**techTFQ**

[**https://www.youtube.com/watch?v=AK7\_m-aThfw&ab\_channel=techTFQ**](https://www.youtube.com/watch?v=AK7_m-aThfw&ab_channel=techTFQ)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 25 SQL interview Question \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Question #1:

**What are DDL and DML languages? Give example.**

**Question Level: *Expect this question for beginner and intermediate level role.***

**DDL** stands for Data Definition Language. They include CREATE, DROP, ALTER and TRUNCATE statements.

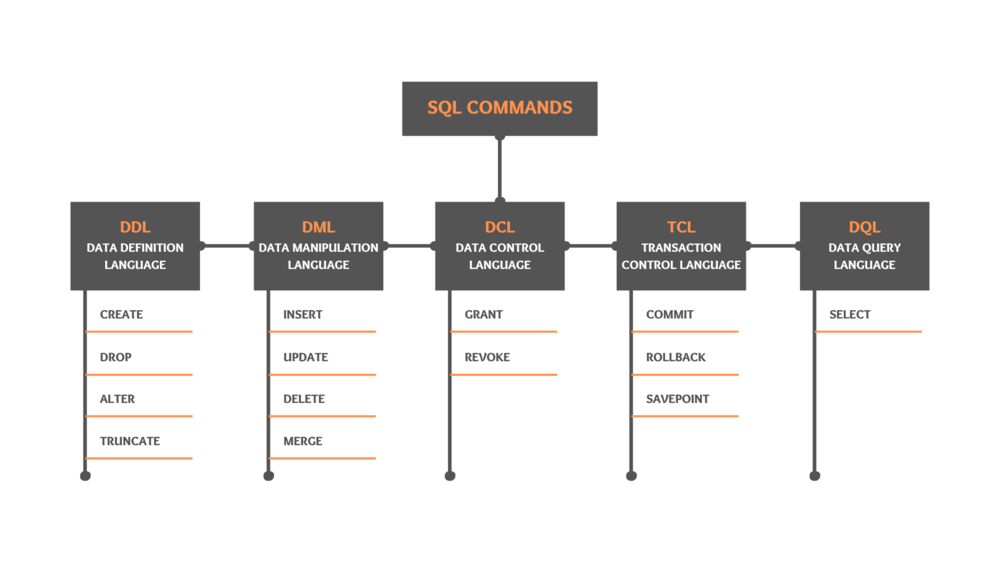
DDL statements are used to create, remove or modify database objects like table. You do not need to commit the changes after running DDL commands.

CREATE statement can be used to create any database objects like tables, views, functions, procedures, triggers etc.

DROP statement can be used to remove any database objects like tables, views, functions, procedures, triggers etc.

ALTER statement can be used to modify the structure of a database objects.

TRUNCATE statement can be used to remove all the data from a table at once.



**DML** stands for Data Manipulation Language. DML includes INSERT, UPDATE, DELETE and MERGE statements.

DML statements are used to add, remove or modify data from database tables. It is mandatory to run the COMMIT command after running a DML statement so as to save the changes to the database (some tools may have auto commit on so you don’t have to manually run the commit command).

INSERT statement will add rows or records to a table.

UPDATE statement will modify the data in the table.

DELETE statement will remove one or multiple rows from a table.

MERGE statement will either do an update or insert to a table based on the available data. If the data is present then it does an update. If data not present then merge will do an insert.

You may also want to learn what is DCL, TCL and DQL languages:

**DCL** stands for Data Control Language. DCL includes GRANT and REVOKE statements.

GRANT statements are used to provide access privileges to a database object to any database or schema.

REVOKE statements are used to remove access privileges from a database object from any database or schema.

**TCL** stands for Transaction Control Language. TCL includes COMMIT, ROLLBACK and SAVEPOINT.

COMMIT statement will permanently save any open transactions in the current session to the database. By transaction, I mean any changes done to any database table using any of the DML statements like INSERT, UPDATE, DELETE and MERGE.

ROLLBACK statement will remove (unsave) any open transactions in the current session to the database. So all un committed transactions in the current session will be lost.

SAVEPOINT statement can be used to create a specific pointer in your session and provide a name to this pointer. You can then either rollback or commit transactions only until this point (savepoint name) rather than committing or rollbacking all the transaction in the session.

**DQL** stands for Data Query Language. It includes only the SELECT statement.

SELECT statement is used to fetch and view data from the database.

Question #2:

**What is the difference between DELETE and TRUNCATE statement?**

**Question Level: *Expect this question for beginner and intermediate level role.***

**Delete** can be used to remove either few or all the records from a table. Whereas **truncate** will always remove all the records from the table. Truncate cannot have WHERE condition.

Delete is a DML statement hence we will need to commit the transaction in order to save the changes to database. Whereas truncate is a DDL statement hence no commit is required.  
For example:

Below statement will delete only the records from employee table where the name is ‘Thoufiq’

DELETE FROM employee WHERE name = ‘Thoufiq’;

COMMIT;

Below statement will delete all records from the employee table.

DELETE FROM employee;

COMMIT;

Below statement will also delete all the records from the employee table. No commit is required here.

