STRUCTURED QUERY LANGUAGE

CONTENT

INTRODUCTION

Structured Query Language (SQL) is a domain-specific programming language used for managing and manipulating relational databases. It enables users to define, retrieve, modify, and manage data stored in tables. SQL's core functions include creating and altering database structures, inserting and updating data, and querying for specific information. By providing a standardized way to interact with databases, SQL plays a crucial role in data management, enabling users to perform tasks like data extraction, sorting, filtering, and aggregation, making it an essential tool for developers, data analysts, and database administrators**.**

What is SQL?

- SQL stands for Structured Query Language.
- SQL lets you access and manipulate databases.
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987.

What is DBMS & RDBMS?

DBMS, as the name suggest, is a management system which is used to manage the entire flow of data, i.e., insertion of data or the retrieval of data, how the data is inserted into the database or how fast the data should be retrieved, so DBMS takes care of all these features, as it maintains the uniformity of the database as well does the faster insertions as well as retrievals.

RDBMS on the other hand is a type of DBMS, as the name suggest it deals with relations as well as various key constraints. So here we have tables which is called as schema and we have rows which are called as tuples. It also aids in the reduction of data redundancy and the preservation of database integrity. Relational Database Management System is an advanced version of a DBMS.

DBMS VS RDBMS

(**D**ata **B**ase **M**anagement **S**ystem) DBMS	(**R**elational **D**ata **B**ase **M**anagement **S**ystem) RDBMS
DBMS stores data as file.	RDBMS stores data in tabular form.
Data elements need to access individually.	Multiple data elements can be accessed at the same time.
No relationship between data	Data is stored in the form of tables which are related to each other.
It stores data in either a navigational or hierarchical form.	It uses a tabular structure where the headers are the column names, and the rows contain values.
It deals with small quantity of data.	It deals with large amount of data.
***Examples for* DBMS VS RDBMS**	
XML, Window Registry, FoxPro, dbase III plus etc.	MySQL, PostgreSQL, SQL Server, Oracle, Microsoft Access etc.

1. DATA TYPES

What are Data Types?

A data type is a classification of data which tells the compiler or interpreter how the user intends to use the data.

A classification or category of various types of data, that states the possible values that can be taken, how they are stored, and what range of operations are allowed on them.

SQL DATA TYPES

DATA TYPE	SIZE
NUMBER	38
LONG	3 GB
CHAR	2000
VARCHAR	2000
VARCHAR2	4000
DATE	
CLOB	4GB
BLOB	4 GB

2. DATA QUERY LANGUAGE

- DQL statements are used to retrieve data stored in relational databases.
- The main purpose of DQL is to perform queries and filter data based on specific criteria using the **SELECT** command.
- DQL is categorized as one of the four main categories of SQL sub-languages, which also include Data Definition Language (DDL), Data Control Language (DCL), and Data Manipulation Language (DML).
- DQL allows you to retrieve specific data from tables, apply filters, sort the results, and perform various operations on the data.

Example: SELECT * FROM TABLE_NAME

3. DATA DEFINITION LANGUAGE

The Data Definition Language (DDL) is a sub-language of SQL used to define and manage the structure of a database. It includes commands for creating, altering, and dropping database objects such as tables, indexes, views, and constraints.

Command	Description
Create	This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
Alter	This is used to alter the structure of the database.
Rename	This is used to rename an object existing in the database
Comment	This is used to add comments to the data dictionary.
Truncate	This is used to remove all records from a table, including all spaces allocated for the records are removed.
Drop	This command is used to delete objects from the database.

4. DATA MANIPULATION LANGUAGE

The **Data Manipulation Language** (DML) is a sub-language of SQL used for retrieving, inserting, updating, and deleting data within a database. It allows users to manipulate the data stored in the database tables.

The SQL commands that deal with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements. It is the component of the SQL statement that controls access to data and to the database.

DML statements are typically executed by application developers or users who need to interact with the data in the database. DML statements can be executed interactively through a database management system or embedded within a programming language like Java, Python, or PHP.

Command	Description
Insert	It is used to insert data into a table.
Update	It is used to update existing data within a table.
Delete	It is used to delete records from a database table.

5. CONSTRAINTS

SQL constraints are used to specify rules for the data in a table. Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

Command	Description
Not null	Ensures that a column cannot have a NULL value.
Unique	Ensures that all values in a column are different
Primary Key	A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.
Foreign Key	Prevents actions that would destroy links between tables.
Check	Ensures that the values in a column satisfies a specific condition.
Default	Sets a default value for a column if no value is specified.
Create Index	Used to create and retrieve data from the database very quickly.

