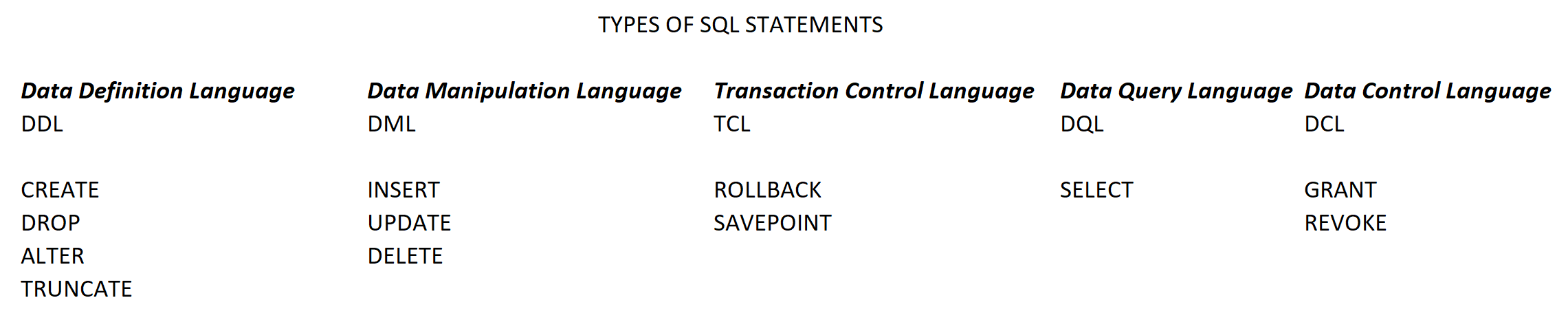
SQL – CONCEPTS & EXAMPLES

## # Type of SQL Statements

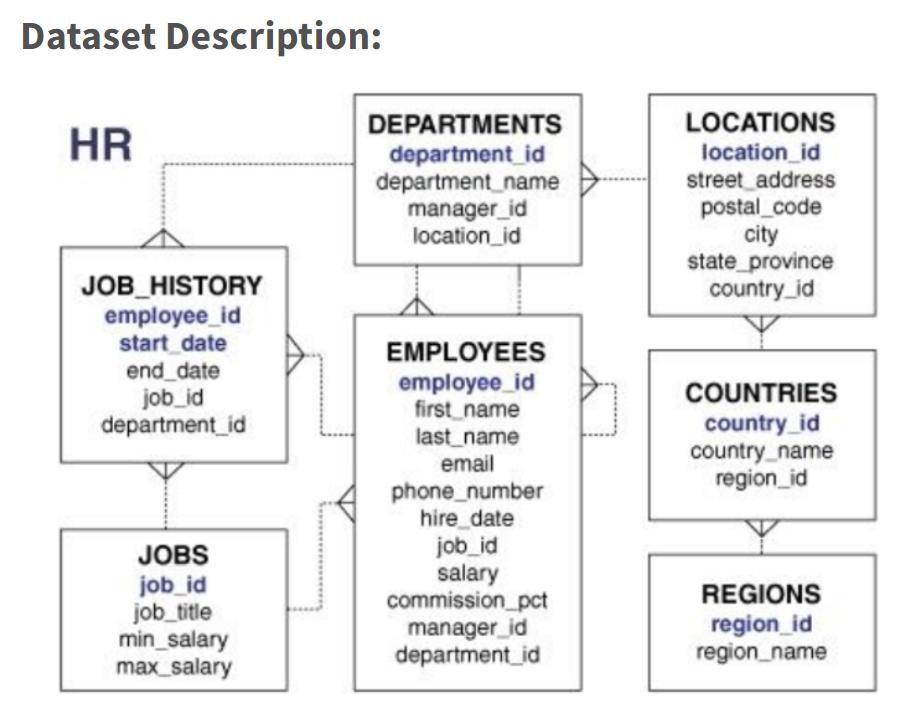


## # Points:

1. What is a Database & why we need it ? How many databases - Today more than 343 databases are out there in the tech world which is a very huge number

* Organized storage – inserting and updating , saving will be very fast
* Rules for storing data - No garbage

1. Data – warehouse
2. SQL – structured query language
3. Data - Tables
4. Normalization



**What is Normalization?**

1. Normalization is the process of organizing the data in the database.
2. Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
3. Normalization divides the larger table into smaller and links them using relationships.
4. The normal form is used to reduce redundancy from the database table.

**Advantages of Normalization**

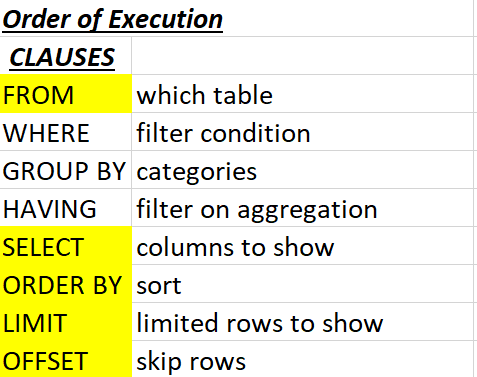
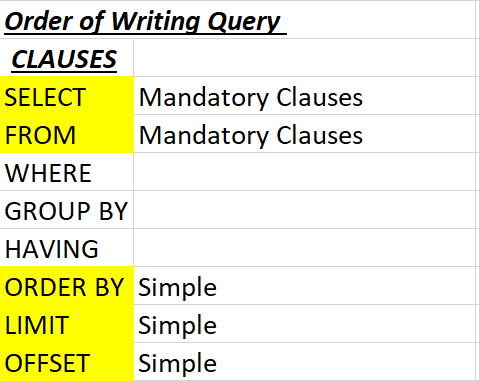
* Normalization helps to minimize data redundancy.
* Greater overall database organization.
* Data consistency within the database.
* Much more flexible database design.
* Enforces the concept of relational integrity.

**Disadvantages of Normalization**

* You cannot start building the database before knowing what the user needs.
* The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
* It is very time-consuming and difficult to normalize relations of a higher degree.
* Careless decomposition may lead to a bad database design, leading to serious problems.

1. Primary key
2. Foreign key
3. Composite primary key
4. Relationships
   1. One to one – Theoretical
   2. Many to many – Theoretical
   3. One to many – Practical
   4. Many to one – same as above

## # Concept 1 – OOE & OOWQ

# Q1 : Display all details for all the products

select \*

FROM `farmers\_market.product`;

# -----------------

# Q2 : Display prod id and prod name for all products

select product\_id, product\_name

from `farmers\_market.product`; # 23 records

# Extras

select product\_id, product\_name

from `farmers\_market.product`

# order by product\_id;

order by product\_id desc

limit 10 offset 5;

# -----------------

# Q3 : Explore vendor\_booth\_assignments.

select \*

from `farmers\_market.vendor\_booth\_assignments` ;

# List down 10 rows of farmer’s market vendor booth assignments,

# displaying the market date, vendor ID, and booth number

# from the vendor\_booth\_assignments table.

select market\_date, vendor\_id, booth\_number

from `farmers\_market.vendor\_booth\_assignments`

limit 10 ;

# -----------------

# Q4 : Sort the vendor booth assignment data by Date

select \*

from `farmers\_market.vendor\_booth\_assignments` ;

select \*

from `farmers\_market.vendor\_booth\_assignments`

order by market\_date;

# -----------------

# Q5 : Sort the customer purchases data by date. Most recent orders on top.

select \*

from `farmers\_market.customer\_purchases`

order by market\_date desc ;

select \*

from `farmers\_market.customer\_purchases`

order by market\_date

limit 20;

# most recent use descending "desc"

select \*

from `farmers\_market.customer\_purchases`

order by market\_date desc

limit 20;

# -----------------

# Q6 : Sort the vendor booth assignment data by Date in decreasing order and Vendor in ascending order

select \*

from `farmers\_market.vendor\_booth\_assignments`

order by market\_date desc, vendor\_id ;

select \*

from `farmers\_market.vendor\_booth\_assignments`

order by vendor\_id , market\_date desc ;

-- changing order of columns in order by will change the result --

# -----------------

# Q7 :  Fetch the details of 3rd most recent order from top (desc by date)

select \*

from `farmers\_market.customer\_purchases` ;

select \*

from `farmers\_market.customer\_purchases`

order by market\_date desc

limit 3;

# give me 1 transaction after skipping the first 2 rows

select \*

from `farmers\_market.customer\_purchases`

order by market\_date desc

limit 1 offset 2 ;

# ----------------- check point 1

# Q7A: Find employee with highest salary (no 2 ppl have same salary)

select \*

from `employees.employees` ;

select \*

from `employees.employees`

order by salary desc

limit 1 ;

# Extras

# Find employee with 3rd highest salary

select \*

from `employees.employees`

order by salary desc

limit 1 offset 2 ;

#----------------------

## # Concept 2 – Inline Calculations

/\*

# Inline Calculations - work on rows

# Given columns  empid, salary, commission

# Find total earnings of each employee.

# Ask is to create a new column TotalEarnings

select salary + Commission as (alias) total\_earning

from employee ;

# Where do we create a new column in a sql query ?

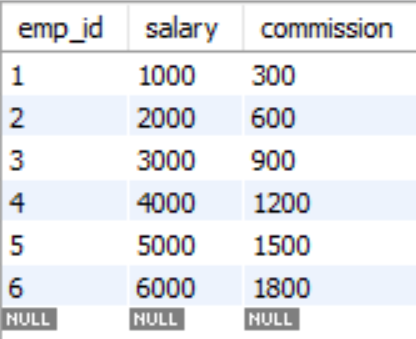
# Select clause

inline calculations dont impact the underlying table

See My SQL 'employee\_table' not 'employees'

\*/

select \* from employee\_table.emp ;

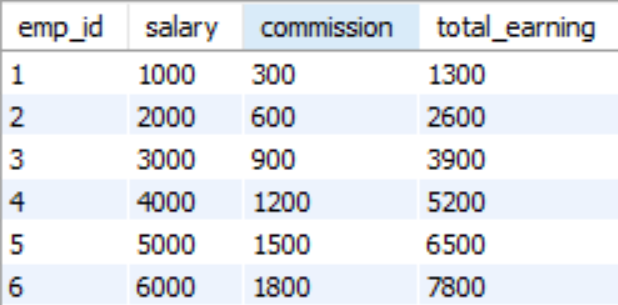


select

emp\_id, salary, commission,

(salary + commission) total\_earning

from employee\_table.emp ;