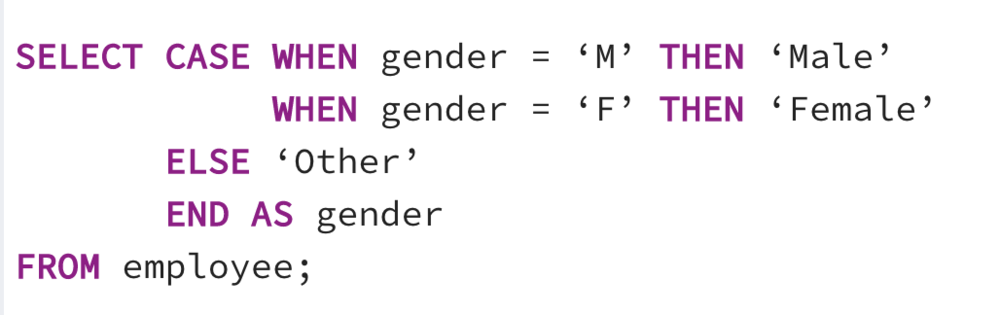
TRUNCATE TABLE employee;

Question #3:

**Why do we use CASE Statement in SQL? Give example**

**Question Level: *Expect this question for beginner and intermediate level role.***

**CASE** statement is similar to IF ELSE statement from any other programming languages. We can use it to fetch or show a particular value based on certain condition.  
For example:



In above example query, we display the gender as ‘Male’ when the gender column in the employee table has value as ‘M’. And if the gender column has value as ‘F’ then we display the value as ‘Female’. If the gender column has values anything other than M or F then the query would return ‘Other’ (which we have defined using the ELSE part in the CASE statement.

Question #4:

**What is the difference between LEFT, RIGHT, FULL outer join and INNER join?**

**Question Level: *Expect this question for beginner, intermediate and senior level role.***

In order to understand this better, let’s consider two tables CONTINENTS and COUNTRIES as shown below. I shall show sample queries considering these two tables.

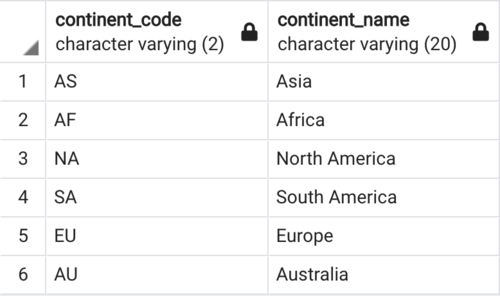


Table Name: **CONTINENTS**

Has data of 6 continents. Please note the continent “Antarctica” is intentionally missed from this table.

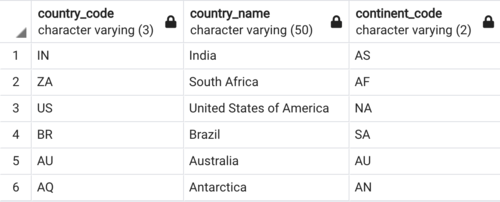
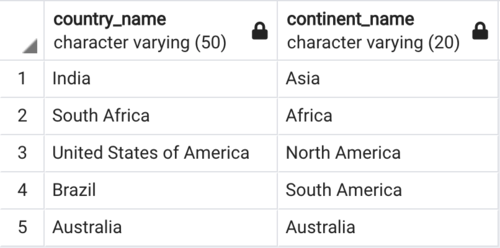


Table Name: **COUNTRIES**

Has data of one country from each continent. Please note that I have intentionally missed to add a country from Europe in this table.

**INNER JOIN** will fetch only those records which are present in both the joined tables. The matching of the records is only based on the columns used for joining these two tables. INNER JOIN can also be represented as JOIN in your SELECT query.



***INNER JOIN Query***

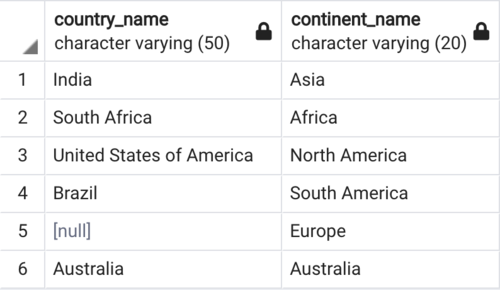
**SELECT cr.country\_name, ct.continent\_name**

**FROM continents ct**

**INNER JOIN countries cr**

**ON ct.continent\_code = cr.continent\_code;**

**LEFT JOIN** will fetch all records from the left table (table placed on the left side during the join) even if those records are not present in right table (table placed on the right side during the join). If your select clause has a column from the right table then for records which are not present in right table (but present in left table), SQL will return a NULL value. LEFT JOIN can also be represented as LEFT OUTER JOIN in your SELECT query.



