**SQL NOTES**

### ****What is a database?****

### Basically, **a database is a collection of interrelated data that is stored in an organized manner for easy access, management, and maintenance.** Each table consists of rows and columns to store data. A database can be defined as a structured form of data storage from which data can be retrieved and managed based on requirements.

### ****What is SQL?**** SQL stands for Structured Query Language. It uses SQL queries to interact with the database, i.e to create a database, to create a table in the database, to retrieve data or update a table in the database, etc

### ****What are the main data types in SQL?**** **In SQL, the main data types are as follows:**

|  |  |  |
| --- | --- | --- |
| Data Types | Definition | Syntax |
| INT (integer) | INT is used for whole numbers. | CREATE TABLE Employees ( ID INT,  Name VARCHAR(30) ); |
| FLOAT (floating point number) | DECIMAL is used for decimal and fractional numbers. | CREATE TABLE Item ( ID INT,  Price Decimal(5,2) ); |
| CHAR (fixed-length character string) | CHAR is used for fixed-length stringsultiple byte. The data-type can be single byte or multiple byte. | CREATE TABLE Employees ( ID INT,  initial CHAR(1) ); |
| VARCHAR (variable-length character string) | VARCHAR is used for variable-length strings. It can accept character strings upto 255 bytes. | CREATE TABLE Employees ( ID INT,  Name VARCHAR(30) ); |
| DATE (date) | DATE is used for dates in the format (YYYY-MM-DD). | CREATE TABLE Employees ( ID INT,  BirthDate DATE ); |
| |  DATETIME (date and time) | DATETIME is used for date and time values in the format (YYYY-MM-DD HH:MM:SS). | CREATE TABLE Employees ( ID INT,  OrderDate DATETIME ); |
| BOOLEAN (true or false) | BOOLEAN is used for boolean values. |  |

**What is SQL injection, and how do you avoid it?**SQL injection is a technique used to exploit user data through web page inputs by injecting SQL commands as statements. Basically, these statements can be used to manipulate the application’s web server by malicious users.  
SQL injection is a malicious attack targeted at an SQL server instance. It is usually sent through strings of statements and passed into the SQL server for execution. To avoid SQL injection, all statements must be verified for malicious vulnerabilities before being allowed for execution.

In addition to that, the following methods can be applied to avoid SQL injections:. They are given as follows:

* Using type-safe SQL parameters
* Using parameterized input with stored procedures
* Filtering inputs
* Reviewing codes
* Wrapping parameters

**Common SQL Injection Examples**

**Basic Injection:** ' OR '1'='1 in a login form  
**Union-Based:** Adding UNION SELECT to retrieve additional data  
**Blind Injection:** Extracting data through true/false responses

### What are tables and fields?

### Tables are the database objects where data is stored logically. A table is an organized collection of data stored in the form of rows and columns. A row/tuple in a table represents a record, and columns represent the different fields. Fields have data types such as text, dates, numbers, and links.

**How do you implement pagination in SQL quaries ?**Implementing pagination in SQL queries allows you to retrieve and display a subset of results from a larger dataset, which is useful for presenting data in manageable chunks such as in web applications. Here's how you can implement pagination using SQL:

In MySQL and PostgreSQL, you can use the LIMIT clause to specify the maximum number of rows to return and the OFFSET clause to skip a certain number of initial rows.

**SYNTAX : SELECT column1, column2 FROM table\_name ORDER BY column1 LIMIT 10 OFFSET 10;  
  
  
SQL Clause Execution Order (Logical Processing Flow)**  
  
**1. FROM/JOIN (Identify Data Sources)**

* First, the database identifies all tables and joins them together
* This creates the initial working dataset

**2. WHERE (Row Filtering)**

* Filters rows based on conditions
* Applied before grouping or aggregation

**3. GROUP BY (Aggregation)**

* Groups rows by specified columns
* Creates one row per unique group

**4. HAVING (Group Filtering)**

* Filters grouped data (after aggregation)
* Unlike WHERE, can reference aggregate functions

**5. SELECT (Column Selection)**

* Determines which columns to return
* Calculates expressions and aliases

**6. DISTINCT (Duplicate Removal)**

* Removes duplicate rows from result set

**7. ORDER BY (Sorting)**

* Sorts the final result set
* Only clause that can reference column aliases

**8. OFFSET/FETCH or LIMIT (Pagination)**

* Limits number of rows returned
* Applied last after all processing

## Visual Flowchart:

FROM/JOIN → WHERE → GROUP BY → HAVING → SELECT → DISTINCT → ORDER BY → LIMIT

**What are ACID properties in SQL?  
ACID Properties :-**

**1. Atomicity:** Any operation performed on the data should be executed either entirely or not executed at all. Partial execution is not possible in SQL transactions.