6. SQL Clauses

Clauses are in-built functions available to us in SQL. With the help of clauses, we can deal with data easily stored in the table.

Clauses help us filter and analyze data quickly. When we have large amounts of data stored in the database, we use Clauses to query and get data required by the user

Clause	Description
Where	It is used to filter records based on a specified condition. It allows you to retrieve only the records that meet the specified criteria.
Order by	It is used to sort the result set in ascending or descending order based on one or more columns. It allows you to arrange the data in a specific order.
Group by	It is used to group rows based on one or more columns. It is often used in conjunction with aggregate functions like SUM, COUNT, AVG, etc.
Having	It is used to filter the grouped rows based on a condition. It is similar to the WHERE clause but operates on the grouped data.
Limit	It is used to restrict the number of rows returned by a query. It is often used with the ORDER BY clause to retrieve a specific number of top or bottom records.

7. OPERATORS

Operators are the foundation of any programming language. We can define operators as symbols that help us to perform specific mathematical and logical computations on operands. In other words, we can say that an operator operates the operands. SQL operators have three different categories.

Operators are used to perform various operations on data, compare values, and combine conditions.

Major Types of SQL Operators

1. Arithmetic operators
2. Comparison operators
3. Logical operators
4. Special Operators

1. Arithmetic Operators

SQL Arithmetic Operators are used to perform mathematical operations on the data stored in SQL tables.

Operator	Description
+	The addition is used to perform an addition operation on the data values.
-	This operator is used for the subtraction of the data values.
/	This operator works with the ‘ALL’ keyword and it calculates division operations.
*	This operator is used for multiplying data values.

%	Modulus is used to get the remainder when data is divided by another.

2. Comparison Operators

Another important operator in SQL is a comparison operator, which is used to compare one expression's value to other expressions.

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

3. Logical Operators

Logical operators are used to combine multiple conditions or expressions and create complex conditions for filtering data. They allow us to perform logical operations on Boolean values or conditions

Operators	Description
AND	True if all the conditions separated by AND is True
NOT	Displays a record if the condition(s) is NOT TRUE
OR	True if any conditions separated by OR is True

4. **Special Operators:**

There are several special operators that provide additional functionality and perform specific operations.

Here are some commonly used special operators in SQL:

Operators	Description
ALL	TRUE if all of the subquery values meet the condition
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
SOME	TRUE if any of the subquery values meet the condition

8. SET OPERATORS

Set operators:

They are used to combine the results of two or more SELECT statements. These operators help in performing operations such as union, intersection, and difference on the result sets.

Different Types of Set Operators:

- UNION
- UNION ALL
- MINUS/EXCEPT
- INTERSECT

```
SELECT your_select_query
set_operator
SELECT another_select_query
```


Rules to use Set Operators:

1. When selecting your columns, the number of columns needs to match between queries, and the data type of each column needs to be compatible. So, if you select three columns in the first query, you need to select three columns in the second query.
2. The data types also need to be compatible, so if you select a number and two-character types in the first query, you need to do the same in the second query.
3. If you want to order your results, the ORDER BY must go at the end of the last query. You can't add ORDER BY inside each SELECT query before the set operator

UNION:

The UNION keyword or set operator will allow you to combine the results of two queries. It removes any duplicate results and shows you the combination of both.

```
SELECT your_select_query
UNION
SELECT another_select_query
```

UNION ALL:

The UNION ALL set operator also combines the results from two queries. It's very similar to UNION, but it does not remove duplicates. It will keep the duplicate values.

```
SELECT your_select_query
UNION_ALL
SELECT another_select_query
```


MINUS OR EXCEPT:

The MINUS set operator will return results that are found in the first query specified that don't exist in the second query. EXCEPT is the same as MINUS – they both show results from one query that don't exist in another query.

However, MINUS is an Oracle-specific keyword, and EXCEPT is in other databases such as SQL Server.

```
SELECT your_select_query
MINUS
SELECT another_select_query
```

INTERSECT:

The INTERSECT keyword allows you to find results that exist in both queries. Two SELECT statements are needed, and any results that are found in both of them are returned if INTERSECT is used.

SELECT your_select_query **INTERSECT** another_select_query

9. TRANSACTION CONTROL LANGUAGE

A transaction is the logical work unit that performs a single activity or multiple activities in a database.

