Advanced SQL Guide

1. Subqueries (Nested Queries)

A **subquery** is a query within another query, usually within the `WHERE`, `FROM`, or `SELECT` clause.

- **Subquery in `WHERE` Clause**:

```
sql
Copy code
SELECT name, salary
FROM employees
WHERE department_id = (SELECT department_id FROM departments WHERE
department_name = 'HR');
```

- **Correlated Subquery**:

```
sql
Copy code
SELECT name, salary
FROM employees e
WHERE salary > (SELECT AVG(salary) FROM employees WHERE department_id =
e.department_id);
```

2. JOINs

JOIN combines rows from two or more tables based on a related column.

- **INNER JOIN**: Returns only the rows that match in both tables.

```
sql
Copy code
SELECT e.name, d.department_name
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id;
```

- **LEFT JOIN**: Returns all rows from the left table and matched rows from the right table.

```
sql
Copy code
SELECT e.name, d.department_name
FROM employees e
LEFT JOIN departments d ON e.department_id = d.department_id;
```

- **RIGHT JOIN**: Returns all rows from the right table and matched rows from the left table.

```
sql
Copy code
SELECT e.name, d.department_name
FROM employees e
RIGHT JOIN departments d ON e.department_id = d.department_id;
```

- **FULL OUTER JOIN**: Returns rows when there is a match in one of the tables.

```
sql
Copy code
SELECT e.name, d.department_name
FROM employees e
FULL OUTER JOIN departments d ON e.department_id = d.department_id;
```

3. GROUP BY and HAVING

GROUP BY groups rows that have the same values into summary rows, like finding the total salary by department.

- **GROUP BY**:

```
sql
Copy code
SELECT department_id, SUM(salary)
FROM employees
GROUP BY department_id;
```

- **HAVING**: Filters the groups created by `GROUP BY`.

```
sql
Copy code
SELECT department_id, COUNT(*) AS num_employees
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 5;
```

4. Window Functions

Window Functions perform calculations across a set of rows related to the current row, allowing operations like ranking and aggregating.

- **ROW_NUMBER()**: Assigns a unique number to each row in a result set.

```
sql
Copy code
SELECT name, salary, ROW_NUMBER() OVER (ORDER BY salary DESC) AS rank
FROM employees;
```

- **RANK()**: Assigns ranks to rows with ties (duplicates get the same rank).

```
sql
Copy code
SELECT name, salary, RANK() OVER (ORDER BY salary DESC) AS rank
FROM employees;
```

- **DENSE_RANK()**: Like `RANK()`, but no gaps in rank values.

```
sql
Copy code
SELECT name, salary, DENSE_RANK() OVER (ORDER BY salary DESC) AS
dense_rank
FROM employees;
```

- **NTILE()**: Divides the result set into a specified number of buckets.

```
sql
Copy code
SELECT name, salary, NTILE(4) OVER (ORDER BY salary DESC) AS quartile
FROM employees;
```

- **LEAD() and LAG()**: Accesses data from the next or previous row in the result set.

```
sql
Copy code
```

```
SELECT name, salary, LEAD(salary) OVER (ORDER BY salary DESC) AS
next_salary
FROM employees;
```

5. Common Table Expressions (CTEs)

A **CTE** is a temporary result set defined within the execution scope of a `SELECT`, `INSERT`, `UPDATE`, or `DELETE` statement.

- **Basic CTE**:

```
sql
Copy code
WITH dept_salary AS (
    SELECT department_id, AVG(salary) AS avg_salary
    FROM employees
    GROUP BY department_id
)
SELECT e.name, e.salary, d.avg_salary
FROM employees e
JOIN dept_salary d ON e.department_id = d.department_id
WHERE e.salary > d.avg_salary;
```

- **Recursive CTE**: Useful for hierarchical data (like organization charts).

```
sql
Copy code
WITH RECURSIVE org_chart AS (
    SELECT employee_id, manager_id, name
    FROM employees
    WHERE manager_id IS NULL
    UNION ALL
    SELECT e.employee_id, e.manager_id, e.name
    FROM employees e
    INNER JOIN org_chart o ON e.manager_id = o.employee_id
)
SELECT * FROM org_chart;
```

6. Indexing

Indexes improve the speed of query processing, especially on large tables.

