AND operator:

* It must have multiple conditions.
* Both the conditions on either side of the AND operator must be True.

AND operator always has higher precedence value.

LIKE operator:

* %-stands for any number of characters.ex:

SELECT \* FROM Customers

WHERE Personal\_name LIKE ‘%U’ (for my name AISHU)

* \_-stands for just the exact number of characters ex:

SELECT \* FROM Customers

WHERE Personal\_name LIKE ‘\_\_\_\_U’ (for my name AISHU)

REGEXP(Regular Expression) operator :

This operator is used instead of LIKE operator

SELECT \* FROM Customers

WHERE Personal\_name LIKE ‘%U’ (for my name AISHU)

Can be written as

SELECT \* FROM Customers

WHERE Personal\_name REGEXP ‘U’ (for my name AISHU)

Here while using this operator we have ^ (Carrot operator ) at the beginning of the string while we use $ operator to denote the end of the string

We can check with two words as well such as ‘field|mac’

* where last\_name REGEXP '[gim]e' returns name with ge ie me it can also be written as
* where last\_name REGEXP 'l[dfm]'
* where last\_name REGEXP '[gim]e'

instead of typing letter by letter which Is very verbose we can write something like A to H in square bracket as [a-h]

1. [] to match any single character listed in the brackets
2. ^beginning of the string
3. $end of the string
4. | logical or
5. [abcd]
6. [a-f] a range of charcters to match from a to f

Get the customers whose

1. First names are ELKA or AMBUR
2. Last names end with EY OR ON
3. last names start with MY or contains SE
4. last names contain B followed by R or U

1. select \*

from customers

where first\_name REGEXP 'ELKA|AMBUR'

2. select \*

from customers

where last\_name REGEXP 'EY$|ON$'

3.select \*

from customers

where last\_name REGEXP '^MY|SE'

4. select \*

from customers

where last\_name REGEXP 'B[RU]'

Null operator:

The null value from the database can be extracted from the IS NULL operator ,

It can be sued as follows

SELECT \*

FROM customers

WHERE phone IS NULL

Query:

Get the orders that are not shipped yet:

select \*

from orders

where shipped\_date is null

can also be written as

select \*

from orders

where shipper\_id is null

ORDER BY:

In order to sort the data in database we use this, by default when we use this statement the data will be arranged in ascending order but in case we want it in descending order we can use DESC

select \*

from customers

order by state,first\_name

in case if you want the data in ascending order we write like this

but in case of ascending order we write the following

select \*

from customers

order by state DESC, first\_name DESC

LIMIT clause:

Limits the data which is supposed to be returned to the user

select order\_id, product\_id,quantity,unit\_price\*quantity as totalprice

from order\_items

limit 1

Whenever we write LIMIT 6,3 it means that skip till 6th and print the next 3

* top three loyal cutomers

select \*

from customers

order by points desc

limit 3

JOINS:

In SQL we have two types of JOINS i.e

* inner join :  This join is based on a logical relationship (or a common field) between the tables and is used to retrieve data that appears in both tables. A intersection B
* outer join

ex: Select \*

from orders(TABLE)

join customers(TABLE) ON on orders.customer\_id(TABLE.COULMN)= customers.customer\_id(TABLE.COLUMN)

INNER JOIN and OUTER JOIN :

Technically when we write normal JOIN keyword it means that its an inner join and When it comes to OUTER JOIN we write OUTER JOIN .

**Always before execution of query make sure that the database is selected before hand.**

Text, letter

Description automatically generated

The table which is boxed its whole table will be displayed irrespective of the condition being satisfied or not in the where clause.

If we are using INNER Keyword or just JOIN then its an INNER JOIN but when it comes to an OUTER Keyword we may use OUTER or just LEFT and RIGHT followed by JOIN.

After the SELECT statement in whichever order you put the column name in the same order the output will be displayed.

Solution for the question 7 – Outer joins :

SELECT

p.product\_id,p.name ,o.quantity

from products p

left join order\_items o

on p.product\_id = o.product\_id

order by p.product\_ids

It is not always necessary to have identical column names in the joins it can vary from time to time as real time data will not always have similar database.

select

o.order\_id,o.order\_date, c.first\_name as customer, sh.name as shipper, os.name as status

from orders o

join customers c

on c.customer\_id= o.customer\_id

left join shippers sh

on o.shipper\_id= sh.shipper\_id

join order\_statuses os

on o.status = os.order\_status\_id

order by order\_id

Self outerjoin:

Either select a particular database or write **USE DATABASE TO BE SELECTED**;

We take an employee table and alaiyaze it with some new name and join it with the same take , this is called self outerjoin.