Transaction Control Language commands are used to manage transactions in the database. These are used to manage the changes made by DML-statements.

Transaction Control Language (TCL) consists of a set of statements that are used to manage transactions within a database. Transactions are used to group multiple database operations into a single logical unit of work that can be executed atomically (all or nothing), ensuring data integrity and consistency

Command	Description
COMMIT	Commit command is used to permanently save any transaction into the database.
ROLLBACK	This command restores the database to last committed state. It is also used with save point command to jump to a save point in a transaction.
SAVEPOINT	This command is used to temporarily save a transaction so that you can roll back to that point whenever necessary.
RELEASE SAVEPOINT	It is used to remove a previously defined save point. It releases the save point and makes it unavailable for rollback
SET TRANSACTION	This statement is used to set properties for the current transaction, such as isolation level and transaction access mode.

10. DATA CONTROL LANGUAGE

Data Control Language (DCL) is a subset of SQL statements that are used to control access to data and manage database privileges. DCL statements are typically used by database administrators or users with administrative privileges.

Keyword	Description
GRANT	allow specified users to perform specified tasks
REVOKE	cancel previously granted
DENY	deny specified tasks to specified users

11. SQL CASE EXPRESSION

The CASE expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

CASE Syntax

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```


Example:

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN 'The quantity is greater than 30'
    WHEN Quantity = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

12. CHAR OR STRING FUNCTIONS (MANIPULATION)

Case Manipulation

In SQL, you can manipulate the case of data using various functions and statements.

Command	Description
Upper()	This function is used to convert a string to uppercase. It takes a string as input and returns the uppercase version of that string.
Lower()	This function is used to convert a string to lowercase. It takes a string as input and returns the lowercase version of that string.
Initcap()	This function is used to convert the first character of each word to uppercase and the remaining characters to lowercase. It takes a string as input and returns the modified string.

OTHER STRING FUNCTIONS

CHARINDEX():

This function searches for a substring in a string, and returns the position. If the substring is not found, this function returns 0. It returns the position of the first occurrence of the substring within the given string. **Note: This function performs a case-insensitive search.**

Syntax: Charindex(*substring*, *string*, *start*)

CONCAT():

This function in SQL is used to concatenate (or join) two or more strings together. It takes multiple string expressions as arguments and returns a single string that is the concatenation of those expressions.

Syntax: CONCAT(*string1*, *string2*,*.....*, *string_n*)

CONCAT_WS():

This function in SQL is used to concatenate multiple strings together with a specified separator. It stands for "concatenate with separator".

Syntax: CONCAT_WS(*separator*, *string1*, *string2*, *....*, *string_n*)

Length() / Len():

This function returns the number of bytes used to represent an expression.

Syntax: Length(expression) / Len(expression)

INSTR() / INSTRING():

This function in SQL is used to find the position of a substring within a string. It returns the position of the first occurrence of the substring within the string.

Syntax: Instr(string, substring) / Instring(string, substring)

SUBSTR() / SUBSTING():

This function in SQL is used to extract a substring from a string. It allows you to retrieve a portion of a string based on a specified starting position.

Syntax: Substr(string, start_position, length) /
Substring(*string*, *start*, *length*)

LTRIM():

Remove leading spaces from the left/beginning of a string.

Syntax: Ltrim(*string*)

RTRIM():

Remove leading spaces from the right/ending of a string.

Syntax: Rtrim(*string*)

LPAD():

This function is used to make the given string of the given size by adding the given symbol to the left side.

Syntax: Lpad(string, string size, padding character)

RPAD():

This function is used to make the given string of the given size by adding the given symbol to the right side.

Syntax: Rpad(string, string size, padding character)

REVERSE():

This function reverses a string and returns the result.

Syntax: Reverse(String)

Replace():

The REPLACE() function replaces all occurrences of a substring within a string, with a new substring.

Syntax: Replace(*string*, *old_string*, *new_string*)

TRANSLATE():

This function in SQL is used to replace occurrences of specified characters in a string with other specified characters. It allows you to perform character-level substitution or deletion within a string.

The TRANSLATE() function returns the string from the first argument after the characters specified in the second argument are translated into the characters specified in the third argument.

Note: The TRANSLATE() function will return an error if *characters* and *translations* have different lengths.