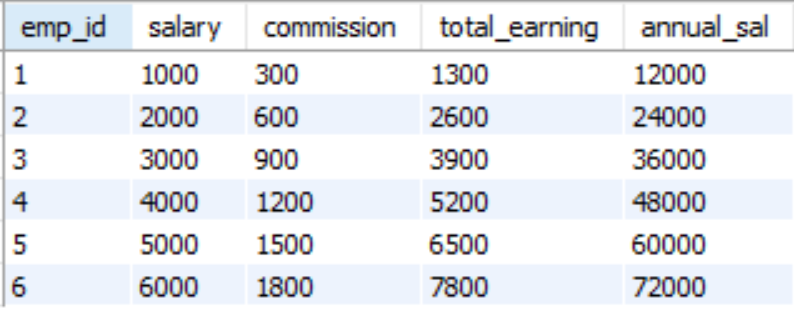
select

emp\_id, salary, commission,

(salary + commission) total\_earning,

(salary \* 12) annual\_sal

from employee\_table.emp



#----------------------

# Q8 : From the customer purchases table, query the total amount that the customer has paid.

select \*

from `farmers\_market.customer\_purchases` ;

select quantity \* cost\_to\_customer\_per\_qty as total\_amt

from `farmers\_market.customer\_purchases` ;

select \*, quantity \* cost\_to\_customer\_per\_qty total\_amt

from `farmers\_market.customer\_purchases` ; # as keyword is optional

#----------------------

# Q9 : Display quantity columns rounded to 1 decimal places.

select

quantity,

round(quantity,1) as quantity\_rounded

from `farmers\_market.customer\_purchases` ;

#----------------------

# Q10 : Display total cost rounded to 2 decimal places.

select

cost\_to\_customer\_per\_qty,

round(cost\_to\_customer\_per\_qty, 2) as cost\_rounded

from `farmers\_market.customer\_purchases` ;

#----------------------

# Q11 : Display total amount rounded to 2 decimal places.

select

round(quantity \* cost\_to\_customer\_per\_qty,2) as total\_amt

from `farmers\_market.customer\_purchases` ;

select

\*,

round(quantity \* cost\_to\_customer\_per\_qty,2) as total\_amt

from `farmers\_market.customer\_purchases` ;

select

\*,

round(quantity \* cost\_to\_customer\_per\_qty,1) as total\_amt

from `farmers\_market.customer\_purchases` ;

#----------------------

# Concept 3 – Mathematical functions

select round(7.33,1) as round,

ceil(7.33) as ceil,

floor(7.33) as floor; # See the syntax

#----------------------

# Q12 : Display a new column with full name of customers.

select \*

from `farmers\_market.customer` ;

select  \*, concat(customer\_first\_name," ", customer\_last\_name) as cust\_full\_name

from `farmers\_market.customer` ;

select  customer\_first\_name, customer\_last\_name,  concat(customer\_first\_name," ", customer\_last\_name) as cust\_full\_name

from `farmers\_market.customer`

limit 10 ;

# not all DBMS's understand double quotes (""), use single quotes (''), mysql will fail

select  customer\_first\_name, customer\_last\_name,  concat(customer\_first\_name,' ', customer\_last\_name) as cust\_full\_name

from `farmers\_market.customer`

limit 10 ;

#----------------------

# Q13 : Display all full names in caps.

select  customer\_first\_name, customer\_last\_name,  upper(concat(customer\_first\_name," ", customer\_last\_name)) as cust\_full\_name

from `farmers\_market.customer`

limit 10 ;

select  customer\_first\_name, customer\_last\_name,  lower(concat(customer\_first\_name," ", customer\_last\_name)) as cust\_full\_name

from `farmers\_market.customer`

limit 10 ;

#----------------------

## # Concept 3 – Where clause

# Q14 : Extract all product names that are part of product category 1.

select \*

from `farmers\_market.product`;

select product\_name

from `farmers\_market.product`

where product\_category\_id = 1

limit 10;

select product\_name,product\_category\_id

from `farmers\_market.product`

where product\_category\_id = 1

limit 10;

#----------------------

# Q15 : Extract all products that have size medium.

select \*

from `farmers\_market.product`;

select product\_name, product\_size

from `farmers\_market.product`

where product\_size = 'medium'

limit 10;

select \*

from `farmers\_market.product`

where product\_size = 'medium'

limit 10;

select product\_name, product\_size, product\_id

from `farmers\_market.product`

where product\_size = 'medium'

limit 10;

#----------------------

# Q16 : Extract all products that have size small and product category 2.

select \*

from `farmers\_market.product`;

select \*

from `farmers\_market.product`

where product\_size = 'small' and product\_category\_id = 2

limit 10;

# let's do product category id = 1

select \*

from `farmers\_market.product`

where product\_size = 'small' and product\_category\_id = 1

limit 10;

#----------------------

# Q17 : Extract all products that have product category 2 or product name carrots.

select \*

from `farmers\_market.product`

where product\_category\_id = 1 or product\_name = 'Carrots'

limit 10;

select \*

from `farmers\_market.product`

where product\_category\_id = 2 or product\_name = 'Carrots'

limit 10;

select \*

from `farmers\_market.product`

where product\_category\_id = 2 and product\_name = 'Carrots'

limit 10;

select \*

from `farmers\_market.product`

where product\_category\_id = 1 and product\_name = 'Carrots'

limit 10;

select \*

from `farmers\_market.product`

where product\_name = 'Carrots'

limit 10;

select \*

from `farmers\_market.product`;

#----------------------

# Q18 : Extract all product id's greater than 3

select \*

from `farmers\_market.product`

where product\_id > 3

limit 20 ;

select \*

from `farmers\_market.product`

where product\_id > 3

order by product\_id

limit 20 ;

# Q19 : Find out what booths vendor no. 3 was assigned to, on or before April 20, 2019.

select \*

from `farmers\_market.vendor\_booth\_assignments`

where vendor\_id = 3 and market\_date <= '2019-04-20' # less than or equal to this date.

limit 20 ;

#----------------------

## # Concept 4 – Keywords and Operators

# Q20 : Extract products with id's between 3 and 8 (both exclusive, no 3 and 8)

select \*

from `farmers\_market.product`

where product\_id >3 and product\_id <8

order by product\_id

limit 20; # 3 and 8 exclusive

select \*

from `farmers\_market.product`

where product\_id >=3 and product\_id <=8

order by product\_id

limit 20; # 3 and 8 exclusive

select \*

from `farmers\_market.product`

where product\_id between 3 and 8

order by product\_id

limit 20; # 3 and 8 inclusive

select \*

from `farmers\_market.product`

where product\_id between 4 and 7 # brute force, if 3.5, then it will fail.

order by product\_id

limit 20;

#----------------------

# Q21 : Find the booth assignments for vendor 7 for any market date that occured between April 3,2019 and may 16,2019 including both the dates.

select vendor\_id, market\_date

from `farmers\_market.vendor\_booth\_assignments`

where vendor\_id = 7 and

market\_date >='2019-04-03' and market\_date <='2019-05-16'

order by market\_date;

--- using between clause

select \*

from `farmers\_market.vendor\_booth\_assignments`

where vendor\_id = 7 and market\_date between '2019-04-03' and '2019-05-16'

order by market\_date;

# where market\_date = '2019-05-16';

select vendor\_id, market\_date

from `farmers\_market.vendor\_booth\_assignments`

where vendor\_id = 7 and market\_date between '2019-04-03' and '2019-05-16'

order by market\_date;

# when inclusive use between, else use greater than and lesser than.

#----------------------

# Q22 : Select products which are small, medium or large

select \*

from `farmers\_market.product`

where product\_size = 'small' or product\_size = 'medium' or product\_size = 'large' ;

--- in keyword ---

select \*

from `farmers\_market.product`

where product\_size in ('small', 'medium', 'large') # multiple 'or' conditions

order by product\_id ;

#----------------------

# Q23 : Find all the products from the product table without sizes

select \*

from `farmers\_market.product` ;

select \*

from `farmers\_market.product`

where product\_size = ' '; # Works fine but does not give null values

select \*

from `farmers\_market.product`

where product\_size is null; # Gives null values

select \*

from `farmers\_market.product`

where product\_size = ' ' or product\_size is null ; # Works fine

select \*

from `farmers\_market.product`

where product\_size = ' ' and product\_size is null ; # Space and null wont be in same column, so no data

select \*

from `farmers\_market.product`;

# where product\_size is nothing # Syntax error: Expected keyword FALSE or keyword NULL or keyword TRUE or keyword UNKNOWN but got identifier "nothing" at [3:23]

select \*

from `farmers\_market.product`

where product\_size is null ; # Works fine but does not give blank values

select \*

from `farmers\_market.product`

where product\_size is null or product\_size = ' '; # Boom 'or' keyword does the trick not 'and' keyword

select \*

from `farmers\_market.product`

where product\_size = ' ' or product\_size is null; # Boom 'or' keyword does the trick not 'and' keyword

select \*

from `farmers\_market.product`

where product\_size = ' ' and product\_size is null; # There is no data to display.

#----------------------

# Q24 : Select products which are not small, medium or large.

select \*

from `farmers\_market.product`

where product\_size <> ('small', 'medium', 'large') ;

# No matching signature for operator != for argument types: STRING, STRUCT<STRING, STRING, STRING>. Supported signature: ANY != ANY at [3:7]

select \*

from `farmers\_market.product`

where product\_size not in ('small', 'medium', 'large') ;

#----------------------

# Q25 : Show products for which size is not missing.

select \*

from `farmers\_market.product`

where product\_size is not null ; # everything except null product size

select \*

from `farmers\_market.product`

where product\_size is not null and product\_size != ' ' ;

# everything except null and blank product size

select \*

from `farmers\_market.product`

where product\_size != ' ' ;

select \*

from `farmers\_market.product`

where product\_size is null ;

#----------------------

# Q26 : Replace missing product size with 'missing\_value'.

--- replace null values with required strings ---

--- missing value handling ---

select product\_id,product\_size, ifnull(product\_size, 'hahaha') as new\_product\_size

from `farmers\_market.product` ; # new column

--- Coalesce ---

select product\_id,product\_size, coalesce(product\_size, 'hahaha') as new\_product\_size

from `farmers\_market.product` ;

# read about coalesce function

#----------------------

# Q26A : Replace a black space with required string

select \*

from `farmers\_market.product` ;

select replace(product\_size, ' ', 'hehehehe')

from `farmers\_market.product` ;

# now we have another problem, where ever there are spaces the string is replaced.

# lets try this

# Ask pratyusha, how to update a blank cell in a column with required string.

select replace(product\_size, '', 'hehehehe')

from `farmers\_market.product` ;

# not working

select replace(trim(product\_size), ' ', 'hehehehe')

from `farmers\_market.product` ;

# not working

select \*, replace(product\_size, ' ', 'hehehehe') as blahblah

from `farmers\_market.product` ;

where product\_size = ' ' # sort of half worked,

# but question remains :  if there is a blankspace in a column, can we replace that with a required string ?

# if you want to eleminate blanks use trim function

# Pattern matching - like keyword - wildcard % - any characters of any length

#----------------------

# Q27 : Find all customers whose first name begins with J

select \*

from `farmers\_market.customer`

where customer\_first\_name like 'J%' ;

# its case sensitive (Jane, search with cap J not small j)

select \*

from `farmers\_market.customer`

where lower(customer\_first\_name) like 'j%' ; # its case sensitive (Jane, search with cap J not small j)

# like 'J%t' - first letter shd be J and last letter shd be t with any number of characters in between. % means 0 or more characters

# like 'J\_t' - first letter shd be J and last letter shd be t with one and exactly letter character in between which can be anything.

