**MySQL datatypes**

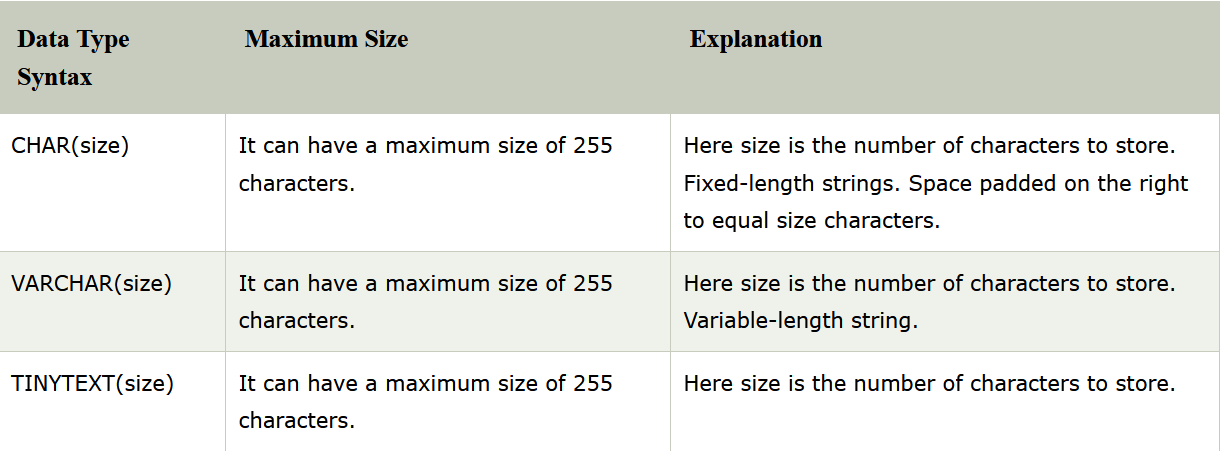
1. BIT
2. BOOLEAN
3. CHAR
4. DATE
5. DATETIME
6. DECIMAL
7. ENUM
8. INT
9. JSON
10. TIME
11. TIMESTAMP
12. VARCHAR

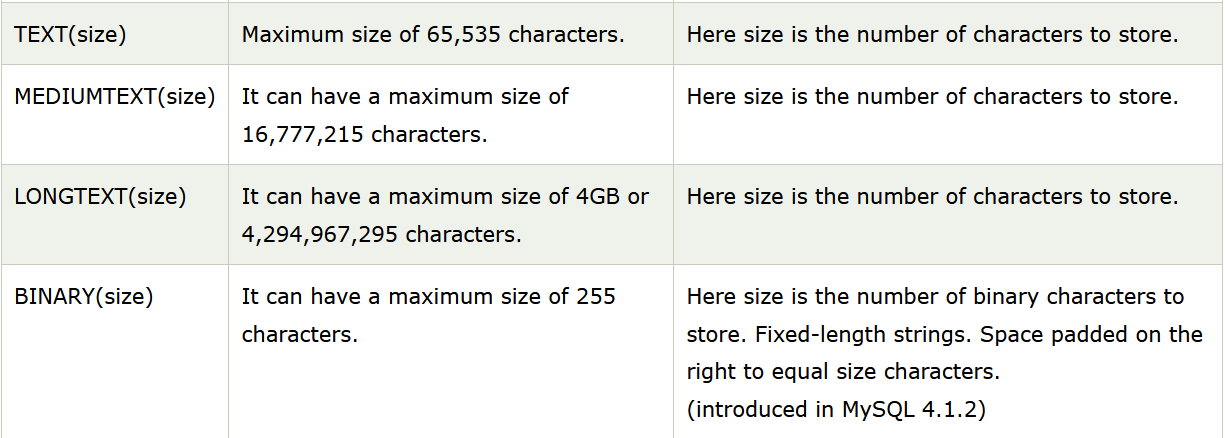
**Boolean data types**

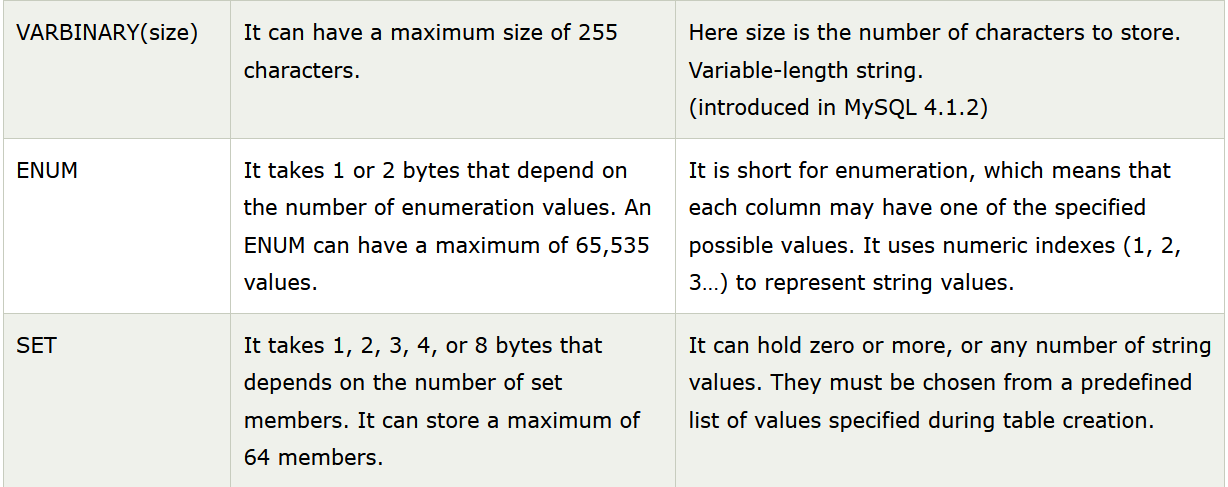
MySQL doesn’t have any specific Boolean datatypes. In MySQL 0 is considered as FALSE/false and non-zero values are considered as TRUE/true.

**String/CHAR/VARCHAR data types**

The string data type is used to hold plain text and binary data, for example, files, images, etc. MySQL can perform searching and comparison of string value based on the pattern matching such as LIKE operator, Regular Expressions, etc







**VARCHAR vs TINYTEXT**

VARCHAR is ISO standard, TINYTEXT is non-standard. TINYTEXT can only have a default value of NULL, where VARCHAR can have any text as a default. TINYTEXT data is stored as a reference that needs to be fetched separately, which may add latency

**Comparing MySQL CHAR values**

* MySQL does not consider trailing spaces when comparing CHAR values using the comparison operator such as =, <>, >, <, etc
* Notice that the LIKE operator does consider the trailing spaces when you do pattern matching with CHAR values

When we use ==, <>, >, <, etc operator for comparing or getting values then it checks for exact character but if we use LIKE operator then it checks for that character, doesn’t care about the trailing spaces.

**MySQL CHAR and UNIQUE index**

If the CHAR column has a UNIQUE index and you insert a value that is different from an existing value in a number of trailing spaces (i.e same value appending with some spaces at end), MySQL will reject the changes because of duplicate-key error.

**MYSQL VARCHAR**

MySQL VARCHAR is the variable-length string whose length can be up to 65,535. MySQL stores a VARCHAR value as a 1-byte or 2-byte length prefix plus actual data.

*If a column requires less than 255 bytes, the length prefix is 1 byte. In case the column requires more than 255 bytes, the length prefix is two length bytes*.

The maximum length (65535), however, is subject to maximum row size (65,535 bytes) and the character set used. It means that the total length of all columns should be less than 65,535 bytes.

**MySQL VARCHAR and spaces**

* MySQL does not pad space when it stores the VARCHAR values. Also, MySQL retains the trailing spaces when it inserts or selects VARCHAR values but trailing spaces will be truncated when string length is more that the specified maximum length.

Let say we have below table:

CREATE TABLE IF NOT EXISTS ajanta.string\_test (

s1 VARCHAR(3) NOT NULL

);

Now we are tying to insert a string of more that specified characters then we will get warnings.

INSERT INTO string\_test(s1)

VALUES('ABC ');

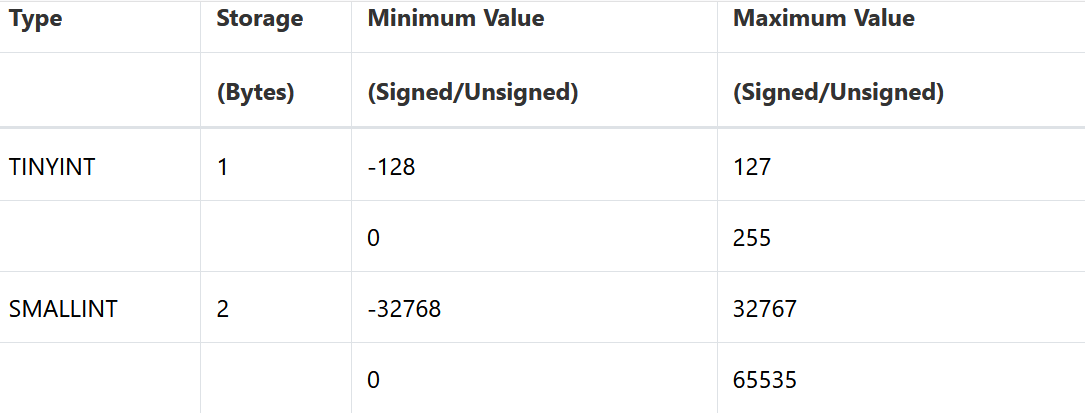
Warning

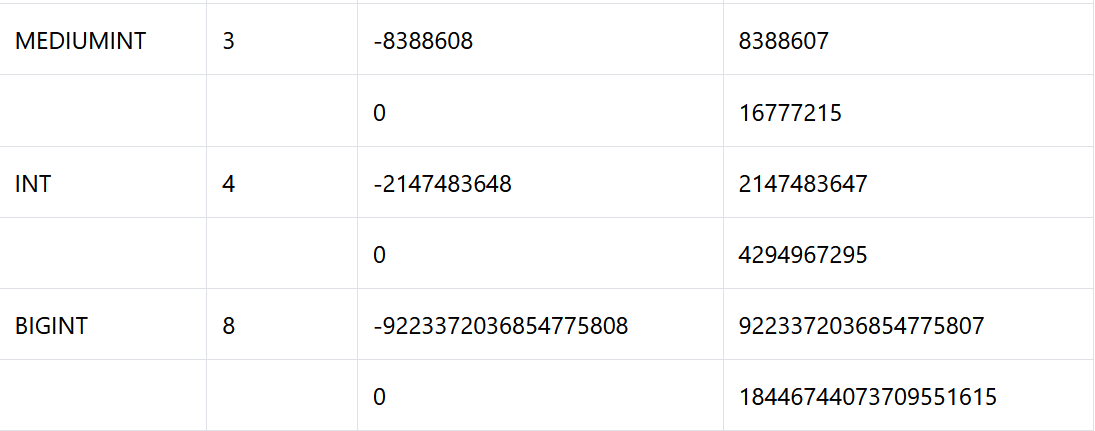
INSERT INTO string\_test(s1)

VALUES('ABC ') 1 row(s) affected, 1 warning(s):

1265 Data truncated for column 's1' at row 1 0.015 sec

**MySQL INT datatypes**





**MySQL DECIMAL data type**

The MySQL DECIMAL data type is used to store exact numeric values in the database. We often use the DECIMAL data type for columns that preserve exact precision

To define a column whose data type is DECIMAL you use the following syntax:

column\_name DECIMAL(P,D);

* P is the precision that represents the number of significant digits. The range of P is 1 to 65.
* D is the scale that that represents the number of digits after the decimal point. The range of D is 0 and 30. MySQL requires that D is less than or equal to (<=) P. default to 0.

**MySQL DATE data type**

MySQL DATE is one of the five temporal data types used for managing date values. MySQL uses yyyy-mm-dd format for storing a date value. This format is fixed and it is not possible to change it.