Syntax: Translate(*string*, *characters*, *translations*)

My SQL String Functions

Function	Description
ASCII	Returns the ASCII value for the specific character
CHAR_LENGTH	Returns the length of a string (in characters)
CHARACTER_LENGTH	Returns the length of a string (in characters)
CONCAT	Adds two or more expressions together
CONCAT_WS	Adds two or more expressions together with a separator
FIELD	Returns the index position of a value in a list of values
FIND_IN_SET	Returns the position of a string within a list of strings
FORMAT	Formats a number to a format like "#,###,###.##", rounded to a specified number of decimal places
INSERT	Inserts a string within a string at the specified position and for a certain number of characters
INSTR	Returns the position of the first occurrence of a string in another string
LCASE	Converts a string to lower-case
LEFT	Extracts a number of characters from a string (starting from left)
LENGTH	Returns the length of a string (in bytes)
LOCATE	Returns the position of the first occurrence of a substring in a string
LOWER	Converts a string to lower-case
LPAD	Left-pads a string with another string, to a certain length
LTRIM	Removes leading spaces from a string
MID	Extracts a substring from a string (starting at any position)
POSITION	Returns the position of the first occurrence of a substring in a string
REPEAT	Repeats a string as many times as specified
REPLACE	Replaces all occurrences of a substring within a string, with a new substring
REVERSE	Reverses a string and returns the result
RIGHT	Extracts a number of characters from a string (starting from right)
RPAD	Right-pads a string with another string, to a certain length
RTRIM	Removes trailing spaces from a string

SPACE	Returns a string of the specified number of space characters
STRCMP	Compares two strings
SUBSTR	Extracts a substring from a string (starting at any position)
SUBSTRING	Extracts a substring from a string (starting at any position)
SUBSTRING_INDEX	Returns a substring of a string before a specified number of delimiter occurs
TRIM	Removes leading and trailing spaces from a string
UCASE	Converts a string to upper-case
UPPER	Converts a string to upper-case

13. NUMBER FUNCTIONS

Types of number functions:

1. **Aggregation functions:** An aggregate function or aggregation function is a function where the values of multiple rows are processed together to form a single summary value.
2. **Analytical Functions:** SQL Analytical functions are a set of powerful tools that enable data analysts and developers to perform complex calculations and transformations on groups of data, without the need for complex sub-queries or multiple queries. They allow you to perform calculations over a set of rows that are related to each other.

1. Aggregation Functions:

AVG(): The AVG() function returns the average value of an expression. Note: NULL values are ignored.

Syntax: Avg(*expression*)

COUNT(): The COUNT() function returns the number of records returned by a select query. **Note: NULL values are not counted.**

Syntax: Count(*expression*)

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

Syntax: Min(*expression*)

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

Syntax: Max(expression)

SUM(): The SUM() function calculates the sum of a set of values.
Note: NULL values are ignored.

Syntax: Sum(expression)

2. Analytical Functions:

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

Syntax: ABS(*number*)

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

Syntax: CEIL(*number*)

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

Syntax: *x* DIV *y*

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

Syntax: FLOOR(number)

PI(): This function returns the value of PI**.**

Syntax: PI()

POW(): This function returns the value of a number raised to the power of another number.

Syntax: POW(x, y)

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

Syntax: ROUND(number, decimals)

TRUNCATE(): This function truncates a number to the specified number of decimal places.

Syntax: Truncate(number, decimals)

RANK(): This function assigns a rank to each row in a result set based on the value of a particular column or set of columns.

DENSE_RANK(): This function is similar to RANK(), but it assigns consecutive ranks to rows with the same value.

ROW_NUMBER(): This function assigns a unique row number to each row in a result set.

LAG(): This function returns the value of a column from the previous row in a result set.

LEAD(): This function returns the value of a column from the next row in a result set.

SIGN(): Returns the sign of a number (-1 for negative, 0 for zero, 1 for positive)

SQRT(): Returns the square root of a number.

14. NULL FUNCTIONS

In SQL, there are several functions that are specifically designed to work with NULL.

ISNULL(): This function is used to check if an expression is NULL and optionally replaces it with a specified value. The ISNULL function checks if the expression is NULL. If it is NULL, it returns the replacement value; otherwise, it returns the original expression.

Syntax: Isnull(expression, replacement value)

IFNULL(): IFNULL function is equivalent to the ISNULL function in SQL Server. Allows us to return the first value if the value is NULL, and otherwise returns the second value.

