DATABASE AND SQL

BY

JANARDHANA BANDI

CONTENT

- What is Data, DataBase?
- What is SQL
- Data Types
- Types of SQL Statements
- Operators
- DDL and DML Commands
- SELECT Statement
- WHERE Clause
- String Matching
- Pre-defined Functions
- Aggregate and Window functions
- Subqueries
- Case and Decode
- Constraints
- Null Handling
- Joins
- Set Operators
- CTEs
- Pivot and Unpivot
- Indexes
- Views
- Access Controls
- Performance Tuning
- Advanced SQL Queries
- What Next?

DATABASE BASICS

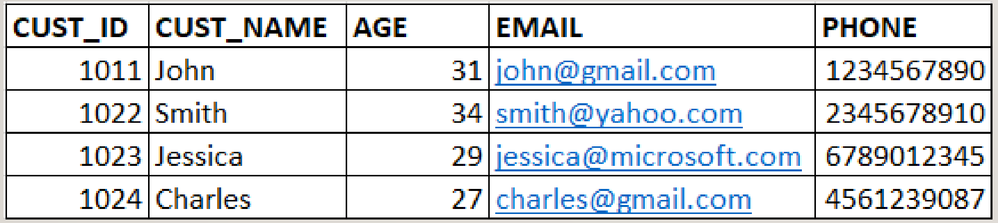
WHAT IS DATA, DATABASE?

- **Data** is any thing that can be represented as text, numbers, or multimedia(image, sound, video etc.)

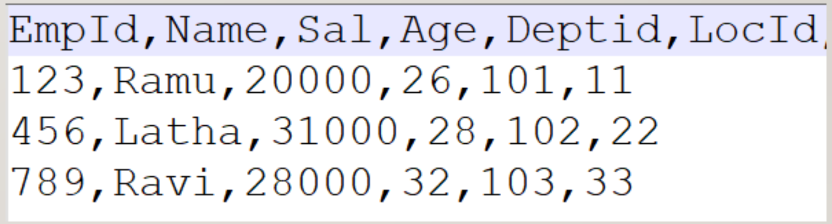
  Examples: ‘Technology’, 243589, ‘2022-09-20’

- **Dataset** is a structured collection of data that contains information of an entity

  Examples:



CUST_ID	CUST_NAME	AGE	EMAIL	PHONE
1011	John	31	john@gmail.com	1234567890
1022	Smith	34	smith@yahoo.com	2345678910
1023	Jessica	29	jessica@microsoft.com	6789012345
1024	Charles	27	charles@gmail.com	4561239087

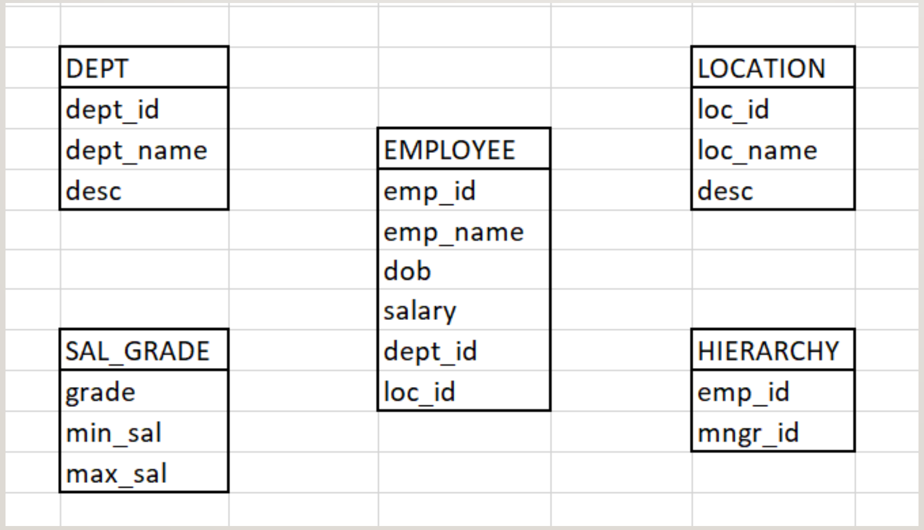


```
EmpId,Name,Sal,Age,Deptid,LocId,
123,Ramu,20000,26,101,11
456,Latha,31000,28,102,22
789,Ravi,28000,32,103,33
```

WHAT IS DATA, DATABASE?

- **Database** is a system that allow users to store, organize and analyze data. It is an organized collection of data stored in multiple related datasets.

  Example: Employee Database contains below datasets.



USES OF DATABASE?

- Can handle huge amounts of data
- Easy to combine different datasets
- Analyze the data for decision making
- Database is backend for many applications
- Easy to share the data
- Data security

WHAT IS RDBMS?

RDBMS – Relational Data Base Management Systems.

- A relational Database is database that stores data in a table form, containing columns and rows.
- RDBMS is a system used to store, manage, query, and retrieve data stored in a relational databases.
- The RDBMS provides an interface between users and applications and the database.

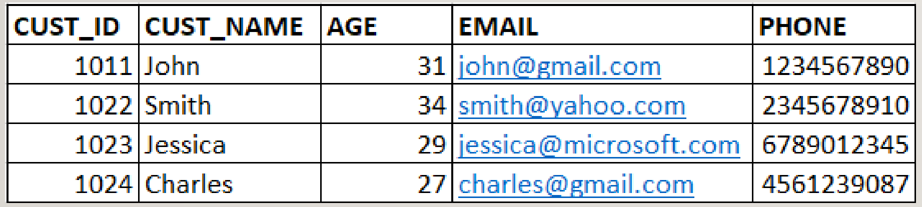
LIST OF RDBMS SYSTEMS

- Oracle
- MySQL
- Teradata
- SQL Server
- PsotgreSQL
- Vertica
- IBM DB2
- Snowflake (Cloud Database)

WHAT IS TABLE?

- **Table** is a database object that is a collection of related data entries, where data is organized in rows and columns.
- Rows are also know as records
- Columns are also known as fields

Example: Customer table



CUST_ID	CUST_NAME	AGE	EMAIL	PHONE
1011	John	31	john@gmail.com	1234567890
1022	Smith	34	smith@yahoo.com	2345678910
1023	Jessica	29	jessica@microsoft.com	6789012345
1024	Charles	27	charles@gmail.com	4561239087

Metadata: Data about data, it gives the info that what data is stored in a table.

HOW TO LOAD DATA INTO DATABASE TABLES?

- Using manual insert statements
- Importing data from files using SQL clients(Dbeaver, Squirrel, Oracle SQL dev, Teradata SQL Assistant, TOAD)
- Load using ETL tools or scripts

HOW TO ACCESS THE DATA FROM DATABASE?

SQL – Structured Query Language

- We can access and manipulate the data from relational databases by using SQL.
- SQL is a programming language used to communicate with these RDBMS systems like Oracle, Teradata, MySQL etc.
- SQL is an ANSI/ISO standard, all databases support ANSI SQL so the syntaxes, keywords and functions or similar in all databases.
- We can build SQL queries and procedures to operate with the data present in Database.
- Queries can be written by using Operators, Keywords, Expressions, Functions..