**2. Consistency:** It ensure that before and after the transaction, the database must be in a consistent state.

**3. Isolation:** It indicates that transactions are isolated from other transactions. That means if multiple transactions take place at the same time, one must not affect another.

**4. Durability:** All updates done by transaction must become permanent.   
 Or  
Durability ensures that data stays in the database permanently when a transaction is successfully completed. **How do you optimize sql quaries ?**Optimizing SQL queries is crucial for improving database performance. Here are several strategies for optimizing SQL queries:

* **Indexing:** Indexes can significantly improve query performance by providing a quick lookup mechanism. Ensure that your tables are appropriately indexed based on the columns frequently used in the WHERE, JOIN, and ORDER BY clauses.
* **Limit the Result Set:** When possible, limit the number of rows returned by a query using the LIMIT clause. This can reduce the strain on the database and improve performance.
* **\*\*Avoid SELECT \*\*\*:** Instead of selecting all columns with "SELECT \*", explicitly specify only the columns needed in the query. This reduces the amount of data the database needs to process.
* **Optimize Joins:** Ensure that join conditions are efficient. Use appropriate join types (e.g., INNER JOIN, LEFT JOIN) and indexing to optimize join operations.
* **Use EXISTS Instead of IN:** When comparing the result of a subquery, use EXISTS instead of IN. EXISTS can be more efficient, especially for large datasets.
* **Avoid Subqueries Where Possible:** Subqueries can negatively impact performance. Consider using JOINs or other techniques to achieve the same result without the need for subqueries.
* **Use WHERE Clause Efficiently:** Place conditions that reduce the result set as early as possible in the query. This can limit the amount of data processed by the database.
* **Parameterize Queries:** Use parameterized queries to avoid SQL injection and benefit from potential performance improvements due to query plan caching.
* **Regular Maintenance:** Regularly analyze and optimize long-running queries, and consider database maintenance tasks such as index rebuilds and statistics updates.
* **Use Stored Procedures:** Stored procedures can improve performance by reducing network traffic and providing potential for query plan caching.  
   **Types of SQL Data Constraints :-**

|  |  |  |
| --- | --- | --- |
| Constraints | Definition | Syntax |
| NOT NULL Constraint | Ensures that all values in a column are different. It can accept null values but only one. There can be multiple unique constraints in a table. | CREATE TABLE Students (  ID int NOT NULL,  Name varchar(255) NOT NULL,  Age int  ); |
| UNIQUE Constraint | Ensures that all values in a column are different. It can accept null values but only one. There can be multiple unique constraints in a table. | CREATE TABLE Students (  ID int NOT NULL UNIQUE,  Name varchar(255) NOT NULL,  Age int  ); |
| CHECK Constraint | The CHECK constraint ensures that all the values in a column satisfies certain conditions. | CREATE TABLE Students (  ID int NOT NULL,  Name varchar(255) NOT NULL,  Age int,  CHECK (Age>=18)); |
| PRIMARY KEY Constraint | Uniquely identifies each record in a table. Primary key must contain UNIQUE values. It connot accept null values. Exactly the same as the UNIQUE constraint but there can be many unique constraints in a table, but only one PRIMARY KEY constraint per table. | CREATE TABLE Students (  ID int NOT NULL,  Name varchar(255) NOT NULL,  Age int,  PRIMARY KEY (ID)  ); |
| FOREIGN KEY Constraint | A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY of another table. | CREATE TABLE Orders (  OrderID int NOT NULL,  OrderNumber int NOT NULL,  ID int,  PRIMARY KEY (OrderID),  FOREIGN KEY (ID) REFERENCES Students(ID)  ); |
| DEFAULT Constraint | It provides a default value for a column when none is specified. | CREATE TABLE Students (  ID int NOT NULL,  Name varchar(255) NOT NULL,  Age int,  City varchar(255) DEFAULT 'Unknown'  ); |