Syntax: Ifnull(first value, second value)

COALESCE(): Helps us to return the first non-null values in the arguments.

NVL(): Helps to replace the NULL value with the desired value given by the user.

Syntax: Nvl(expression,replacement value)

NVL2(): Checks whether the given expression is null if it is null it will replace the first replacement value else it will replace second replacement value.

Syntax: Nvl2(expression,first replacement value, second replacement value)

15. DATE FUNCTIONS

SQL provides various functions to work with dates.

ADD_MONTHS(): This function in SQL is used to add a specified number of months to a date. It is commonly used in databases like Oracle and MySQL.

Syntax: Add_months(date_column, num_months)

MONTHS_BETWEEN(): This function in SQL is used to calculate the number of months between two dates. It measures the difference in months between a start date and an end date.

Syntax: Months_between(end_date, start_date)

LAST_DAY(): Extracts the last day of the month for a given date

Syntax: Last_day(date)

NEXT_DAY(): This function in SQL is used to find the next specified day of the week after a given date. It is commonly used to determine the next occurrence of a specific day in a week.

Syntax: Next_day(date,day)

ADDDATE() / DATE_ADD(): This function in SQL is used to add a specified interval to a given date. It allows you to add years, months, days, hours, minutes, and seconds to a date value.

Syntax: Date_add / Add_date (date_column, INTERVAL value interval_type)

CURRENT_TIME(): Returns the current time

CURRENT_TIMESTAMP(): Returns the current date and time

DATE(): This function in SQL is used to extract the date part from a given datetime or timestamp expression. It allows you to retrieve only the date portion of a datetime value.

Syntax: DATE(datetime_expression)

DATEDIFF(): Returns the number of days between two date values.

Syntax: Datediff(date1, date2)

DATE_FORMAT(): function in SQL is used to format a date or datetime value into a specific string representation. It allows you to customize the output format according to your requirements.

Syntax: Date_format(date, format)

%Y: Represents the year in 4 digits (e.g., 2023).

%y: Represents the year in 2 digits (e.g., 23).

%m: Represents the month in 2 digits (e.g., 08 for August).

%d: Represents the day in 2 digits (e.g., 23).

%H: Represents the hour in 24-hour format (e.g., 14 for 2 PM).

%h: Represents the hour in 12-hour format (e.g., 02 for 2 PM).

%i: Represents the minutes in 2 digits (e.g., 30).

%s: Represents the seconds in 2 digits (e.g., 45).

%p: Represents AM or PM.

Date_format(order_date, '%Y-%m-%d')

ADDDATE	**Adds a time/date interval to a date and then returns the date**
ADDTIME	Adds a time interval to a time/datetime and then returns the time/datetime
CURDATE	Returns the current date
CURRENT_DATE	Returns the current date
CURRENT_TIME	Returns the current time
CURRENT_TIMESTAMP	Returns the current date and time
CURTIME	Returns the current time
DATE	Extracts the date part from a datetime expression
DATEDIFF	Returns the number of days between two date values
DATE_ADD	Adds a time/date interval to a date and then returns the date
DATE_FORMAT	Formats a date
DATE_SUB	Subtracts a time/date interval from a date and then returns the date

DAY	Returns the day of the month for a given date
DAYNAME	Returns the weekday name for a given date
DAYOFMONTH	Returns the day of the month for a given date
DAYOFWEEK	Returns the weekday index for a given date
DAYOFYEAR	Returns the day of the year for a given date
EXTRACT	Extracts a part from a given date
FROM_DAYS	Returns a date from a numeric datevalue
HOUR	Returns the hour part for a given date
LAST_DAY	Extracts the last day of the month for a given date
LOCALTIME	Returns the current date and time
LOCALTIMESTAMP	Returns the current date and time
MAKEDATE	Creates and returns a date based on a year and a number of days value
MAKETIME	Creates and returns a time based on an hour, minute, and second value
MICROSECOND	Returns the microsecond part of a time/datetime
MINUTE	Returns the minute part of a time/datetime
MONTH	Returns the month part for a given date
MONTHNAME	Returns the name of the month for a given date
NOW	Returns the current date and time
PERIOD_ADD	Adds a specified number of months to a period
PERIOD_DIFF	Returns the difference between two periods
QUARTER	Returns the quarter of the year for a given date value
SECOND	Returns the seconds part of a time/datetime
SEC_TO_TIME	Returns a time value based on the specified seconds
STR_TO_DATE	Returns a date based on a string and a format
SUBDATE	Subtracts a time/date interval from a date and then returns the date
SUBTIME	Subtracts a time interval from a datetime and then returns the time/datetime
SYSDATE	Returns the current date and time
TIME	Extracts the time part from a given time/datetime
TIME_FORMAT	Formats a time by a specified format
TIME_TO_SEC	Converts a time value into seconds