DATA TYPES

DATA TYPES

- Data types specify the type of data we storing or querying.
- There are many data types to store
  - Characters or strings
  - Numeric values
  - Date and Time values
  - Boolean (True or False)
  - Images, audio, video data

DATA TYPES

CHAR(size)	A FIXED length string, it can contain letters, numbers, and special characters. The size parameter specifies the column length in characters.
VARCHAR(size)	A VARIABLE length string, it can contain letters, numbers, and special characters. The size parameter specifies the maximum string length in characters.
INT	It is used for the integer value. Its range is from -2147483648 to 2147483647.
INTEGER	Same as INT
BIGINT	A large integer. Its range is from -9223372036854775808 to 9223372036854775807.
FLOAT(size, d)	A floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter.
DOUBLE(size, d)	A normal-size floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter
NUMBER(size, d)	Number can be used instead of Integers and decimals
DECIMAL(size, d)	Same as Number

DATA TYPES

DATE	It is used to specify date format YYYY-MM-DD
TIME	It is used to specify the time format. Its format is hh:mm:ss
DATETIME	It is used to specify date and time combination. Its format is YYYY-MM-DD hh:mm:ss Supported range for DATETIME is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP	It is used to specify the timestamp. Supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC.
BOOLEAN	Just to indicate True or False, 0 is False, any non-zero number is True

OPERATORS IN SQL

OPERATORS IN SQL

SQL Arithmetic Operators:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo

SQL Comparison Operators:

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<> or !=	Not equal to

OPERATORS IN SQL

SQL Logical Operators:

Operator	Description
AND	TRUE if all the conditions separated by AND is TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
NOT	Displays a record if the condition(s) is NOT TRUE
ALL	TRUE if all of the subquery values meet the condition
ANY	TRUE if any of the subquery values meet the condition
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records

OPERATORS IN SQL

SQL Bit-wise Operators:

Operator	Description
&	Bit-wsie AND
|	Bit-wsie OR
^	Bit-wsie Exclusive OR
~	Bit-wsie NOT

SQL Set Operators:

UNION
UNION ALL
INTERSECT
MINUS

HOW TO PRACTICE?

3 WAYS

1. Using Online SQL Editors
2. By Installing Postgre SQL
3. From Snowflake – Easiest and Advanced

EMPLOYEES DATA FOR PRACTICE

Set of 7 employee tables given for Practice.

Create and Insert data into those table for our practice

Given 3 versions of Tables(Online, Postgre, Snowflake)

I strongly suggest to practice in Snowflake as it is very advanced and cloud database

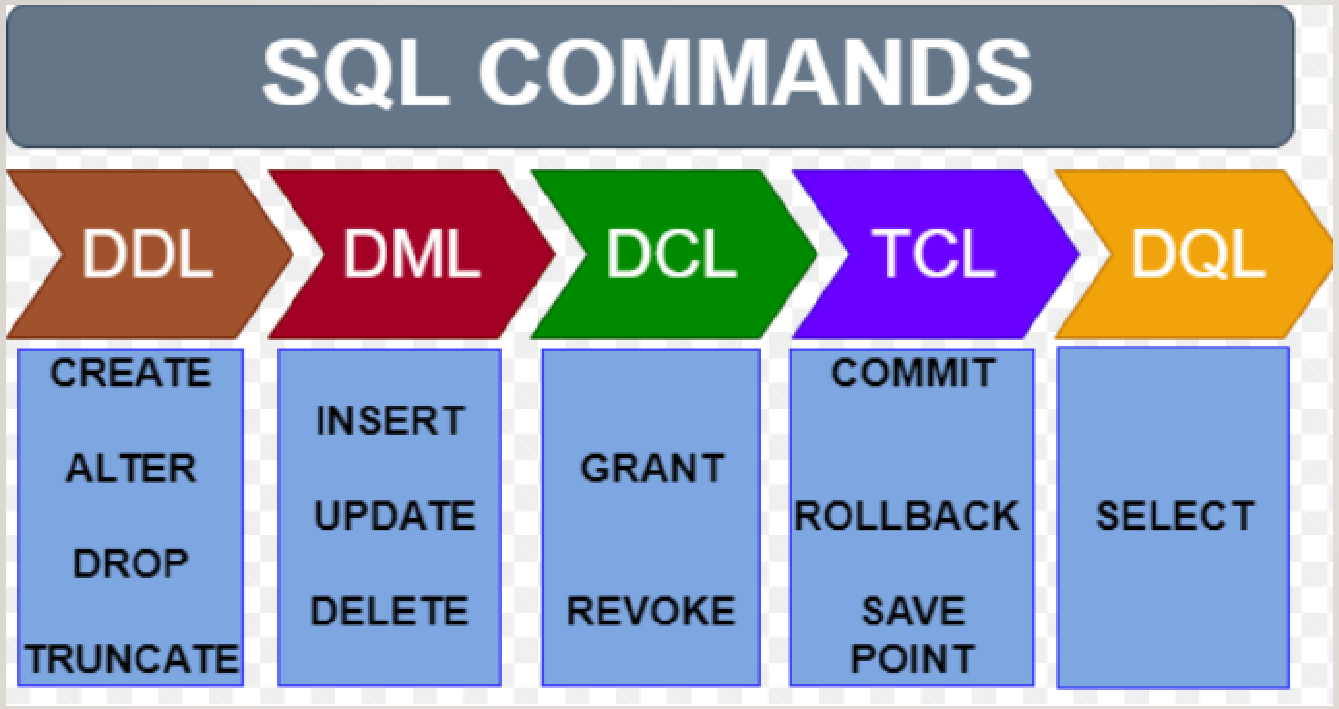
TYPES OF SQL STATEMENTS

TYPES OF SQL STATEMENTS

There are 5 types of SQL statements.

1. DDL – Data Definition Language
2. DML – Data Manipulation Language
3. DCL – Data Control Language
4. TCL – Transaction Control Language
5. DQL – Data Query Language

TYPES OF SQL STATEMENTS



SQL COMMANDS

DDL	DML	DCL	TCL	DQL

CREATE
ALTER
DROP
TRUNCATE
	INSERT
	UPDATE
	DELETE
		GRANT
		REVOKE
			COMMIT
			ROLLBACK
			SAVE POINT
				SELECT

DDL (DATA DEFINITION LANGUAGE)

DDL: Deals with the definition of database objects, these commands used to create, modify and remove the structure of the database objects like tables, views, functions etc.

Below are the DDL commands

- Create
- Alter
- Drop
- Truncate

CREATE

CREATE: To creates objects in the database like tables, indexes, functions, views, procedures and triggers etc.

Syntax: CREATE TABLE *TAB_NAME*

```
( Column1 DataType,
  Column 2 DataTpe,
  …….
);
```

CREATE VIEW *VIEW_NAME* as SELECT … FROM *TAB_NAME*;

ALTER

ALTER: Used to alter the structure of table like adding new columns, dropping existing column, changing the length or datatype of a column and to rename the table.

Syntax:

ALTER TABLE *TABLE_A* RENAME TO *TABLE_B*;

ALTER TABLE *TABLE_NAME* ADD new_column datatype;

ALTER TABLE *TABLE_NAME* MODIFY COLUMN column_name;

ALTER TABLE *TABLE_NAME* DROP COLUMN column_name;

DROP

DROP: Used to drop database objects like tables, views, functions, procedures, indexes etc.

Syntax:

DROP TABLE *TABLE_NAME*; or DROP *TABLE_NAME*;

DROP VIEW *VIEW_NAME*;

DROP FUNCTION *FUN_NAME*;

DOPR INDEX *INDEX_NAME*;

TRUNCATE

TRUNCATE: It deletes all rows from the table and we can’t rollback the deleted records.

Syntax:

TRUNCATE TABLE *TABLE_NAME*;

Now, Lets Practice DDL Statements..

DML (DATA MANIPULATION LANGUAGE)

DML: DML statements are used to add the data, modify data and to delete the data from tables.

Below are the DML commands

- Insert
- Update
- Delete

INSERT

INSERT: To insert data into a table, by using INSERT we can

1. Insert single record manually
2. Insert multiple records manually
3. Insert bulk data from other processes
4. Insert data from one table to another table

Syntax: INSERT INTO *table_name*(columns)
VALUES(……);

INSERT INTO *table_a*(columns) SELECT columns FROM *table_b*;

UPDATE

UPDATE: To modify data from the table. We use SET keyword for updating data.
By using UPDATE we can

1. Update one or more columns data
2. Update one or more records based on condition
3. Update entire table

Syntax: UPDATE TABLE table_name SET column1=… , column2=…;

```
UPDATE TABLE table_name  SET column1=…
WHERE condition;
```

DELETE

DELETE: To remove records from table, by using DELETE we can

1. Delete one or more records based on condition
2. Delete all records

Syntax: DELETE FROM *table_name*;

DELETE FROM *table_name* WHERE condition;

INTERVIEW QUESTION

Important interview question:

What is the difference between Delete and Truncate?