MySQL uses 3 bytes to store a DATE value. The DATE values range

MySQL Date values with two-digit years

Year values in the range 00-69 are converted to 2000-2069

Year values in the range 70-99 are converted to 1970 – 1999

**Some methods for date in MySQL.**

* To get the current system date, you use CURDATE() and current time stamp CURRENT\_TIMESTAMP function.
* To get only date part of a DATETIME value, you use the DATE() function.
* To format a date value, you use DATE\_FORMAT(date object, format) function (same as python).
* To calculate the number of days difference in two data we use DATEDIFF function.
* DATEDIFF(date2,date1) ----- to get number of days difference in two data.

**JSON datatypes:**

MySQL supports the native JSON data type since version 5.7.8. The native JSON data type allows you to store JSON documents more efficiently than the JSON text format in the previous versions.

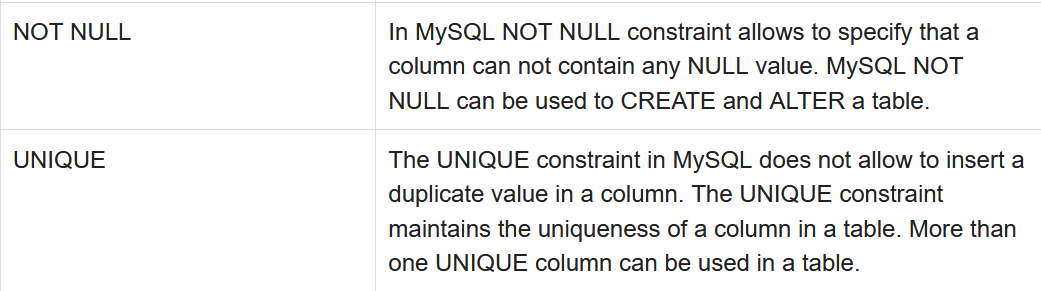
The storage of a JSON document is approximately the same as the storage of LONGBLOB or LONGTEXT data.

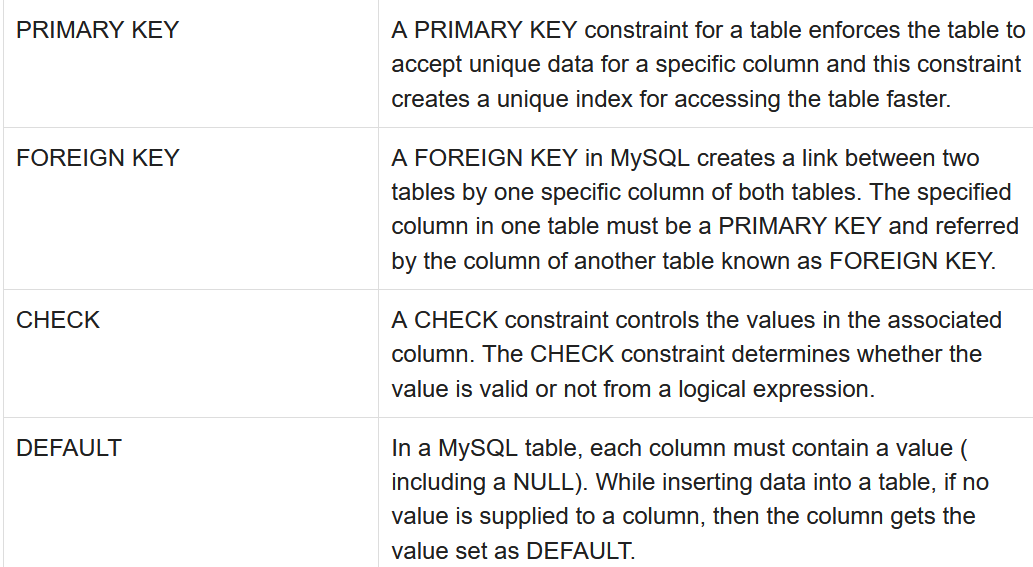
**MySQL constraints**

MySQL CONSTRAINT is used to define rules to allow or restrict what values can be stored in columns.

MySQL CONSTRAINTS are used to limit the type of data that can be inserted into a table.

MySQL CONSTRAINTS can be classified into two types - column level and table level.





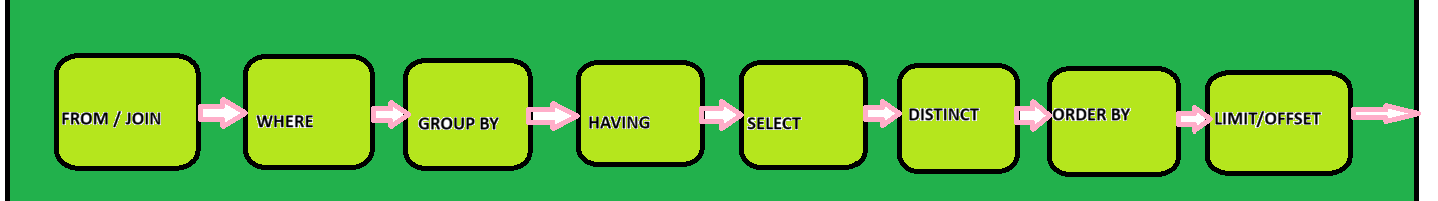
**Primary Key vs Unique key**

Unique key allows one NULL value but Primary key doesn’t allow any null value, both ensures to maintain unique value (no duplicate value) in any column

CREATE TABLE [table name]

([column name] [data type]([size]) [column constraint]….

[table constraint] ([[column name]……])……);



**ORDERBY**

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

select\_list

FROM

table\_name

ORDER BY

column1 [ASC|DESC],

column2 [ASC|DESC],

...;

**MySQL clauses**

There are generally five kinds of SQL Clauses in MySQL Server. They are listed as follows:

* WHERE Clause
* ORDER BY clause
* HAVING Clause
* LIMIT Clause
* GROUP BY Clause

**Group By**

The GROUP BY clause is used to group rows that have the same values in the result set.

**Order by**

The ORDER BY clause is used in SQL for sorting records

**Having clause**

This clause is introduced to apply functions in the query with the WHERE clause. In SQL, the HAVING clause was added because the WHERE clause could not be applied with aggregate functions.

**Limit clause**

The LIMIT clause is used in the SELECT statement to constrain the number of rows to return. The LIMIT clause accepts one or two arguments. The values of both arguments must be zero or positive integers.

SELECT

select\_list

FROM

table\_name

LIMIT row\_count;

row\_count----> maximum number of rows to return.

**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

**Question ----- LeetCode ---- Good**

Write an SQL query to report the patient\_id, patient\_name and conditions of the patients who have Type I Diabetes. Type I Diabetes always starts with DIAB1 (not doesn’t mean first 5 character will be DIAB1 b/c this is code for diabetic and it could be second code name in contions column) prefix from Patient table.

Note:

condition contains 0 or more conditions code separated by spaces.

For diabetic, first 5 characters will be always - 'DIAB1'

Solution1:

select patient\_id,patient\_name,conditions from Patients where conditions like '%DIAB1%';

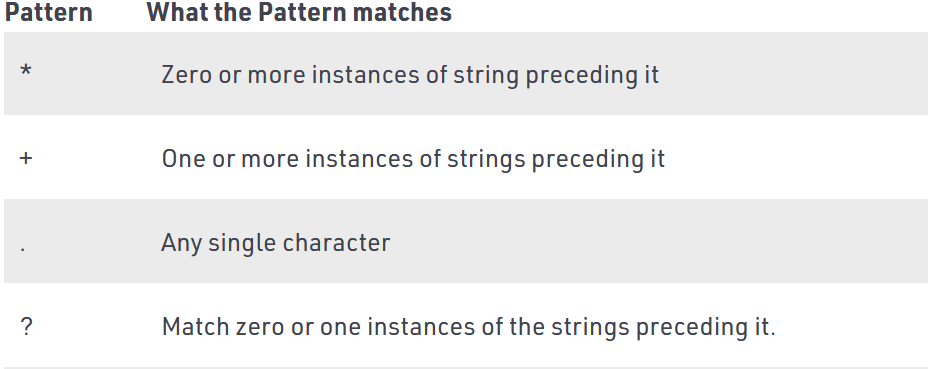
----- This is wrong solution/Query b/c if it will fetch all data which contains ‘DIAB1’ anywhere in condition column.

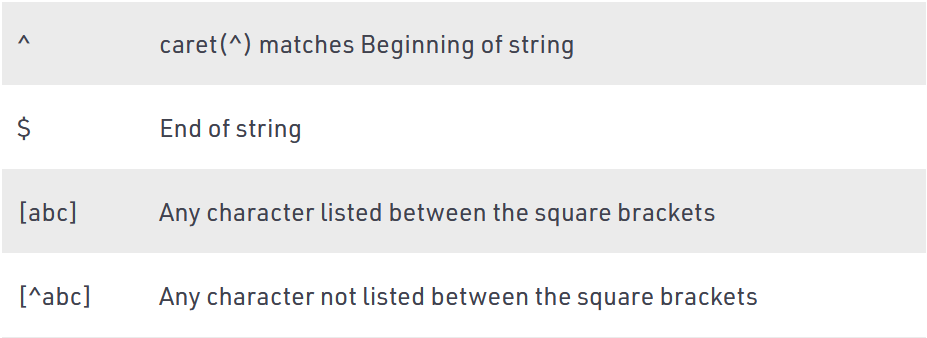
Solution 2:

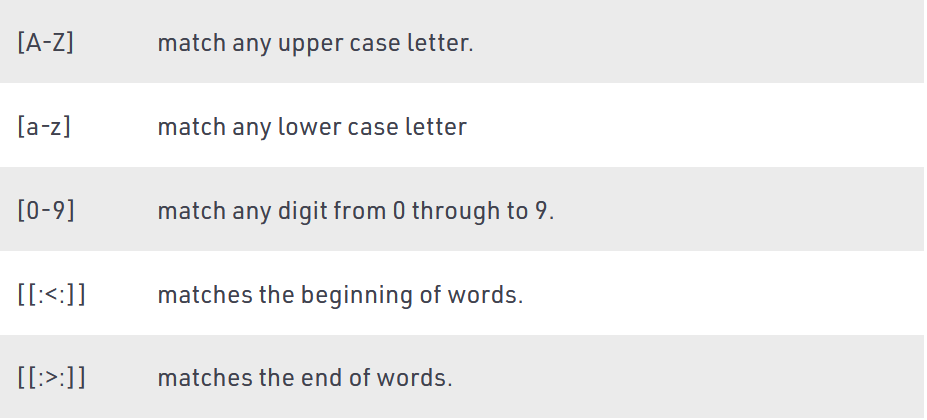
select patient\_id,patient\_name,conditions from Patients where conditions like 'DIAB1%' or conditions like '% DIAB1%';

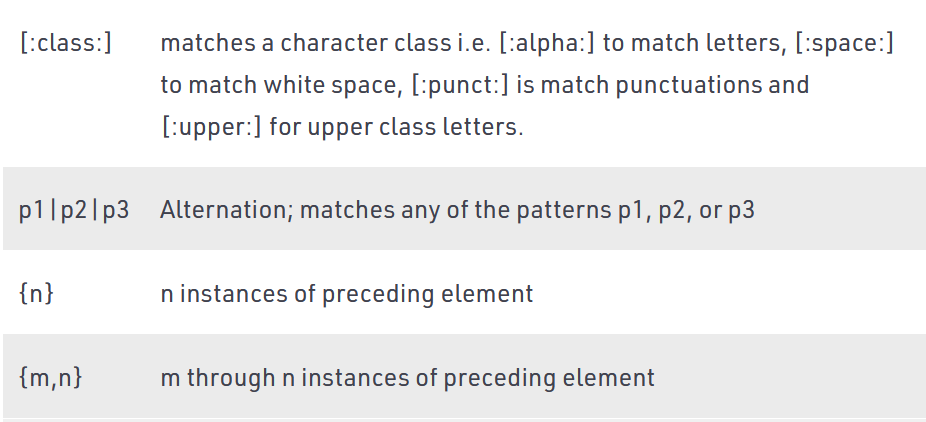
**MySQL Regular expressions (Regexp)**

MySQL supports another type of pattern matching operation based on the regular expressions and the REGEXP operator.









Examples:

**Match beginning of string(^):**

Gives all the names starting with ‘sa’.Example- sam,samarth.