TIMEDIFF	Returns the difference between two time/datetime expressions
TIMESTAMP	Returns a datetime value based on a date or datetime value
TO_DAYS	Returns the number of days between a date and date "0000-00-00"
WEEK	Returns the week number for a given date
WEEKDAY	Returns the weekday number for a given date
WEEKOFYEAR	Returns the week number for a given date
YEAR	Returns the year part for a given date
YEARWEEK	Returns the year and week number for a given date

16. PSUDO COLUMNS

In SQL, pseudo columns are special read-only columns that represent additional information about rows or provide metadata related to the query or database.

A pseudo-column behaves like a table column but is not actually stored in the table. You can select from pseudo-columns, but you cannot insert, update, or delete their values. A pseudo-column is also similar to a function without arguments.

CURRVAL and NEXTVAL: A sequence is a schema object that can generate unique sequential values. These values are often used for primary and unique keys. You can refer to sequence values in SQL statements with these psudo columns:

CURRVAL: Returns the current value of a sequence.

NEXTVAL: Increments the sequence and returns the next value.

Examples:

```
SELECT STUDENTSEQ.currval FROM DUAL;

INSERT INTO STUDENT VALUES (STUDENTSEQ.nextval,
'BISHAL', 'JAVA', 7902);
```

LEVEL: For each row returned by a hierarchical query, the LEVEL pseudocolumn returns 1 for a root node, 2 for a child of a root, and so on.

ROWNUM: Oracle engine maintains the number of each record inserted by users in table. By the help of ROWNUM clause we can access the data according to the record inserted.

Example:

```
SELECT * FROM EMP WHERE ROWNUM <= 3;
```

ROWID: For each row in the database, the ROWID pseudocolumn returns a row\'s address. The ROWID contains 3 information about row address:

FileNo: FileNo means Table Number.

DataBlockNo: DataBlockNo means the space assigned by the oracle SQL engine to save the record.

RecordNo: Oracle engine maintains the record number for each record.

Example:

```
SELECT ROWID, ename FROM emp WHERE deptno = 20;
```

17. VIEW

View: In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

Here are some key points about views in SQL:

- Views are kind of virtual tables that don't hold the actual data.
- Views can be created from one or many tables, depending on the written SQL query.
- Views allow you to simplify complex queries by creating a virtual table with the desired columns and rows.
- Views can be used to restrict access to certain columns or rows of a table, providing an additional layer of security.
- Views can be used to present data in a customized format, combining columns from different tables or applying calculations.
- Views can be updated, inserted into, or deleted from, depending on the underlying tables and the view's definition.

Creating View:

CREATE VIEW view_name as (Select Query)

Example: CREATE VIEW Customers

```
SELECT CustomerName, ContactName

FROM Customers

WHERE Country = 'India';
```

Updating a View:

A view can be updated with the CREATE OR REPLACE VIEW statement.

Syntax: Replace VIEW view_name as (Select Query)

Deletinng a View:

A view is deleted with the DROP VIEW statement.

Syntax: DROP VIEW view_name;

Let's break down the **Syntax:**

CREATE VIEW: This keyword is used to indicate that you want to create a view.

[view_name] Specify the name you want to give to the view.

AS: This keyword is used to indicate that you're defining the view.

SELECT [column1, [column2]), ...: Specify the columns you want to include in the view.

FROM [table_name] Specify the table(s) you want to retrieve data from.

WHERE [condition] (Optional) You can add a condition to filter the data if needed.

18. MATERIALIZED VIEW

Materialized View: A materialized view in SQL is a database object that contains the results of a query. It takes the regular view, which is a virtual table based on an SQL statement, and materializes it by proactively computing the results and storing them in a "virtual" table. Here are some key points about materialized views in SQL:

- Materialized views are used when data needs to be accessed frequently and the data in the underlying tables does not get updated on a frequent basis.
- They are useful when the view is accessed frequently, as they save computation time by storing the precomputed results in the database.
- Materialized views can be created using the CREATE MATERIALIZED VIEW statement in SQL.
- The FROM clause of the query used to create a materialized view can reference tables, views, and other materialized views.
- Materialized views can improve the performance of complex queries, especially those involving joins and aggregations.
- They are particularly beneficial in scenarios where performance tuning for queries is needed.