USING CLAUSE:

As the joins become complex it becomes more difficult to analyze the problem, there is some useful feature to write code that being USING CLAUSE.

If the column name is exactly same like **ON O.CUSTOMER\_ID = C.CUSTOMER\_ID** we can replace the ON clause with **USING(CUSTOMER\_ID),** This using keyword can be used for both inner and outer joins, the condition being the column name must be identical.

It reduces the clumsiness of the code.

use sql\_invoicing;

select

p.date, c.name as client,p.amount,pm.name

from payments p

on p.client\_id = c.client\_id

using(client\_id)

left join payment\_methods pm

on p.payment\_method=pm.payment\_method\_id

can be written as

use sql\_invoicing;

select

p.date, c.name as client,p.amount,pm.name

from payments p

join clients c

using(client\_id)

left join payment\_methods pm

on p.payment\_method=pm.payment\_method\_id

**Natural join:**

**This method is usually not recommended as it gives different results each time apart from that it itself takes the common cioolumns. They produce unexpected results.**

Cross join:

Each and every column in first table is matched with each and every column in table 2. There are two types in declaring them those are as follows:

* implicit
* Explicit

Typing multiple tables in the from clause can also be a method of implicitly calling **CROSS JOIN.**

Unions:

Like we combinecloumns from multiple table in sql we combine multiple rows as well and this is extremely powerful. With union we can combine results from multiple queries.whatever is in the first query its column name will be the first query.

use sql\_store;

select customer\_id,first\_name,points,

'bronze' as type

from customers

where points<2000

union

select

customer\_id,

first\_name,

points,

'silver' as type

from customers

where points>=2000 and points<= 3000

union

select

customer\_id,

first\_name,

points,

'gold' as type

from customers

where points>3000

order by first\_name

Insert:

Inserting a row into a table

INSERT INTO column\_name(VALUES ACCORDING TO THE COULMNS)

VALUES(VALUES OF EVERY COLUMN IS SUPPLIED ACCORDINGLY)

To insert multiple rows in one go:

INSERT INTO column\_name(NAME)

VALUES(‘AISHWARYA1’),

(‘Aishwarya2’),

(‘Aishwarya3’)

Inserting heirarchial rows:

LAST\_INSERT\_ID= This is the function which is built in , basically a function is a piece of code that is called whenever it is necessary in the code. Example being that TV has few builtin function of which power on and power off are functions we change it whenever we want to change.

Before inserting this function the row which is being inserted should end with ; and

**Select last\_insert\_id()**

insert into orders(customer\_id,order\_date,status)

values(1,'2019-01-12',1);

insert into order\_items

values

( last\_insert\_id(),1,1,2.95),

( last\_insert\_id(),2,1,2.95)

By seeing the orders table we inserted the first row later we saw the order\_items table and made use of the Table.

**Copying the complete TABLE:**

CREATE TABLE AISHWARYA\_RAMA AS

SELECT \* FROM ORDERS

Subquery: It Is A Select Statement Which Is Part Of Another Sql Statement

INSERT INTO aishwarya\_rama

SELECT \*

FROM ORDERS

WHERE ORDER\_DATE<'2019-01-01'

Excersize:

USE SQL\_INVOICING;

create table invoice\_archive as

SELECT

i.invoice\_id,

i.number,

c.name as client\_name,

i.invoice\_total,i.payment\_total,i.invoice\_date,i.due\_date,i.payment\_date

FROM INVOICES I

left JOIN CLIENTS C

ON I.client\_id = c.client\_id

where payment\_date is not null

Updating data in the records:

We use update statement inorder to update one or more rows in the table

**UPDATE INVOICES**

**SET PAYMENT\_TOTAL =10,PAYMENT\_DATE = ‘2019-03-01’**

**WHERE INVOICE\_ID =1**

**USE SQL\_STORE;**

**UPDATE CUSTOMERS**

**SET POINTS=POINTS+50**

**WHERE BIRTH\_DATE < '1990-01-01'**

Using subqueries in update statement:

UPDATE ORDERS

SET COMMENTS =’GOLD CUSTOMER’

WHERE CUSTOMER\_ID IN

(SELECT CUSTOMER\_ID

FROM CUSTOMERS

WHERE POINTS >3000)

Delete Statement :

DELETE FROM statement is used to DELETE Records from Table, after which we specify with the WHERE clause where we specify where we are deleting the statement, we should be very cautious while we should be very cautious cause it might delete the entire table.