Ans:

TCL (TRANSACTION CONTROL LANGUAGE)

TCL: A transaction is set of SQL statements executed on the data stored in Database.
TCL commands are used to manage those transactions in the database and mainly to control the transactions made by DML commands like INSERT, UPDATE and DELETE.

COMMIT
ROLLBACK
SAVEPOINT

TCL (TRANSACTION CONTROL LANGUAGE)

COMMIT: It is used to permanently save any transaction into the database.

Syntax: COMMIT;

ROLLBACK: It is used to rollback the transactions made after last commit.

Syntax: ROLLBACK;

SAVEPOINT: It is used to save transactions up to a point and we can rollback the transactions made after that point.

```
SAVE TRANSACTION SP1;
transactions….
ROLLBACK TRANSACTION SP1;
```

DCL (DATA CONTROL LANGUAGE)

DCL: DCL commands are used to handle access privileges for users and roles on database objects to. Only database administrator's & owner's of the database object can provide and remove the privileges.

GRANT: It is used to provide permissions on database objects to users and roles.

Syntax: GRANT *privilege_name* ON *db_object* TO *user|role*

REVOKE: It is used to remove permissions on database objects to users and roles.

Syntax: REVOKE *privilege_name* ON *db_object* FROM *user|role*

Privileges: Privileges let the user know what are all operations that user can perform on that object.

Table/View level privileges: SELECT, INSERT, UPDATE, DELETE and ALL

Schema/Database level privileges: CREATE, ALTER and DROP

DQL (DATA QUERY LANGUAGE)

DQL:

- Select is the only DQL command we use to retrieve data from Database.
- Select is the very important command that we use to fetch data from Tables and Views.
- Select command plays key role in Data analysis.
- Syntax: SELECT **/column_list* FROM *table/view_name*;

Let's see what are the things we can do with SELECT..

SELECT STATEMENT

Select:

- Select command is used to retrieve data from Tables and Views in Database.
- Syntax: SELECT **/column_list* FROM *table/view_name*;

With Select Statement we can

1. Fetch all columns(*) or only required columns from a table
2. Fetch all records or subset of records using where filter condition
3. Write expression
4. Call pre-defined functions and user defined functions
5. Calculate aggregations
6. Order data in ascending or descending order

EMPLOYEES DATABASE

Lets Create Employees data related tables and insert sample data for our practice.

WHERE AND ORDER BY

WHERE CLAUSE

Where:

- We use WHERE clause to filter data from tables/views.
- We can mention one or more conditions in WHERE clause based on our requirements.
- The SELECT statement displays all the records satisfying the conditions mentioned in WHERE.

Syntax: SELECT **/column_list* FROM *table/view_name* WHERE *condition*;

Note: SQL keywords are case in-sensitive, we can use select or SELECT, WHERE or where.
But the data we are operating is case sensitive, so ‘a’ is different from ‘A’ in data.

ORDER BY CLAUSE

Order by:

- To order the data in ascending order or descending order.
- Default is ascending order, have to specify ‘desc’ for descending order.
- Can be applied on any data type(Numeric, Strings, Date)
- Can be applied on multiple columns

Syntax:

SELECT **/column_list* FROM *table/view_name* ORDER BY *Col2;*

SELECT **/column_list* FROM *table/view_name* ORDER BY *Col3 desc;*

SELECT **/column_list* FROM *table/view_name* ORDER BY *Col1 desc, col2;*

Lets Practice Where and Order By

STRING MATCHING

Like - To match strings

Not like - Opposite match of strings

Wildcard operators: _ and %

'_’ - To match exactly single character

'%’ - To match Zero or more characters

Lets Practice String Matching..

PRE-DEFINED FUNCTIONS

PREDEFINED FUNCTIONS

SQL has so many predefined functions..

1. Numeric Functions
2. String Functions
3. Date and Time Functions
4. Conversion Functions
5. Null handling Functions
6. Aggregate Functions
7. Window Functions

NUMERIC FUNCTIONS

ABS - Returns the absolute(positive) value of a numeric expression.

MOD - Equivalent to the % which returns remainder.

SQRT - Returns the square-root of a non-negative number or expression

SQUARE - Returns the square of a numeric expression.

POWER(x,y) – Returns x to the power of y value

CEIL – Returns the nearest equal or larger integer

FLOOR - Returns the nearest equal or smaller integer

ROUND – Returns the nearest integer based on the decimal part

SIGN – Returns sign of its argument(-1 for negative, 1 for positive, 0 for 0)

LOG - Returns the logarithm of a number or expression

LN - Returns the natural logarithm of a number or expression

EXP(x) – Returns Euler number (e – 2. 718282) to the power of x

FACTORIAL – Returns factorial of a number or expression

STRING FUNCTIONS

LENGTH – To calculate length of given string

CONCAT – To combine two or more strings

TRIM – To remove spaces on both left and right sides of the given string

LTRIM – To remove spaces on the left side of the given string

RTRIM – To remove spaces on the right side of the given string

LPAD – To add spaces or characters on the left side of the given string

RPAD – To add spaces or characters on the right side of the given string

SUBSTR(SUBSTRING) – To get a part of the string

REVERSE – Returns reverse of given string

LOWER – Converts given string to lower case

UPPER – Converts given string to UPPER case

INITCAP – Converts first letter of each word in the given string to UPPER case

REPLACE – To replace one or more characters/words with other words

DATE AND TIME FUNCTIONS

CURRENT_DATE() – Returns today’s date in server time zone

CURRENT_TIME() – Returns current time in server time zone

CURRENT_TIMESTAMP() - Returns current timestamp in server time zone

YEAR(date or ts) – Returns Year from given date or timestamp

MONTH(date or ts) - Returns Month from given date or timestamp

DAY(date or ts) – Returns Day of the month from given date or timestamp

WEEK(date or ts) – Returns Week number(1 to 53) of the year from given date or ts

TO_DATE(expr,format) – Converts given expr of certain format to Date

TO_TIMESTAMP(expr,format) – Converts given expr of certain format to Timestamp

DATEDIFF – To calculate difference between 2 dates in years, months, days, hours etc..

TIMESTAMPDIFF – To calculate difference between 2 timestamps in years, months, days, hours..

DATEADD – To add years, months, days, hours etc. to a Date

TIMESTAMPADD – To add years, months, days, hours etc. to a Timestamp

MONTHS_BETWEEN – To calculate the months between two dates or timestamps.

Other Dates:

SYSDATE()

DATETIME

LONG_DATE

NOW()

CONVERSION FUNCTIONS

TO_CHAR – To convert a Decimal or Date expression to Char type

TO_DECIMAL or TO_NUMBER – To Convert a string with only numeric characters to Decimal, If string has non-numeric characters, conversion will fail.

TO_DATE(expr, format) – Converts given expr of certain format to Date

TO_TIMESTAMP(expr, format) – Converts given expr of certain format to Timestamp

CAST – To convert any datatype to other datatype.

AGGREGATE FUNCTIONS

Aggregate functions are used to calculate summaries after grouping the data.

Summaries like Count, Minimum, Maximum, Sum, Average, Mode, Median.

Aggregate function performs calculation on multiple values of a column and returns a single value.

If we want to use Aggregate functions over some groups of data, we have to group the data first based on some column(s) or expression(s).

We use GROUP BY clause to group the data.

Syntax: SELECT *column/expr, aggr_fun* FROM *table_name* GROUP BY *column/expr*;

Having: Having clause is used to filter out the data from groups based on conditions, it is used after GROUP BY clause.