SELECT name FROM student\_tbl WHERE name REGEXP '^sa';

**Match the end of a string($):**

Gives all the names ending with ‘on’.Example – norton,merton

*SELECT name FROM student\_tbl WHERE name REGEXP 'on$';*

**Match zero or one instance of the strings preceding it(?):**

Gives all the titles containing ‘com’.Example – comedy , romantic comedy.

*SELECT title FROM movies\_tbl WHERE title REGEXP 'com?';*

**matches any of the patterns p1, p2, or p3(p1|p2|p3):**

Gives all the names containing ‘be’ or ‘ae’.Example – Abel, Baer.

*SELECT name FROM student\_tbl WHERE name REGEXP 'be|ae' ;*

**Matches any character listed between the square brackets([abc]):**

Gives all the names containing ‘j’ or ‘z’.Example – Lorentz, Rajs.

SELECT name FROM student\_tbl WHERE name REGEXP '[jz]' ;

**Matches any lower case letter between ‘a’ to ‘z’- ([a-z]) ([a-z] and (.)):**

Retrieve all names that contain a letter in the range of ‘b’ and ‘g’, followed by any character, followed by the letter ‘a’.Example – Tobias, sewall.

**Matches any single character(.)**

SELECT name FROM student\_tbl WHERE name REGEXP '[b-g].[a]' ;

**Matches any character not listed between the square brackets.([^abc]):**

Gives all the names not containing ‘j’ or ‘z’. Example – nerton, sewall.

SELECT name FROM student\_tbl WHERE name REGEXP '[^jz]' ;

**Matches the end of words[[:>:]]:**

Gives all the titles ending with character “ack”. Example – Black.

SELECT title FROM movies\_tbl WHERE REGEXP 'ack[[:>:]]';

**Matches the beginning of words[[:<:]]:**

Gives all the titles starting with character “for”. Example – Forgetting Sarah Marshal.

SELECT title FROM movies\_tbl WHERE title REGEXP '[[:<:]]for';

**Matches a character class[:class:]:**

i.e [:lower:]- lowercase character ,[:digit:] – digit characters etc.

Gives all the titles containing alphabetic character only. Example – stranger things, Avengers.

SELECT title FROM movies\_tbl WHERE REGEXP '[:alpha:]' ;

**GROUP BY**

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.

We often use the GROUP BY clause with aggregate functions such as SUM, AVG, MAX, MIN, and COUNT

*The GROUP BY clause must appear after the FROM and WHERE clauses*. Following the GROUP BY keywords is a list of comma-separated columns or expressions that you want to use as criteria to group rows.

SELECT

c1, c2,..., cn, aggregate\_function(ci)

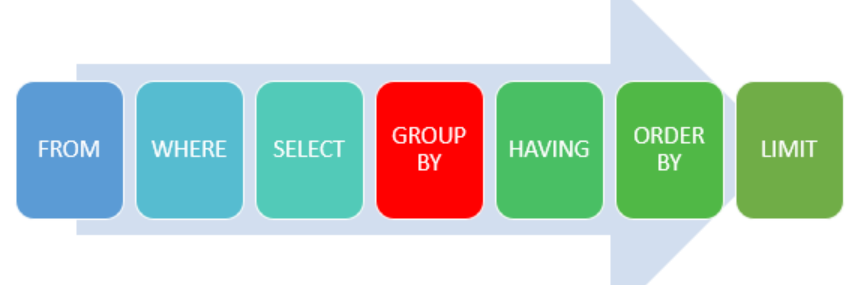
FROM

table

WHERE

where\_conditions

GROUP BY c1 , c2,...,cn;



MySQL evaluates the GROUP BY clause after the FROM, WHERE and SELECT clauses and before the HAVING , ORDER BY and LIMIT clauses.

**MySQL DISTINCT clause**

When querying data from a table, you may get duplicate rows. In order to remove these duplicate rows, you use the DISTINCT clause in the SELECT statement.

SELECT DISTINCT

select\_list(s)

FROM

table\_name

**MySQL DISTINCT with multiple columns**

You can use the DISTINCT clause with more than one column. In this case, *MySQL uses the* ***combination of values*** *in these columns to determine the uniqueness of the row in the result set*

SELECT DISTINCT

state, city

FROM

customers

WHERE

state IS NOT NULL

ORDER BY

state,

city;

Above query unique combination of city and state from the customers table.

**DISTINCT clause vs. GROUP BY clause**

If you use the GROUP BY clause in the SELECT statement without using aggregate functions, the GROUP BY clause behaves like the DISTINCT clause.

MySQL Data Manipulation

AND

OR

IN

BETWEEN --- down

IS NULL

TABLE & COLUMN ALIASES ---------- use as keyword for aliasing table to column

jNOINS

INNER JOIN

LEFT JOIN

RIGHT JOING

SELF JOIN

CROSS JOIN

GROUP BY

HAVING

RULLUP

Subquery -------- a query nested within another query

Derived tables

EXIXTS

UNION

MINUES

INTERSECT

INSERT

Insert Multiple ROWS

INSER INTO SELECT

Insert On Duplicate Key update

INSERT IGNORE

UPDATE

UPDATE JOIN

DELETE

DELETE JOIN

ON DELETE CASCADE

REPLACE

**Between Operator**

The BETWEEN operator is a logical operator that allows you to specify whether a value in a range or not. The BETWEEN operator is often used in the WHERE clause of the SELECT, UPDATE, and DELETE statements.

expr [NOT] BETWEEN begin\_expr AND end\_expr; ---- syntax

begin\_expr -- this is starting point of range

end\_expr ---- this is end point of range

e.g—

SELECT

productCode,

productName,

buyPrice

FROM

products

WHERE

buyPrice < 20 OR buyPrice > 100;

**MySQL HAVING clause**

The HAVING clause is used in the SELECT statement to apply filter conditions for a group of rows or aggregates.

SELECT

select\_list

FROM

table\_name

WHERE

search\_condition

GROUP BY

group\_by\_expression

HAVING

group\_condition;

*MySQL evaluates the HAVING clause after the FROM, WHERE, SELECT and GROUP BY clauses and before ORDER BY, and LIMIT clauses.*

**MySQL update statement**

The UPDATE statement updates data in a table. It allows you to change the values in one or more columns of a single row or multiple rows.

UPDATE table\_name

SET

column\_name1 = expr1,

column\_name2 = expr2,

...

[WHERE

condition];

**Example 1: Updating one column data**

Update the email of a employee whose employee id is 1056.

update Employee

SET email="new email.com"

where employee\_id=1056

Example 2 : Multiple column update

Update the last\_name,first\_name of employee whose id if 1056.

update employee

SET first\_name="Mohan",last\_name="hare"

where employee\_id=1056

**Question --- Good, Leetcode**

Write an SQL query to swap all 'f' and 'm' values of Salary (i.e., change all 'f' values to 'm' and vice versa) with a single update statement and no intermediate temporary tables.

Schema detail of Salary table-

+-------------+----------+

| Column Name | Type |

+-------------+----------+

| id | int |

| name | varchar |

| sex | ENUM |

| salary | int |

+-------------+----------+

Solution:

update Salary

set sex=IF(sex='m','f','m');

**MySQL REPLACE statement**

The MySQL REPLACE statement is an extension to the SQL Standard. The MySQL REPLACE statement works as follows:

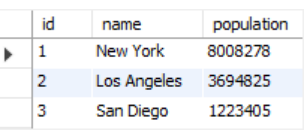
1.) Insert a new row into the table

2.) If the insertion fails due to a duplicate-key error occurrence-

Delete the conflicting row that causes the duplicate key error from the table.

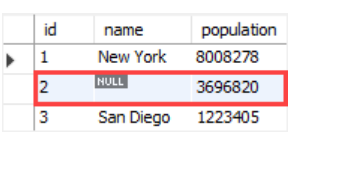
Insert the new row into the table again.

Let say we have below table



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

After replace



**MySQL DELETE statement**

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

**Simple delete statement**

DELETE FROM table\_name

WHERE condition;

**Delete with joins statement**

DELETE col1,col2 FROM table1

INNER JOIN table2

ON table1.coln = table2.coln

WHERE condition

* WHERE clause is optional. If you omit the WHERE clause, the DELETE statement will delete all rows in the table
* Besides deleting data from a table, the DELETE statement returns the number of deleted rows.
* *If don’t want to know how many of rows deleted without any condition then use TRUNCATE TABLE*



**Questions**

Delete customer from customer table whose country is France, sorts them by credit limit in from low to high and delete first 5 customers

**Solution**:

delete from customers

where country="france"

order by credit

limit 5

**Question --- LeetCode**

Write an SQL query to delete all the duplicate emails from Person table, keeping only one unique email with the smallest id. Note that you are supposed to write a DELETE statement and not a SELECT one.

Person table is-

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| id | int |

| email | varchar |

+-------------+---------+

**Solution**

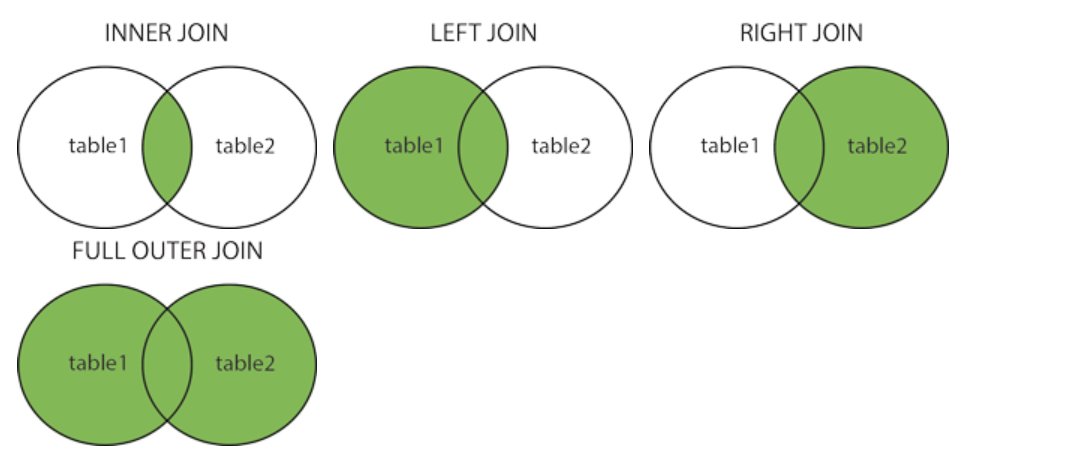
delete p1.\* from Person p1

JOIN Person p2

on p1.email=p2.email and p1.id>p2.id --- p1.id>p2.id because we are deleting from p1 table.

**JOINS**

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



**INNER join**

The INNER JOIN clause compares each row in the t1 table with every row in the t2 table based on the join condition.

SELECT

select\_list

FROM t1

INNER JOIN t2 ON join\_condition1

INNER JOIN t3 ON join\_condition2

...;

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2ON table1.column\_name = table2.column\_name;

e.g..

SELECT Orders.OrderID, Customers.CustomerName  
FROM Orders  
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;\

**LEFT / LEFT OUTER join**

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

*In some databases it is called as left outer join*

SELECT column\_name(s)

FROM table1

LEFT JOIN table2

ON table1.column\_name = table2.column\_name;

**Example**

SELECT Customers.CustomerName, Orders.OrderID

FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID

ORDER BY Customers.CustomerName;

**RIGHT join**

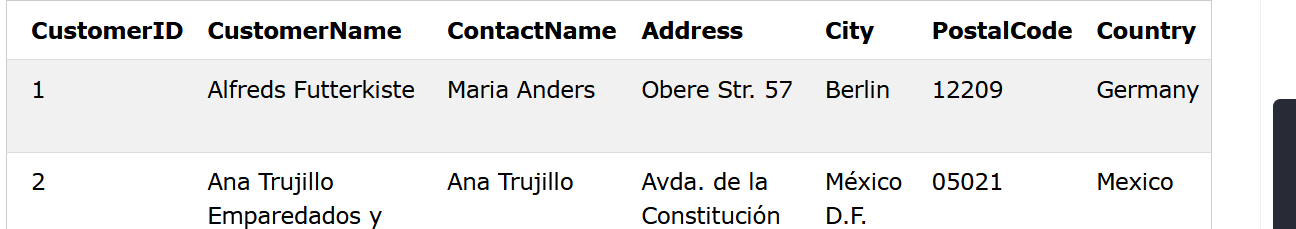
Same as left joins

**Self join**

A self join is a regular join, but the table is joined with itself.