select \*

from `farmers\_market.customer`

where customer\_first\_name like 'J%' or customer\_first\_name like 'j%' ;

select \*

from `farmers\_market.customer`

where lower(customer\_first\_name) like 'j%' ;

#----------------------

# Q 28 : Find all customer first names that end with e

select \*

from `farmers\_market.customer`

where customer\_first\_name like '%e';

select \*

from `farmers\_market.customer`

where lower(customer\_first\_name) like '%e'; # also known as text cleaning

#----------------------

# Q 29 : Get customers with first name starting with J and last name end with e, and sort the results by first and last name respectively.

select \*

from `farmers\_market.customer`

where customer\_first\_name like 'J%' and customer\_last\_name like '%e'

order by customer\_first\_name, customer\_last\_name ;

#----------------------

# Q 30 : For every vendor identify the ones who deal with 'fresh' produce and those which dont.

select \*

from `farmers\_market.vendor` ;

select \*

from `farmers\_market.vendor`

where lower(vendor\_type) like '%fresh%' ;

# we missed the 'and those who dont'. Dont Filter data , categorize data.

select

\*, if(lower(vendor\_type) like '%fresh%','Fresh','NOT Fresh') as ResultColumn

from `farmers\_market.vendor`;

select \*,

case

  when lower(vendor\_type) like '%fresh%'

  then 'Yes'

  else 'No'

end as Fresh\_Produce # single word no spaces

from `farmers\_market.vendor` ;

#----------------------

## # Concept 5 – Sub-query – Query inside a query

# Q31 : Get all the details of the products where product\_category contains fresh

# We want to query products table but condition is on product category table

# Does product table have product category column in it ?

select \*

from `farmers\_market.product`;

select \*

from `farmers\_market.product\_category` ; # its product category name that we are after.

# Inner query

select product\_category\_id

# prod cat id not name, coz its the common column name in both the tables

from `farmers\_market.product\_category`

where lower(product\_category\_name) like '%fresh%' ; # inner query returned 1,4,6

# Outer query

select \*

from `farmers\_market.product`

where product\_category\_id in

(

) ;

# Full Query

select \*

from `farmers\_market.product`

where product\_category\_id in

(

  select product\_category\_id

from `farmers\_market.product\_category`

where lower(product\_category\_name) like '%fresh%'

) ;

# Full query does not have 4 prod cat id why ? Coz outer query does not have 4 in its table, it has only 1 and 6.

#----------------------

# Q32 : Give details of all customers who bought products from vendor 7 on '2019-04-03'

# Query 1

select \*

from `farmers\_market.customer`; # customer\_id

# Query 2

select \*

from `farmers\_market.customer\_purchases` ; # vendor\_id, customer\_id

# Inner Query

select customer\_id

from `farmers\_market.customer\_purchases`

where vendor\_id = 7  and market\_date = '2019-04-03'

# Full Query

select \*

from `farmers\_market.customer`

where customer\_id in

(

select customer\_id

from `farmers\_market.customer\_purchases`

where vendor\_id = 7  and market\_date = '2019-04-03'

);

# Trying to get 2 columns out of the inner query

# Full Query

select \*

from `farmers\_market.customer`

where customer\_id in

(

select customer\_id, vendor\_id # Subquery of type IN must have only one output column at [5:1]

from `farmers\_market.customer\_purchases`

where vendor\_id = 7  and market\_date = '2019-04-03'

);

#----------------------

## # Rules

# Rules for Sub-query

# 1. Columns from single table, condition for 1 or more other tables

# 2. Cheat the where clause by writing aggregation function in inner query

# 3. Only one column output from inner query, Error : Subquery of type IN must have only one output column at [5:1]

#----------------------

/\*

# JOINS : used to merge 2 or more tables together. Vertical Merging. Columns will increase.

1. Inner Join  - Join             - Display results only when there is a match

2. Left Join - left outer join    - Include customers even if they did not purchase anything

3. Right Join - right outer join  - Include customers even if they did not purchase anything, opposite to left join.

4. Full Outer Join - outer join   - Include all results, match and dont match

\*/

# Sub-query Vs Join

# Subquery takes more processing power than joins, Joins are much faster.

# Subquery only gets a single output column , # Subquery of type IN must have only one output column at [5:1]

#----------------------

## # Co-related sub-queries – not required, instead use joins.

## # Proof

# order by executes after select clause

select product\_id, product\_name

from `farmers\_market.product`

order by product\_name ;

select product\_id, count(product\_name) prod\_count

from `farmers\_market.product`

order by prod\_count ; # Error :SELECT list expression references column product\_id which is neither grouped nor aggregated at [1:8]

select count(product\_id) alias, count(product\_name) prod\_count

from `farmers\_market.product`

order by prod\_count ;

select count(product\_name) prod\_count

from `farmers\_market.product`

order by prod\_count ;

select product\_id,product\_id \* 2 as new\_prod\_id

from `farmers\_market.product`

order by new\_prod\_id ;

#----------------------

## # Concept 6 –Joins – From clause

SYNTAX :

SELECT [Cols]

FROM [Table 1]

[type of join] [Table 2]

ON table1.column = table2.column

QUE 1 : Some condition we can write in ON itself, not need to create a where condition and write in it.

\*/

#----------------------

# Q33 : List all products along with their product category name.

# Inspect the first table

select \*

from `farmers\_market.product` ; # product\_id, product\_name, product\_size, product\_category\_id, product\_qty\_type # prod cat 4 not available here

# Inspect the second table

select \*

from `farmers\_market.product\_category` ; # product\_category\_id, product\_category\_name # prod cat 4 available here

# Full Query

select \*

from

`farmers\_market.product` as p

JOIN

`farmers\_market.product\_category` as pc

ON p.product\_category\_id = pc.product\_category\_id ;

# Easy way

select \*

from

`farmers\_market.product` as p

JOIN

`farmers\_market.product\_category` as pc

using(product\_category\_id) ;

# Inner Join # result 23 rows

select \*

from

`farmers\_market.product` as p  INNER JOIN `farmers\_market.product\_category` as pc

ON p.product\_category\_id = pc.product\_category\_id

order by p.product\_id ;

# Left Join # result 23 rows

select \*

from

`farmers\_market.product` as p  LEFT JOIN `farmers\_market.product\_category` as pc

ON p.product\_category\_id = pc.product\_category\_id

order by p.product\_id ;

# Right Join # result 24 rows

select \*

from

`farmers\_market.product` as p  RIGHT JOIN `farmers\_market.product\_category` as pc

ON p.product\_category\_id = pc.product\_category\_id

order by p.product\_id ;

#----------------------

# Q34 : List all products along with their product category name.

# Right Join # result 24 rows - include product categories that don't have any products associated to them.

select

p.product\_name,

p.product\_name,

p.product\_category\_id,

pc.product\_category\_id,

pc.product\_category\_name

from

`farmers\_market.product` as p  RIGHT JOIN `farmers\_market.product\_category` as pc

ON p.product\_category\_id = pc.product\_category\_id ;

# Left Join # result 23 rows - include product id's that don't have any product\_category\_names associated to them.

select p.product\_name, p.product\_name, p.product\_category\_id, pc.product\_category\_id, pc.product\_category\_name

from

`farmers\_market.product` as p  LEFT JOIN `farmers\_market.product\_category` as pc

ON p.product\_category\_id = pc.product\_category\_id ;

# List all product categories that do not have a product associated with it.

SELECT

p.product\_id, p.product\_name, pc.product\_category\_id, pc.product\_category\_name

FROM

`farmers\_market.product` as p RIGHT JOIN `farmers\_market.product\_category`as pc

ON p.product\_category\_id = pc.product\_category\_id ;

#----------------------

# Q35 : List all product cat names that dont have a match in the product table

/\*

Left table

1. p.product\_id : null

2. p.product\_name : null

Right table

3. pc.product\_category\_name : Freshly Prepared Food

\*/

# If you need only the column that dont match,

# its like teacher in a class asking everybody who are absent raise their hands so i can count those who are present.

SELECT

p.product\_id,

p.product\_name,

p.product\_category\_id as t1 ,

pc.product\_category\_id as t2 ,

pc.product\_category\_name

FROM

`farmers\_market.product` as p

RIGHT JOIN `farmers\_market.product\_category`as pc

ON p.product\_category\_id = pc.product\_category\_id

WHERE p.product\_category\_id is null ;

SELECT

p.product\_id,

p.product\_name,

p.product\_category\_id as t1 ,

pc.product\_category\_id as t2 ,

pc.product\_category\_name

FROM

`farmers\_market.product` as p

RIGHT JOIN `farmers\_market.product\_category`as pc

ON p.product\_category\_id = pc.product\_category\_id # we can write the where condition here also

WHERE p.product\_category\_id != pc.product\_category\_id ; # There is no data to display. (<> same result)

#----------------------

# Q36 : List all customers who have placed at least 1 order.

# Table 1

select \*

from `farmers\_market.customer` ; # customer\_id, customer\_first\_name, customer\_last\_name, customer\_zip

# Table 2

select \*

from `farmers\_market.customer\_purchases` ; # product\_id, vendor\_id, market\_date, customer\_id, quantity, cost\_to\_customer\_per\_qty, transaction\_time

# Full Query

# 1st Try

SELECT

c.customer\_id,

concat(c.customer\_first\_name,' ', c.customer\_last\_name) as customer\_full\_name,

cp.customer\_id,

cp.product\_id,

cp.market\_date,

cp.cost\_to\_customer\_per\_qty

FROM

`farmers\_market.customer` as c INNER JOIN `farmers\_market.customer\_purchases` as cp

ON c.customer\_id = cp.customer\_id ;

# 2nd Try

SELECT

c.\*

FROM

`farmers\_market.customer` as c INNER JOIN `farmers\_market.customer\_purchases` as cp

ON c.customer\_id = cp.customer\_id ;

# 3rd Try

SELECT

distinct

c.\*

FROM

`farmers\_market.customer` as c INNER JOIN `farmers\_market.customer\_purchases` as cp

ON c.customer\_id = cp.customer\_id ;

# 4th Try

SELECT

distinct

c.customer\_id,

concat(c.customer\_first\_name,' ', c.customer\_last\_name) as customer\_full\_name

FROM

`farmers\_market.customer` as c INNER JOIN `farmers\_market.customer\_purchases` as cp

ON c.customer\_id = cp.customer\_id ;

# 5th Try

SELECT

distinct

c.customer\_id,

concat(c.customer\_first\_name,' ', c.customer\_last\_name) as customer\_full\_name

FROM

`farmers\_market.customer` as c INNER JOIN `farmers\_market.customer\_purchases` as cp

ON c.customer\_id = cp.customer\_id

ORDER BY c.customer\_id ;

# Can we solve this question using a sub-query

# Inner Query

select customer\_id

from `farmers\_market.customer\_purchases`

# Outer Query

select \*

from `farmers\_market.customer`

where customer\_id in

(

) ;

# Full Query

select \*

from `farmers\_market.customer`

where customer\_id in

(

select customer\_id

from `farmers\_market.customer\_purchases`

)

order by customer\_id ;