Syntax: SELECT *column/expr, aggr_fun* FROM *table_name* GROUP BY *column/expr* HAVING *conditions*;

AGGREGATE FUNCTIONS

COUNT – To get the number of records from a table or view

SUM – To get the sum of all values of a column

AVG – To get average of all values of a column

MAX – To get the biggest values of a column

MIN – To get smallest value of a column

MODE – To get the most frequent value of values of a column

MEDIAN – To get middle value of column

Note: Except COUNT(*) none of the other aggregate functions consider nulls.

INTERVIEW QUESTIONS

Important Interview Questions:

1. What is the difference between WHERE and HAVING?

Ans: Where is used to filter out the data from a table or view based on conditions, but Having is used to filter out the data from groups of records based on specified conditions, Having is used along with GROUP BY clause.

2. Can you use both WHERE and HAVING in same SQL statement?

Ans: Yes, we can use

3. What is the order of execution of WHERE and HAVING in the query?

Ans: SELECT … FROM WHERE …. GROUP BY …. HAVING ….

NULL HANDLING FUNCTIONS

What is a Null?

- A column with NULL value means that column has no value.
- A Null value is neither Zero(0) nor a blank space.
- Length of a null value is null where length of a blank value is 0.
- We can define nullability of a column in the table definition, called NOT NULL constraint.
- If we define any column as NOT NULLABLE then it will not accept nulls, if you try to load nulls into that, the query will fail.
- Null never matches with another null.
- We should be very careful with nulls.

NULL HANDLING FUNCTIONS

IFNULL(col1/expr1, col2/expr2) – If col1 is null then returns col2 otherwise returns col1

NVL (col1/expr1, col2/expr2) – If col1 is null then returns col2 otherwise returns col1

NVL2 (col1/expr1, col2/expr2, col2/expr2) – If col1 is not null then returns col2 otherwise returns col3

COALESCE(<expr1> , <expr2> [, ... , <exprN>]) – Returns first not-null expression

ALL ABOUT NULLS

Microsoft
werPoint Presentat

WINDOW FUNCTIONS

WINDOW FUNCTIONS

A window is a group of related rows. A window can consist of one, or multiple rows.

A window function is any function that operates over a window of rows.

RANK – Returns rank over a group of values, skips the series in case of duplicates

DENSE_RANK – Returns rank over a group of values, doesn’t skips the series in case of duplicates

ROW_NUMBER – Returns a unique row number for each row within a group of values

LEAD – To get the next row information

LAG – To get the next row information

FIRST_VALUE – Returns the first value within an ordered group of values.

LAST_VALUE – Returns the last value within an ordered group of values.

NTH_VALUE – Returns the nth value within an ordered group of values.

RANK

Rank:

- Returns the rank of a value within an ordered group of values.
- The rank value starts at 1 and continues up sequentially.
- If two values are the same, they have the same rank but skips the next sequence number.
- Used to assign rank numbers, find the top values etc.

Syntax:

```
RANK() OVER ([partition by col/expr] order by col/expr [asc/desc])
```

DERIVE THE RANK BASED ON SALARY

EMPID	FIRST_NAME	LAST_NAME	AGE	SALARY
1	Anitha	Rao	28	60000
2	Vinay	Kumar	31	70000
3	Divya	Reddy	25	50000
4	Rama	Devi	28	60000
5	Naresh	Raju	30	55000
6	Uma	Shankar	26	55000
7	Tilak	Varma	29	60000

```
SELECT EMPID, FIRST_NAME, LAST_NAME, AGE,
SALARY,
RANK() OVER(ORDER BY SALARY DESC) as Rank
FROM EMPLOYEES;
```

EMPID	FIRST_NAME	LAST_NAME	AGE	SALARY	Rank
2	Vinay	Kumar	31	70000	1
1	Anitha	Rao	28	60000	2
4	Rama	Devi	28	60000	2
7	Tilak	Varma	29	60000	2
5	Naresh	Raju	30	55000	5
6	Uma	Shankar	26	55000	5
3	Divya	Reddy	25	50000	7

DENSE_RANK

Desne_Rank:

- Returns the rank of a value within a group of values, without gaps in the ranks.
- The rank value starts at 1 and continues up sequentially.
- If two values are the same, they have the same rank, but doesn’t skips the next sequence number.
- Used to assign rank numbers, find the top values etc.

Syntax:

DENSE_RANK() OVER ([partition by col/expr] order by col/expr [asc/desc])

DERIVE DENSE RANK BASED ON SALARY

EMPID	FIRST_NAME	LAST_NAME	AGE	SALARY
1	Anitha	Rao	28	60000
2	Vinay	Kumar	31	70000
3	Divya	Reddy	25	50000
4	Rama	Devi	28	60000
5	Naresh	Raju	30	55000
6	Uma	Shankar	26	55000
7	Tilak	Varma	29	60000

```
SELECT EMPID, FIRST_NAME, LAST_NAME, AGE, SALARY,

DENSE_RANK() OVER(ORDER BY SALARY DESC) as Rank

FROM EMPLOYEES;
```

EMPID	FIRST_NAME	LAST_NAME	AGE	SALARY	DRank
2	Vinay	Kumar	31	70000	1
1	Anitha	Rao	28	60000	2
4	Rama	Devi	28	60000	2
7	Tilak	Varma	29	60000	2
5	Naresh	Raju	30	55000	3
6	Uma	Shankar	26	55000	3
3	Divya	Reddy	25	50000	4

ROW_NUMBER

Row_Number:

- Returns a unique row number for each row within a window partition.
- The row number starts at 1 and continues up sequentially.
- If two values are the same, assigns row number randomly.
- Used to assign a sequence number irrespective of duplicates.

Syntax:

ROW_NUMBER() OVER ([partition by col/expr] order by col/expr [asc/desc])

DERIVE ROW NUMBER BASED ON SALARY

EMPID	FIRST_NAME	LAST_NAME	AGE	SALARY
1	Anitha	Rao	28	60000
2	Vinay	Kumar	31	70000
3	Divya	Reddy	25	50000
4	Rama	Devi	28	60000
5	Naresh	Raju	30	55000
6	Uma	Shankar	26	55000
7	Tilak	Varma	29	60000

```
SELECT EMPID, FIRST_NAME, LAST_NAME, AGE, SALARY,
ROW_NUMBER() OVER(ORDER BY SALARY DESC) as Rank
FROM EMPLOYEES;
```

EMPID	FIRST_NAME	LAST_NAME	AGE	SALARY	Rank
2	Vinay	Kumar	31	70000	1
7	Tilak	Varma	29	60000	2
4	Rama	Devi	28	60000	3
1	Anitha	Rao	28	60000	4
5	Naresh	Raju	30	55000	5
6	Uma	Shankar	26	55000	6
3	Divya	Reddy	25	50000	7

WHICH ONE TO CHOOSE?

It's completely depends on our requirements.

1. Suppose if I want to find winners/toppers based on their score/marks then I have to choose Dense_Rank for sure.
2. If I want to choose particular number of winners, I will go with Rank.
3. If I want to just generate a sequence number to rows based on some column/s, I will go with Row Number.
4. If we are sure there are no duplicate values, we can choose any one but use Row Number to avoid confusions.

LAG AND LEAD

LAG: Can fetch the value for a particular column from previous row in the same table/group after sorting in some order, without using a self join.

Syntax: LAG (col/expr, [, <offset> , <default>])

```
        OVER ( [ PARTITION BY <col/expr> ] ORDER BY <col/expr> [ { ASC | DESC } ] )
```

LEAD: Can fetch the value for a particular column from subsequent(next) row in the same table/group after sorting in some order, without using a self join.