SELECT column\_name(s)  
FROM table1 T1, table1 T2  
WHERE condition;

Let say we have below table structure



**Question**:- Write a SQL query to find customer from same city.

selet A.customer\_name, B.customer\_name

from customer A , customer B

where A.customername<>B.customername

where A.city=B.city

**CROSS join**

The CROSS JOIN clause returns the Cartesian product of rows from the joined tables.

The result set will include all rows from both tables, where each row is the combination of the row in the first table with the row in the second table. In general, if each table has n and m rows respectively, the result set will have nxm rows.

SELECT \*/column(s) FROM t1

CROSS JOIN t2;

Number of rows returned by FULL OUTER JOIN = (No. of Rows by LEFT OUTER JOIN) + (No. of Rows by RIGHT OUTER JOIN) - (No. of Rows by INNER JOIN).

**CROSS join vs FULL outer join**

**Full Outer Join:** A full outer join is a combination of a left outer and right outer join. It returns all rows in both tables that match the query's where clause, and in cases where the ON condition can't be satisfied for those rows it puts NULL values in for the unpopulated fields.

**CROSS Join:** A cross join produces a cartesian product between the two tables, returning all possible combinations of all rows. It has no on clause because you're just joining everything to everything.

**UNION:** Creates union of both the table for which all column names and data types in two tables should be same. There is no chance of NULL values

**INSERT INTO**

INSERT INTO table(c1,c2,...)

VALUES (v1,v2,...);

Inserting multiple rows

INSERT INTO table(c1,c2,...)

VALUES

(v11,v12,...),

(v21,v22,...),

...

(vnn,vn2,...);

**INSERT INTO SELECT**

Here instead of value statement we specifies values using select statement

INSERT INTO table\_name(column\_list)

SELECT

select\_list

FROM

another\_table

WHERE

condition;

**INSERT ON DUPLICATE KEY UPDATE**

INSERT INTO table (column\_list)

VALUES (value\_list)

ON DUPLICATE KEY UPDATE

c1 = v1,

c2 = v2,

...;

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.

Basically, the statement first tries to insert a new row into the table. If a duplicate error occurs, it will update the existing row with the value specified in the ON DUPLICATE KEY UPDATE clause.

**# Join when no direct relation between two tables # ---- Interview**

We can join two table where there is no direct relation between two table. In such case we can use two or more column to do join operations.

**Example:**

Given below Student and Grade table there is no direct relation but we can use Min\_Mark and Max\_Mark from Grade table to perform join.

|  |  |
| --- | --- |
|  | Here we can use Mark from Student and Min\_Mark and Max\_Mark from Grade table to establish relation |

**Sample query:**

SELECT STUDENTS.NAME, GRADES.GRADE, STUDENTS.MARKS FROM STUDENTS

JOIN GRADES ON MARKS BETWEEN MIN\_MARK AND MAX\_MARK

**Questions:**

WAQ to get the name , mark and Grade.

SELECT STUDENTS.NAME, GRADES.GRADE, STUDENTS.MARKS FROM STUDENTS

JOIN GRADES ON MARKS BETWEEN MIN\_MARK AND MAX\_MARK

################################################################################# MYSQL keywords #

################################################################################

<https://www.w3schools.com/sql/sql_ref_keywords.asp>

1. ADD CONSTRAINT

The ADD CONSTRAINT command is used to create a constraint after a table is already created. As table was already created then then adding constraint then we need to use it with ALTER TABLE keyword.

ALTER TABLE tbl\_name

[alter\_option| ADD CONSTRAINT]

**alter option**

there are many alter option available, few of them are-

1. Add [column]
2. Add [constraint]
3. Drop [constraint]
4. Drop [column]
5. Drop Primary Key

Example:

ALTER TABLE Persons

ADD CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName);

**Note:**

If we are modifying anything property (adding column, keys, deleting) then we will have to use ALTER TABLE keyword.

1. ADD

Adds column in existing table. This must be also used with ALTER TABLE keyword.

ALTER TABLE Customers

ADD table\_name data\_type;

1. ALTER /ALTER TABLE / ALTER COLUMN

ALTER command is used to change table or column property of table.

Adds, deletes, or modifies columns in a table, or changes the data type of a column in a table.

1. **ALTER TABLE**

This is used with alter/change the table property. (already above)

1. **ALTER COLUMN**

This is used to change the column data type of given table. Always used with ALTER TABLE.

ALTER TABLE table\_name

ALTER COLUMN col\_name new\_data\_type;

1. ALL

The ALL command returns true if all of the subquery values meet the condition.

The word ALL, which must follow a comparison operator, means “return TRUE if the comparison is TRUE for ALL of the values in the column that the subquery returns.”

SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);

1. AND ------ Only includes rows where conditions are TRUE
2. ANY --- Returns true if any of the subquery values meet the condition
3. AS --- this is for giving any alais name
4. ASC ---- Sorts the result set in ascending order
5. BETWEEN

The MYSQL BETWEEN condition specifies how to retrieve values from an expression within a specific range (values will be inclusive). It is used with SELECT, INSERT, UPDATE and DELETE statement.

expression BETWEEN value1 AND value2; ------- syntax

**example**

SELECT \* FROM Products

WHERE Price BETWEEN 10 AND 20;

1. CHECK

The CHECK constraint limits the value that can be placed in a column. It is used with CREATE table option.

CHECK (expr) ----------- Syntax

**Example**

CREATE TABLE Persons (

Age int,

CHECK (Age>=18)

);

1. DELETE

DELETE is a DML statement that removes rows from a table.

The DELETE statement deletes rows from tbl\_name and returns the number of deleted rows. To check the number of deleted rows, call the ROW\_COUNT() function

DELETE FROM tbl\_name [[AS] tbl\_alias]

[WHERE where\_condition | expression]

[ORDER BY ...]

[LIMIT row\_count]

**Delete with join condtion.**

DELETE <table\_name>

FROM table1

<join\_type> tabl1 I ON tabl1.Id = table2.ID

WHERE condition

**Note:**

Based on the condition what ever record will be returned as output will be deleted

Example:

DELETE w

FROM WorkRecord2 w

INNER JOIN Employee e

ON EmployeeRun=EmployeeNo

WHERE Company = '1' AND Date = '2013-05-06'

**Example**

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

1. SELECT DISTINCT

The SELECT DISTINCT command returns only distinct (different) values in the result set.

**Example**

SELECT DISTINCT Country FROM Customers;

1. DROP /DROP TABLE / DROP COLUMN
2. **DROP COLUMN**

DROP COLUMN deletes a column from table, as it is modifying the table then we must use it with ALTER TABLE keyword as mention above.

ALTER TABLE table\_name

DROP COLUMN col\_name; -------- Drop column

ALTER TABLE table\_name

DROP PRIMARY KEY; ------- Drop Primary key

ALTER TABLE Orders

DROP FOREIGN KEY foreign\_key\_name; ---- Drop Foreign key

ALTER TABLE Persons

DROP CHECK check\_statement; ---- Drop check constrains

ALTER TABLE Persons

ALTER coln\_name DROP DEFAULT; --- Drop default value for a given col\_name

1. DROP TABLE

This is used for deleting table.

DROP table tbl\_name ------ Dropping table

1. EXISTS

The EXISTS command tests for the existence of any record in a **subquery**, and returns true if the subquery returns one or more records

Query where EXISTS (sub query) --------- Syntax

**Example**

SELECT SupplierName

FROM Suppliers

WHERE EXISTS (SELECT ProductName FROM Products WHERE SupplierId = Suppliers.supplierId AND Price < 20);

**Questions**

Write a query to find the name (first\_name, last\_name) of all employees who works in the IT department

<https://www.w3resource.com/mysql-exercises/subquery-exercises/find-the-names-of-all-employees-who-works-in-the-it-department.php>

**Solutions 1**

select first\_name, last\_name from employees where department\_id= (select department\_id from departments where department\_name='IT');

**Solution 2:**

select first\_name, last\_name from employees where department\_id IN (select department\_id from departments where department\_name='IT');

**Questions:**

Write a query to find the name (first\_name, last\_name) of the employees who are managers.

<https://www.w3resource.com/mysql-exercises/subquery-exercises/find-the-names-of-the-employees-who-are-managers.php>

**Solutions**

select \* from employees where employee\_id in (select distinct(manager\_id) from employees);

1. IN

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

The IN operator is a shorthand for multiple OR conditions.

select statement WHERE col\_name IN (val1, val2,...valn)

**IN vs BETWEEN**

BETWEEN --------- It takes only two value ( inclusive) for specifying range

IN ------- It takes any number of value of any type.

1. TOP, LIMIT and ROWNUM

These commands are used to specify the number or X percent of records to return.

These three works same but are database dependent keyword

LIMIT ------ MYSQL

ROWNUM ---- Oracle

SELECT FROM table\_name

WHERE [condition]

LIMIT number|percent column\_name(s) ----------- Syntax

**Example:**

SELECT \* FROM employees

ORDER BY salary DESC LIMIT 3;

**Setting Row Offset in LIMIT Clause**

The LIMIT clause accepts an optional second parameter.

the first parameter specifies the offset of the first row to return i.e. the starting point, whereas the second parameter specifies the maximum number of rows to return

SELECT FROM table\_name

WHERE [condition]

LIMIT offset\_number, number\_of\_records\_to\_fetch column\_name(s) ----------- Syntax

offset\_number start with 0

offset\_number=0 -----> first record

offset \_number=1 -----> second record

**Example**

SELECT \* FROM employees

ORDER BY salary DESC LIMIT 2, 1;

1. SET

The SET command is used with UPDATE to specify which columns and values that should be updated in a table.

UPDATE Customers

SET var\_name1|col\_name1= vaue

WHERE condition

Example

UPDATE Customers  
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'  
WHERE CustomerID = 1;

###############################################################################

# FUNCTIONS IN MYSQL #

###############################################################################

Function can be divided into—

1. String function
2. Numerical Function
3. DATE function
4. Server functions

###################################

# String function #

###################################

1. UPPER / UCASE

UPPER() function converts a string to upper-case.

UPPER(text)

Example

SELECT UPPER(CustomerName) AS UppercaseCustomerName

FROM Customers;

1. TRIM

Removes leading and trailing spaces from String. ( Same as python trim() of string)

TRIM(text) ----------

Example:

SELECT TRIM(' SQL Tutorial ') AS TrimmedString;

1. LTRIM / RTRIM

These functions are used to remove spaces from left or right of string. (same in python )

1. STRCMP

The STRCMP() function compares two strings

STRCMP( string1 , string2)

String1 and string2 are two string to be compared

1. SUBSTRING() ---------- this was asked in interview

The SUBSTRING() function extracts a substring from a string (starting at any position).

SUBSTRING(string, start, length)

String ----- string function where substring will be extracted

Start ---- start index for getting substring, it’s minimum value is 1

Length ---- number of character to be extracted, if omitted then will get all string from start.

**Example:**

SELECT SUBSTRING(CustomerName, 2, 5) AS ExtractString

FROM Customers; ------------ this was asked in interview

1. SPACE

The SPACE() function returns a string of the specified number of space characters.

**Example**

SELECT SPACE(10);

1. REVERSE

The REVERSE() function reverses a string and returns the result. (same function in python also but we can do by using slicing concept also)

REVERSE(string) -------- Syntax

**Example**

SELECT REVERSE(CustomerName)

FROM Customers;

1. REPLACE

The REPLACE() function replaces all occurrences of a substring within a string, with a new substring. In python we have sub() or subn() function

REPLACE (string, string\_to\_be\_replaced, new\_string) ------ Syntax

String ---- Required. The original string, in which replacement will be done

String\_to\_be\_replaced ---- Required. The substring to be replaced

New\_string ----- Required. The new replacement substring

1. LOWER / LCASE --------- Converts string into lower case ( same function I python also)

LOWER ( text)

**Example**

SELECT LOWER("SQL Tutorial is FUN!");

10. REPEAT

The REPEAT() function repeats a string as many times as specified.

REPEAT(string, number) -------- Syntax

String ------- string to be repeated. Required

Number ---- number to times to be repeated. Repeated

**Example**

SELECT REPEAT(CustomerName, 2)

FROM Customers;

11. CONCAT

The CONCAT() function adds two or more expressions together.