***LEFT JOIN Query***

**SELECT cr.country\_name, ct.continent\_name**

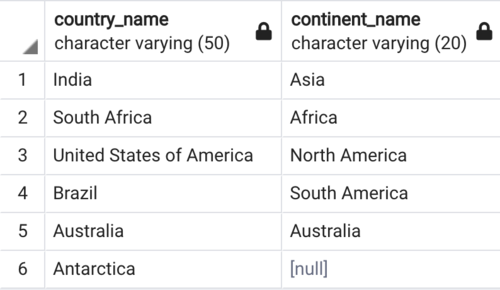
**FROM continents ct**

**LEFT JOIN countries cr**

**ON ct.continent\_code = cr.continent\_code;**

**RIGHT JOIN** will fetch all records from the right table (table placed on the right side during the join) even if those records are not present in left table (table placed on the left side during the join). If your select clause has a column from the left table then for records which are not present in left table (but present in right table), SQL will return a NULL value. RIGHT JOIN can also be represented as RIGHT OUTER JOIN in your SELECT query.

***\*Note:*** LEFT and RIGHT join depends on whether the table is placed on the left side of the JOIN or on the right side of the JOIN.



***RIGHT JOIN Query***

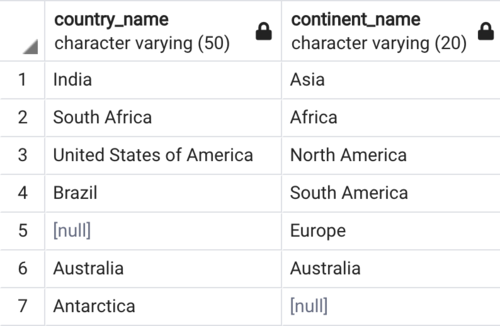
**SELECT cr.country\_name, ct.continent\_name**

**FROM continents ct**

**RIGHT JOIN countries cr**

**ON ct.continent\_code = cr.continent\_code;**

**FULL JOIN** will fetch all records from both left and right table. It’s kind of combination of INNER, LEFT and RIGHT join. Meaning FULL JOIN will fetch all the matching records in left and right table + all the records from left table (even if these records are not present in right table) + all the records from right table (even if these records are not present in left table). FULL JOIN can also be represented as FULL OUTER JOIN in your SELECT query.



***FULL OUTER JOIN Query***

**SELECT cr.country\_name, ct.continent\_name**

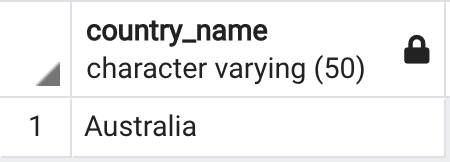
**FROM continents ct**

**FULL OUTER JOIN countries cr**

**on ct.continent\_code = cr.continent\_code;**

Also check for what is SELF join, NATURAL join and CROSS join?

**SELF JOIN** is when you join a table to itself. There is no keyword like SELF when doing this join. We just use the normal INNER join to do a self join. Just that instead of doing an inner join with two different table, we inner join the same table to itself. Just that these tables should have different alias name. Other than this, SELF join performs similar to INNER join.



***SELF JOIN Query***

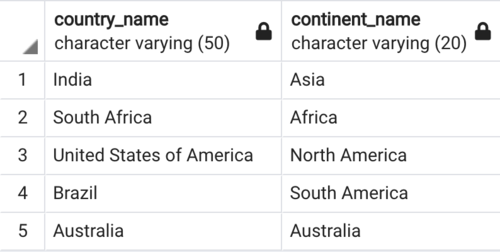
**SELECT cr1.country\_name**

**FROM countries cr1**

**JOIN countries cr2**

**ON cr1.country\_code = cr2.continent\_code;**

**NATURAL JOIN** is similar to INNER join but we do not need to use the ON clause during the join. Meaning in a natural join we just specify the tables. We do not specify the columns based on which this join should work. By default when we use NATURAL JOIN, SQL will join the two tables based on the common column name in these two tables. So when doing the natural join, both the tables need to have columns with same name and these columns should have same data type.



***NATURAL JOIN Query***

**SELECT cr.country\_name, ct.continent\_name**

**FROM continents ct**

**NATURAL JOIN countries cr;**

**CROSS JOIN** will join all the records from left table with all the records from right table. Meaning the cross join is not based on matching any column. Whether there is a match or not, cross join will return records which is basically number of records in left table multiplied by number of records in right table. In other words, cross join returns a Cartesian product.



***CROSS JOIN Query***

**SELECT cr.country\_name, ct.continent\_name**

**FROM continents ct**

**CROSS JOIN countries cr;**

Question #5:

**What is the difference between DISTINCT and GROUP BY?**

**Question Level: *Expect this question for beginner and intermediate level role.***

**DISTINCT** clause will return unique column values. Depending on the list of columns you provide I the DISTINCT clause, it will fetch the unique combination of values for all those combined columns. If you provide just a single column in DISTINCT then it fetches just the unique values in that specific column. Example below:

Below query returns unique employee names from the employee table:

SELECT DISTINCT name FROM employee;

Whereas below query would return unique combination of values based on all the columns from the employee table.

SELECT DISTINCT \* FROM employee;

**GROUP BY** clause will group together the data based on the columns specified in group by. Which will then return just one record for each unique value in the column specified in group by. In other words GROUP BY can also be used to fetch unique records from a table but this is not why group by clause is used for. The main purpose of group by clause is to perform some aggregation (using the aggregate functions like MIN, MAX, COUNT, SUM, AVG) based on the grouped by column values. Example below:

Below query would group together the data from employee table based on name column and then for each name value, it would count how many records have the same name.