Syntax: LEAD (col/expr, [, <offset> , <default>])

```
        OVER ( [ PARTITION BY <col/expr> ] ORDER BY <col/expr> [ { ASC | DESC } ] )
```

FIRST_VALUE, LAST_VALUE AND NTH_VALUE

FIRST_VALUE: Returns the first value within an ordered group of values.
Syntax: FIRST_VALUE(<col/expr>)
OVER ([PARTITION BY <col/expr>] ORDER BY <col/expr> [{ ASC | DESC }]

LAST_VALUE: Returns the last value within an ordered group of values.
Syntax: LAST_VALUE(<col/expr>)
OVER ([PARTITION BY <col/expr>] ORDER BY <col/expr> [{ ASC | DESC }]

NTH_VALUE: Returns the nth value within an ordered group of values.
Syntax: NTH_VALUE(<col/expr> , n)
OVER ([PARTITION BY <col/expr>] ORDER BY <col/expr> [{ ASC | DESC }]

SUBQUERIES

SUBQUERY

- A Subquery is a query within another SQL query, also called as nested query or inner query.
- Can be used with the SELECT, INSERT, UPDATE, and DELETE statements.
- Can be used in Where clause, From clause and in Joins.
- Subquery must be enclosed in parentheses.
- The subquery generally executes first if it has no co-relation with the main query, so the result of a outer query is dependent on subquery or inner query.

Co-related subquery:

- The Subquery that uses the values from outer query or dependent on the outer query is called Co-related subquery.
- In co-related subquery, the result of the outer query is dependent on inner query and result of the inner query is dependent on outer query, hence called co-related.

IN, EXISTS

IN: To match a column or expression against a list of values, the values can be direct or from result of a subquery.

Syntax:

SELECT **/column_list* FROM *table* WHERE *col1* IN (values..);

SELECT **/column_list* FROM *table* WHERE *col1* IN (subquery);

EXISTS: Returns True or False based on the subquery result that has condition to evaluate.

Syntax:

SELECT **/column_list* FROM *table* WHERE EXISTS (SELECT * FROM *table* WHERE *condition);*

***Note**: NOT IN is just opposite to IN AND NOT EXISTS is just opposite to EXISTS.*

CASE AND DECODE

CASE STATEMENT

- The CASE statement go through the conditions specified and returns the value where the condition is true and stops going through the next conditions, if none of the conditions are true then returns the value specified in ELSE clause.
- CASE statement works like IF-ELSE (We can't use IF-ELSE in SQL but we can use in Pl-SQL).

Syntax:

```
SELECT
CASE WHEN condition1 THEN result1
     WHEN  condition2 THEN result2
     WHEN  conditionN THEN resultN
     ELSE result
END as col_name
FROM table_name;
```

DECODE STATEMENT

- Decode compares the column/expression with search value and returns the result value where it matches and if none of the search values are matching with expression it returns the default value specified in the statement.
- It works similar to IF-ELSE and CASE but the difference in CASE and IF-ELSE we can write any conditions, but in DECODE the conditions are based on the column/expression specified.
- This is useful when we need to compare same col/expr with multiple values.

Syntax:

SELECT

DECODE (*col/expr, search_1, result_1 [, search_2, result_2].....[,search_n,result_n] [, default]*)

FROM *table_name;*

DIFF BETWEEN CASE AND DECODE

Important interview question:

What is the difference between CASE and DECODE?

CASE	**DECODE**
Case is a statement	Decode is system function
CASE works with other relational operators like (>, <,>=, <=) along with equal to (=)	Decode only works with equality(=) checks
Case can be used as parameter in Procedure calls	Decode doesn’t work in procedure calls
Case statement can be used in PL SQL blocks	Decode works in PL SQL programs but only in select statements
Case expects data type consistency of returning values	Decode does not expects

CONSTRAINTS

CONSTRAINTS IN SQL

Constraints: Constraints are used to specify conditions or rules or restrictions for data in a table. In case of any violation to these constraints, the SQL statements will fail.

Below are the constraints in SQL.

- UNIQUE
- NOT NULL
- DEFAULT
- CHECK
- PRIMARY KEY
- FOREIGN KEY

CONSTRAINTS IN SQL

UNIQUE: Unique constraint can be applied at column level, this constraint ensures that all values in that column are different and it won't allow duplicates. It allows null values.

NOT NULL: Not Null can be applied at column level, this constraint won't allow nulls into the column, if you try to load null value then SQL statement gives error.

DEFAULT: Default can be applied at column level, where we have to specify a default value for that column, for any record if we do not insert/load a value to that column then the default value will be stored in that column.

CHECK: Check constraint can be applied at column level, where we have to specify a condition. This allows only the values that are satisfying the condition. If the condition is false then the SQL statement gives error.

Note: We can specify all above constraints at the time of table creation or can add constraints later by using ALTER TABLE statement.

CONSTRAINTS IN SQL

PRIMARY KEY:

- Primary Key helps us to identify each row uniquely in a table.
- It is combination of UNIQUE and NOT NULL constraints.
- Primary Key can be applied on a single column or combination of multiple columns.
- This is a table level constraint and can be added at the time of table creation or later by using ALTER TABLE statement.

FOREIGN KEY:

- Foreign Key is used to specify referential integrity between tables.
- That means same column present in two tables, in the first table that column is a Primary Key and in the other table it can be used as foreign key to refer the record from that other table.
- Foreign Key is basically used to provide the links between database tables.
- We use REFERENCES key word to specify foreign key relationships.

CONSTRAINTS IN SQL

```
CREATE TABLE employee (
emp_id INT UNIQUE,
first_name VARCHAR(30) NOT NULL,
last_name VARCHAR(30),
salary FLOAT DEFAULT 0,
age INT CHECK (age > 18 and age < 60),
location VARCHAR(20),
dept_id INT REFERENCES dept(dept_id),
PRIMARY KEY (emp_id)
);
```

```
CREATE TABLE dept (
dept_id INT UNIQUE,
dept_name VARCHAR(30) NOT NULL,
manager_id INT,
loc_id INT REFERENCES location(loc_id),
PRIMARY KEY (dept_id)
);
```

CONSTRAINTS IN SQL

Important Interview Question:

How many unique constraints and primary key constraints a table can have ?

Ans: A table can have any number of unique keys but only one primary key. That means we can define multiple unique keys on different columns, but can define single primary key at table level.

JOINS

JOINS IN SQL

JOIN: Join clause is used to combine rows from two or more tables by joining them based on a common filed(s) between the tables. We call them as Join Keys. Join Key can be a single column or set of columns.

empid	empname	salary	deptid
1	John	48000	101
2	Charles	55000	102
3	Tina	62000	101

deptid	deptname	loc_cd
101	Sales	BNG
102	Marketing	HYD
103	Finance	CHN

loc_cd	location
BNG	Bangalore
HYD	Hyderabad
CHN	Chennai
MUM	Mumbai

empid	empname	salary	deptname	location
1	John	48000	Sales	Bangalore
2	Charles	55000	Marketing	Hyderabad
3	Tina	62000	Sales	Bangalore

JOINS IN SQL

Below are the four types of joins in SQL

1. INNER JOIN
2. LEFT OUTER JOIN
3. RIGHT OUTER JOIN
4. FULL OUTER JOIN

Other than these 4 we have

- SELF JOIN
- CARTESIAN JOIN (or) CROSS JOIN

INNER JOIN

INNER JOIN: Returns all matching rows between both the tables based on the Join Key.