CONCAT(expression1, expression2, expression3,...)

NOTE:

If any of expression is null then result will be null.

12. GROUP\_CONCAT

This function is used to concatenate string from multiple rows into a single string.

SELECT col1, col2, ..., colN

GROUP\_CONCAT ([DISTINCT] col\_name1

[ORDER BY clause] [SEPARATOR str\_val] )

FROM table\_name GROUP BY col\_name2;

Default value of separator is comma(,)

**Question --- Leetcode**

Write an SQL query to find for each date the number of different products sold and their names from Activities table.

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| sell\_date | date |

| product | varchar |

+-------------+---------+

Hint:- Use GROUP\_CONCAT function.

Solution:

select sell\_date,

count(distinct product) as num\_sold,

group\_concat(distinct product) as products

from Activities

group by sell\_date

**Note:**

we are using 'distinct product' b/c we want count for unique product name.

12. CHAR\_LENGTH/CHARACTER\_LENGTH()

The CHARACTER\_LENGTH() function return the length of a string (in characters).

CHARACTER\_LENGTH(string) ------ Syntax

13. LEFT

The LEFT() function extracts a number of characters from a string (starting from left).

LEFT(string, number\_of\_chars) ------ Syntax

string--------Required. The string to extract from.

number\_of\_chars----Required. The number of characters to extract. If this parameter is larger than the number of characters in string, this function will return string

**Example**

SELECT LEFT(CustomerName, 5) AS ExtractString

FROM Customers;

14.RIGHT

Same as LEFT but it return from right.

**What is DBMS and RDBMS**

**DBMS** --- Data base management System that is used to define, create and maintain a database and provides the controlled access to the data

It Stores the data in form of file.

**RDBMS**--- This is stores data in a tabular format (key-value pair)

**What is Primary Key:**

Column which stores unique value and not null value in a table to uniquely identify each row.

Primary key column can’t contain Null value.

PKEY= > Unique + Not Null

**What is unique key:**

Column which contains unique value. It can contain only one null value.

**What is Foreign Key:**

Column (s) which refer to primary key of another table. Column with foreign key is called child table.

**What are constraints:**

These are rules for any column that defines kind of data it can store. e.g.-

NOT NULL ---- Column will not contain any null value

UNIQUE ------- Column will contain unique values

PRIMARY KEY

FOREIGN KEY

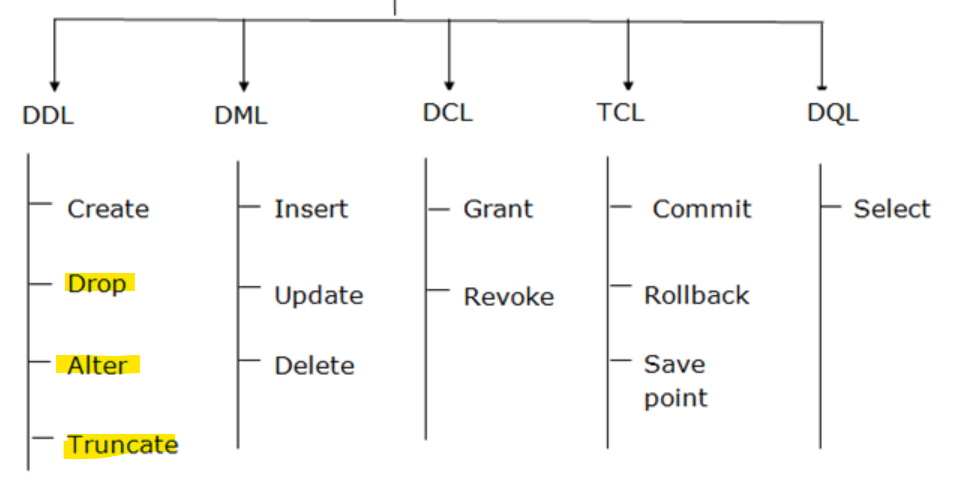
CHECK ----- Ensures that value in column full fills specified condition

DEFAULT ---- Default value for that column

**Explain the type of SQL commands**

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL as-

* Data Definition Language (DDL)
* Data Manipulation Language (DML)
* Data Control Language(DCL)
* Transaction Control Language(TCL)
* Data Query Language (DQL)



**What is difference between DELETE, DROP and TRUNCATE in sql.**



**What is GROUP BY and ORDER BY**

GROUP BY – For grouping of data

ORDER BY --- Sorting data in ascending or descending order

**What are all possible types of join in SQL.**

CROSS JOIN

INNER JOIN

LEFT JOIN (LEFT OUTER JOIN)

RIGHT JOIN (RIGHT OUTER JOIN)

FULL JOIN

NATURAL JOIN

**Difference between Nested Query, Correlated Query and Joins**



**What is natural join**

**Natural Join:**

Natural Join in SQL combines records from two or more tables based on the common column between them. The common column must have the same name and data type in both the tables.

We don’t need to specify the ON condition.

Natural join can be of below type –

|  |  |
| --- | --- |
| Natural Inner join | SELECT \* FROM tableA **NATURAL JOIN** tableB where condition |
| Left Natural join | SELECT \* FROM tableA **NATURAL LEFT JOIN** tableB where condition |
| Right Natural join | SELECT \* FROM tableA **NATURAL RIGHT JOIN** tableB where condition |
| Full Outer join | SELECT \* FROM tableA **NATURAL FULL JOIN** tableB where condition |

**Example:**

Let say we have below two tables and perform natural inner join

Employee table Department table

|  |  |
| --- | --- |
| **| EmployeeID | FirstName | LastName | DeptID |**  |:----------:|:---------:|:--------:|:------:|  | E62549 | John | Doe | D1001 |  | E82743 | Priya | Sharma | D3002 |  | E58461 | Raj | Kumar | D1002 |  | E95462 | Ravi | | D1001 |  | E25947 | Shreya | P | D3000 | | **| DeptID | DeptName | Location |**  |:------:|:----------:|:---------:|  | D1001 | Technology | Bangalore |  | D1002 | Technology | Hyderabad |  | D3001 | Sales | Gurugram |  | D3002 | Operations | Hyderabad | |

Answer:

SELECT \* FROM employee NATURAL JOIN department

Note:

Using natural join so no need to specify the column names on which joins will be performed.

| EmployeeID | FirstName | LastName | DeptID | DeptName | Location |

|:----------:|:---------:|:--------:|:------:|:----------:|:---------:|

| E62549 | John | Doe | D1001 | Technology | Bangalore |

| E82743 | Priya | Sharma | D3002 | Operations | Hyderabad |

| E58461 | Raj | Kumar | D1002 | Technology | Hyderabad |

| E95462 | Ravi | | D1001 | Technology | Bangalore |

| E42650 | Jane | Scott | D3001 | Sales | Gurugram |

**Difference between natural join and inner/other join**

|  |  |
| --- | --- |
| Natural Join | Other Join |
| Needs a column name, data types in two table should be same. | Column name and data types need not to be same, we have to specify the column name on which join will be performed. |

**Questions--- LeetCode**

Write an SQL query to fix the names in User table so that only the first character is uppercase and the rest are lowercase.

Users table schema is-

+----------------+---------+

| Column Name | Type |

+----------------+---------+

| user\_id | int |

| name | varchar |

+----------------+---------+

Solution:

select user\_id,concat(UPPER(substring(name,1,1)),LOWER(substring(name,2))) as name from Users order by user\_id;

##########################

# Numerical Function #

##########################

1. ABS

The ABS() function returns the absolute (positive) value of a number.

ABS (number)

Example:

SELECT ABS(-243.5);

1. AVG

The AVG() function returns the average value of an expression.

AVG(expression) ------- Syntax

expression ---- Required. A numeric value (can be a field or a formula)

Example

SELECT \* FROM Products

WHERE Price > (SELECT AVG(Price) FROM Products);

1. CEIL

The CEIL() function returns the smallest integer value that is bigger than or equal to a number.

CEIL(number) ----- Syntax

Number ----------- required, A numeric value.

Example:

SELECT CEIL(25);

1. COUNT

The COUNT() function returns the number of records returned by a select query.

count (expression) --- Synntax

select count(expression|col\_name) from table ----- Syntax

**Example**

SELECT COUNT(ProductID) AS NumberOfProducts FROM Products;

-----Return the number of products in the "Products" table:

1. FLOOR

The FLOOR() function returns the largest integer value that is smaller than or equal to a number.

FLOOR(number) ------- Syntax

Example:

SELECT FLOOR(25);

1. DIV

The DIV function is used for integer division (x is divided by y). An integer value is returned.

DIV (x,y) ------- syntax

X --- Required, A value that will be divided

Y --- Required, The divisor

1. GREATEST

The GREATEST() function returns the greatest value of the list of arguments.

GREATEST(arg1, arg2, arg3, ...) ---- Syntax

arg1, arg2….. -------- Required. The list of arguments to be evaluated

**Example**

SELECT GREATEST("w3Schools.com", "microsoft.com", "apple.com");

1. LEAST

The LEAST() function returns the smallest value of the list of arguments.

LEAST(arg1, arg2, arg3, ...) ---- Syntax

arg1, arg2….. -------- Required. The list of arguments to be evaluated

Example:

1. SELECT LEAST(3, 12, 34, 8, 25);
2. SELECT LEAST("w3Schools.com", "microsoft.com", "apple.com");
3. MAX/MIN

The MAX() function returns the maximum value in a set of values.

The MIN() function returns the minimum value in a set of values.

MIN(expression) ------- Syntax

MAX(expression) ------- Syntax

**Example:**

SELECT MIN(Price) AS SmallestPrice FROM Products;

1. ROUND

The ROUND() function rounds a number to a specified number of decimal places.

ROUND(number, decimals) ------- Syntax

number ------ Required. The number to be rounded.

decimal ------ Optional. The number of decimal places to round number to. If omitted, it returns the integer (no decimals)

**Example**

1.)SELECT ROUND(345.156, 0);

2.)SELECT ProductName, Price, ROUND(Price, 1) AS RoundedPrice FROM Products;

1. SIGN ---- return the sign of number
2. SQRT ---- returns the square root of number
3. SUM --- Calculte sum of number
4. TRUNCATE ---- Truncates a number to the specified number of decimal places

TRUNCATE(number, decimals) ---- Syntax

number ----- Required. The number to be truncated

decimal ---- Required. The number of decimal places to truncate to

**Example**

1.)SELECT TRUNCATE(345.156, 0);

2.)SELECT TRUNCATE(135.375, 2);

**# Decision making statement #**

For decision making we use **IF, IFNULL, ISNULL, NULLIF etc** statement.

**IF statement**

IF(condition, value\_if\_true, value\_if\_false)

IF(condition, value\_if\_true, value\_if\_false) as col\_name

Example1:

SELECT OrderID, Quantity, IF(Quantity>10, "MORE", "LESS")  
FROM OrderDetails;

**Question: ---- Good, LeetCode**

WAQ to calculate the bonus for all employees on below condition from Employee table.

The bonus of an employee is 100% of their salary if the ID of the employee is an odd number and the employee name does not start with the character 'M'. The bonus of an employee is 0 otherwise.

We have employee table with below schema definition-

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| employee\_id | int |

| name | varchar |

| salary | int |

+-------------+---------+

Solution:

# Write your MySQL query statement below

select employee\_id,IF(employee\_id%2<>0 and substring(name,1,1)<>'M', salary, 0) as bonus from Employees

order by employee\_id;

**IFNULL statement**

The IFNULL() function returns a specified value(alt\_value) if the expression is NULL.

If the expression is NOT NULL, this function returns the expression.

IFNULL(expression, alt\_value)

Example:

SELECT IFNULL("Hello", "W3Schools.com"); #Output will be Hello

SELECT IFNULL(NULL, 500); # Output will be 500

**NULLIF statement**

The NULLIF() function compares two expressions and returns NULL if they are equal. Otherwise, the first expression is returned.

NULLIF(expr1, expr2)

**ISNULL statement**

The ISNULL() function returns 1 or 0 depending on whether an expression is NULL.

ISNULL(expression)

**UNION statement**

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 and same datatype

The column name must be in same order in each select statement.