**What is Subquery in SQL?**  
A subquery (also called an inner query or nested query**) is a SQL query embedded within another SQL statement.** Sub-queries allow you to perform operations in multiple steps by using the result of one query as input for another.### \*\*SQL vs MySQL\*\*

|  |  |  |
| --- | --- | --- |
| Feature | SQL | MySQL |
| Type | SQL is a standard language for retrieving and manipulating structured databases. | MySQL is a relational database management system, like SQL Server, Oracle, or IBM DB2, that is used to manage SQL databases. |
| Usage | It is used for querying relational database systems | Used to store, modify and delete data |
| Licensing | It is a licensed product of Microsoft. | It is an open-source platform managed by Oracle corporation |
| Security | High (enterprise-grade) | As it is an open-source platform, security cannot be reliable |
| Connectors | It doesn’t support any connectors | Support connectors such as the Workbench tool to build databases |

**### \*\*SQL vs NoSQL\*\***

|  |  |  |
| --- | --- | --- |
| Feature | SQL | NoSQL |
| Structure | Works on relational databases. i.e Table-based (schema) | Works on non- relational databases. which don’t follow a fixed schema for data storage.  Flexible structure (document/key-value or graphs ) |
| Scaling | Vertical scaling by adding more resources to server. | Horizontal scaling by partitioning data across multiple servers (sharding). |
| ACID Compliance | SQL satisfies ACID Properties such as atomicity, consistency, isolation, and durability. | Follows CAP theory – according to this, any two of the following need to be satisfied – Consistency, Availability, and Partition tolerance. |
| quaries | Easy to execute complex queries | Difficult to execute complex queries |
| Speed |  | Fast: Designed for faster read and writes, often with a simpler query language. |

**### \*\*CHAR vs VARCHAR\*\***

|  |  |
| --- | --- |
| CHAR | VARCHAR |
| It is a fixed-length character string data type | It is a variable-length character string data type. |
| The data type can be a single byte or multiple-byte | It can accept character strings up to 255 bytes |
| This data type can be used when the character length is known | This data type is used when the character length is not clear |
| It uses static memory location | It uses dynamic memory location |
| When length is known (e.g., ZIP codes) | When length is unknown. (e.g., names) |

**### \*\*DELETE vs TRUNCATE\*\***

|  |  |  |
| --- | --- | --- |
| Feature | DELETE | TRUNCATE |
| Definition | The DELETE statement removes rows one at a time and records an entry in the transaction log for each deleted row. | TRUNCATE cmd removes the data by deallocating the data pages used to store the table data and records only the page deallocations in the transaction log. |
| Speed | The DELETE command is slower. | TRUNCATE command is faster. |
| Permissions | To use Delete you need DELETE permission on the table. | To use Truncate on a table we need at least ALTER permission on the table. |
| Rollback | possible | Not possible |

**### \*\*TRUNCATE vs DROP\*\***

|  |  |  |
| --- | --- | --- |
| Feature | DROP | Truncate |
| Definition | The DROP command is used to remove the table definition and its contents (structure too). Yes (fully removes) | Whereas the TRUNCATE command is used to delete all the rows (not structure) from the table. No (keeps structure) |
| Speed | Slower | Faster. |
| Space Freeing | table space is freed from memory. | While the TRUNCATE command does not free the table space from memory. |
| DDL Type | DDL | DDL |
| Effect | All the constraints will be removed after the execution of the DROP function. | Constraints don’t get affected because of the execution of this statement |

**### \*\*Delete vs TRUNCATE vs DROP\*\***

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | Drop | Delete | Truncate |
| Removes | Entire table (structure & data) | Specific rows based on WHERE clause | All rows (data only) |
| Speed | Slower (depends on table size) | Slower | Fastest |
| Transaction Log | NO | Logs deleted rows | Doesn't log deleted rows |
| Rollback | Not possible | possible | Not possible |
| Type of Statement | DDL | DML | DDL (in some DB) |

**### \*\*Primary Key vs Unique Constraint\*\***

|  |  |  |
| --- | --- | --- |
| Feature | Primary Key | Unique Constraint |
| NULL Values | Not allowed | Allowed (single NULL) |
| Quantity per Table | One | Multiple |
| Index | Creates clustered index | Creates non-clustered index |

**### \*\*BETWEEN vs IN Operators\*\***

|  |  |  |
| --- | --- | --- |
| Operator | Usage Example | Purpose |
| `BETWEEN` | WHERE age BETWEEN 20 AND 30 | Range-based filtering |
| `IN` | `WHERE id IN (1, 5, 9)` | Value-list matching |