# Why was this possible? Coz, columns from only one table were required

# and you only need condition from the other table.

# Just to be sure apply distinct here also

# 2nd Try

# Full Query

select \*

from `farmers\_market.customer`

where customer\_id in

(

select distinct customer\_id

from `farmers\_market.customer\_purchases`

)

order by customer\_id ;

#----------------------

# Q37 : List all customer details along with their orders. Include customers that haven't placed any orders.

# Table 1

select \*

from `farmers\_market.customer` ; # customer\_id, customer\_first\_name, customer\_last\_name, customer\_zip

# Table 2

select \*

from `farmers\_market.customer\_purchases` ; # product\_id, vendor\_id, market\_date, customer\_id, quantity, cost\_to\_customer\_per\_qty, transaction\_time

# Join Solution

select distinct

c.customer\_id,

concat(c.customer\_first\_name,' ',c.customer\_last\_name) as customer\_full\_name,

# cp.customer\_id,

# cp.cost\_to\_customer\_per\_qty,

# cp.product\_id

from

`farmers\_market.customer` c LEFT JOIN `farmers\_market.customer\_purchases` cp

ON c.customer\_id = cp.customer\_id;

# In Left join, if i include columns from right table, then duplicate records appear inspite of using distinct.

# In Right join, if i include columns from left table, then duplicate records appear inspite of using distinct.

# if we add 1 column aslo from right table duplicate records appear

# Sub-query Solution

# Not possible because it does not satisfy the below condition.

# Sub-query Solution - Condition :

# Why was this possible? Coz, columns from only one table were required

# and you only need condition from the other table.

#----------------------

# Q38 : Find the customers who are

# 1. New to the market (recently signed up)

# 2. Have deleted their account from the market

select

\*

from

`farmers\_market.customer` c

FULL JOIN

`farmers\_market.customer\_purchases` cp

ON c.customer\_id = cp.customer\_id

where c.customer\_id is null OR cp.customer\_id is null ;

# idea is if someone is new or have deleted their account, the customer id will be null

#----------------------

## # Concept 7 – Self Join

/\*

SYNTAX :

select

\*

from

TAB T1

LEFT JOIN

TAB T2

ON T1.common\_col = T2.common\_col;

Note :  read from SQL.doxc "C:\Users\vivek\Desktop\SQL E2E\SQL.docx"

Application of self-join :

1. When dealing with self-referring tables

2. When comparing different rows of same table with each other.

Catch: Here Manager is also an employee,

so manager column acts as a foreign key to the employee column

\*/

# Q39 : Display the details of all employees along with manager's name.

select \*

from `employees.employees` ;

SELECT

e.first\_name as emp\_name,

m.first\_name as Mgr\_name

from

`employees.employees` e

left JOIN

`employees.employees` m

on e.manager\_id = m.employee\_id;

# Synapses : 1 Column has all info, we need to join this table to itself why ?, coz manager is also an employee

# emp\_id has ~ mgr\_id as its counter part, infact both are same, so use this in ON condition

# To distinguish the mgr and employee, use first name

#----------------------

# Q40 : Compare the salary of each employee with next employee and write the difference. (Lead function shd be used)

# Self Join

select

e.employee\_id,

e.first\_name,

e.salary,

m.employee\_id,

m.first\_name,

m.salary

from `employees.employees` e

left join

`employees.employees` m

on e.manager\_id = m.employee\_id

order by e.first\_name ; # not exactly what is asked

#----------------------

# Q41 : List emp\_id, name, salary along with manager name

SELECT

e.employee\_id as empl\_id,

e.first\_name as employee\_name,

e.salary as emp\_sal,

m.first\_name as manager\_name

from

`employees.employees` e

left join

`employees.employees` m

on e.manager\_id = m.employee\_id;

#----------------------

# Q42 : Display all employees whose salary is greater than their managers ?

SELECT

e.employee\_id as emp\_id,

e.first\_name as empl\_name,

e.salary as emp\_sal,

m.employee\_id as mgr\_id,

m.first\_name as mgr\_name,

m.salary as mgr\_sal

from

`employees.employees` e

left join

`employees.employees` m

on e.manager\_id = m.employee\_id

where e.salary > m.salary;

#----------------------

# Q43 : Write a query to display employees who are managers.

SELECT # distinct

/\*

e.employee\_id as emp\_id,

e.first\_name as empl\_name,

e.salary as emp\_sal,

m.employee\_id as mgr\_id,

m.first\_name as mgr\_name,

m.salary as mgr\_sal

\*/

e.manager\_id,# Mandatory

count(e.employee\_id) emp\_count # Optional

from

`employees.employees` e

left join

`employees.employees` m

on e.manager\_id = m.employee\_id

group by e.manager\_id; #solved 1:43 am 10 May 2023

# where m.manager = m.first\_name; # Don't think this is right.

select \*

from `employees.employees`

where e.manager\_id not null ; # Not good logic, just tried

#----------------------

## # Unsolved

# Q44 : Compare salaries of all employees. Display results such that salary of left employee is <= salary of right employee.

--- Sub string ---

select substr('Vivek', 2,3) as sub\_string ;

# Assignment questions

# Regex - Pattern matching

## # Concept 8 – Aggregations - Select

# Q45 : Get count of all employees in the company

select \* from `employees.employees` ; # 107 count

select count(employee\_id) employee\_count

from `employees.employees` ;

#----------------------

# Q46 : Display the min, max and avg salary of employees.

select

min(salary) as minimum\_salary,

max(salary) as maximum\_salary,

avg(salary) as average\_salary

from `employees.employees` ;

#----------------------

# Q47 : Count different product sizes available

select \* from `farmers\_market.product` # Count 23

select count(product\_size) product\_size\_count

from `farmers\_market.product` ; # Count 22 , count does not count nulls

select count(\*) product\_size\_count

from `farmers\_market.product` ; # Count 23 ,\* counts nulls also

# Extras

select distinct(product\_size) unique\_count

from `farmers\_market.product` # Unique Count 16

select count(distinct(product\_size)) unique\_count

from `farmers\_market.product` # Unique Count 15, does not count nulls

select product\_size from `farmers\_market.product` order by 1; # pick the first column from select and order by that column.

select distinct product\_size from `farmers\_market.product`; # Count 16 - only distinct , counts nulls

#----------------------

/\*

# Notes:

1. count(\*) - counts everything including nulls in the column

2. count(column\_name) - counts everything excluding nulls in the column

3. count(distinct(column\_name)) - counts only unique values excluding nulls in the column

\*/

select \* from `farmers\_market.product` order by 2; # order by second column in select clause.

## # Concept 9 – Group by

/\*

# Imagine you work at Amazon category team and asked to answer the following questions.

1. How many products are there in each category

2. What is the avg price of products in each category

3. Provide details of products purchased by more than 20 customers.

\*/

# aggregation on entire data

# aggregation on sub-set of data. Its called Group by

# Group by - helps group data by various categories and provide aggregate metrics.

# GROUP BY Syntax

select [column] [aggragate function] # aggregation of a column, show which column you are aggregating on

where

group by [column]

#----------------------

# Q48 : Find total number of products with product size medium

select \* from `farmers\_market.product` ;

select count(product\_id) as medium\_count # count(product\_size) also

from `farmers\_market.product`

where product\_size = 'medium' ; # count 5

#----------------------

# Q49 : Find total number of products for different product sizes

select product\_size,

count(product\_id ) product\_count

from `farmers\_market.product`

group by product\_size ;

select product\_size,

count(product\_id ) product\_count

from `farmers\_market.product`

group by product\_size

order by product\_count desc;

#----------------------

# Q49A : Find total number of products for different product sizes and display results for small, medium and large.

select product\_size, # coz group by on this column

count(product\_id ) product\_count # aggregation

from `farmers\_market.product` # actual table

where product\_size = 'small' or product\_size = 'medium' or product\_size = 'large' # filter data

group by product\_size ; # group things together

# Alternatively

select product\_size, # coz group by on this column

count(product\_id ) product\_count # aggregation

from `farmers\_market.product` # actual table

where product\_size in ('small','medium','large') # filter data - use single quotes

group by product\_size ; # group things together

#----------------------

# Q50 : Count number of orders by each customer.

select \* from `farmers\_market.customer\_purchases` ;

select customer\_id, count(product\_id) customer\_orders

from `farmers\_market.customer\_purchases`

group by customer\_id; # which customer has max num of orders

# alternatively

select customer\_id, count(\*) customer\_orders

from `farmers\_market.customer\_purchases`

group by customer\_id

order by customer\_orders ; # desc to get the large one first.

select customer\_id, count(\*) customer\_orders

from `farmers\_market.customer\_purchases`

group by customer\_id

order by customer\_orders desc; # while this is good, i need 1 answer not all results

select customer\_id, count(\*) customer\_orders

from `farmers\_market.customer\_purchases`

group by customer\_id

order by customer\_id ;

# Alternatively

select customer\_id, count(\*) customer\_orders

from `farmers\_market.customer\_purchases`

group by customer\_id

order by customer\_orders desc

limit 1; # no comma before limit, can use alias names here,

                                        #coz order by is after select clause, alias names are available to order by.

select customer\_id, count(\*) orders\_count

from `farmers\_market.customer\_purchases`

group by customer\_id

order by 2 desc

limit 1; # no comma before limit

# This is not possible

select customer\_id, max(count(product\_id)) order\_count

from `farmers\_market.customer\_purchases`

group by customer\_id ;

# Error : Aggregations of aggregations are not allowed at [1:21]

# aggregations happen on the original table, not on another aggregation.

# Dont know what i wanted here, its all good

# Lets see why its not possible

select customer\_id, count(product\_id) order\_count

from `farmers\_market.customer\_purchases`

group by customer\_id ;

#----------------------

# Q51 : Count number of orders by each customer per market date.

select \*

from `farmers\_market.customer\_purchases`;

select customer\_id, market\_date, count(product\_id) customer\_orders

from `farmers\_market.customer\_purchases`

group by customer\_id, market\_date

order by market\_date, customer\_id;

select customer\_id, market\_date, count(\*) customer\_orders

from `farmers\_market.customer\_purchases`

group by customer\_id, market\_date # the order does not matter.

order by 1,2;

#----------------------

# Q52 : Calculate total quantity purchased by each customer per market date

select customer\_id, market\_date, sum(quantity) tot\_quantity

from `farmers\_market.customer\_purchases` # good

group by customer\_id, market\_date

order by 1,2;

select customer\_id, market\_date, round(sum(quantity),1) tot\_quantity # Round operates at row level, so its not an aggregate function.

from `farmers\_market.customer\_purchases` # good

group by customer\_id, market\_date ;

#----------------------

# Q53 : How many kinds of products were purchased by each customer on each market date.

select \*

from `farmers\_market.customer\_purchases`;

select customer\_id, market\_date, count(product\_id) prod\_count

from `farmers\_market.customer\_purchases` # good

group by customer\_id, market\_date ;

# coz product\_id should appear only once, use distinct keyword also in your aggregation.

select customer\_id, market\_date, count(distinct product\_id) prod\_count # read the question - How many kinds of products, not count of products.