SELECT column\_name(s) FROM table1  
UNION  
SELECT column\_name(s) FROM table2;

Note:

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

**UNION ALL operator**

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;

**Question --- Leetcode**

Write an SQL query to report the IDs of all the employees with missing information from Employees and Salaries. The information of an employee is missing if.

* The employee's name is missing, or
* The employee's salary is missing.

|  |  |
| --- | --- |
| Employees table  +-------------+---------+  | Column Name | Type |  +-------------+---------+  | employee\_id | int |  | name | varchar |  +-------------+---------+ | Salaries table  +-------------+---------+  | Column Name | Type |  +-------------+---------+  | employee\_id | int |  | salary | int |  +-------------+---------+ |

Hint:

Use UNION operator.

**Solution**

select distinct employee\_id from Salaries  where Salaries.employee\_id not in (select distinct employee\_id from Employees)

UNION

select distinct employee\_id from Employees where Employees.employee\_id not in (select distinct employee\_id from Salaries)

order by employee\_id;

Question ---- Leetcode

Write an SQL query to rearrange the Products table so that each row has (product\_id, store, price). If a product is not available in a store, do not include a row with that product\_id and store combination in the result table.

Product table is-

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| product\_id | int |

| store1 | int |

| store2 | int |

| store3 | int |

+-------------+---------+

Example:

**Input:**

Products table:

+------------+--------+--------+--------+

| product\_id | store1 | store2 | store3 |

+------------+--------+--------+--------+

| 0 | 95 | 100 | 105 |

| 1 | 70 | null | 80 |

+------------+--------+--------+--------+

**Output:**

+------------+--------+-------+

| product\_id | store | price |

+------------+--------+-------+

| 0 | store1 | 95 |

| 0 | store2 | 100 |

| 0 | store3 | 105 |

| 1 | store1 | 70 |

| 1 | store3 | 80 |

+------------+--------+-------+

Hint:

Use UNION ALL operator.

Solution

# Write your MySQL query statement below

select product\_id,'store1' as store,store1 as price from Products where store1 is not null

UNION ALL

select product\_id,'store2' as store,store2 as price from Products where store2 is not null

UNION ALL

select product\_id,'store3' as store,store3 as price from Products where store3 is not null

**Question --- Leetcode**

Write an SQL query to report the type of each node in the Tree table.

Each node in the tree can be one of three types:

* "Leaf": if the node is a leaf node.
* "Root": if the node is the root of the tree.
* "Inner": If the node is neither a leaf node nor a root node.

Example-

**Input:**

Tree table:

+----+------+

| id | p\_id |

+----+------+

| 1 | null |

| 2 | 1 |

| 3 | 1 |

| 4 | 2 |

| 5 | 2 |

+----+------+

**Output:**

+----+-------+

| id | type |

+----+-------+

| 1 | Root |

| 2 | Inner |

| 3 | Leaf |

| 4 | Leaf |

| 5 | Leaf |

+----+-------+

**Solution**

# Write your MySQL query statement below

select id,'Root' as type from Tree where p\_id is null

union

select id,'Leaf' as type from Tree where id not in (select distinct p\_id from Tree where p\_id is not null) and p\_id is not null

union

select id,'Inner' as type from Tree where p\_id is not null and  id in (select distinct p\_id from Tree where p\_id is not null)

**MINUS operator**

The Minus Operator in SQL is used with two SELECT statements. The MINUS operator is used to subtract the result set obtained by first SELECT query from the result set obtained by second SELECT query.

SELECT column1 , column2 , ... columnN

FROM table\_name

WHERE condition

**MINUS**

SELECT column1 , column2 , ... columnN

FROM table\_name

WHERE condition;

Note:

* The number of columns in both SELECT statements must be same.
* The data type of corresponding columns of both SELECT statement must be same.
* The WHERE clause is optional in the above query.

Example 1:

Select second highest salary from Employee table.

Method 1: -------- Use of MINUS operator

SELECT name, MAX(salary) AS salary

FROM employee

WHERE salary IN

(SELECT salary FROM employee **MINUS** SELECT MAX(salary)

FROM employee);

Method 2:

SELECT salary FROM Employee ORDER BY salary DESC LIMIT 1, 1

Note: offset for LIMIT clause start with 0

**Question ----- Searching a target value in a sorted list ---- Using binary search O(Log(n))**

Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1.

Algo:

We will use two pointers left and right and a middle (middle=(l+r)//2)

If value as middle index <target ----- > target will be in right half, update left=middle+1

If value at middle index > target ---- > target will be in left half, update rigt=middle-1

If value at middle index==target ---- > return middle

Keep doing above three steps till left<right.

class Solution:

    def search(self, nums: List[int], target: int) -> int:

        if nums==[]:

            return -1

        N=len(nums)

        l=0

        r=N-1

        while l<r:

            middle=(l+r)//2

            #if target is in left

            if target<nums[middle]:

                r=middle-1

            elif target>nums[middle]:

                l=middle+1

            else:

                return middle

        if nums[l]==target:

            return l

        else:

            return -1

**#Question # ----- Check the first bad version of product ----- Leetcode**

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which returns whether version is bad. Implement a function to find the first bad version. **You should minimize the number of calls to the API.**

Hint:

Same as previous set three pointer left and right. Left=1 and right=n and bad\_vr=None

Find middle as (left+right)//2, if version as middle is good then bad version will be in right, update the left pointer as left=middle+1 , if it is bad then update right =middle-1 and bad\_vr=middle

Keep on looping this algo till left<right to find the root bad version.

Note:

When loop breaks then l==r but loop will not check if it is bad to at end , check if it is bad. In case bad then return l (that version) else return the version which was found as bad (bad\_vr)

# The isBadVersion API is already defined for you.

# def isBadVersion(version: int) -> bool:

class Solution:

    def firstBadVersion(self, n: int) -> int:

        l=1

        r=n

        bad\_vr=''

        while l<r:

            middle=(l+r)//2

            print(l,r,middle)

            if isBadVersion(middle)==True:

                r=middle-1

                bad\_vr=middle

            else:

                l=middle+1

        #here both l and l will be same, check if this is bad or not

        #loop does not checks this value of l and r

        print(l,r)

        if isBadVersion(l)==True:

            return l

        else:

            return bad\_vr

**Question ------- LeetCode --- Good**

Write an SQL query to report the names of all the salespersons who did not have any orders related to the company with the name **"RED"**.

Table: SalesPerson

+-----------------+---------+

| Column Name | Type |

+-----------------+---------+

| sales\_id | int |

| name | varchar |

| salary | int |

| commission\_rate | int |

| hire\_date | date |

+-----------------+---------+

sales\_id is the primary key column for this table.

Each row of this table indicates the name and the ID of a salesperson alongside their salary, commission rate, and hire date.

Table: Company

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| com\_id | int |

| name | varchar |

| city | varchar |

+-------------+---------+

com\_id is the primary key column for this table.

Each row of this table indicates the name and the ID of a company and the city in which the company is located.

Table: Orders

+-------------+------+

| Column Name | Type |

+-------------+------+

| order\_id | int |

| order\_date | date |

| com\_id | int |

| sales\_id | int |

| amount | int |

+-------------+------+

order\_id is the primary key column for this table.

com\_id is a foreign key to com\_id from the Company table.

sales\_id is a foreign key to sales\_id from the SalesPerson table.

Each row of this table contains information about one order. This includes the ID of the company, the ID of the salesperson, the date of the order, and the amount paid

Solution 1: --- Using joins

# Write your MySQL query statement below

select name from SalesPerson where SalesPerson.sales\_id not in (select SalesPerson.sales\_id from SalesPerson

left join Orders on SalesPerson.sales\_id=Orders.sales\_id

left join Company

on Orders.com\_id=Company.com\_id where Company.name="RED")

Solution 2: ----Without joins

SELECT name FROM salesperson

WHERE sales\_id NOT IN

    (SELECT sales\_id FROM orders WHERE com\_id IN

    (SELECT com\_id FROM company WHERE name='RED'))

**#Qestuin# ---- Interview ---- Join when no direct relation**

Get the name of student,mark and grade from Student and Grades table.

<https://www.hackerrank.com/challenges/the-report/problem?isFullScreen=true&h_r=next-challenge&h_v=zen>

|  |  |
| --- | --- |
| Student table | Grade table |

Solution:

SELECT NAME, GRADES.GRADE,STUDENTS.MARKS FROM STUDENTS

JOIN GRADES ON MARKS BETWEEN MIN\_MARK AND MAX\_MARK

**Question ---- Top three salary for each department**

[**https://leetcode.com/problems/department-top-three-salaries/description/**](https://leetcode.com/problems/department-top-three-salaries/description/)

A company's executives are interested in seeing who earns the most money in each of the company's departments. A **high earner** in a department is an employee who has a salary in the **top three unique** salaries for that department.

Write an SQL query to find the employees who are **high earners** in each of the departments.

Return the result table **in any order**.

|  |  |
| --- | --- |
| Employee table  +--------------+---------+  | Column Name | Type |  +--------------+---------+  | id | int |  | name | varchar |  | salary | int |  | departmentId | int |  +--------------+---------+  id is the primary key column for this table.  departmentId is a foreign key of the ID from the Department table. | Department table  +-------------+---------+  | Column Name | Type |  +-------------+---------+  | id | int |  | name | varchar |  +-------------+---------+  id is the primary key column for this table.  Each row of this table indicates the ID of a department and its name. |

Solution:

select tbl1.dep\_name as Department,tbl1.emp\_name as Employee, tbl1.Salary from

(select emp.name as emp\_name,dep.name as dep\_name,emp.salary,

DENSE\_RANK() over(partition by dep.name order by emp.salary desc) as num

from Employee emp

join Department dep on emp.departmentId=dep.id) as tbl1

where num <=3

**##################**

**# Window Function #**

**##################**

We have below window function in SQL-

* RANK()
* DENSE\_RANK()
* ROW\_NUMBER()

**Syntax:**

SELECT coulmn\_name1,

**window\_function**(cloumn\_name2)

OVER([PARTITION BY column\_name1] [ORDER BY column\_name3]) AS new\_column

FROM table\_name;

**window\_function** – this could be window function or aggregation function

**RANK()**

As the name suggests, the rank function assigns rank to all the rows within every partition. Rank is assigned such that rank 1 given to the first row and rows having same value are assigned same rank. For the next rank after two same rank values, one rank value will be skipped.

**DENSE\_RANK()**

It assigns rank to each row within partition. Just like rank function first row is assigned rank 1 and rows having same value have same rank. The difference between RANK() and DENSE\_RANK() is that in DENSE\_RANK(), for the next rank after two same rank, consecutive integer is used, no rank is skipped

**ROW\_NUMBER()**

It assigns consecutive integers to all the rows within partition. Within a partition, no two rows can have same row number.

**Note –**

ORDER BY() should be specified compulsorily while using rank window functions.

**####################**

**# PARTITION BY Clause #**

**####################**

A PARTITION BY clause is used to partition rows of table into groups. It is useful when we have to perform a calculation on individual rows of a group using other rows of that group.

It is always used inside OVER() clause.

* The partition formed by partition clause are also known as Window.
* This clause works on windows functions only. Like- RANK(), LEAD(), LAG() etc.
* If this clause is omitted in OVER() clause, then whole table is considered as a single partition.

Window\_function ( expression )

Over ( partition by col1, col2 [order by col\_a, co\_b.. coln] [frame\_clause] )

**Order by:**

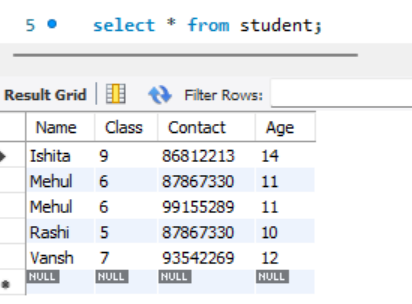
This is used to order each group row data and then assign the rank, row\_number based on given column

**NOTE:**

* We should always use order by in case if we using RANK or DENSE\_RANK function else ranking will same for each row in a group.
* For ROW\_NUMBER() it’s not mandatory to use order\_by.

