET income = 0;
 label1: LOOP
  SET income = income + val;
  IF income < 3000 THEN
   ITERATE label1;
  END IF;
  LEAVE label1;
 END LOOP label1;
 RETURN income;
```

END; //
DELIMITER;

select cal_income(100);

Control Statements

While



```
DELIMITER //
CREATE FUNCTION CalcIncome (starting_value
INT )
RETURNS INT
BEGIN
 DECLARE income INT;
 SET income = 0;
 label1: WHILE income <= 3000 DO
  SET income = income + starting_value;
 END WHILE label1;
 RETURN income;
END; //
DELIMITER;
```

select CalcIncome(100);



Procedure



- It is just like procedures in other programming languages.
- ➤ **Header:** The header contains the name of the procedure and the parameters or variables passed to the procedure.
- ▶ Body: The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

Ways to pass parameter

IN OUT

Syntax



```
CREATE PROCEDURE procedure_name (parameters...)

BEGIN
.....statements
END //
DELIMITER;
```

Example



```
CREATE PROCEDURE insert_user
(IN p_id int(10),IN p_name varchar(100))

BEGIN
insert into user_reg(id,name)values(p_id,p_name);
END //

DELIMITER;
```

call insert_user(101, 'abc');



Function



- > A function is same as a procedure except that it returns a value.
- > a function must always return a value, and on the other hand a procedure may or may not return a value.





```
CREATE FUNCTION function_name
(parameters...)
RETURNS datatype
BEGIN
.....statements
return value;
END //
DELIMITER;
```

Example



```
DELIMITER //
CREATE FUNCTION my_square(val int)
RETURNS int
BEGIN
      DECLARE result int;
     set result=0;
     set result=val * val;
     return result;
END //
DELIMITER;
```

select my_square (5);



Trigger



- > Triggers are stored programs, which are automatically executed or fired when some event occurs.
- > Trigger generates some derived column values automatically
- > Event logging and storing information on table access
- > Imposing security authorizations







```
DELIMITER //

CREATE TRIGGER trigger_name BEFORE INSERT ON table_name

BEGIN

.....statements

END //

DELIMITER;
```

Example



```
CREATE TRIGGER before_insert_money BEFORE INSERT ON money
FOR EACH
ROW
BEGIN
UPDATE employee SET salary='Credited';
END //
DELIMITER;
```

insert into money values('100');



Cursors



- A **Cursor** is a temporary memory that is allocated by the database server at the time of performing the **D**ata **M**anipulation **L**anguage operations on a table, such as INSERT, UPDATE and DELETE etc.
- In MySQL, you cannot declare a cursor directly outside of a stored procedure or function. Cursors are generally declared within stored procedures, functions, or blocks of SQL code in MySQL database.

Life Cycle of the Cursor

Declare Open Fetch Empty Close





- Cursors allow us to process data row-by-row, which can be useful when we need to perform complex calculations or transformations on the data.
- > Cursors are especially useful when processing data from multiple tables where the relationships are not straightforward.

Cursor Limitations

- It imposes locks on a subset or the entire table.
- > It updates table records one row at a time, which slows down its performance.



Steps 1

```
CREATE TABLE customers (
id int(255),
name varchar(50),
addresss varchar(100),
PRIMARY KEY (id)
);
```

Steps 2

```
INSERT INTO customers VALUES (1, 'John', 'Pune'), (2, 'Allen', 'Mumbai'), (3, 'Anderson', 'Pune');
```

Steps 3

```
CREATE TABLE customers_backup (
id int(255),
name varchar(100),
PRIMARY KEY (id)
);
```

Steps 4



```
DELIMITER //
CREATE PROCEDURE FetchCustomers()
BEGIN
DECLARE done INT DEFAULT FALSE;
DECLARE customer_id INT;
DECLARE customer_name VARCHAR(255);
DECLARE auto_id INT;
```

-- Declare cursor

DECLARE MY_CURSOR CURSOR FOR SELECT id, name FROM CUSTOMERS;

-- Declare exit handler

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

-- Open cursor

OPEN MY_CURSOR;

```
-- Fetch and insert rows
 read loop: LOOP
   FETCH MY CURSOR INTO customer id, customer name;
   IF done = 1 THEN
    LEAVE read_loop;
   END IF;
   -- Insert the fetched data into the backup table
   INSERT INTO customers_backup VALUES (customer_id,
customer name);
   -- Get the last auto-generated ID used in the insertion
   SET auto id = LAST INSERT ID();
 END LOOP;
  -- Close cursor
  CLOSE MY CURSOR;
END //
DELIMITER;
```



Steps 5

CALL FetchCustomers();

Steps 6

Select * from customers_backup



THANKONDU