Syntax:

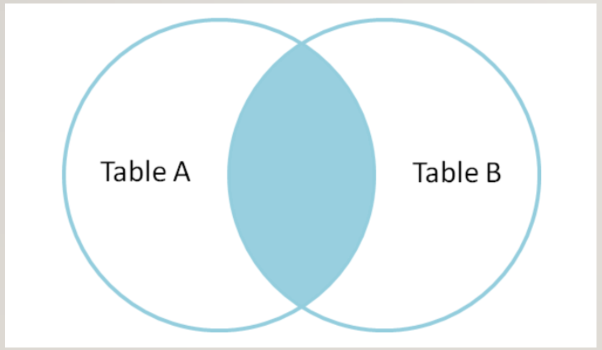
SELECT *A.*, B.**

FROM *TABLE_A A* INNER JOIN *TABLE_B B*

ON *A.Key = B.Key;*

Employee Table			
empid	**empname**	**salary**	**deptid**
1	John	48000	101
2	Charles	55000	102
3	Tina	62000	103

Dept Table	
deptid	**deptname**
101	Sales
102	Marketing
104	Finance



Employee INNER JOIN Dept On deptid					
empid	**empname**	**salary**	**E.deptid**	**D.deptid**	**deptname**
1	John	48000	101	101	Sales
2	Charles	55000	102	102	Marketing

LEFT OUTER JOIN

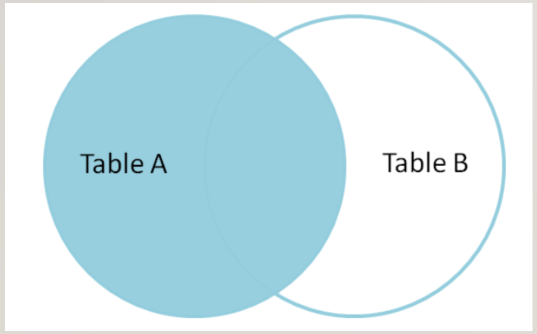
LEFT OUTER JOIN: Returns all rows from Left table and matching rows from Right table. For all non-matching rows from Right table, it will return nulls.

Syntax:

SELECT *A.*, B.**

FROM *TABLE_A A* LEFT OUTER JOIN *TABLE_B B*

ON *A.Key = B.Key;*



Employee Table

empid	**empname**	**salary**	**deptid**
1	John	48000	101
2	Charles	55000	102
3	Tina	62000	103

Dept Table

deptid	**deptname**
101	Sales
102	Marketing
104	Finance

Employee LEFT OUTER JOIN Dept On deptid

empid	**empname**	**salary**	**E.deptid**	**D.deptid**	**deptname**
1	John	48000	101	101	Sales
2	Charles	55000	102	102	Marketing
3	Tina	62000	103		

RIGHT OUTER JOIN

RIGHT OUTER JOIN: Returns all rows from Right table and matching rows from Left table. For all non-matching rows from Left table, it will return nulls.

Syntax:

SELECT *A.*, B.**

FROM *TABLE_A A* RIGHT OUTER JOIN *TABLE_B B*

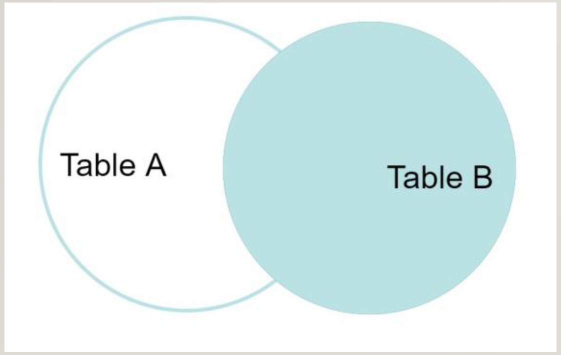
ON *A.Key = B.Key;*

Employee Table

empid	empname	salary	deptid
1	John	48000	101
2	Charles	55000	102
3	Tina	62000	103

Dept Table

deptid	deptname
101	Sales
102	Marketing
104	Finance



Employee RIGHT OUTER JOIN Dept On deptid

empid	empname	salary	E.deptid	D.deptid	deptname
1	John	48000	101	101	Sales
2	Charles	55000	102	102	Marketing
				104	Finance

FULL OUTER JOIN

LEFT OUTER JOIN: Returns all rows from both Left and Right tables. For all non-matching rows from both tables, it will return nulls.

Syntax:

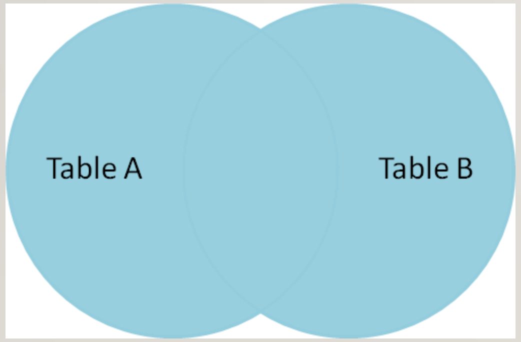
SELECT *A.*, B.**

FROM *TABLE_A A* FULL OUTER JOIN *TABLE_B B*

ON *A.Key = B.Key;*

Employee Table			
empid	**empname**	**salary**	**deptid**
1	John	48000	101
2	Charles	55000	102
3	Tina	62000	103

Dept Table	
deptid	**deptname**
101	Sales
102	Marketing
104	Finance



Employee FULL OUTER JOIN Dept On deptid					
empid	**empname**	**salary**	**E.deptid**	**D.deptid**	**deptname**
1	John	48000	101	101	Sales
2	Charles	55000	102	102	Marketing
3	Tina	62000	103		
				104	Finance

SELF JOIN

SELF JOIN: Sometimes we may need to join a table with same table, we call that join as self join

Syntax:

SELECT *A.*, B.**

FROM ***TABLE_A*** *A* INNER JOIN ***TABLE_A*** *B*

ON *A.Key = B.Key;*

(ON *A.managerid = B.empid*)

Employee Table			
empid	**empname**	**salary**	**managerid**
1	John	78000	
2	Charles	65000	1
3	Tina	61000	1
4	Bobby	40000	2

empid	**empname**	**salary**	**B.empname as managername**
1	John	78000	
2	Charles	65000	John
3	Tina	61000	John
4	Bobby	40000	Charles

CARTESIAN JOIN

CARTESIAN JOIN: Every row of a table will be joined with every row of other table.

There will be no join keys between two tables or no need to specify them.

We get the output row count as multiplication of both tables row count.

Syntax:

SELECT *A.*, B.**

FROM *TABLE_A A* CROSS JOIN *TABLE_B B;*

(or simply)

SELECT *A.*, B.**

FROM *TABLE_A , TABLE_B B;*

Employee Table

empid	**empname**	**salary**	**deptid**
1	John	48000	101
2	Charles	55000	102

Dept Table

deptid	**deptname**
101	Sales
102	Marketing

Employee CROSS JOIN Dept

empid	**empname**	**salary**	**E.deptid**	**D.deptid**	**deptname**
1	John	48000	101	101	Sales
1	John	48000	102	102	Marketing
2	Charles	55000	101	101	Sales
2	Charles	55000	102	102	Marketing

JOINS IN SQL – POINTS TO REMEMBER

1. We can achieve INNER JOIN with WHERE clause, but INNER JOIN is preferred.

   SELECT *A.*, B.** from *TABLE_A A, TABLE_B B* WHERE *A.Key=B.Key*
2. Be careful where there are nulls in Join keys, null never matches with another null.
3. In real time we mostly use INNER, LEFT OUTER and SELF joins.
4. *TABLE_A* LEFT OUTER JOIN *TABLE_B* is same as *TABLE_B* RIGHT OUTER JOIN *TABLE_A*.
5. Avoid using Cartesian joins as they are costly that means performance will be very slow.
6. If there are multiple matches in the joining table, we will get multiple records in the output.
7. While writing joins we should consider some important points.
   1. What we need in the output exactly
   2. The performance of the query should be good
   3. Query should be in readable format that any one can understand it in future.

JOINS IN SQL

Important Interview question:

Below are two tables with one column in each, what is the output number of records after each type of join?

Table A
1
2
1
3
null

Table B
1
2
2
4
null
null

INNER JOIN –

LEFT JOIN –

RIGHT JOIN –

FULL OUTER JOIN –

CARTESIAN JOIN –

JOINS IN SQL

Important Interview question:

Below are two tables with one column in each, what is the output number of records after each type of join?

