SQL Notes

UNION vs UNION ALL

- **UNION**: Combines results from multiple queries, excluding duplicates.
- **UNION ALL**: Combines results from multiple queries, including duplicates.

WHERE vs HAVING

- **WHERE**: Filters individual rows before aggregation.
- **HAVING**: Filters groups after aggregation.

Clustered vs Non-Clustered Index

- **Clustered Index**: Determines the physical order of data in a table; can only be created on one column.
- **Non-Clustered Index**: A separate data structure; allows for multiple indexing strategies and does not affect physical data order.

Finding Duplicate Rows

```sql
SELECT salary, department, count(*)
FROM employees
GROUP BY salary, department
HAVING count(*) > 1;
```

Correlated vs Non-Correlated Subqueries

- **Correlated Subqueries**: Depend on values from the outer query and are re-evaluated for each row of the outer query.

- **Non-Correlated Subqueries**: Independent of the outer query and executed only once.

PRIMARY Key vs UNIQUE Key

- **PRIMARY Key**: Uniquely identifies each row and creates a clustered index.
- **UNIQUE Key**: Enforces uniqueness without necessarily identifying each row uniquely; creates a non-clustered index.

View vs Materialized View

- **View**: A virtual table that does not store data.
- **Materialized View**: A physical copy of the query result stored in a separate table.

TRUNCATE vs DELETE vs DROP

- **TRUNCATE**: Removes all rows from a table.
- **DELETE**: Removes specific rows based on conditions.
- **DROP**: Removes entire database objects.

Referential Integrity

- Ensures that when a record is deleted from the primary table, all associated records are deleted from the related table.

CHECK Constraint

- Defines a condition that must be true for any data inserted or updated in the column.

CHAR vs VARCHAR

- **CHAR**: Fixed length. Padding spaces will be inserted to complete the length.

- VARCHAR: Variable length.

INNER JOIN vs LEFT OUTER JOIN

- INNER JOIN: Returns only matching rows from both tables.
- LEFT OUTER JOIN: Returns all rows from the left table and matching rows from the right table; NULL for no match in the right table.

Self-Join

- Joining two instances of the same table.

COUNT Functions

- COUNT(*): Counts all rows.
- COUNT(1): Counts all rows.
- COUNT(column_name): Counts non-null values in the specified column.

Database Statistics

- Information about data distribution and structure used by the query optimizer. Updating database statistics regularly can optimize database performance.

Query Optimizer

- Analyzes possible execution plans and chooses the most efficient one.

Compound Index Column Order

- The order of columns in a compound index affects query performance and utilization by the query optimizer.

Wildcards in LIKE Operator

- `_`: Matches one character.
- `%`: Matches any number of characters.

Influencing Index Usage

- **Best Practices**:
  - Create the right index.
  - Keep statistics up to date.
  - Use index hints if necessary.
  - Create covering indexes meaning indexes for often queried columns.
  - Avoid indexing overkill.

NULL Comparisons

- The expression `NULL = NULL` returns `NULL`.

Temp Table

- A base table that exists only while the current database session is active.

Copying Tables

```
SELECT * INTO new_table FROM old_table;

INSERT INTO new_table SELECT * FROM old_table;
```

Changing Column Data Type

- Use `ALTER TABLE` with `ALTER COLUMN`.

Storing Monetary Values

- Use `DECIMAL` (also known as `NUMERIC`) data type.

Precision and Scale

- **Precision**: Total number of digits.
- **Scale**: Number of digits to the right of the decimal point.
  - Example: `123.45` has a precision of `5` and a scale of `2`.

Maximum Value for DECIMAL(6, 5)

- Maximum value: `9.99999`.

SQL Best Practices

- Use parameterized prepared statements to prevent SQL injection.
- Index appropriately but avoid too many indexes.
- Update statistics regularly.
- Normalize data.

Safest Date Format

- **ISO 8601 Format**:
  - Date: `"YYYY-MM-DD"`
  - Timestamp: `"YYYY-MM-DDTHH:MM:SS"`

IN vs EXISTS

- **IN**: Compares a value with a set of values.
- **EXISTS**: Checks for the existence of rows in a subquery.
- **NOT IN**: Compares a value with a set of values for non-matching.
- **NOT EXISTS**: Checks for the absence of rows in a subquery.

- **NOT IN and NOT EXISTS**: Not always interchangeable, especially with NULL values. When null values are present NOT IN will not behave as expected and better use NOT EXISTS instead.

Non-Equi Join

- Uses join conditions other than equality, e.g., greater than or less than operators.

Handling NULL Values in Joins

- Use `IS NULL` or `IS NOT NULL` conditions, or `COALESCE` function.

Cartesian Join

- Occurs when joining tables without join conditions.

Anti-Join

- Retrieves records that exist in one table but not in another.

```
SELECT * FROM table1
LEFT JOIN table2 ON table1.column_name = table2.column_name
WHERE table2.column_name IS NULL;
```

CTEs vs Derived Tables

- **CTEs**: Can be referenced multiple times within the same query.
- **Derived Tables**: Used once in the query in which they are defined.

```
-- Derived Tables
SELECT ...
FROM (SELECT ...) AS DerivedTable_name;
```

```
-- CTE
WITH CTE_name AS (SELECT ...)
SELECT ... FROM CTE_name;
```

ROW_NUMBER() and OVER Clause

- **ROW_NUMBER()**: Assigns a unique sequential integer to rows within a partition.
- **OVER Clause**:
  - `PARTITION BY`: Divides the result set into partitions.
  - `ORDER BY`: Specifies the order of rows within each partition.

Deleting Duplicates Using ROW_NUMBER()

```
WITH CTE AS (
   SELECT emp_id, ROW_NUMBER() OVER (PARTITION BY emp_name
ORDER BY emp_id) AS duplicate_count
   FROM employee
)
DELETE FROM employee
WHERE emp_id IN (
   SELECT emp_id
   FROM CTE
   WHERE duplicate_count > 1
);
```

Index Scan vs Index Seek

- **Index Scan**: Reads all rows in the index; used for large data retrieval and will be used in absence of WHERE condition.
- **Index Seek**: Selectively retrieves data; faster and used with conditions.

Table Scan vs Index

- **Table Scan**: Fast for small tables; inefficient for large tables.
- **Index**: Efficient for selective data retrieval.

COUNT Function Behavior

- **COUNT(field)**: Does not count null values.
- **COUNT(*)**: Counts null values.

Group By Clause Restrictions

- Non-aggregated columns in the SELECT list must be in the GROUP BY clause.

Composite Index Order

- Order of columns in a composite index affects query performance and utilization.

SQL Operation Order:

- `FROM` and `JOIN`
- `WHERE`
- `GROUP BY`
- `ROLLUP`, `CUBE`, `GROUPING SETS`
- `HAVING`
- `OVER` (e.g., Window Functions)
- `SELECT`
- `DISTINCT`
- `UNION`, `INTERSECT`, `EXCEPT`
- `ORDER BY`
- `OFFSET`

- `FETCH FIRST/NEXT ROWS ONLY`, `LIMIT`, `TOP`