Syntax:

```
CREATE MATERIALIZED VIEW [view_name] AS (select query)
```

Deletinng a View

A view is deleted with the DROP VIEW statement.

Syntax: DROP MATERIALIZED VIEW view_name

Views	Materialized Views
Query expression are stored in the databases system, and not the resulting tuples of the query expression.	Resulting tuples of the query expression are stored in the databases system.
Views needs not to be updated every time the relation on which view is defined is updated, as the tuples of the views are computed every time when the view is accessed.	Materialized views are updated as the tuples are stored in the database system. It can be updated in one of three ways depending on the databases system as mentioned above.
It does not have any storage cost associated with it.	It does have a storage cost associated with it.
It does not have any updating cost associated with it.	It does have updating cost associated with it.
There is an SQL standard of defining a view.	There is no SQL standard for defining a materialized view, and the functionality is provided by some databases systems as an extension.
Views are useful when the view is accessed infrequently.	Materialized views are efficient when the view is accessed frequently as it saves the computation time by storing the results beforehand.

19. JOINS

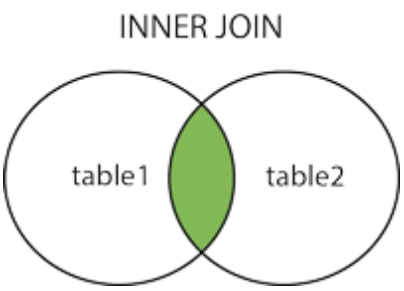
A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Joins allow you to retrieve data from multiple tables in a single query and make it possible to establish relationships between tables.

Types of Joins:

1. **Inner Join:** Returns records that have matching values in both tables
2. **Right Join:** Returns all records from the right table, and the matched records from the left table
3. **Left Join:** Returns all records from the left table, and the matched records from the right table
4. **Full Outer Join:** Retrieves all records when there is a match in either the left or right table. If there is no match, NULL values are returned for the non-matching table.
5. **Cross Join:** Returns all records from both tables

1. **Inner Join:** This join / keyword selects records that have matching values in both tables.



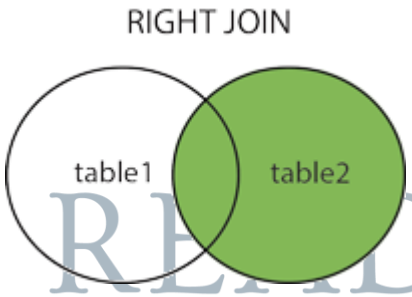
Syntax:

```
SELECT column_name(s) FROM table1

INNER JOIN table2

ON table1.column_name = table2.column_name;
```

2. **Right Join:** This join / keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1).



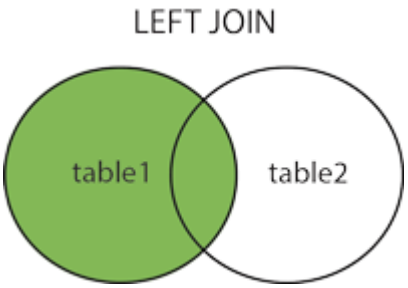
Syntax:

```
SELECT column_name(s) FROM table1

RIGHT JOIN table2

ON table1.column_name = table2.column_name;
```

3. **LEFT JOIN:** This join / keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2).



Syntax:

```
SELECT column_name(s) FROM table1

LEFT JOIN table2

ON table1.column_name = table2.column_name;
```

4. **Full Outer Join:** Retrieves all records when there is a match in either the left or right table. If there is no match, NULL values are returned for the non-matching table.

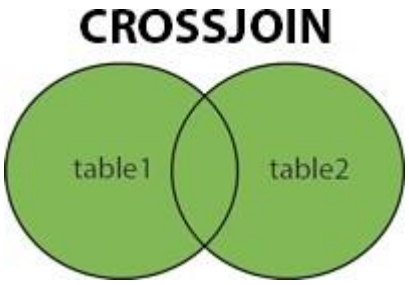
Syntax: Syntax:

```
SELECT column_name(s) FROM table1

FULL OUTER JOIN table2

ON table1.column_name = table2.column_name;
```

5. **Cross Join:** Produces the Cartesian product of both tables, resulting in a combination of every row from the first table with every row from the second table.