Table A
1
2
1
3
null

Table B
1
2
2
4
null
null

INNER JOIN – 1+2+1 = 4

LEFT JOIN – 4+1+1 = 6

RIGHT JOIN – 4+1+1+1 = 7

FULL OUTER JOIN –
4+1+1+1+1+1 = 9

CARTESIAN JOIN – 5*6 = 30

SET OPERATORS

SET OPERATORS IN SQL

- Set Operators in SQL are same as Set Operators in Mathematics.
- Used to combine or compare result sets of two or more Select queries.
- Below are the rules to perform Set operations
  - Number of columns must be same in all Select queries
  - Order of columns must be same in all Select queries
  - Datatypes must be compatible

Below are the 4 set operators is SQL:

1. UNION
2. UNION ALL
3. INTERSECT
4. MINUS

UNION

UNION: Union is used to combine result sets of two or more Select queries. After combining the result sets it eliminate duplicates if any, that means if same record exists in both result sets it shows only one record in the output.

Syntax:

SELECT *col1, col2, …col_n* FROM *TABLE_A*

UNION

SELECT *col1, col2, …col_n* FROM *TABLE_B*

UNION

SELECT *col1, col2, …col_n* FROM *TABLE_C*

…..

TABLE_A

empid	**name**
21	Raja
22	Gopal

TABLE_B

empid	**name**
22	Gopal
25	Seetha
28	Kamal

TABLE_A **UNION** TABLE_B

empid	**name**
21	Raja
22	Gopal
25	Seetha
28	Kamal

UNION ALL

UNION ALL: Same as Union but Union all but it doesn't eliminate duplicates.

Syntax:

SELECT *col1, col2, …col_n* FROM *TABLE_A*

UNION ALL

SELECT *col1, col2, …col_n* FROM *TABLE_B*

UNION ALL

SELECT *col1, col2, …col_n* FROM *TABLE_C*

…..

TABLE_A

empid	**name**
21	Raja
22	Gopal

TABLE_B

empid	**name**
22	Gopal
25	Seetha
28	Kamal

TABLE_A **UNION ALL** TABLE_B

empid	**name**
21	Raja
22	Gopal
22	Gopal
25	Seetha
28	Kamal

INTERSECT

INTERSECT: Used to combine result sets and returns only common rows between both the tables.

Syntax:

SELECT *col1, col2, …col_n* FROM *TABLE_A*

INTERSECT

SELECT *col1, col2, …col_n* FROM *TABLE_B*

INTERSECT

SELECT *col1, col2, …col_n* FROM *TABLE_C*

…..

TABLE_A

empid	name
21	Raja
22	Gopal

TABLE_B

empid	name
22	Gopal
25	Seetha
28	Kamal

TABLE_A **INTERSECT** TABLE_B

empid	name
22	Gopal

MINUS

MINUS: Minus operator returns the records that exists in first result set and do not exists in the next result set. Very useful to data validation between two tables.

Syntax:

SELECT *col1, col2, …col_n* FROM *TABLE_A*

MINUS

SELECT *col1, col2, …col_n* FROM *TABLE_B*

TABLE_A	
empid	**name**
21	Raja
22	Gopal

TABLE_B	
empid	**name**
22	Gopal
25	Seetha
28	Kamal

TABLE_A **MINUS** TABLE_B	
empid	**name**
21	Raja

TABLE_B **MINUS** TABLE_A	
empid	**name**
25	Seetha
28	Kamal

SET OPERATORS IN SQL

Important Interview question:

What is the difference between UNION and UNION ALL, which gives better performance and which one you choose?

Ans: Union combines two or more result sets and eliminate the duplicate records, but Union All doesn’t eliminate duplicates.

Performance wise UNION ALL is better as the task of duplicate elimination is not required here.

So if you are concerned with duplicates better choose UNION and if you are sure that there are no duplicates then choose UNION ALL which gives better performance.

COMMON TABLE EXPRESSIONS (CTES)

CTES

- A CTE (common table expression) is a named subquery defined in WITH clause.
- We can think of the CTE as a temporary table. The result of the query expression is effectively a table.
- We can define CTE once in the query and can be used at multiple places if needed.
- CTEs can be used in Joins as well.
- We can use CTEs in all DML statements along with Select statement.

Syntax:

```
With
cte1 as ( … ),
cte2 as (…) …. cteN(…)
SELECT * FROM cte;
```

PIVOT AND UNPIVOT

PIVOT

Pivot operator converts the rows data of the table into the column data.
This operator supports the built-in aggregate functions AVG, MIN, MAX, COUNT, SUM.
Example:

Input:

```
(empid, amount, month)
  (1, 10000, 'JAN’),
  (1, 400, 'JAN'),
  (2, 4500, 'JAN'),
  (2, 35000, 'JAN'),
  (1, 5000, 'FEB'),
  (1, 3000, 'FEB'),
  (2, 200, 'FEB'),
  (2, 90500, 'FEB'),
  (1, 6000, 'MAR'),
  (1, 5000, 'MAR'),
  (2, 2500, 'MAR'),
  (2, 9500, 'MAR'),
```

Output:

```
+-------+-------+-------+-------+-------+
| EMPID | 'JAN' | 'FEB' | 'MAR' |
|-------+-------+-------+-------+-------+
|    1 | 10400 |  8000 | 11000 |
|    2 | 39500 | 90700 | 12000 |
+-------+-------+-------+-------+-------+
```

```
select * from monthly_sales
  pivot (sum(amount) for month in ('JAN', 'FEB', 'MAR'))  as p
order by empid;
```

UNPIVOT

Unpivot operator converts the column based data into rows, this is opposite to Pivot.

Example:

Input:

```
(empid, dept, jan, feb, mar);
(1, 'electronics', 100, 200, 300),
(2, 'clothes', 100, 300, 150),
(3, 'cars', 200, 400, 100);
```

```
select * from monthly_sales
 unpivot (sales for month in (jan, feb, mar))
order by empid;
```

Output:

```
+-------+------------+-------+-------++-------
| EMPID | DEPT       | MONTH | SALES |
|-------+------------+-------+-------+-------
|     1 | electronics | JAN   |   100 |
|     1 | electronics | FEB   |   200 |
|     1 | electronics | MAR   |   300 |
|     2 | clothes     | JAN   |   100 |
|     2 | clothes     | FEB   |   300 |
|     2 | clothes     | MAR   |   150 |
|     3 | cars        | JAN   |   200 |
|     3 | cars        | FEB   |   400 |
|     3 | cars        | MAR   |   100 |
+-------+------------+-------+-------++-----
```

INDEXES

INDEXES IN SQL

INDEXES:

- Index is a special look up table used to speedup the data retrieval from Database tables.
- Indexes are just pointers to original data, they are similar to Index page in the book, to search particular topic in the book (column in the database).
- Indexes improves the performance of SQL queries a lot when we are dealing with millions of rows.
- Index can be created on a single column or combination of multiple columns in a table.

Syntax:

CREATE INDEX *index_name* ON *TABLE_NAME(col_name)*;

Note: Indexes improves the performance of Select queries but degrades the performance of Insert and Update queries. Sometimes we drop and recreate indexes once the data load is completed.

INDEXES IN SQL

UNIQUE INDEXES: Unique index on a column doesn’t allow duplicate values into that column. It works both as Index and Unique constraint.

Syntax:

CREATE UNIQUE INDEX *index_name ON TABLE_NAME(col_name)*;

COMPOSITE INDEXES: An index created on two or more columns of a table is called Composite Index.

Syntax:

CREATE INDEX *index_name* ON *TABLE_NAME(col_1, col_2, ….)*;

Dropping an Index:

DROP INDEX *index_name*;

INDEXES IN SQL

Important Interview Questions:

1. What are indexes and unique indexes?
2. What are the advantages of Indexes?
3. When do you create indexes (or) on which columns you create indexes?

Ans: We create indexes in below scenarios..