SELECT name, COUNT(1) FROM employee GROUP BY name;

Question #6:

**What are the rules to follow when using UNION operator?**

**Question Level: *Expect this question for beginner and intermediate level role.***

**UNION** operator can be used to combine two different SQL Queries. The output would be the result combined from both these queries. Duplicate records would not be returned.

You can combine two queries using UNION operator if they follow the below rules:

* Both queries must return same same no of columns.
* The columns in both the queries must be in same order.
* Data type of all the columns in both the queries must be same.

Question #7:

**What are aggregate functions? Name and explain different types of aggregate functions in SQL?**

**Question Level: *Expect this question for beginner and intermediate level role.***  
Aggregate function can be used to perform calculation on a set of values, which will then return a single value. We can use aggregate function either with GROUP BY clause or without it.

**SUM**: Calculates the sum of given values.

Below query will find the sum of all the salaries from the entire employee table:

SELECT SUM(salary) as total\_salary FROM employee;

Whereas the below query will return the sum of salaries for each department in the employee table:

SELECT dept\_id, SUM(salary) as total\_salary\_per\_dept

FROM employee

GROUP BY dept\_id;

**AVG**: Calculate the average from the given set of values.

Below query will find the average salary from the entire employee table:

SELECT AVG(salary) as average\_salary FROM employee;

Whereas the below query will return the average salary for each department in the employee table:

SELECT dept\_id, AVG(salary) as avg\_salary\_per\_dept

FROM employee

GROUP BY dept\_id;

**MIN**: Find the minimum value in the given set of values.

Below query will find the minimum salary from the entire employee table:

SELECT MIN(salary) as min\_salary FROM employee;

Whereas the below query will return the minimum salary for each department in the employee table:

SELECT dept\_id, MIN(salary) as min\_salary\_per\_dept

FROM employee

GROUP BY dept\_id;

**MAX**: Find the maximum value in the given set of values.

Below query will find the maximum salary from the entire employee table:

SELECT MAX(salary) as max\_salary FROM employee;

Whereas the below query will return the maximum salary for each department in the employee table:

SELECT dept\_id, MAX(salary) as max\_salary\_per\_dept

FROM employee

GROUP BY dept\_id;

**COUNT**: Returns the number of records or rows.

Below query will find the total number of records (or employees) from the entire employee table:

SELECT COUNT(emp\_id) as no\_of\_emp FROM employee;

Whereas the below query will return the maximum salary for each department in the employee table:

SELECT dept\_id, COUNT(emp\_id) as no\_of\_emp\_per\_dept

FROM employee

GROUP BY dept\_id;

Question #8:

**What is the difference between RANK, DENSE\_RANK and ROW\_NUMBER window function?**

**Question Level: *Expect this question for intermediate and senior level role.***

Let’s look at the below sample table to explain the difference.

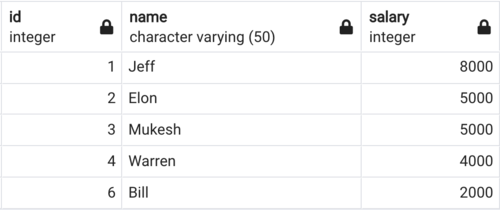


Table Name: **MANAGERS**

Contains salary details of 5 different managers.

**RANK()** function will assign a rank to each row within each partitioned result set. If multiple rows have the same value then each of these rows will share the same rank. However the rank of the following (next) rows will get skipped. Meaning for each duplicate row, one rank value gets skipped.

**DENSE\_RANK()** function will assign a rank to each row within each partitioned result set. If multiple rows have the same value then each of these rows will share the same rank. However the dense\_rank of the following (next) rows will NOT get skipped. This is the only difference between rank and dense\_rank. RANK() function skips a rank if there are duplicate rows whereas DENSE\_RANK() function will never skip a rank.

**ROW\_NUMBER()** function will assign a unique row number to every row within each partitioned result set. It does not matter if the rows are duplicate or not.

By using the managers table, let’s write a query to get the rank, dense rank and row number for each manager based on their salary.

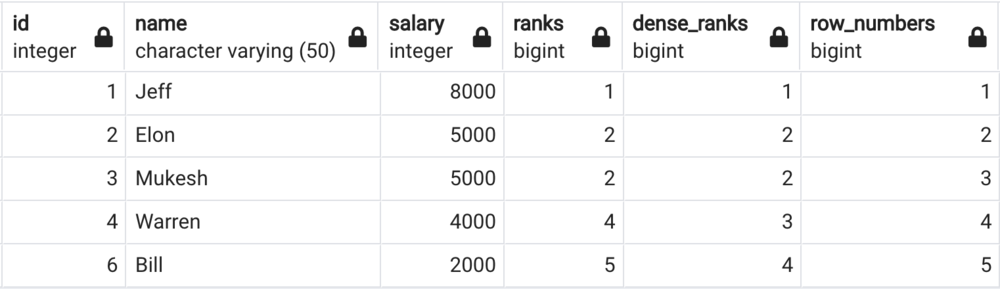
**SELECT \***

**, RANK() OVER(ORDER BY salary DESC) AS ranks**

**, DENSE\_RANK() OVER(ORDER BY salary DESC) AS dense\_ranks**

**, ROW\_NUMBER() OVER(ORDER BY salary DESC) AS row\_numbers**

**FROM managers;**



Result from above query. Check the difference in column values for ranks, dense\_ranks and row\_numbers.