**### \*\*INNER JOIN vs OUTER JOIN\*\***

|  |  |  |
| --- | --- | --- |
| Join Type | Result | Use-cases |
| Inner-Join | Matching rows only | When you need exact matches |
| Outer-Join | All rows + NULLs for non-matches | Preserving all records |

|  |  |
| --- | --- |
| Keys | Definition |
| Primary Key | A primary key is a field or combination of many fields that helps to uniquely identify each record in a table. It must contain **UNIQUE** values and has an implicit **NOT NULL** constraint. Note that there can be only one primary key for a table, which is comprised of single or multiple fields (columns). The table that has the primary key is known as the parent table.  https://noteboardapp.s3.amazonaws.com/users/images/722582_da2d28fa725111d428974d55be40bdf3.png |
| Foreign Key | A foreign key comprises a single or combination of fields in a table that essentially refers to the **PRIMARY KEY** of another table. The foreign key constraint ensures referential integrity in the relation between two tables. Unlike a primary key, a table can have one or many foreign keys. The table that has a foreign key is known as the child table.  For example, ID (1) is the primary key of the student table, and customer ID (2) in the Orders table is identified as the foreign key to the customer's table.  https://noteboardapp.s3.amazonaws.com/users/images/722582_691070772ba251fe8a0fd53c1ee31db4.png |
| Super Key | It is a set of one or more attribute that are taken collectively to identify uniquely an entity in a entity set. Know that super keys can have one or more attributes, even though all the attributes are not necessary to identify the records. |
| Candidate Key | A candidate key is a subset of Superkey, which can have one or more attributes to identify records in a table. Unlike Superkey, all the attributes of the candidate key must be helpful to identify the records. The minimal superkey for which there is no proper subset of a superkey are called candidate key.x  Note that all the candidate keys can be Super keys, but all the super keys cannot be candidate keys. |
| Composite Key | A composite key is the combination of two or more attribute in a table used to identify a row in a table. Know that a combination of columns is essential in creating composite keys because a single column in a composite key cannot identify a row in a table. We can say that the composite key is the primary key, with a few more attributes or columns. Also, a composite key can be a combination of candidate keys. |

**SQL JOINS**SQL joins combine rows from two or more tables based on related columns. Understanding the different join types is essential for querying relational databases effectively.

|  |  |  |
| --- | --- | --- |
| Names | Definition | Syntax |
| NATURAL JOIN | * Joins tables on columns with same names automatically * Not recommended (implicit behavior can be dangerous) | SELECT \*  FROM employees  NATURAL JOIN departments; |
| SELF JOIN | * Joins a table to itself * Useful for hierarchical data or comparing rows within same table | SELECT e1.name AS employee, e2.name AS manager  FROM employees e1  JOIN employees e2 ON e1.manager\_id = e2.employee\_id; |
| CROSS JOIN | * Returns Cartesian product (all possible combinations) * No join condition specified * Can generate large result sets | * SELECT products.name, colors.color\_name * FROM products * CROSS JOIN colors; |
| INNER JOIN (Standard Join) | * Returns only matching rows from both tables * Most commonly used join type | SELECT orders.order\_id, customers.customer\_name  FROM orders  INNER JOIN customers ON orders.customer\_id = customers.customer\_id; |
| LEFT JOIN (LEFT OUTER JOIN) | * Returns all rows from left table + matching rows from right table * Non-matches get NULL values for right table columns | SELECT employees.name, departments.dept\_name  FROM employees  LEFT JOIN departments ON employees.dept\_id = departments.dept\_id; |
| RIGHT JOIN (RIGHT OUTER JOIN) | * Returns all rows from right table + matching rows from left table * Non-matches get NULL values for left table columns * Less commonly used than LEFT JOIN | SELECT employees.name, departments.dept\_name  FROM employees  RIGHT JOIN departments ON employees.dept\_id = departments.dept\_id; |
| FULL JOIN  (FULL OUTER JOIN) | * Returns all rows when there's a match in either table * Combines LEFT and RIGHT JOIN results * Non-matches get NULL values for the opposite table | * SELECT employees.name, departments.dept\_name * FROM employees FULL JOIN departments ON  employees.dept\_id = departments.dept\_id; |

|  |
| --- |
| **### \*\*COALESCE() vs ISNULL()\*\*** |
| |  |  |  | | --- | --- | --- | | Function | Behavior | Example | | `COALESCE()` | Returns first non-NULL value | `COALESCE(NULL, 'default')` | | `ISNULL()` | Replaces NULL with a value | `ISNULL(column, 0)` | |