from `farmers\_market.customer\_purchases` # good

group by customer\_id, market\_date

order by 1,2;

# both aggregations side by side

select customer\_id, market\_date, count(product\_id) prod\_count, count(distinct product\_id) prod\_count

from `farmers\_market.customer\_purchases` # good

group by customer\_id, market\_date

order by 1,2;

#----------------------

# Q54 : Find total price paid by customer\_id 3 per market date.

select \*

from `farmers\_market.customer\_purchases`;

select market\_date, round(sum(quantity \* cost\_to\_customer\_per\_qty),2) tot\_price  # Aggregation (column) on top of in-line calculation(row)

from `farmers\_market.customer\_purchases`

where customer\_id = 3

group by market\_date

order by 1;

# Which customer , how do you know

select customer\_id, market\_date, round(sum(quantity \* cost\_to\_customer\_per\_qty),2) tot\_price  # Aggregation (column) on top of in-line calculation(row)

from `farmers\_market.customer\_purchases`

where customer\_id = 3

group by customer\_id, market\_date

order by 1,2; # no need 1, its all cust id 3

#----------------------

# Q55 : Find the least and max quantity purchased by any customer

select

customer\_id,

min(quantity) min\_qty,

max(quantity) max\_qty

from `farmers\_market.customer\_purchases`

group by customer\_id

order by 1;

#----------------------

# Q56 : Display all product categories that have more than 2 products.

select \*

from `farmers\_market.product\_category` # not this table.

select \*

from `farmers\_market.product` # this table.

select product\_category\_id, count(product\_id) prod\_count

from `farmers\_market.product`

group by product\_category\_id ; # filtering is still not done here yet

select product\_category\_id, count(product\_id) prod\_count

from `farmers\_market.product`

# where count(product)id

group by product\_category\_id

having count(product\_id) >= 2 ; # 4 rows of data (>=)

select product\_category\_id, count(product\_id) prod\_count

from `farmers\_market.product`

group by product\_category\_id

having count(product\_id) > 2 ; # 2 rows of data (>)

# Use of alias in having clause

select product\_category\_id, count(product\_id) prod\_count

from `farmers\_market.product`

group by product\_category\_id

having prod\_count > 2 ; # It works but dont use it. It only works in Big Query, not in mysql or other DB's.

# Select executes last ,row , after having clause , but coz big query did some optimization it will work in BQ only.

# write proper aggregate function only inside having clause.

#----------------------

# Q57 : Display all products along with total number of orders for that product if number of orders exceed 10

select \*

from `farmers\_market.product` # has nothing on orders

select \*

from `farmers\_market.customer\_purchases` # this is orders table

table: customer purchases

columns : product id, orders (count(\*))

clauses : where :  orders > 10 ; group by (product\_id), condition on aggregation, having clause not where clause

select product\_id, count(\*) order\_count

from `farmers\_market.customer\_purchases`

group by product\_id

having count(\*) > 10 ; # 4 records, prod id 1,2,3,4

select product\_id, count(product\_id) order\_count

from `farmers\_market.customer\_purchases`

# where count(product\_id) > 10 # Error : Aggregate function COUNT not allowed in WHERE clause at [3:7]

group by product\_id

having count(product\_id) > 10 ; # 4 records, prod id 1,2,3,4

# Count cannot be in where clause ever

#----------------------

# Q58 : Find the average amount spent on each market day. We want to consider only those days where the number of purchases were

# more than 3 and the transaction amount of an order was greater than 5. [HARD]

# Table : cust purchases

# average amount spent : Amount, Avg

# on each market day : group by - market\_date

select \*

from `farmers\_market.customer\_purchases`;

table : cust purchases

columns : amount - in-line calc - row level,

aggreagation:  avg

select : market\_date, aggregation : avg(quantity \* cost\_to\_customer\_per\_qty) average\_amount, count(\*)

group by : market date

having

count(\*) > 3, avg(quantity \* cost\_to\_customer\_per\_qty) > 5 ;

select market\_date, avg(quantity \* cost\_to\_customer\_per\_qty) average\_amount, count(\*) num\_of\_purchases

from `farmers\_market.customer\_purchases`

where sum(quantity \* cost\_to\_customer\_per\_qty) > 5

group by market\_date

having count(\*) > 3,  ;

# not necessary that having shd have same aggregate function that appears in select clause. Both can have different aggregation functions.

# Full Query

select

market\_date,

round(avg(quantity \* cost\_to\_customer\_per\_qty),2) average\_amt,

count(\*) purchase\_count\_more\_than\_three,

round(sum(quantity \* cost\_to\_customer\_per\_qty),2) pur\_amt\_more\_than\_five

from `farmers\_market.customer\_purchases`

where quantity \* cost\_to\_customer\_per\_qty > 5

group by market\_date

having count(\*) > 3 ;

## # Concept 10 – Having Clause

# Having clause

# Display departments with more than 2 employees

select \*

from `employees.employees` ;

select department\_id, count(department\_id) count\_dept

from `employees.employees`

where count(department\_id) >= 2 # First principles - where is executed no 2 in ooe, how can it use 'count(dept\_id)' which is calculated later.

group by department\_id ;

# How can we apply filter on a group by object

# Having clause :  filtering after grouping is done, on top of a grouped object, on top of aggregated value

# Special clause used to filter results after group by. Having cannot exist without group by. Group by can exist without having,

select department\_id, count(employee\_id) count\_dept

from `employees.employees`

# where count(dept\_id) >= 2

group by department\_id

having count(employee\_id) >= 2 # it's correct now.

order by 1 ;

# count\_dept shd not work but big query must have done some kind of optimization , it will certainly not work in other DB's

select department\_id, count(employee\_id) count\_dept

from `employees.employees`

# where count(dept\_id) >= 2

group by department\_id

having count(employee\_id) >= 2 # it's correct now

order by 1 ;

# OOE - where - group by – having

## # Concept 11 – Window Functions

# How many employees have salary > avg salary ?

# How many employees have salary > avg salary of their department ?

# Q59 : Display avg salary of all empoloyees in a new column

select \* from `employees.employees`;

# Wrong way

select

employee\_id,

first\_name,

salary,

avg(salary) as average\_salary # SQL is expecting a summarization, a group by, but we want to keep the original table and do summarization

from `employees.employees`;

# SELECT list expression references column employee\_id which is neither grouped nor aggregated at [2:1]

# Aggregation loses original table an compresses the values.

# Window functions keep original table and creates a new loumn with aggregated values right next to original values

# correct way

select

employee\_id,

first\_name,

salary,

avg(salary) over() as average\_salary

from `employees.employees`;

# Round off average salary

select

employee\_id,

first\_name,

salary,

round(avg(salary),2) over() as average\_salary

from `employees.employees`; # ,2 should come after over()

# Error: Function ROUND does not support an OVER clause at [5:1]

# Correct way

select

employee\_id,

first\_name,

salary,

round(avg(salary) over(),2) as average\_salary

from `employees.employees`;

#----------------------

# Q60 : Display the avg salary of employees in each department as a new column next to the respective employee belonging in the corresponding department.

# Window Function solution

select

employee\_id,

first\_name,

department\_id,

salary,

round(avg(salary) over(partition by department\_id),2) as average\_salary

from `employees.employees`

order by department\_id;

# Join solution

# Sub-query

# Inner Query

select department\_id, round(avg(salary),2) as dep\_avg\_sal

from `employees.employees`

group by department\_id ;

# Outer Query

SELECT

\*

FROM `employees.employees` e

JOIN

(

) ;

# Full Query

SELECT

e.employee\_id,

e.first\_name,

e.department\_id,

e.salary,

das.department\_id,

das.dep\_avg\_sal

FROM `employees.employees` e

left JOIN

(

select department\_id, round(avg(salary),2) as dep\_avg\_sal

from `employees.employees`

group by department\_id

) as das # a calculated table, not an original table, so give alias name to it.

ON e.department\_id = das.department\_id

order by e.department\_id;

# Writing window function is very straight forward and very easy

# With in DB also window function operates very efficient compared to your join.

#----------------------

# Q60A:  Display details of employees with salary > avg salary.

select

\*,

round(avg(salary),2) over() as average\_salary

from `employees.employees`

where salary > round(avg(salary),2) ; # OOE, anything which is part of select cannot be used in where, where executes first.

# Aggregate functions are not part of where clause. Error: Aggregate function AVG not allowed in WHERE clause at [6:22].

# Here we are comparing the (original+aggregation column) with (original Column)

# We know that the below query gives (original+aggregation column) - run it and see.

select

employee\_id,

first\_name,

salary,

round(avg(salary) over(),2) as average\_salary

from `employees.employees`;

# is this a table, YES. Treat it like a TABLE, thats it. Give it an alias name.

select

\*

from

(select

employee\_id,

first\_name,

salary,

round(avg(salary) over(),2) as average\_salary # for outer query average\_salary is a column

from `employees.employees`) window\_table

where salary > average\_salary

order by salary ;

# where clause only works on original columns. For where clause window function's columns are original columns.

# So fool the where clause by

# using window function to create a extra column average salary.

# use the concept of sub-query, to write where clause on your window functions

#----------------------

# Q61 : Display details of employees with salary > department avg salary

select \*

from

(

select

employee\_id,

first\_name,

department\_id,

salary,

round(avg(salary) over(partition by department\_id),2) dep\_avg\_sal

from `employees.employees` ) as dep\_table

where salary > dep\_avg\_sal

order by department\_id;

# If you want all departments to show

select

\*

from

(

select

employee\_id,

first\_name,

department\_id,

salary,

round(avg(salary) over(partition by department\_id),2) dep\_avg\_sal

from `employees.employees` ) dep\_table

# where salary > dep\_avg\_sal

order by department\_id;

#-----------------------------

## # Concept 12 – Rank Functions

# Find the top 5 earning employees in the company ?

select

employee\_id,

first\_name,

department\_id,

salary

from `employees.employees`

order by salary desc

limit 5;

# rank, dense\_rank & row\_number

select

employee\_id,

first\_name,

department\_id,

salary,

row\_number() over(order by salary desc) as row\_num, # if you want exact rows use row\_number

rank() over(order by salary desc) as rankk, # If values are same, rank is same, but counts in the back,1,2,2,4

dense\_rank() over(order by salary desc) as dense\_rankk # If values are same, but does not count in the back,1,2,2,3

from `employees.employees`

order by salary desc;

# Testing order by row\_num

select

employee\_id,

first\_name,

department\_id,

salary,

row\_number() over(order by salary desc) as row\_num, # if you want exact rows use row\_number

rank() over(order by salary desc) as rankk, # If values are same, rank is same, but counts in the back,1,2,2,4

dense\_rank() over(order by salary desc) as dense\_rankk # If values are same, but does not count in the back,1,2,2,3

from `employees.employees`

order by row\_num;

#-----------------------------

# Q62 : Fetch top 2 employees from each department based on salary. No more than 2 employees in each department.

# Table : Employee table

# function : row number

# order by salary desc needed

# partition by department

select

employee\_id,

first\_name,

department\_id,

salary,

row\_number() over(partition by department\_id order by salary desc) as rrr

from `employees.employees`

where rrr <= 2 # Work, Error ; Unrecognized name: rrr at [8:7]

order by department\_id ;

select \*

from

(select

employee\_id,

first\_name,

department\_id,

salary,

row\_number() over(partition by department\_id order by salary desc) as rankk

from `employees.employees` ) tbl

where rankk <= 2

order by department\_id;

# when there is a aggregation function which you cannot use in where clause, to cheat the where clause by using a sub-query.