***If you wish to learn more about all the available window functions in SQL then check out my video on my YouTube channel*** [***techTFQ***](https://www.youtube.com/c/techtfq) ***covering all the window function in detail with examples. There are two video,***

***First video covers RANK, DENSE\_RANK, ROW\_NUMBER, LEAD, LAG function. Watch*** [***here***](https://www.youtube.com/watch?t=621s&v=Ww71knvhQ-s)

***Second video covers CUME\_DIST, NTILE, NTH\_VALUE, PERCENT\_RANK, FIRST\_VALUE, LAST\_VALUE and also Frame clause. Watch*** [***here***](https://www.youtube.com/watch?t=2398s&v=zAmJPdZu8Rg)

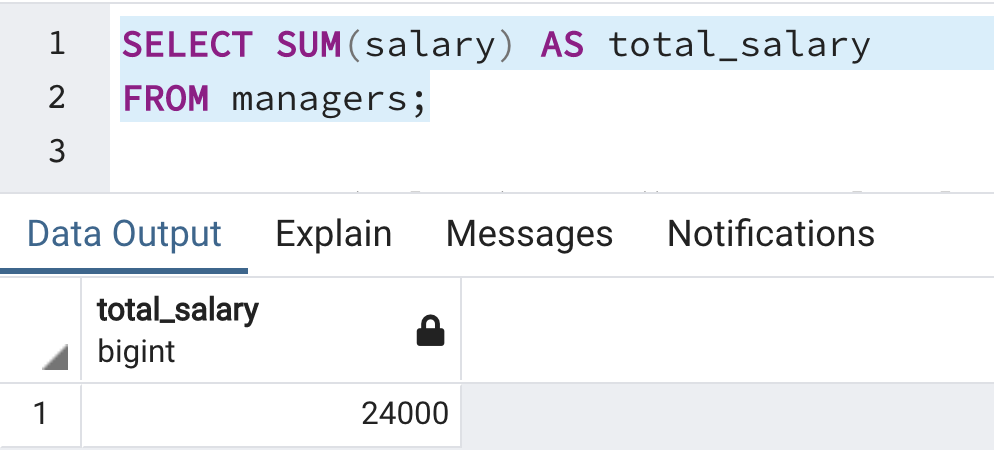
Question #9:

**Can we use aggregate function as window function? If yes then how do we do it?**

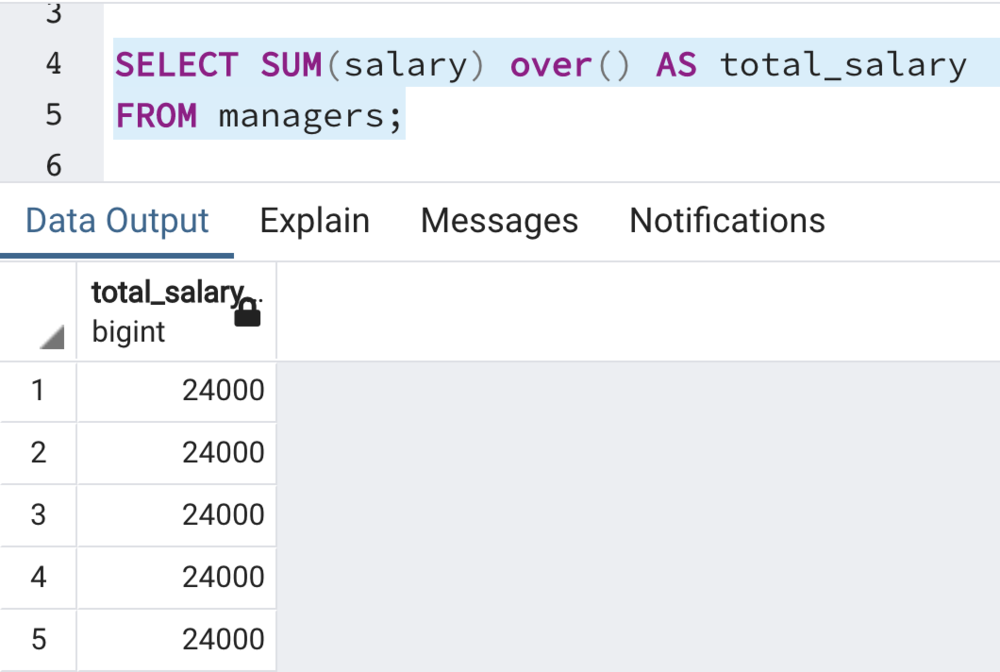
**Question Level: *Expect this question for intermediate and senior level role.***

Yes, we can use aggregate function as a window function by using the OVER clause. Aggregate function will reduce the number of rows or records since they perform calculation of a set of row values to return a single value. Whereas window function does not reduce the number of records.