Example:

Let say we have below data in student table-



**Question1**:

**Group the record by name and class column and assign the rank for each record/row**

**CASE 1:** DENSE\_RANK with ORDER BY

select Name,Class,Contact,Age,

DENSE\_RANK() OVER( partition by Name,Class order by Contact ) as rnk

from student;

|  |  |
| --- | --- |
| Table data | Query output    **Note:**  We can we it have assigned a rank for each group based on unique value of contact in each row |

**CASE 2:** **DENSE\_RANK without order by not used**

If order by is not used in rank functions ( RANK, DENSE\_RANK ) then it will assign same value (1) for each row in a group.

select Name,Class,Contact,Age,

DENSE\_RANK() OVER( partition by Name,Class ) as rnk

from student;

|  |  |
| --- | --- |
| Table data | Query output |

**CASE 3**: ROW\_NUMBER --- it always assigns the row number starting from 1 for each group.

select Name,Class,Contact,Age,

**ROW\_NUMBER() OVER( partition by Name ) as rnk**

from student;

|  |  |
| --- | --- |
| Table data | Query output |

**################################################################################**

**# CTE ( Common Table expression ) #**

**################################################################################**

* CTE was introduced in SQL 2005 version.
* A common table expression, or CTE, is a temporary named result set created from a SELECT statement that can be used in a subsequent SELECT statement.
* We can define CTEs by adding a WITH clause directly before the SELECT, INSERT, UPDATE, DELETE, or MERGE statement.
* CTE can not be used with below clause-

1. ORDER BY unless you also use as TOP clause
2. INTO
3. OPTION clause with query hints
4. FOR BROWSE

|  |  |
| --- | --- |
| **One CTE** | **More than one CTE / Nested CTE** |
| with cte\_name AS (  select col1, col2 . . . coln from tbl  )  Select c1, c2, . . c<n> from cte\_name  **Note:**  C1, c2 . . c<n> --- must be in cte\_name1 | with cte\_name1 AS (  select col1, col2 . . . coln from tbl  ),  cte\_name1 AS (  select col1, col2 . . . coln from tbl  )  Select c1, c2, . . c<n> from cte\_ name1 , cte\_ name2  **Note:**  C1, c2 . . c<n> --- must be in cte\_name1, cte\_name2 |

**Some of its use cases are given below/ Why we need CTE-**

1. It is useful when we need to define a derived table multiple times within a single query.
2. It is useful when we need to create an alternative to a view in the database.
3. It is useful when we need to perform the same calculation multiple times on multiple query components simultaneously.
4. It is useful when we need to use ranking functions like ROW\_NUMBER(), RANK(), and NTILE().

**Some of its advantages are given below -**

1. CTE facilitates code maintenance easier.
2. CTE increases the readability of the code.
3. It increases the performance of the query.
4. CTE makes it possible to implement recursive queries easily.

**Disadvantage of CTE-**

1. CTE members are unable to use the keyword clauses like Distinct, Group By, Having, Top, Joins, etc.
2. The CTE can only be referenced once by the Recursive member.
3. Since it's just a shortcut for a query or subquery, it can't be reused in another query.

**################**

**# MERGER in sql ##**

**################**

* MERGE statement in SQL is *used to perform insert, update, and delete operations on a target table based on the results of JOIN with a source table*. This allows users to synchronize two tables by performing operations on one table based on results from the second table.
* MERGE statement *provides a flexible way to manage data changes and is commonly used in scenarios like maintaining Slowly Changing Dimensions (SCD) in data warehouses*.

MERGE TOP (value) <target\_table>

USING <table\_source>

ON <merge\_search\_condition>

    [ WHEN MATCHED [ AND <clause\_search\_condition> ]

        THEN <merge\_matched> ]

    [ WHEN NOT MATCHED [ BY TARGET ] [ AND <clause\_search\_condition> ]

        THEN <merge\_not\_matched> ]

    [ WHEN NOT MATCHED BY SOURCE [ AND <clause\_search\_condition> ]

        THEN <merge\_matched> ]

    [ <output\_clause> ]

    [ OPTION ( <query\_hint> ) ]

**Note:**

* **WHEN MATCHED:**
  + WHEN MATCHED clause in SQL Server MERGE statement is used to update, delete the rows in the target table when the rows are matched with the source table based on the join condition.
  + we can specify only two WHEN MATCHED clauses in the MERGE statement, for updating and deleting each one.
* **WHEN NOT MATCHED BY SOURCE:**
  + this is used to delete or Update the rows in the target table that does not match join condition with a source table.
  + WHEN NOT MATCHED BY SOURCE can be maximum two times in sql. If it is two times then one must be delete and other must be UPDATE statement.
* **WHEN NOT MATCHED BY TARGET** 
  + This clause is used to insert rows into target table that does not match join condition with a source table. WHEN NOT MATCHED BY TARGET clause can be specified only once in the SQL Server MERGE statement.

Example:1 # **WHEN MATCHED ---- delete and update**

MERGE Locations T

USING Locations\_stage S ON T.LocationID=S.LocationID

WHEN MATCHED AND T.LocationID =3 THEN

DELETE

WHEN MATCHED AND T.LocationID =1 THEN

UPDATE SET LocationName=S.LocationName;

**Example:2 # WHEN NOT MATCHED BY SOURCE ---- delete and update**

MERGE Locations T

USING Locations\_stage S ON T.LocationID=S.LocationID

WHEN NOT MATCHED BY SOURCE AND LocationID =2

THEN

DELETE

WHEN NOT MATCHED BY SOURCE AND LocationID =6

THEN UPDATE SET LocationName ='Test';

Example:3 **WHEN NOT MATCHED BY TARGET ---- insert**

MERGE Locations T

USING Locations\_stage S ON T.LocationID=S.LocationID

WHEN NOT MATCHED BY TARGET

THEN

INSERT (LocationID,LocationName)

VALUES (S.LocationID,S.LocationName);

**################################**

**# PIVOTING and UNPIVOTING in SQL #**

**################################**

In SQL, Pivot and Unpivot are relational operators that are used to transform one table into another in order to achieve more simpler view of table. Conventionally we can say that Pivot operator converts the rows data of the table into the column data. The Unpivot operator does the opposite.

########

# PIVOT #

########

SELECT (ColumnNames)

FROM (TableName)

PIVOT

(

AggregateFunction(ColumnToBeAggregated)

FOR PivotColumn IN (PivotColumnValues)

) AS alias\_name ------------ Syntax

**#Example1 #**

Let say we have below table data. Now, specify the aggregation function SUM(SalesAmount), and the pivot column (Month), and show the values you expect in the pivot columns (for example, January, February, March, April,..., and December).

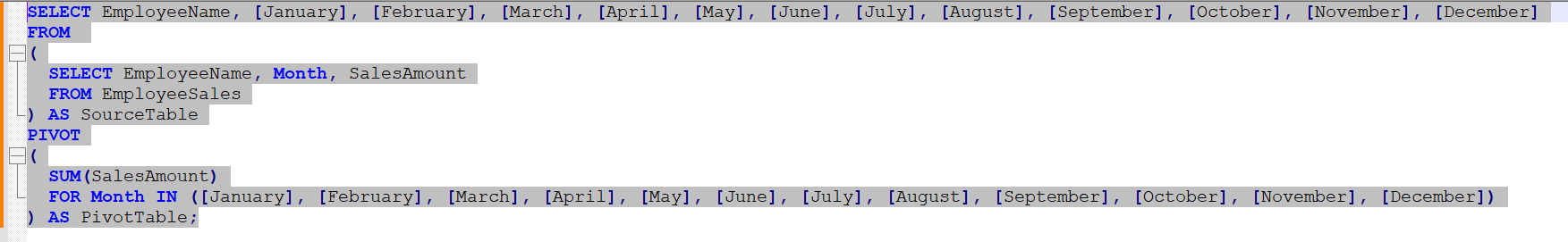
**Input table**

|  |  |  |
| --- | --- | --- |
| **EmployeeName** | **Month** | **SalesAmount** |
| Ramesh | January | 1000 |
| Suresh | January | 1500 |
| Ramesh | February | 1200 |
| Suresh | February | 1100 |

**Expected output**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Employee Name** | **January** | **February** | **March** | **April** | **May** | **June** | **July** | **August** | **September** | **October** | **November** | **December** |
| **Rames** | 1000 | 1200 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Suresh** | 1500 | 1100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Solution:**



SELECT EmployeeName, [January], [February], [March], [April], [May], [June], [July], [August], [September], [October], [November], [December]

FROM

(

-- Create alias table for pivot

SELECT EmployeeName, Month, SalesAmount

FROM EmployeeSales

) AS SourceTable

PIVOT

(

SUM(SalesAmount)

FOR Month IN ([January], [February], [March], [April], [May], [June], [July], [August], [September], [October], [November], [December])

) AS PivotTable;

**#Example 2#**

Let say we have below bookstore table. "Page Turners" wanted to analyze their sales performance across different book genres over several months. The goal was to understand consumer demand trends to optimize inventory and tailor marketing campaigns for different times of the year.

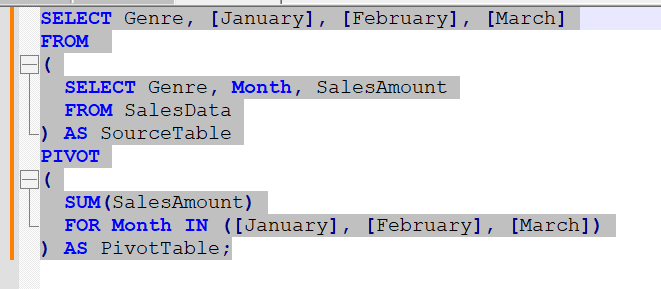
**Input table**

|  |  |  |  |
| --- | --- | --- | --- |
| **BookID** | **Genre** | **Month** | **SalesAmount** |
| 1 | Fiction | January | 1200 |
| 2 | Non-Fiction | January | 800 |
| 1 | Fiction | February | 1100 |
| 3 | Mystery | January | 500 |
| 2 | Non-Fiction | February | 900 |
| 4 | Romance | January | 650 |
| 3 | Mystery | February | 700 |
| 1 | Fiction | March | 1500 |
| 4 | Romance | February | 400 |
| 2 | Non-Fiction | March | 950 |

**Expected output**

|  |  |  |  |
| --- | --- | --- | --- |
| **Genre** | **January** | **February** | **March** |
| Fiction | 1200 | 1100 | 1500 |
| Non-Fiction | 800 | 900 | 950 |
| Mystery | 500 | 700 | 0 |
| Romance | 650 | 400 | 0 |

**Solution:**



SELECT Genre, [January], [February], [March]

FROM

(

SELECT Genre, Month, SalesAmount

FROM SalesData

) AS SourceTable

PIVOT

(

SUM(SalesAmount)

FOR Month IN ([January], [February], [March])

) AS PivotTable;

**What is clustered and uncluttered index.**

**Clustered Index:**

A clustered index is an index that specifies the physical arrangement of a database's table records. There can only be one clustered index per table since there can only be one method that records are physically maintained in a database table. It stores the records in sorted order.

**Non-Clustered Index:**

A non-clustered index is an index that doesn't specify the physical arrangement of the records maintained in the database’s table. The Non-Clustered Indexes are stored in a different table that contains the pointer to data. Therefore, as they are maintained in a different table, there can be numerous non-clustered indexes that can be created for a single table.

In Non-Clustered data is stored in one place, and the index is stored in another place. Since the data and non-clustered index is stored separately, then you can have multiple non-clustered indexes in a table. In a non-clustered index, the index contains the pointer to data.

**Key Difference between Clustered and Non-clustered Index**

* A cluster index is a type of index that sorts the data rows in the table on their key values, whereas the Non-clustered index stores the data at one location and indices at another location.
* Clustered index stores data pages in the leaf nodes of the index, while the Non-clustered index method never stores data pages in the leaf nodes of the index.
* The cluster index doesn’t require additional disk space, whereas the Non-clustered index requires additional disk space.
* Cluster index offers faster data access and slower insert operation, on the other hand, the Non-clustered index is slower.

**What is query optimizer:**

The query optimizer (called simply the optimizer) is built-in database software that determines the most efficient method for a SQL statement to access requested data.

A query optimizer generates one or more query execution plans for each query, each of which may be a mechanism used to run a query. The most efficient query plan is selected and used to run the query.