#-----------------------------

# Q63 : Fetch details of employees with top 3 salaries. Display all people if salaries are same.

select \*

from

(select

employee\_id,

first\_name,

department\_id,

salary,

dense\_rank() over(order by salary desc) as dens\_rankk

from `employees.employees`) tbl

where dens\_rankk <= 3 # limit cannot be used, think abt it.

order by dens\_rankk;

/\*

Simple funda :

Repetition allowed - Dense rank

Repetition not allowed - rank

Exact values needed - row number

\*/

#-----------------------------

# Q64 :  From each department display the person with max salary. Display multiple people if they have same salary

select \*

from

(select

employee\_id,

first\_name,

department\_id,

salary,

row\_number() over(partition by department\_id order by salary desc) as rankk

from `employees.employees`) tbl

where rankk <= 1

order by department\_id;

# Dense Rank

select \*

from

(select

employee\_id,

first\_name,

department\_id,

salary,

dense\_rank() over(partition by department\_id order by salary desc) as dens\_rankk

from `employees.employees`) tbl

where dens\_rankk <= 2

order by department\_id;

#-----------------------------

# Q65:  From each department display the person with 3rd highest salary.

select \*

from

(select

employee\_id,

first\_name,

department\_id,

salary,

row\_number() over(partition by department\_id order by salary desc) as rankk

from `employees.employees`) tbl

where rankk = 3

order by department\_id;

#-----------------------------

# Q66: From each department display the person with 3rd highest salary. Display multiple folks if thats the case

select \*

from

(select

employee\_id,

first\_name,

department\_id,

salary,

dense\_rank() over(partition by department\_id order by salary desc) as rankk

from `employees.employees`) tbl

where rankk = 3

order by department\_id;

#-----------------------------

# Q67: Fetch details of employees with top 3 salaries in each department.

# Display all employees with same salary

select

\*

from

(

  select

employee\_id,

first\_name,

department\_id,

salary,

dense\_rank() over(partition by department\_id order by salary desc) as rankk

from `employees.employees`) tbl1

where rankk <= 3

order by department\_id ;

## # Concept 13 – Lead and Lag

# Q: Compare each employee's salary with the next highest salary.

# Self Join has 2 applications

# 1. When self referring child lable

# 2. Compare multiple rows with in the same table

# Ranking functions are also known as non-aggregate functions window functions, works on Rows

# aggregate functions works on columns

# Non- aggregate functions - LAG() and LEAD()

# Lag means previous, lead means next

# We can also skip Lead(salary,2), jumps 2 steps

#---------

# 68: For every employee display their details. Also display the salary of the next highest earning employee.

# Use the new column to calculate salary difference.

select \*

from `employees.employees`;

select

employee\_id,

department\_id,

salary,

LEAD(salary) over(order by salary ) as next\_salary,

( LEAD(salary) over(order by salary) - salary )as sal\_diff

from

`employees.employees`

order by salary desc ;

#---------

select

employee\_id,

department\_id,

salary,

LEAD(salary) over(order by salary) as next\_salary

from

`employees.employees`

order by salary ; # Why 2400 is coming in 3rd row. Lets sort by salary and next salary.

# Highest salary on top, use salary desc

#---------

select

employee\_id,

department\_id,

salary,

LEAD(salary) over(order by salary) as next\_salary

from

`employees.employees`

order by salary, next\_salary ; # first solution is the best so far.

#---------

# Leave lead order by empty and see what happens

select

employee\_id,

department\_id,

salary,

LEAD(salary) over() as next\_salary, # Error : Window ORDER BY is required for analytic function lead at [5:18]

from

`employees.employees`

order by salary, next\_salary ;

# Error : Window ORDER BY is required for analytic function lead at [7:18]

# Lead and lag functions like your rank functions dont make any sense without the order by clause.

#---------

# Experiment with different functions

select

employee\_id,

department\_id,

salary,

LEAD(salary) over(order by salary) as next\_salary,

LEAD(salary,2) over(order by salary) as next\_salary\_skip,

LAG(salary) over(order by salary) as prev\_salary,

LAG(salary,2) over(order by salary) as prev\_salary\_skip,

(next\_salary - salary) as diff\_salary # Error: Unrecognized name: next\_salary at [9:2]

from

`employees.employees`

order by salary, next\_salary ;  # Error

#---------

# Use a sub-query

select # why cant we use the column names here. Error : Unrecognized name: employee\_id at [2:1]

employee\_id,department\_id,salary,next\_salary,

(next\_salary - salary) as diff\_salary

from

( select

employee\_id, department\_id, salary,

LEAD(salary) over(order by salary) as next\_salary,

LEAD(salary,2) over(order by salary) as next\_salary\_skip,

LAG(salary) over(order by salary) as prev\_salary,

LAG(salary,2) over(order by salary) as prev\_salary\_skip

from `employees.employees`) as tbl1

order by salary, next\_salary ;

#---------

# Another way

select

\*,

(next\_salary - salary) as diff\_salary

from

( select

salary,

LEAD(salary) over(order by salary) as next\_salary,

from `employees.employees`) tbl1

order by salary desc, next\_salary  ;

#---------

# there are 2 kinds of window functions

# 1. Aggregate - same functions that we used in group by also - Min, max, sum, count, average

# 2. Non-aggregate - dont exist on their own. They can only exist with window functions - rank, dense rank, lag, lead.

# The above can be written in a inline calculation also.

select

\*,

from

( select

employee\_id, # this works fine

lead(salary) over(order by salary) as next\_salary

from `employees.employees`) as tbl1

right join `employees.employees`e

on tbl1.employee\_id = e.employee\_id;

#---------

# Yet another way

select

employee\_id, department\_id, salary,

from

( select

employee\_id, # Error: Column name employee\_id is ambiguous at [2:1]

lead(salary) over(order by salary) as next\_salary

from

`employees.employees`) as tbl1

right join `employees.employees`e

on tbl1.employee\_id = e.employee\_id; This failed

# I want to have empid, dept id, salary and next salary and diff as columns , is it possible

select

e.employee\_id, e.department\_id, e.salary,

tbl1.next\_salary,

next\_salary-salary as diff\_salary

from

(select

employee\_id,

lead(salary) over(order by salary) as next\_salary

from

`employees.employees`) as tbl1

right join `employees.employees`e

on tbl1.employee\_id = e.employee\_id

order by e.salary, tbl1.next\_salary;

## # Concept 14 – Cumulative Sum

# Keep adding to the previous number.

# sum(salary) over(partition by dept\_id) - this will add another column with aggregated value of the entire column for each row for comparison

# sum(salary) over(partition by dept\_id order by emp\_id) - this triggers cumulative sum till the current row.

# When ever order by is NOT mentioned in window function, calculations happen for the entire window.

# as soon as the order by is seen it triggers the concept of window frames.

# order by calculations are happening between current row and all rows above.

# if you actually think about it, this is exactly what is happening in rank function also.

# Q69: Display running sum or cumulative sum of salaries in each department in the order of joining dates of employees.

select \*

from `employees.employees` ; # Hire date

select

employee\_id,

department\_id,

salary,

hire\_date,

sum(salary) over(partition by department\_id order by hire\_date desc) as cumu\_sal

from `employees.employees`  # WOW

order by department\_id ;

select

employee\_id,

department\_id,

salary,

hire\_date,

sum(salary) over(partition by department\_id order by employee\_id) as cumu\_sal

from `employees.employees`  # WOW

order by department\_id ;

# What if order by is not there

select

employee\_id,

department\_id,

salary,

hire\_date,

sum(salary) over(partition by department\_id order by hire\_date) as cumu\_sal,

sum(salary) over(partition by department\_id) as col\_sal

from `employees.employees`

order by department\_id ; # Now you see , dept total is in every dept data row

# Window Frames (Advanced Control)

# particular clause that is getting executed within over - "rows between unbounded preceding and current row"

# We can manipulate it further.

# Concept of moving averages -  Time Series

# Window Frames

select

employee\_id,

department\_id,

salary,

hire\_date,

sum(salary) over(partition by department\_id order by hire\_date) as cumu\_sal,

sum(salary) over(partition by department\_id order by hire\_date rows between unbounded preceding and current row) as cumu\_sal\_new,

sum(salary) over(partition by department\_id order by hire\_date rows between 1 preceding and 1 following) as moving\_sum\_3 # Window size 3

from `employees.employees`

order by department\_id ;

# Real life example , Stock market prices fluctuate every sec to sec. Its hard to consider second to second values. So we take averages.

# we keep a window of 5 days and including current day we take average of 5 days , 2 above and 2 below. so tomorrow also same calc is

# considered, hence the average moves because the stock prices fluctuate.

# Example Screener - asain paints, 50 DMA - Daily Moving Average.

# Order by is mandatory for window functions rank, dense rank, lag and lead

## # Concept 15 – DATE & TIME Functions

# Date Time and Writing Efficient Queries

# How many employees were hired in 2020.

# Data Types associated with Dates

# 1. Date - '2023-04-01'

# 2. Time - '08-14-32........00000'

# 3. Date time - date + time

# 4. timestamp - '2023-04-01 08-14-32 IST' # Timezone

# https://cloud.google.com/bigquery/docs/reference/standard-sql/data-types

# https://cloud.google.com/bigquery/docs/reference/standard-sql/date\_functions

# mysql data and time functions

# D&T Functions

1. Extract - uses and limitations - extract (year from hire\_date) # day, month, dayofweek

# Q70 : Tell the joining month of all the employees

select

employee\_id,

department\_id,

salary,

hire\_date,

extract(month from hire\_date) as Hire\_month,

date(hire\_date) Date

from `employees.employees` ;

# Trying this on my own

select

employee\_id,

department\_id,

salary,

extract(year from hire\_date) as hire\_year,

extract(month from hire\_date) as hire\_month,

extract(dayofweek from hire\_date) as hire\_dayofweek, # week starts on Sunday, 4 is wednesday.

extract(day from hire\_date) as hire\_day,

case  # when hire\_month = 1 then 'January' # Error : Unrecognized name: hire\_month; Did you mean hire\_date? at [7:8]

  when extract(month from hire\_date) = 1 then 'January'

  when extract(month from hire\_date) between 1 and 3 then 'First Quarter'

end as month\_names

from `employees.employees` ;

# When dealing with dates there can be a requirement for generating dates.

# Provide tenure of all employees (current date, current time and so on)

# big query gives 1 function each for different date related data types.

# Generate Dates

CURRENT - DATE()

CURRENT - TIME()

CURRENT - DATETIME()

CURRENT - TIMESTAMP()