**###\*\*Where vs Having\*\***

|  |  |
| --- | --- |
| Where | Having |
| Filters rows before grouping. | Filters groups after aggregation |
| Used with SELECT, UPDATE, DELETE | Used with GROUP BY (only with aggregate functions) |
| Cannot use aggregate functions (SUM, AVG, etc.) | Requires aggregate functions (SUM, AVG, etc.) |
| Executes before GROUP BY | Executes after GROUP BY |
| Faster (filters early) | Slower (filters after grouping) |

**### \*\*Clustered vs Non-Clustered Index\*\***

|  |  |  |
| --- | --- | --- |
| Feature | Clustered Index | Non-Clustered Index |
| Storage | Physically reorders data | Separate index structure |
| Quantity | One per table | Multiple per table |
| Speed | Faster for retrieval | Slower (extra lookup) |
| Defination | Clustered index modifies the way records are stored in a database based on the indexed column. | A non-clustered index creates a separate entity within the table which references the original table. |

**### \*\*OLAP vs OLTP\*\***

|  |  |  |
| --- | --- | --- |
| Feature | OLAP | OLTP |
| Purpose | Analytics (historical data) | Transactions (real-time) |
| Speed | Slower | Faster |
| Example | Netflix recommendations | Online ticket booking |

**### \*\*NVL vs NVL2\*\***

|  |  |  |
| --- | --- | --- |
| Function | Behavior | Example |
| `NVL()` | Returns expr2 if expr1 is NULL | `NVL(NULL, 'backup')` |
| `NVL2()` | Returns expr3 if expr1 is NULL | `NVL2(NULL, 'keep', 'replace')` |

**### \*\*Commit vs Rollback\*\***

|  |  |  |
| --- | --- | --- |
| Command | Effect | When to use |
| COMMIT | Saves changes permanently | After successful transaction |
| ROLLBACK | Undoes changes | On transaction failure |

# ****What are the various commands used in SQL subsets?****

|  |  |
| --- | --- |
| ****DDL**** | CREATE, ALTER, DROP, TRUNCATE, ADD COLUMN, and DROP COLUMN |
| ****DML**** | INSERT, UPDATE and DELETE |
| ****DCL**** | GRANT and REVOKE |
| ****TCL**** | COMMIT, ROLLBACK, SAVEPOINT, and SET TRANSACTION |
| ****DQL**** | SELECT |

**Various set operators :**

|  |  |
| --- | --- |
| Union | This operator allows combining result sets of two or more SELECT statements. It doesn't contain duplicates. The columns in the SELECT statements must have the same data types.  SELECT column1 FROM table1  UNION  SELECT column1 FROM table2; |
| Union All | This operator allows combining result sets of two or more SELECT statements along with duplicates. |
| Intersect | This operator returns the common records of the result sets of two or more SELECT statements.  The columns in the SELECT statements must have the same data types. SELECT column1 FROM table1  INTERSECT SELECT column1 FROM table2; |
| Minus | The EXCEPT (or MINUS) operator returns the set difference of the result sets of two SELECT statements.  Returns all rows from the first SELECT statement that are not present in the second.  The columns in the SELECT statements must have the same data types. SELECT column1 FROM table1  EXCEPT  SELECT column1 FROM table2; |