Below two images shows the output of SUM aggregate function used when used as a normal aggregate function and when used as a window function. We are using the same managers table as used in previous question.



Using SUM as aggregate function returns just 1 record.



Using SUM as window function (with the OVER clause), does not reduce the no of records. Please note it’s not mandatory to use ORDER BY or PARTITION BY inside the OVER clause as shown here.

Question #10:

**How can you convert a text into date format? Consider the given text as “31-01-2021“.**

**Question Level: *Expect this question for beginner intermediate and senior level role.***

Different RDBMS would have different date functions to convert a text to date format. Let’s see the date functions to be used in the four most popular RDBMS as of today.

**Oracle:**

SELECT TO\_DATE('31-01-2021', 'DD-MM-YYYY') as date\_value FROM DUAL;

**MySQL:**

SELECT DATE\_FORMAT('31-01-2021', '%d-%m-%Y') as date\_value;

**Microsoft SQL Server (MSSQL):**

SELECT CAST('31-01-2021' as DATE) as date\_value;

**PostgreSQL:**

SELECT TO\_DATE('31-01-2021', 'DD-MM-YYYY') as date\_value;

***\*Note:*** *Please note, there will be few other date functions in each of these RDBMS to perform the same operation. The above functions are just one of such functions.*

Question #11:

**Imagine there is a FULL\_NAME column in a table which has values like “Elon Musk“, “Bill Gates“, “Jeff Bezos“ etc. So each full name has a first name, a space and a last name. Which functions would you use to fetch only the first name from this FULL\_NAME column? Give example.**

**Question Level: *Expect this question for beginner and intermediate level role.***

Again there would be different functions in different RDBMS to perform the same operation. Let’s solve this using the four most popular RDBMS.

**Oracle:** We can use SUBSTR() function to get a sub string from a given text based on the start and end position. And we can use INSTR() function to find the position of a particular string in the given text. Example query below:

SELECT SUBSTR(full\_name, 1, INSTR(full\_name, ' ', 1, 1) - 1) as first\_name FROM dual;

**MySQL:** We can use SUBSTRING() function to get a sub string from a given text based on the start and end position. And we can use INSTR() function to find the position of a particular string in the given text. Example query below:

SELECT SUBSTRING(full\_name, 1, INSTR(full\_name, ' ', 1, 1) - 1) as first\_name FROM dual;

**Microsoft SQL Server (MSSQL):** We can use SUBSTRING() function to get a sub string from a given text based on the start and end position. And we can use CHARINDEX() function to find the position of a particular string in the given text. Example query below:

SELECT SUBSTRING(full\_name, 1, CHARINDEX(' ', full\_name) - 1) as first\_name;

**PostgreSQL:** We can use SUBSTR() function to get a sub string from a given text based on the start and end position. And we can use POSITION() function to find the position of a particular string in the given text. Example query below:

SELECT SUBSTR(full\_name, 1, POSITION(' ' IN full\_name) - 1) as first\_name;

Question #12:

**What are subqueries? Where can we use them?**

**Question Level: *Expect this question for beginner and intermediate level role.***

A SELECT query statement which is placed inside another SELECT query is termed as a subquery. Subquery can also be termed as inner query.

The SELECT query which holds the subquery can be termed as main query or outer query. Subquery can also reside within a INSERT, UPDATE, DELETE statement or inside another subquery.

In a SELECT statement, subquery may occur in the SELECT clause, FROM clause or the WHERE clause.

When a query statement containing subquery execute, the inner query or the subquery will execute first. The outer query will then use the result from the subquery while processing the outer query.

**Example:**

**SELECT \* FROM continents ct**

**WHERE ct.continent\_code IN (SELECT cr.continent\_code FROM countries cr);**

*In the above query, the text highlighted in* ***RED*** *is the subquery whereas the query highlighted in* ***BLUE*** *is termed as the outer query or the main query.*

Question #13:

**Is it good to have the same subquery multiple times in your query? If no then how can you solve this?**

**Question Level: *Expect this question for intermediate and senior level role.***

It’s not a good practice to use the same subquery multiple times in your query. Repeating the same subquery multiple times in your query can impact the query performance (since the same query would execute multiple times) and also becomes difficult to maintain (since any change to the subquery will need to be made in multiple different places).

We can avoid this by using the WITH clause. We can place the subquery just once inside the WITH clause and then use this multiple times in our query. This way SQL will execute the subquery just once (which is at the start of the query execution).

***If you wish to learn more about WITH clause in SQL, understands its advantages and when to use them then check out my video on this topic in my YouTube channel*** [***techTFQ***](https://www.youtube.com/c/techTFQ)***. Click*** [***here***](https://www.youtube.com/watch?t=715s&v=QNfnuK-1YYY) ***to watch that video.***

Question #14:

**Difference betwen WHERE and HAVING clause**

**Question Level: *Expect this question for beginner and intermediate level role.***

**WHERE** clause is used to filter records from the table. We can also specify join conditions between two tables in the WHERE clause. If a SQL query has both WHERE and GROUP BY clause then the records will first get filtered based on the conditions mentioned in WHERE clause before the data gets grouped as per the GROUP BY clause.