# How many products sold till date

# Q71 : Display current date, time etc.

# Isolated values

select

current\_date() cur\_date,

current\_time() cur\_time,

current\_datetime() cur\_datetime,

current\_timestamp() cur\_timestamp # 2023-04-25 07:26:03.757799 UTC

# Big query is cloud platform, so where ever the cloud servers are located that time is displayed.

# With a table

select

employee\_id,

hire\_date,

current\_date() curr\_date,

current\_time() curr\_Time,

current\_datetime() cur\_datetime,

from `employees.employees`;

# Q72: Provide tenure of all employees

select

employee\_id,

hire\_date,

current\_date() curr\_date,

(current\_date() - hire\_date) as emp\_tenure # cannot be understood  - 0-0 13096 0:0:0

from `employees.employees`;

# Date diff

select

employee\_id,

current\_date() curr\_date,

hire\_date,

date\_diff(current\_date(),hire\_date, day) as emp\_tenure\_days,

date\_diff(current\_date(),hire\_date, month) as emp\_tenure\_months,

date\_diff(current\_date(),hire\_date, year) as emp\_tenure\_years

from `employees.employees`

order by emp\_tenure\_years desc ; # Oldest employees first

# How many products are sold till date ?

select \*

from `employees.sales` ; # odin, thor

select \*

from `employees.employees` ;

# How many tenure years are there?

select

date\_diff(current\_date(),hire\_date, year) as emp\_tenure\_years

from `employees.employees`

order by emp\_tenure\_years desc ; # oldest on top, multiple values present

# make them unique

select

date\_diff(current\_date(),hire\_date, year) as emp\_tenure\_years

from `employees.employees`

group by emp\_tenure\_years

order by emp\_tenure\_years desc ; # 12 values

# Count the num of empl for each 12 categories of tenure.

# Count of employees for each unique tenure year ? How many pple are serving for how many years ?

select

count(\*) as emp\_count

from

(select

date\_diff(current\_date(),hire\_date, year) as emp\_tenure\_years

from `employees.employees`

group by emp\_tenure\_years

order by emp\_tenure\_years desc ) tenure\_tbl ; # Incorrect

# Error : Unrecognized name: employee\_id at [2:7]

# Count of employee id will not work here, as it in snot part of the internal query in our sub-query, so do count(\*)

# Also group by emp\_tenure\_years in the outer query of your sub-query.

# no need to do sub-query

select

date\_diff(current\_date(),hire\_date, year) as emp\_tenure\_years,

count(employee\_id) as emp\_count

from `employees.employees`

group by emp\_tenure\_years

order by emp\_tenure\_years desc ; # Well done

# Using sub-query

select

emp\_tenure\_years,

count(\*) as emp\_count

from

(select

date\_diff(current\_date(),hire\_date, year) as emp\_tenure\_years

from `employees.employees`

order by emp\_tenure\_years desc ) tenure\_tbl

group by emp\_tenure\_years ; # Well done

# Can we add dates ? Does not make any sense (3rd april + 10th june), what does that even mean ?

# What makes sense is what is the date going to be if i add 5 days to current date or some dat ?

# date\_add() & date\_sub()

select date\_add(current\_date(), interval 12 day) as some\_days\_later ;  # 2023-05-07

select date\_add(current\_date(), interval 12 month) as some\_months\_later ; # 2024-04-25

select date\_add(current\_date(), interval 12 year) as some\_years\_later ; # 2035-04-25

select date\_sub(current\_date(), interval 12 day) as some\_days\_before ;  # 2023-04-13

select date\_sub(current\_date(), interval 12 month) as some\_months\_before ; # 2022-04-25

select date\_sub(current\_date(), interval 12 year) as some\_years\_before ; # 2011-04-25

# Single Select clause

select

date\_add(current\_date(), interval 12 day) as some\_days\_later,  # 2023-05-07

date\_add(current\_date(), interval 12 month) as some\_months\_later, # 2024-04-25

date\_add(current\_date(), interval 12 year) as some\_years\_later, # 2035-04-25

date\_sub(current\_date(), interval 12 day) as some\_days\_before,  # 2023-04-13

date\_sub(current\_date(), interval 12 month) as some\_months\_before, # 2022-04-25

date\_sub(current\_date(), interval 12 year) as some\_years\_before # 2011-04-25

# Brilliant

# Q73: For each employee, find the time period between the company's start date and the employee's joining date.

select \*

from `employees.employees`

select

employee\_id,

min(hire\_date) over() as company\_start\_date, # Window function, no partition - coz no category required, no order by coz no sorting required

hire\_date,

date\_diff(hire\_date, min(hire\_date) over(), day) as gap

from

`employees.employees`

order by hire\_date , employee\_id ;

# Sub-query

select

min(hire\_date) over() as company\_start\_date,

from

`employees.employees` ;  # 107 rows with same date '1987-06-17'

#------

# Full Query

select

employee\_id,

date\_diff(hire\_date, tbl1.company\_start\_date, day) as joined\_days\_later,

date\_diff(hire\_date, tbl1.company\_start\_date, year) as joined\_years\_later

from

(select

employee\_id,

min(hire\_date) over() as company\_start\_date,

hire\_date

from

`employees.employees`) as tbl1

order by joined\_days\_later ;

# Q74: Find all employees who joined within 2 years of company's inception.

# Full Query

select

employee\_id,

tbl1.joined\_years\_later

from

(select

employee\_id,

date\_diff(hire\_date, min(hire\_date) over(), year) as joined\_years\_later,

min(hire\_date) over() as company\_start\_date,

hire\_date

from

`employees.employees`) as tbl1

where joined\_years\_later < 2

order by joined\_years\_later ; # WOW

# without sub-query

select

employee\_id,

min(hire\_date) over() as company\_start\_date,

hire\_date,

date\_diff(min(hire\_date) over(), hire\_date, day) as gap

from

`employees.employees`

# where date\_diff(min(hire\_date) over(), hire\_date, day) <=4 # Error: Analytic function not allowed in WHERE clause at [10:17]

group by hire\_date , employee\_id

having date\_diff(min(hire\_date) over(), hire\_date, day) <=4 ; # Analytic function not allowed in HAVING clause at [15:18]

# Direct query without sub query

select

employee\_id,

hire\_date,

min(hire\_date) over() as company\_start\_date,

date\_add(min(hire\_date) over(), interval 2 year) as comp\_start\_2yrs

from `employees.employees`

where hire\_date <= comp\_start\_2yrs ; # cannot use 'comp\_start\_2yrs' directly here, OOE problem, use sub-query.

#----Sub-query

select

\*

from

(select

employee\_id,

hire\_date,

min(hire\_date) over() as company\_start\_date,

date\_add(min(hire\_date) over(), interval 2 year) as comp\_start\_2yrs

from `employees.employees`)

where hire\_date <= comp\_start\_2yrs ;

select

employee\_id

from

(select

employee\_id,

hire\_date,

min(hire\_date) over() as company\_start\_date,

date\_add(min(hire\_date) over(), interval 2 year) as comp\_start\_2yrs

from `employees.employees`)

where hire\_date <= comp\_start\_2yrs ;

# Timezone

select current\_time([IST])

https://stackoverflow.com/questions/12482637/bigquery-converting-to-a-different-timezone

## # Concept 16 – CTE – Common Table Expressions

/\*

CTE and VIEWS – you can write queries without these also.

Points –

1.  Writing efficient queries

2.  When writing complex queries, the structure will be complex

3.  You forget the reason why you wrote such a complex query in a month

4.  No structure, nested sub-queries

For solving the above issues these 2 concepts were defined

CTE – common table expressions

\*/

# Q76 :

# Day 1

# Manager asks, get :

# total orders, total customers, total revenue

# per product, per date

# Without CTE

select \*

from `farmers\_market.customer\_purchases` ;

select

product\_id, market\_date,

count(\*) as tot\_orders,

count(customer\_id) as tot\_cust,

round(  sum(quantity \* cost\_to\_customer\_per\_qty),1) as tot\_rev

from `farmers\_market.customer\_purchases`

group by product\_id, market\_date

order by product\_id, market\_date,tot\_rev desc;

# with CTE

with cte as (

select

product\_id, market\_date,

count(\*) as tot\_orders,

count(customer\_id) as tot\_cust,

round(  sum(quantity \* cost\_to\_customer\_per\_qty),1) as tot\_rev

from `farmers\_market.customer\_purchases`

group by product\_id, market\_date

)

select \* from cte

order by product\_id, market\_date,tot\_rev desc;

# output of the query is stored in a variable (CTE)

# structured way

# No performance gain

# simplifying code readability

# Improves readability

# Avoid redundancy

# when you cannot create a template table in DB, due to access issues

# Create any number of CTE's but single use only.

# Limitations / disadvantages

# Very very limited scope, Need query and CTE in the same environment.

# Only one select statement, multiple times.

# Not in another select query.

# Data Scientists do not have any permisson to create a table

# Read the data and present analysis.

# Multiple CTE's

with cte as (

select

product\_id, market\_date,

count(\*) as tot\_orders,

count(customer\_id) as tot\_cust,

round(  sum(quantity \* cost\_to\_customer\_per\_qty),1) as tot\_rev

from `farmers\_market.customer\_purchases`

group by product\_id, market\_date

),

haha as(

select \* from cte

order by product\_id, market\_date,tot\_rev desc )

select \* from haha

# In case of self joins it will give you some performnce gain

# as the answer is already in the memory.

## # Concept 17 – Views

# VIEWS

# Creates a virtual table to our query

# No table is actually created in the background

CREATE VIEW as view\_name ( select \* from `farmers\_market.customer\_purchases`)

/\*

Advantages and disadvantages

1. Saved in DB, asks for dataset qualifier

2. can be used in multiple places

3. tables will have previews, views will not as they are not actual tables, only query is saved

4. run just the query with the view name, it executes fine. Runs fine in another window also.

5. where ever the DB is accessible, views are accessible too.