**##Triggers, Stored Procedure, Cursor, Views, Collation, Indexes**

|  |  |  |
| --- | --- | --- |
| Trigger | ****Triggers are nothing, but they are special stored procedures that executes when t a specified event(Insert, Update, Delete) occurs on a table.**** | CREATE TRIGGER trigger\_name  ON table\_name  [AFTER | INSTEAD OF] [INSERT | UPDATE | DELETE]  AS  BEGIN  -- Trigger logic  END; |
| Stored Procedure | It is a function that consists a group of statements which can be stored and executed whenever they are required. Know that stored procedures are compiled only once. They are stored as ‘Named Object’ in the SQL server database. Stored procedures can be called at any time during program execution. Moreover, a stored procedure can be called another stored procedure. **Stored Procedures** are created to perform one or more DML operations on databases | CREATE PROCEDURE sp\_name  @param1 datatype,  @param2 datatype  AS  BEGIN  -- SQL statements  END;  -- Execute  EXEC sp\_name; |
| Views | Views are virtual table that do not store any data of their own but display data stored  in another table. VIews improves security of database by showing only intended data to authorised users and hide sensitive data. | **Basic Syntax:**  CREATE VIEW view\_name AS  SELECT column1, column2  FROM table\_name  WHERE condition; |
| Cursor | Database objects used to process rows one-by-one. |  |
| **collation in SQL** | It is a set of rules determining how character data can be sorted and compared. Rules defining character encoding, case sensitivity, and sorting. | **Common Collations**  SQL\_Latin1\_General\_CP1\_CI\_AS (Case-Insensitive, Accent-Sensitive)  Latin1\_General\_BIN (Binary, Case-Sensitive) |
| Indexes | Indexes are special lookup tables that are used to retrieve data from databases very fast. An index is used to speed up select quaries and where clauses. | CREATE INDEX index\_name /\* Create Index \*/ ON table\_name (column\_1, column\_2); DROP INDEX index\_name; /\* Drop Index \*/ |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| How NULL Values Affect SQL Aggregate Functions. ?  |  |  |  | | --- | --- | --- | | **COUNT(\*):** | Counts ALL rows, including those with NULL values. | SELECT COUNT(\*) FROM employees; | | **COUNT(column\_name):** | Counts only NON-NULL values in the specified column. | SELECT COUNT(manager\_id) FROM employees; | | **SUM() Behavior with NULLs** | • Ignores NULL values completely  • Returns NULL if all values are NULL | SELECT SUM(salary) FROM employees; **-- NULLs in salary column are excluded** | | **AVG() Behavior with NULLs** | * Excludes NULL values from calculation * Denominator is count of non-NULL values only | SELECT AVG(salary) FROM employees; | | **MIN() and MAX()** | * Ignore NULL values in their calculations * Return NULL only if ALL values are NULL | SELECT MIN(commission) FROM sales; -- Returns lowest non-NULL commission  SELECT MAX(commission) FROM sales | | **GROUP BY and NULLs** | NULLs are treated as a distinct group in GROUP BY operations | SELECT department\_id, COUNT(\*)  FROM employees **GROUP BY department\_id; -- NULL department\_id will be its own group.** | |
| **Difference between stored procedure and Functions. ?**   |  |  |  | | --- | --- | --- | | Features | Stored Procedure | Funations | | Return Value | May or may not return a value, and can return multiple values via output parameters | Must return a single value (which can be a scalar or table) | | Usage Context | Can be executed independently with EXEC/EXECUTE | Must be called within a SQL statement (SELECT, WHERE, etc.) | | DML Operations | Can perform INSERT, UPDATE, DELETE operations | Typically cannot modify database state (except table variables in some cases) | | Transaction Management | Can manage transactions (BEGIN/COMMIT/ROLLBACK) | Cannot manage transactions | | Error Handling | Can use TRY/CATCH blocks | Limited error handling capabilities | | Temporary Tables | Can create and use temporary tables | Can only use table variables | |
| Window functions apply to aggregate and ranking functions over a particular window (set of rows). OVER clause is used with window functions to define that window. OVER clause does two things :  Partitions rows into form set of rows. (PARTITION BY clause is used)  Orders rows within those partitions into a particular order. (ORDER BY clause is used)  Note : If partitions are not done, then ORDER BY orders all rows of the table.  **Syntax :**  SELECT coulmn\_name1, window\_function(cloumn\_name2), **OVER([PARTITION BY column\_name1] [ORDER BY column\_name3]) AS new\_column** FROM table\_name;    **Explain SQL Window Ranking functions like RANK(), DENSE\_RANK(), ROW\_NUMBER()**   |  |  |  |  | | --- | --- | --- | --- | | Names | Role | Output | Syntax | | ROW\_NUMBER() | **Always unique sequential numbers** | **1,2,3,4,5,6  Useful for pagination** |  | | RANK() | **Same rank for ties, then skips**  (same values get same rank, next rank is skipped). | **1,2,2,4,5,6,  3 is skipped** |  | | DENSE\_RANK() | **Same rank for ties, no skips**  (same values get same rank, next rank is consecutive). | **1,2,2,3,4,5  Consecutive ranking** |  | | LAG() | Accesses previous row | **Great for period-over-period comparisons** |  | | LEAD() | Accesses next row | **Useful for looking ahead** |  |  |  |  |  | | NTILE() | Divides into buckets | **Good for percentiles, quartiles** |  |  |  |  |  | |

**How do we avoid getting duplicate entries in a query without using the distinct keyword?**DISTINCT is useful in certain circumstances, but it has drawbacks in that it can increase the load on the query engine to perform the sort (since it needs to compare the result set to itself to remove duplicates). We can remove duplicate entries using the following options:

-> Remove duplicates using row numbers.  
-> Remove duplicates using self-join.  
-> Remove duplicates using group by.  
-> Remove duplicates using Union.  
-> Remove duplicates using Self-join.