- On large tables (avoid indexes on small tables)
- On a column that contains wide range of distinct values
- On column or set of columns those are frequently used in Where clause
- On column or set of columns used frequently as Join keys
- On columns where we don’t perform frequent Update operations

VIEWS

VIEWS IN SQL

VIEWS:

- A view is a virtual table that contains rows and columns. It is a schema object that stores SQL statement pointing to original tables and fetches data from these original tables when we query the view.
- A view can be created on multiple tables by joining them.
- A view can be created over another view.
- In some databases we can perform DML operations on a view that has no transformations or joins, but in most of the databases views or read-only and we can’t perform DML operations on them.

VIEWS IN SQL

Creating a view:

CREATE VIEW *view_name* as *SELECT col1, col2, ..col_n* FROM *table_name*;

Dropping a view:

DROP VIEW *view_name*;

Altering or Updating a view:

CREATE OR REPLACE VIEW *view_name* as *SELECT col1, col2, ..col_n* FROM *table_name*;

VIEWS IN SQL

Advantages or Uses of Views:

1. Restricting the access to original database tables.
2. Simplifying the SQL queries to end users.
3. We can create multiple views on same table based on the requirement.
4. If we want to hide the original column names, we can rename them in the view definition.
5. We can summarize the data from multiple tables.
6. Views take very little space to store the queries.

VIEWS IN SQL

Materialized Views:

- A normal view just stores the SQL query not the data or results set.
- A Materialized view is a view that stores the results of the query in that view.
- So when we query the materialized view it returns the data from results stored in that view directly and it will not fetch the data from underlying tables.
- Performance wise materialized views are better than views.

Syntax:

CREATE MATERIALIZED VIEW *view_name* AS SELECT *col1, col2, .. col_n* FROM *table_name*;

VIEWS IN SQL

Data Refresh in Materialized views:

- So, how the data(stored results) get refreshed when underlying tables gets updated?
- There is no standard way of refreshing data but it is database specific.
- In some databases there is Auto refresh(SQL Server, Snowflake)
- In some databases we have to refresh manually at regular intervals(PostgreSQL, MongoDB)
- Some databases supports both(Oracle, Redshift)

Advantages:

- When we are frequently running same query, better create a materialized view over it.
- When we query large dataset which returns few rows, better use materialized views.

VIEWS IN SQL

Important Interview Question:

What are the differences between views and materialized views?

Views	**Materialized Views**
It is a virtual tables, just stores the SQL query	It stores the result set of the query
Doesn't occupy storage	Storage space needed to store result set
Query fetches data from underlying tables, so performance may be slow	Performs better than Views as it fetches data from pre computed result sets
No data refresh headache	Data refresh is cost oriented

ACCESS CONTROLS

DCL (DATA CONTROL LANGUAGE)

Access controlling can be done by DCL commands in SQL

DCL: DCL commands are used to handle access privileges for users and roles on database objects to. Only database administrator's & owner's of the database object can provide and remove the privileges.

GRANT: It is used to provide permissions on database objects to users and roles.

Syntax: GRANT privilege_name ON db_object TO user|role

REVOKE: It is used to remove permissions on database objects to users and roles.

Syntax: REVOKE privilege_name ON db_object FROM user|role

DCL (DATA CONTROL LANGUAGE)

USERS: User can be a person or batch id who is operating or running SQL queries on database tables.

PRIVILEGES: Privileges let the user know what are all operations that user can perform on that object.

ROLES: A Role is a database object that group together a set of privileges that can be assigned to users.

Ex: Developer, Tester, Analyst, Administrator, EndUser etc.

Table/View level privileges: SELECT, INSERT, UPDATE, DELETE and ALL

Schema/Database level privileges: CREATE, ALTER and DROP

PERFORMANCE TUNING

PERFORMANCE TUNING

Performance Tuning:

- Tuning is the process of improving your SQL queries to run faster.
- Also called as Query Optimization.
- To reduce the consumption of resources and to reduce waiting time of data retrieval.

Reasons for slow performance:

- Huge tables
- More number of Joins
- More aggregations
- Inefficient query writing

PERFORMANCE TUNING

Tuning Process:

1. First analyze the query why it is taking long time to run.
2. Find which step or which part of the code is taking long time.

   (Check Execution plan or Query plan or Query profile to find out this)
3. Apply tuning on that part or rewrite the query in other way.
4. Follow some standard tuning techniques.

SQL TUNING TECHNIQUES (ALL ON-PREMISE DATABASES)

- First and important step is, create proper indexes, make use of function based indexes as well.
- Select only required fields, don’t use select * always.
- Avoid using multiple OR condition in Where clause, instead you can use Union or IN based on the conditions.
- Avoid using not equal to operator (<> or !=), Instead use Not In.
- Avoid using Distinct, instead we can use Group By on All columns.
- Use Union All instead of Union if you are sure there are no duplicates.
- Avoid joining tables using Where clause it will perform cross join first and then filters the records, Use Inner Join.
- If you want to see sample data then restrict the data using Top 10 or Limit 10.
- If possible avoid writing co-related subqueries.
- Use CTEs instead of subqueries if you want to use same subquery at multiple places in the query.
- Collecting the missing stats on table will help in Teradata.
- Remove unnecessary ORDER BY clauses.

PERFORMANCE IN SNOWFLAKE

There are lot of in-built features in snowflake to improve the performance, so first read and understand the concepts.

1. Data Storage:

The data is stored in micro-partitions after compression, if we have large tables then define cluster keys on the frequently used columns of the table, they will help to scan only required micro-partitions to fetch the data instead of scanning all partitions. Cluster keys behave like indexes in on-premise databases.

2. Metadata:

Snowflake manages meta data like number of rows, distinct values of each column, minimum and maximum of values stored in each column. So when you try to fetch the table count or minimum or maximum values, the query will use metadata instead of scanning entire table.

PERFORMANCE IN SNOWFLAKE

3. Caching:

Snowflake has 2 types results cache and disk cache. The result of a query will be stored in the results cache until 24 hours, so if we run the same query again with-in 24 hours, the query will fetch the data from results cache instead of scanning entire table again unless until the data is not updated. In case of disk cache the results will be stored in the memory and this memory will be effective until we close that session.

4. Vertical scaling and Horizontal scaling:

In Snowflake we have flexibility to increase the size of warehouse(Actual processing unit). We have sizes from X-Small to 6XL and the cost will get doubled when we upgrade to next level. But if you are dealing with very large tables and have multiple joins then increasing the size of warehouse to next level will help the query to run faster. This is called Vertical scaling. Suppose if we have so many concurrent queries running and the queries are going into waiting queues then we can go for multi clustering where we increase the number of clusters which is called as Horizontal scaling.

PERFORMANCE IN SNOWFLAKE

5. Dedicated virtual warehouses:

If we have multiple teams working or accessing same database then better use dedicated virtual warehouse for each team like one for Development team, one for Analyst team, One for End users etc. And always use dedicated warehouse for batch processing.

After following all above things, if your query is still running slow then use the SQL performance tuning techniques discussed in previous slide.

BEST PRACTICES

BEST PRACTICES IN WRITING SQL QUERIES

- Select only required fields.
- Write appropriate comments.
- Use proper alias names.
- Follow standard naming conventions for your table names, view names and field names.
- Format your queries, it should be easily understandable to others.

ADVANCED SQL QUERIES

ADVANCED QUERIES

1. How to Calculate Cumulative Sum?
2. Ho to Eliminate Duplicate pair of records
3. Ho to Eliminate Duplicate pair of records using SPLIT_PART
4. How to add up values into single row using LISTAGG?
5. Working with Regular Expressions
6. How to generate Sequence Number for existing records?

WHAT NEXT? - PL/SQL

- Introduction to Programming languages
- What is PL/SQL?
- Variables and Constants
- Conditional Statements(if-else)
- Looping Statements(for, while)
- Cursors
- User Defined Functions
- Stored Procedures
- Exception handling
- Packages
- Triggers

Thank You &
Wish you all the very best

Janardhana Bandi