6. CTE scope limitation is resolved

7. Why view, instead create a table right ? memory, less use, dynamic, customisable,

8. Query saved as a temp\_table is saving the output of the table and not the query

9. Views on other hand saves the query not output, so when you execute a view,

#  its queries the latest data from relevant tables. if underlying table is modified , we get latest information.

# Advantages of VIEWS

1. can be used as normal table

2. store the query not output

3. scope problem of CTE's is resolved

\*/

# CTE           views

adhoc           repeated query

see the blue book snip

## # Concept 18 – Union

there are suppliers table with supp\_city and customers table with cust\_city

Q. we want to show all cities from both tables

(select supp\_city from suppliers) union (select cust\_city from customers)

Big query has 2 unions

UNION ALL - keeps all duplicates

union distinct - removes duplicates

column names has to be the same

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

SELECT Name FROM `original-glyph-321514.table1.Customers`

UNION ALL

SELECT Name FROM `original-glyph-321514.table1.Sales`

SELECT Name FROM `original-glyph-321514.table1.Customers`

UNION DISTINCT

SELECT Name FROM `original-glyph-321514.table1.Sales`

## # Concept 19 – Intersection

interested in cities where seller is living or customer is living but not both

use intersection

SELECT supplier\_id

FROM suppliers

INTERSECT

SELECT supplier\_id

FROM orders;

SELECT supplier\_id

FROM suppliers

WHERE supplier\_id > 78

INTERSECT

SELECT supplier\_id

FROM orders

WHERE quantity <> 0;

https://www.techonthenet.com/sql/intersect.php

## # Misc Concepts

# How to approach a problem

# 1. Identify the required tables

# 2. How to join the tables if more than 1 tables exist

# 3. Which clauses / concepts will be needed

# 4. Logical flow or structure in mind

# Remaining topics - google it

# 4. Stored procedures - Interview Questions

A stored procedure – Huge task creating A SCHEMA, TABLE, INSERT INTO statements

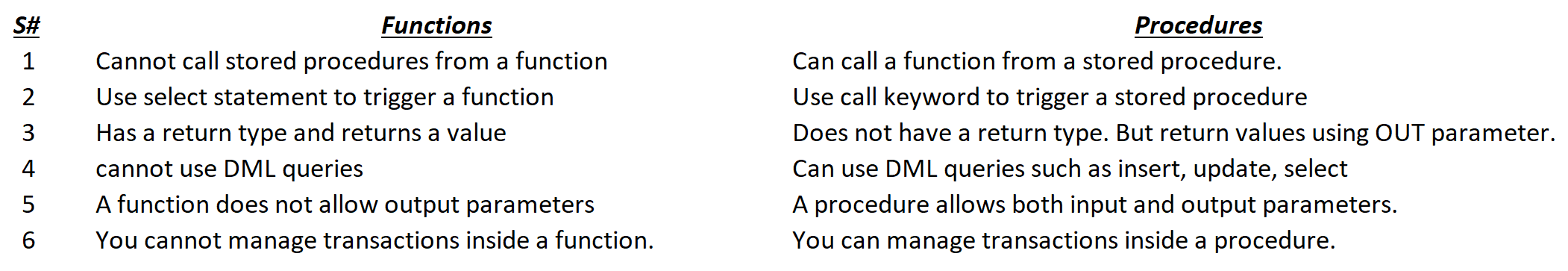
A Stored procedure is a re-usable code. So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it. Return value is optional

# See stored procedure example in Mysql saved queries

Functions – one action

A function is a sub-program to perform an action (complex calculation)

Should return a value. Use select statement to run a function



MySQL supports stored routines (procedures and functions). A stored routine is a set of SQL statements that can be stored in the server. Once this has been done, clients don't need to keep reissuing the individual statements but can refer to the stored routine instead.

When and why would you use a stored procedure?

Use of stored procedures can reduce network traffic between clients and servers, because the commands are executed as a single batch of code. This means only the call to execute the procedure is sent over a network, instead of every single line of code being sent individually.

Why would you use stored procedure in SQL?

By grouping SQL statements, a stored procedure allows them to be executed with a single call. This minimizes the use of slow networks, reduces network traffic, and improves round-trip response time. OLTP applications, in particular, benefit because result set processing eliminates network bottlenecks.

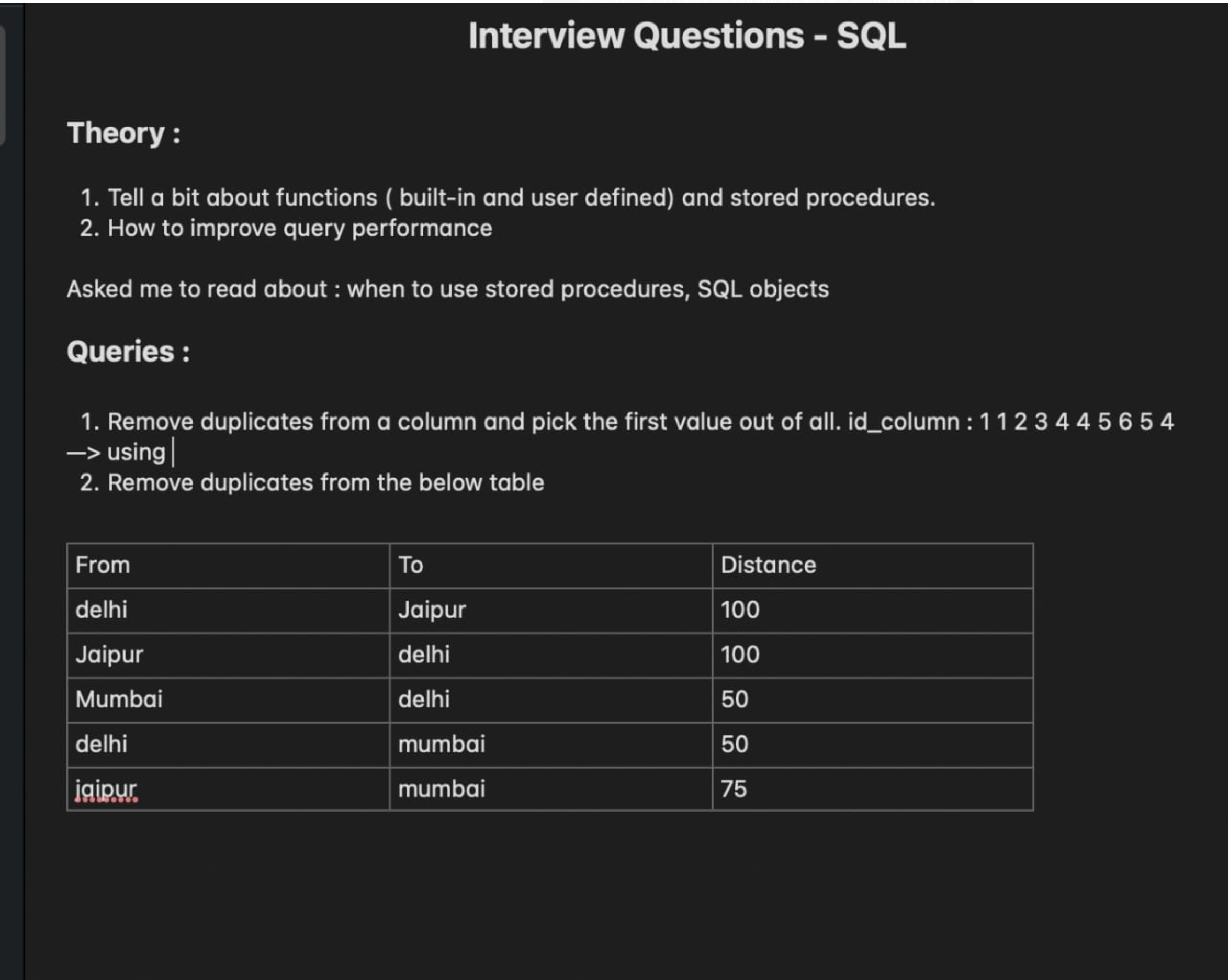
When and why would you use a stored functions?

MySQL stored function is basically a special kind of stored program that returns a single value. We can use stored functions in MySQL to encapsulate mainly simple formulas or business rules that are reusable among SQL statements or stored programs.

# 5. Query Optimization techniques - Interview Questions

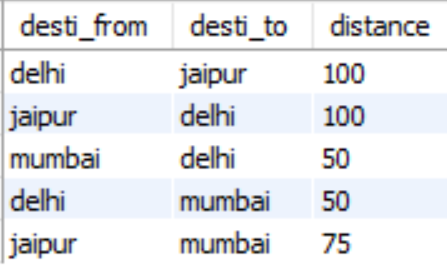
<https://blog.devart.com/how-to-optimize-sql-query.html#missing-indexes>

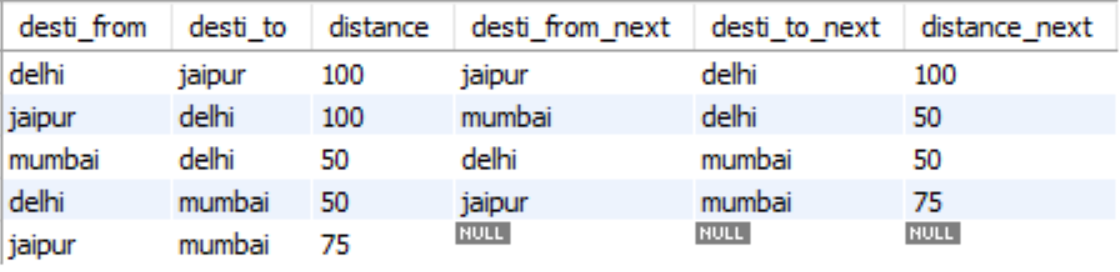
* [12 Query optimization tips for better performance](https://blog.devart.com/how-to-optimize-sql-query.html#query-optimization-tips-for-better-performance)
  + [Tip 1: Add missing indexes](https://blog.devart.com/how-to-optimize-sql-query.html#missing-indexes)
  + [Tip 2: Check for unused indexes](https://blog.devart.com/how-to-optimize-sql-query.html#Non-used-indexes)
  + [Tip 3: **Avoid using multiple OR in the FILTER** predicate](https://blog.devart.com/how-to-optimize-sql-query.html#or-in-join-predicate)
  + [Tip 4: Use wildcards at the end of a phrase only](https://blog.devart.com/how-to-optimize-sql-query.html#use-wildcards)
  + [Tip 5: **Avoid too many JOINs**](https://blog.devart.com/how-to-optimize-sql-query.html#high-table-count)
  + [Tip 6: **Avoid using SELECT DISTINCT**](https://blog.devart.com/how-to-optimize-sql-query.html#avoid-using-select-distinct)
  + [Tip 7: Use SELECT fields instead of SELECT \*](https://blog.devart.com/how-to-optimize-sql-query.html#use-select-fields-instead-of-select-all)
  + [Tip 8: Use TOP to sample query results](https://blog.devart.com/how-to-optimize-sql-query.html#use-top-to-sample-query-results)
  + [Tip 9: Run the query during off-peak hours](https://blog.devart.com/how-to-optimize-sql-query.html#run-query-during-offpeak-hours)
  + [Tip 10: Minimize the usage of any query hint](https://blog.devart.com/how-to-optimize-sql-query.html#minimize-usage-of-query-hint)
  + [Tip 11: Minimize large write operations](https://blog.devart.com/how-to-optimize-sql-query.html#minimize-large-write-operations)
  + [Tip 12: Create joins with INNER JOIN (not WHERE)](https://blog.devart.com/how-to-optimize-sql-query.html#create-joins-with-inner-join)

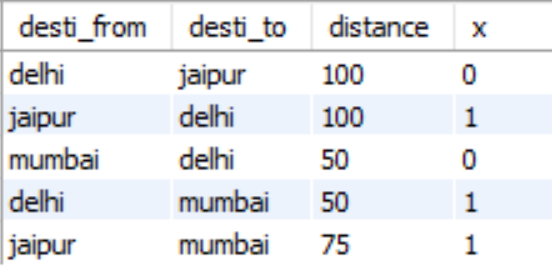


<https://www.interviewbit.com/sql-interview-questions/>

<https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html>

0

1

2