Whereas **HAVING** clause is used to filter records returned from the GROUP BY clause. So if a SQL query has WHERE, GROUP BY and HAVING clause then first the data gets filtered based on WHERE condition, only after this grouping of data takes place. Finally based on the conditions in HAVING clause the grouped data again gets filtered.

Question #15:

**What are indexes? Why do we use it?**

**Question Level: *Expect this question for intermediate and senior level role.***

**Index** is a database object which is applied on one or more columns of a table. When a column (or list of columns) from the table is Indexed, database creates a pointer to each value stored in that column. This significantly improves the query execution time since the database will have a more efficient way to find a particular value from the column based on its index.

Imagine you have a table with one million records and there is an ID column in it along with many other columns. Let’s say you are given a task to write a SQL query which is expected to return just 100 records from this table.

Imagine if you did not create an index on this ID column then the SQL Query which you write will have to scan through all the one million records to find the desired 100 records. This is going to be a very slow and the performance of the query would take a blow.

Now, let’s imagine you did create an index on the ID column, then SQL will have a pointer to every value stored in this column hence the SQL Query you write will be much faster since the database will know where to find the 100 required records by referring to the index of this ID column.

This in a nutshell is what index is used for. Basically, Index creates a pointer to each value in the column which in turn helps in finding any specific value from this column in a much faster way.

Please note, different indexes have different functionalities and will behave differently from each other but in a nutshell index is used to identify any value faster from the table column(s).

Question #16:

**What are steps you would take to tune a SQL query?**

**Question Level: *Expect this question for senior level role.***

When it comes to tuning SQL Queries, below are the list of steps you need to consider:

1. **Check the SQL Query.**

First thing is to write the SQL Query in the best way possible.

* + Make sure all the table joins are correct and all the filter conditions are applied as intended.
  + Also check for any cartesian joins that may happen unintentionally.
  + Avoid any repeated subqueries by using a WITH clause.
  + If using tables with huge list of columns then make sure to only fetch columns which are required for the current query.
  + If required check the columns used in join conditions are similar to how index are created. Just to make sure you give the best possible chance for the optimiser to use indexes.

1. **Check if index is created for the desired columns.**
   * Make sure correct indexes are created on the desired columns. Following the correct type of indexes.
   * Avoid creating unnecessary indexes.
2. **Check if table statistics are upto date.**

Statistics will help the optimizer to have the upto date information about the table which in turn helps the optimiser to create the best possible explain plan.

* + Check if statistics are generated for all the used tables..
  + If there were some updates to table structure or data then its better to create statistics again.

1. **Check the explain plan.**

When we execute a SQL Query, the first thing database does it to parse the query i.e. it will validate the query by looking for any syntax errors and also by checking the validity of the tables and its columns.

The next thing that happens is that the database optimizer will generate an explain plan for the query.

Explain plan is something like a step by step guide on how the query execution will happen. Explain plan will mention which index to use and what sort of joins to follow. So if the explain plan is not using a particular index from a huge table then this is a good indication why the query performance may be slow.

You may then need to again follow the steps from the top to improve the written query.

Question #17:

**What is the difference between primary key, unique key and foreign key**

**Question Level: *Expect this question for beginner and intermediate level role.***

Primary key, unique key and foreign key are constraints we can create on a table.

When you make a column in the table as **primary key** then this column will always have unique or distinct values. Duplicate values and NULL value will not be allowed in a primary key column. A table can only have one primary key. Primary key can be created either on one single column or a group of columns.

When you make a column in the table as **unique key** then this column will always have unique or distinct values. Duplicate values will not be allowed. However, NULL values are allowed in a column which has unique key constraint. This is the major difference between primary and unique key.

**Foreign key** is used to create a master child kind of relationship between two tables. When we make a column in a table as foreign key, this column will then have to be referenced from another column from some other table.

Imagine we have two tables A and B. Both have just 1 column let’s call it COLUMN\_1. If we create foreign key in COLUMN\_1 of table A which references the COLUMN\_1 from table B then the only values COLUMN\_1 in table A can have is the values which are already present in COLUMN\_1 of table B.

This means table B becomes the master table and table A is the child table. COLUMN\_1 of Table A can only have values which are already present in COLUMN\_1 of table B.

Question #18:

**What is the difference between a view and a synonym?**

**Question Level: *Expect this question for beginner and intermediate level role.***

**View** is a database object which is created based on a SQL Query. It’s like giving a name to the results returned from a SQL Query and storing it in the database as a view.

If the query result changes then the data in view also changes. View is directly linked to the SQL Query over which it was created.

**Synonym** on the other hand is just an alias or an alternate name that you can provide to any database objects such as tables, views, sequences, procedures etc.

Synonym is created for a single database object whereas view can be created on a query where the query may have been formed by multiple tables.

Question #19:

**When can a function NOT be called from SELECT query?**

**Question Level: *Expect this question for beginner and intermediate level role.***

* If the function includes DML operations like INSERT, UPDATE, DELETE etc then it cannot be called from a SELECT query. Because SELECT statement cannot change the state of the database.