Syntax:

```
SELECT column_name(s)

FROM table1

CROSS JOIN table2;
```

20. SUB-QUERIES

In SQL, a subquery (also known as a nested query or inner query) is a query that is embedded within another query. It allows you to use the result of one query as a part of another query, enabling you to write more complex and powerful SQL statements.

1. **Syntax:** A subquery is enclosed within parentheses and can be placed in various parts of a SQL statement, such as the SELECT, FROM, WHERE, or HAVING clauses. The result of the subquery is then used in the outer query to perform further filtering, calculations, or comparisons.
2. **Usage**: Subqueries can be used for different purposes, such as:
   - Filtering: Using a subquery in the WHERE clause to filter rows based on a condition from another table or query.
   - Comparison: Comparing a value to the result of a subquery using comparison operators like =, >, <, etc.
   - Calculation: Performing calculations or aggregations on a subset of data obtained from a subquery.
   - Nesting: Nesting multiple levels of subqueries to build complex queries.

A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select.

Subquery rules

A subquery is subject to the following restrictions:

- The select list of a subquery introduced with a comparison operator can include only one expression or column name (except that EXISTS and IN operate on SELECT * or a list, respectively).
- If the WHERE clause of an outer query includes a column name, it must be join-compatible with the column in the subquery select list.
- The **ntext**, **text**, and **image** data types can't be used in the select list of subqueries.
- Because they must return a single value, subqueries introduced by an unmodified comparison operator (one not followed by the keyword ANY or ALL) can't include GROUP BY and HAVING clauses.
- The DISTINCT keyword can't be used with subqueries that include GROUP BY.
- The COMPUTE and INTO clauses can't be specified.
- ORDER BY can only be specified when TOP is also specified.
- A view created by using a subquery can't be updated.
- The select list of a subquery introduced with EXISTS, by convention, has an asterisk (*) instead of a single column name.

The rules for a subquery introduced with EXISTS are the same as those for a standard select list, because a subquery introduced

with EXISTS creates an existence test and returns TRUE or FALSE, instead of data.

TYPES OF SUB-QUERIES:

1. Scalar Sub-Query
2. Single Row Sub-Query
3. Multiple Row Sub-Query
4. Corelated Sub-Query

1. Scalar Subquery

A scalar subquery is a subquery that returns a single value. It is typically used in situations where you need to retrieve a single value for comparison or calculation purposes.

For example:

```
SELECT column1

FROM table1

WHERE column2 = (SELECT MAX(column2) FROM table2);
```

In this example, the subquery (SELECT MAX(column2) FROM table2) returns the maximum value of column2 from table2, which is then used for comparison in the outer query.

2. Single-Row Subquery

A single-row subquery is a subquery that returns a single row of data. It is commonly used in situations where you need to retrieve a single row for filtering or comparison purposes.

For example:

```
SELECT name

FROM customers

WHERE customer_id = (SELECT customer_id FROM orders
WHERE order_id = 123);
```

In this example, the subquery (SELECT customer_id FROM orders WHERE order_id = 123) returns the customer_id for a specific order, which is then used in the outer query to retrieve the corresponding customer name.

3. Multi-Row Subquery

A multi-row subquery is a subquery that returns multiple rows of data. It is often used with operators like IN, ANY, or ALL to compare a value to a set of values returned by the subquery.

For example:

```
SELECT name

FROM customers
```

```
WHERE customer_id IN (SELECT customer_id FROM orders
WHERE order_date = '2022-01-01');
```

In this example, the subquery (SELECT customer_id FROM orders WHERE order date = '2022-01-01') returns a set of customer IDs for orders placed on a specific date. The outer query then retrieves the customer names for those IDs.

4. Correlated Subquery

A correlated subquery is a subquery that refers to the outer query, allowing data from the outer query to be used in the subquery. It is useful when you need to retrieve data based on values from the outer query.

For example:

```
SELECT name

FROM customers c

WHERE EXISTS (SELECT 1 FROM orders o WHERE
o.customer_id = c.customer_id);
```

In this example, the subquery (SELECT 1 FROM orders o WHERE o.customer_id = c.customer_id) correlates with the outer query by using the customer ID from the outer query. The outer query retrieves the names of customers who have placed orders.

These are the main types of subqueries in SQL. Each type serves a different purpose and can be used to solve specific problems when querying your database. Understanding these types of subqueries will allow you to write more advanced and efficient SQL queries.