📚 SQL Cheat Sheet 🎉  
  
🔍 Searching Data:  
SELECT 📊 - Retrieve specific columns from a table  
FROM 🗄️ - Specify the table to retrieve data from  
WHERE 🎯 - Apply conditions to filter data  
LIKE 🔍 - Perform pattern matching  
ORDER BY 🔢 - Sort data in ascending or descending order  
LIMIT 🔒 - Limit the number of results  
  
🔢 Manipulating Data:  
INSERT ➕ - Add new rows into a table  
UPDATE ✏️ - Modify existing data in a table  
DELETE ❌ - Remove data from a table  
🔁 Joining Tables:  
  
INNER JOIN ➕🗄️ - Combine rows from different tables based on matching values  
LEFT JOIN ➕🗄️ - Retrieve all records from the left table and matching records from the right table  
RIGHT JOIN ➕🗄️ - Retrieve all records from the right table and matching records from the left table  
  
📂 Grouping and Aggregating Data:  
GROUP BY 📚 - Group rows based on a column  
HAVING 🎛️ - Apply conditions to grouped data  
COUNT 🔢 - Count the number of rows  
SUM ➕ - Calculate the sum of a column  
AVG 📈 - Calculate the average of a column  
MAX ⬆️ - Find the maximum value in a column  
MIN ⬇️ - Find the minimum value in a column  
  
🔀 Modifying Table Structure:  
CREATE TABLE ➕🗄️ - Create a new table  
ALTER TABLE ✏️🗄️ - Modify the structure of an existing table  
DROP TABLE ❌🗄️ - Delete a table from the database  
📊 Database Management:  
CREATE DATABASE ➕🗄️ - Create a new database  
USE 🗄️ - Select a specific database to work with  
SHOW DATABASES 📚 - Display all available databases  
DESCRIBE ✏️🗄️ - Show the structure of a table  
  
🔒 Transaction Control:  
COMMIT ✅ - Save changes permanently  
ROLLBACK 🔄 - Undo changes made during the current transaction  
SAVEPOINT ⏳ - Set a point in a transaction to roll back to later

Aggregate Functions:

In sql we have some built In functions which are applied for few set of values these are called aggregate functions. Ex: Max(),min(),avg(),sum(),count(), like any other function in nay other programming language we make use of ‘()’ after the usage of aggregate function.

use sql\_invoicing;

select

'First Half of 2019' as Date\_Range,

sum(invoice\_total) as Total\_Sales,

sum(payment\_total) as Total\_payments,

sum(invoice\_total - payment\_total) as What\_we\_expect

from invoices

where invoice\_date between '2019-01-01' and '2019-06-30'

union

select

'Second Half of 2019' as Date\_Range,

sum(invoice\_total) as Total\_Sales,

sum(payment\_total) as Total\_payments,

sum(invoice\_total - payment\_total) as What\_we\_expect

from invoices

where invoice\_date between '2019-07-01' and '2019-12-30'

union

select

'Total' as Date\_Range,

sum(invoice\_total) as Total\_Sales,

sum(payment\_total) as Total\_payments,

sum(invoice\_total - payment\_total) as What\_we\_expect

from invoices

Group by :

Here we group the values based on their respective categories.

use sql\_invoicing;

select

date, payment\_method , sum(amount) as total\_payments

from payments p

join payment\_methods pm

on p.payment\_method = pm.payment\_method\_id

where date > '2019-01-01'

group by date

order by date asc

Having Clause:

We use the WHERE clause before grouping the rows and we use the HAVING clause after grouping the rows and even in this we can type out one or more conditions. In where clause we can access the columns even if its not mentioned in the select clause cloumns, but in case of where clause its not like that it has to be mentioned else it cant be used.

Rollup operator is used to summarize data like summation or so on but this function is only available in MySQL.

FT and RIGHT JOIN