- **Create Index**:

```
sql
Copy code
CREATE INDEX idx_employee_name ON employees (name);
```

- **Drop Index**:

```
sql
Copy code
DROP INDEX idx_employee_name;
```

- **Unique Index**:

```
sql
Copy code
CREATE UNIQUE INDEX idx_unique_email ON employees (email);
```

- **Composite Index**:

```
sql
Copy code
CREATE INDEX idx_composite ON employees (department_id, salary);
```

7. Transactions and ACID Properties

A **transaction** is a sequence of one or more SQL operations that are executed as a unit.

- **Start a Transaction**:

```
sql
Copy code
BEGIN TRANSACTION;
```

- **Commit the Transaction**:

```
sql
Copy code
COMMIT;
```

- **Rollback the Transaction**:

```
sql
Copy code
ROLLBACK;
```

ACID Properties:

- **Atomicity**: Ensures all operations in a transaction are completed successfully.
- **Consistency**: Ensures the database moves from one valid state to another.
- **Isolation**: Ensures transactions are executed independently.
- **Durability**: Ensures changes are permanent even after a system crash.

8. Data Manipulation

- **INSERT**:

```
sql
Copy code
INSERT INTO employees (name, salary, department_id)
VALUES ('John Doe', 50000, 2);
```

- **UPDATE**:

```
sql
Copy code
UPDATE employees
SET salary = salary * 1.1
WHERE department_id = 2;
```

- **DELETE**:

```
sql
Copy code
DELETE FROM employees WHERE department_id = 3;
```

9. Views

A **view** is a virtual table created by a query. It can be queried just like a regular table.

- **Create View**:

```
sql
Copy code
CREATE VIEW employee_view AS
SELECT name, salary, department_id
FROM employees
WHERE salary > 50000;
```

- **Query View**:

```
sql
Copy code
SELECT * FROM employee_view;
```

- **Drop View**:

```
sql
Copy code
DROP VIEW employee_view;
```

10. Stored Procedures and Functions

- **Stored Procedure**: A set of SQL statements that can be executed together.

```
sql
Copy code
CREATE PROCEDURE GetEmployeeSalary (@emp_id INT)
AS
BEGIN
    SELECT name, salary
    FROM employees
    WHERE employee_id = @emp_id;
END;
```

- **Execute Stored Procedure**:

```
sql
Copy code
EXEC GetEmployeeSalary @emp_id = 1;
```

- **Function**: Similar to stored procedures but returns a value.

```
sql
Copy code
CREATE FUNCTION GetSalary (@emp_id INT)
RETURNS DECIMAL(10, 2)
AS
BEGIN
    DECLARE @salary DECIMAL(10, 2);
    SELECT @salary = salary
    FROM employees
    WHERE employee_id = @emp_id;
    RETURN @salary;
END;
```

- **Execute Function**:

```
sql
Copy code
SELECT dbo.GetSalary(1);
```

11. Triggers

A **trigger** is an automatic action executed in response to certain events on a table or view (e.g., insert, update, delete).

- **Create Trigger**:

```
sql
Copy code
CREATE TRIGGER trg_after_insert
ON employees
AFTER INSERT
AS
BEGIN
    PRINT 'A new employee has been added';
END;
```

- **Drop Trigger**:

```
sql
Copy code
DROP TRIGGER trg_after_insert;
```

12. Normalization and Denormalization

Normalization is the process of organizing data to minimize redundancy. The common normal forms are:

- **1NF**: Eliminate duplicate columns.
- **2NF**: Ensure all non-key attributes depend on the entire primary key.
- **3NF**: Ensure no transitive dependencies.

Denormalization is the process of combining tables to improve query performance at the expense of some redundancy.