Question #20:

**What is a trigger?**

**Question Level: *Expect this question for intermediate and senior level role.***

**Trigger** is a database object which is similar to a stored procedure which will automatically get invoked or executed when the specified event occurs in the database.

The most common type of triggers are DML triggers, DDL triggers and Database triggers (also referred as Logon triggers).

DML triggers are invoked when a DML operation (INSERT, UPDATE, DELETE) occurs on the respective table (table on which the trigger was created). Trigger can be configured to invoke either before the DM operation or after the DML operation.

DDL triggers are invoked when a DDL operation (CREATE, ALTER, DROP) occurs on the respective table (table on which the trigger was created).

Database trigger is invoked when the database session is established or shut down.

Question #21:

**What is the difference between a views and a materialized views?**

**Question Level: *Expect this question for intermediate and senior level role.***

Similar to views, **materialized views** are also database objects which are formed based on a SQL Query however unlike views, the contents or data of the materialized views are periodically refreshed based on its configuration.

The contents of view will get updated automatically when the underlying table (forming the query) data gets changed. However, materialised views can be configured to refresh its contents periodically or can be manually refreshed when needed.

Creating materialized views can be a very good approach for performance tuning especially when dealing with remote tables.

Question #22:

**What is MERGE statement?**

**Question Level: *Expect this question for beginner, intermediate and senior level role.***

**Merge** is part of the DML commands in SQL which can be used either perform INSERT or UPDATE based on the data in the respective table.

If the desired data is present then merge will update the records. If desired data is not present then merge will insert the records.

Sample merge statement is shown below. Here if the managers and directors table have matching records based the ID field then UPDATE command will be run else if there are no matching records then INSERT statement will be executed.

MERGE INTO managers m

USING directors d ON (m.id = d.id)

WHEN MATCHED THEN

UPDATE SET name = 'TEST'

WHEN NOT MATCHED THEN

INSERT VALUES (d.id, d.name, 0);

Question #23:

**Which function can be used to fetch yesterdays date? Provide example.**

**Question Level: *Expect this question for beginner and intermediate level role.***

Different RDBMS would have different date functions to add or subtract a day value from the current date. Let’s see the date functions to be used in the four most popular RDBMS.

**Oracle:** In Oracle we can simple subtract an integer from a date value hence we do not need to use a function to find yesterday’s date as shown in below query. SYSDATE will return today’s date.

SELECT SYSDATE - 1 as previous\_day FROM DUAL;

**MySQL:** SYSDATE() will returns today’s date along with timestamp value.

Below query would return the date and timestamp.

SELECT DATE\_SUB(SYSDATE(), INTERVAL 1 DAY) as previous\_day;

Below query only returns the date.

SELECT DATE\_SUB(CAST(SYSDATE() AS DATE), INTERVAL 1 DAY) as previous\_day;

**Microsoft SQL Server (MSSQL):** GETDATE() will returns today’s date along with timestamp value.

Below query would return the date and timestamp.

SELECT DATEADD(DAY, -1, GETDATE());

Below query only returns the date.

SELECT DATEADD(DAY, -1, CAST(GETDATE() AS DATE));

**PostgreSQL:** In PostgreSQL as well, we can simple subtract an integer from a date value hence we do not need to use a function to find yesterday’s date as shown in below query. CURRENT\_DATE will return today’s date.

SELECT CURRENT\_DATE - 1 as previous\_day FROM DUAL;

***\*Note:*** *Please note, there will be few other date functions in each of these RDBMS to perform the same operation. The above functions are just one of such functions.*

Question #24:

**What is the difference between a function and a procedure?**

**Question Level: *Expect this question for beginner and intermediate level role.***

* **Function** should always return a value whereas for a **procedure** it’s not mandatory to return a value.
* Function can be called from a SELECT query whereas procedure cannot be called from a SELECT query.
* Function is generally used to perform some calculation and return a result. Whereas procedure is generally used to implement some business logic.

Question #25:

**What is PRAGMA AUTONOMOUS TRANSACTION?**

**Question Level: *Expect this question for senior level role.***

We can declare the stored program like a procedure as a **PRAGMA AUTONOMOUS TRANSACTION** which means that any transaction committed or rolled back in this procedure will not impact any open transactions in the program from where this procedure was called from.

To understand this further, let’s imagine we have two procedures, pr\_main and pr\_log. PR\_MAIN is a normal procedure whereas PR\_LOG is declared as PRAGMA AUTONOMOUS TRANSACTION.

In the execution block of PR\_MAIN, let’s imagine we do some DML operations like INSERT 100 records into a test table and then within the exception handling block of PR\_MAIN we call the PR\_LOG procedure. PR\_LOG procedure will do some inserts into the log table and then do some commits.

Now when we call the PR\_MAIN procedure, if there was an unexpected exception then the PR\_LOG procedure will get called. However, any commit done in the PR\_LOG procedure will not impact the open transactions in PR\_MAIN procedure because PR\_LOG is declared as autonomous transaction so the commits and rollback within a autonomous transaction procedure only impacts its own transaction.

It’s like autonomous transaction procedure will have its own database session so any transactions commit or rollback only impacts its internal database session.