Optimizer have below three component-

1. **Query Transformer**

The optimizer determines whether it is helpful to change the form of the query so that the optimizer can generate a better execution plan.

1. **Estimator**

The optimizer estimates the cost of each plan based on statistics in the data dictionary.

1. **Plan generator**

The optimizer compares the costs of plans and chooses the lowest-cost plan, known as the execution plan, to pass to the row source generator.

**What is Execution Plan**

It is generated by query optimizer.

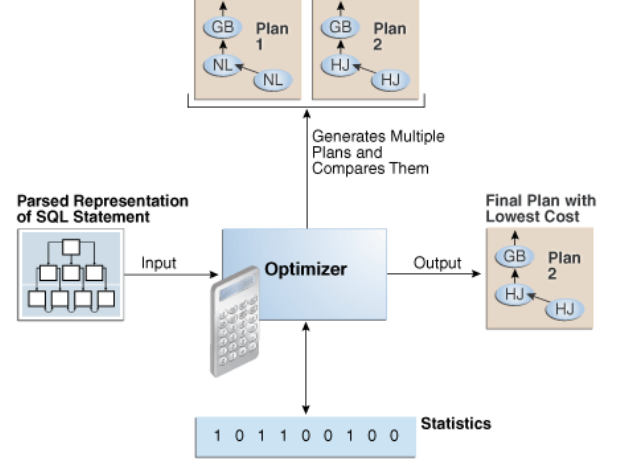
The plan shows the combination of the steps Database uses to execute a SQL statement.

There are mainly two types of execution plan-

**Actual Execution Plan**: When a query is executed then the Actual Execution Plan comes into the picture. At the time of query execution, the real process and strategy included are shown.

**Estimated Execution Plan:** The query processor only guesses the precise actions involved at the time of the result returning. Sometimes, it is generated before the query execution.

In the following graphic, the optimizer generates two possible execution plans for an input SQL statement, uses statistics to estimate their costs, compares their costs, and then chooses the plan with the lowest cost.



**A query execution plan is the may contain definition of:-**

1. **The sequence in which the source tables are accessed.**

In case we are using more than one source table then in which sequence source table must be queried.

1. **The methods used to extract data from each table.**
2. **The methods used to compute calculations, and how to filter, aggregate, and sort data from each table.**

**What is Query Block**

An SQL statement can consist of several subqueries, which are represented in the access plan diagram by query blocks. The subquery can be a SELECT, INSERT, UPDATE, or DELETE. A subquery can contain other subqueries in the FROM clause, the WHERE clause, or a subselect of a UNION or UNION ALL.

Optimizer generates the execution plan for each subquery, query block.

**What is query subplan.**

For each query block, the optimizer generates a query subplan.

The database optimizes query blocks separately from the bottom up. Thus, the database optimizes the innermost query block first and generates a subplan for it, and then generates the outer query block representing the entire query.

**Full table scan:**

**Types of testing in ETL:**

* ETL Functional Testing
* ETL Data Transformation testing
* ETL Data Validation testing
* ETL Integration Testing
* ETL Regression Testing
* ETL Validation

**Function Testing:**

The ETL functional test are designed to ensure that the ETL is following the business rules defined by the uses.

**Integration Testing:**

Integration Testing is defined as a type of testing where software modules,mapping, flows are integrated logically and tested as a group.

**Regression Testing:**

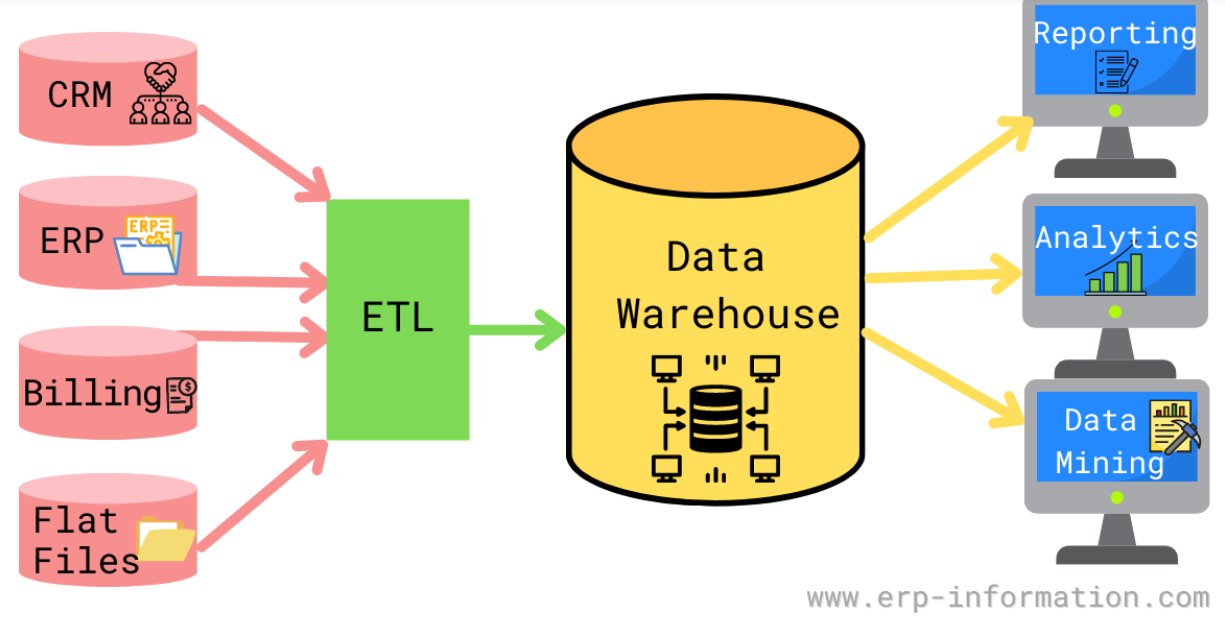
Regression testing is a type of software testing conducted after a code update to ensure that the update introduced no new bugs.

########################## Data Warehouse Concept ################################

**Data Warehouse**

A Data warehouse is a digital location to store data from many sources such as databases and files.

The Data warehouse consolidates data from many sources while ensuring data quality, consistency, and accuracy.



**Advantages of Data Warehousing:**

1. It makes data more accessible to businesses and organizations.
2. we have all data located at one place which makes easier to understand data, decision making etc.

**Disadvantage of Data warehouse**

1. As it contains huge data then maintaining those data is difficult
2. There is a great risk of accumulating irrelevant and useless data. Data loss and erasure are other potential issues.
3. Data is gathered from various sources in a data warehouse. Cleansing and transformation of the data are required. This could be a difficult task.

**What is data mining**

The primary goal of data mining is to discover hidden patterns, information, relationships in the data that can be used to make informed decisions or predictions.

**Example/Application of data mining**

1. In marketing, data mining can be used to identify customer segments and target marketing campaigns
2. In healthcare, it can be used to identify risk factors for diseases and develop personalized treatment plans.

**What is data modelling?**

Data modelling is the process of analyzing and defining all the different data types your business collects and produces, as well as the relationships between those bits of data. By using text, symbols, and diagrams, data modelling concepts create visual representations of data as it’s captured, stored, and used at your business. As your business determines how data is used and when, the data modelling process becomes an exercise in understanding and clarifying your data requirements.

**Types of data modelling**

We have below three type of data modelling-

1. Conceptual data modelling
2. Logical data modelling
3. Physical data modelling

**Schema in Data Warehouse**

Schema means the logical description of the entire database. It gives us a brief idea about the link between different database tables through keys (foreign key, pkey, composite key) and values.

A data warehouse also has a schema like that of a database.

In database modeling, we use the relational model schema.

Whereas in the data warehouse, we use modeling Star, Snowflake, and Galaxy schema.

**Key Concepts of Schemas**

1. Primary Key
2. Foreign Key
3. Dimensions

Dimensions are the column names in a dimension table

1. Measure

Quantitative attributes in the fact table. We perform calculations like average and sum on them. Example: No. of products, discount.

1. Fact table

A fact table contains a dimension key from the dimension table and measures.

**Types of schema**

In Data warehousing we can divide schema mainly in three types-

1. Schema
2. Snowflake Schema
3. Galaxy Schema

**Dimension table**

A dimension table is a table that contains information about the dimensions or categories of the data but not their amount or measurement, such as time, location, customer, product, etc.

A dimension table usually has a primary key that uniquely identifies each row, and one or more attributes that describe the dimension

**Example:**

Product table, which contains product name, productid, description but no measurement column.

**Fact table:**

A fact table is a table that contains the facts or measurement of the data, such as sales, revenue, costs etc

A fact table usually has a composite primary key that consists of foreign keys that reference the dimension tables, and one or more numeric or quantitative columns that store the facts.

A fact table is typically normalized, meaning that it contains only the essential data and avoids redundancy

**Example:**

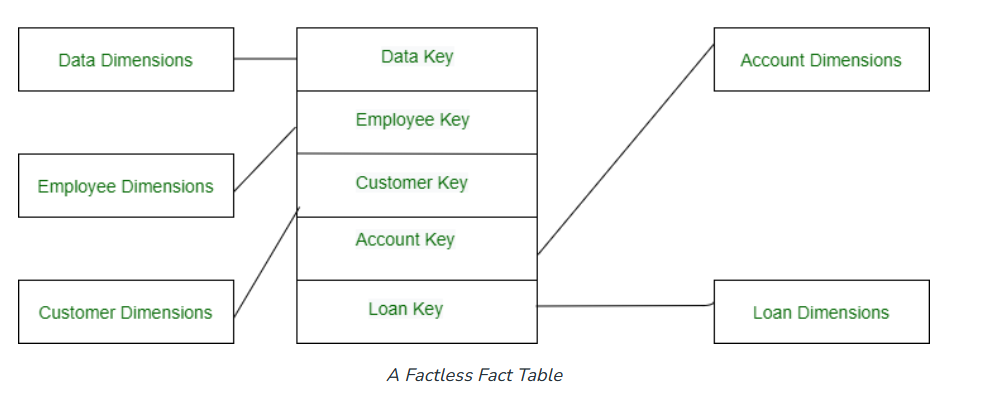
A sales fact table may have foreign keys such as date\_id, product\_id, customer\_id, etc., and facts such as quantity, price, discount, etc

**Fact less fact table**

The fact tables with no measurements linked with the transaction are known as factless fact tables. They are a straightforward grouping of dimensional keys that identifies transactions.

Factless tables simply mean the key available in the fact that no remedies are available. Factless fact tables are only used to establish relationships between elements of different dimensions

Example:



**How do dimension tables and fact tables relate?**

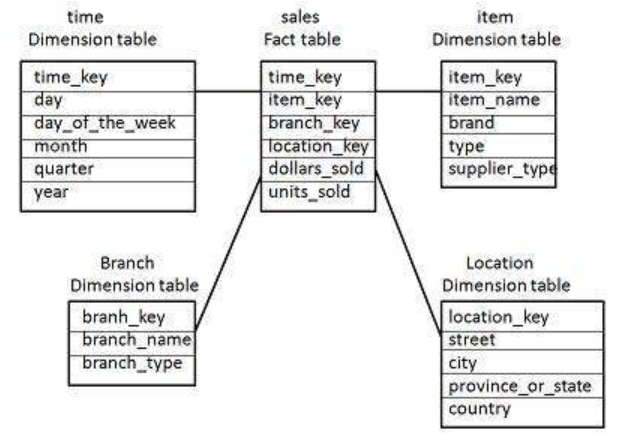
Dimension tables and fact tables are related by a star schema or a snowflake schema, which are common data modeling techniques for data warehousing.

**Start Schema**

Here are some of the basic points of star schema which are as follows:

1. This structure resembles a star and hence it is known as a star schema.
2. In a star schema, as the structure of a star, there is one fact table in the middle and a numbers of associated dimension tables.

**Example**



**Snowflake Schema**

1. Snowflake schema acts like an extended version of a star schema. There are additional dimensions added to Star schema. This schema is known as snowflake due to its structure.
2. There are multiple levels of relationships and child tables involved that have multiple parent tables. In snowflake schema, the affected tables are only the dimension tables and not the fact tables.
3. The difference between star and snowflake schema is that the dimensions of snowflake schema are maintained in such a way that they reduce the redundancy of data. The tables are easy to manage and maintain. They also save storage space.

**Example**

