SQL Essentials for Beginners — Deep Dive with Examples

1. What is SQL? (DDL, DML, DCL, TCL, DQL)

Structured Query Language (SQL) is the standard language used to communicate with relational databases. SQL is categorized into several types of commands:

DDL – Data Definition Language

Defines the structure of the database schema.

- **CREATE**: Creates new objects (tables, views, etc.)
- **ALTER**: Modifies existing objects
- **DROP**: Deletes objects
- **TRUNCATE**: Removes all records quickly

Example:

```
CREATE TABLE employees (
  id INT PRIMARY KEY,
  name VARCHAR(100),
  department VARCHAR(50)
);

ALTER TABLE employees ADD COLUMN salary INT;

DROP TABLE employees;

TRUNCATE TABLE employees;
```

DML – Data Manipulation Language

Used to work with the data itself.

- **INSERT**: Adds new data
- **UPDATE**: Modifies existing data
- **DELETE**: Removes existing records

Example:

```
INSERT INTO employees (id, name, department) VALUES (1, 'Ashish', 'IT');

UPDATE employees SET department = 'Finance' WHERE id = 1;

DELETE FROM employees WHERE id = 1;
```

DCL – Data Control Language

Controls access to data.

- **GRANT**: Provides privileges
- **REVOKE**: Removes privileges

Example:

```
GRANT SELECT, INSERT ON employees TO hr_user;

REVOKE INSERT ON employees FROM hr_user;
```

TCL – Transaction Control Language

Manages transactions and ensures data integrity.

- **COMMIT**: Saves changes
- **ROLLBACK**: Reverts to the last commit
- **SAVEPOINT**: Marks a point within a transaction

Example:

```
BEGIN;
UPDATE employees SET department = 'HR' WHERE id = 2;
SAVEPOINT before_update;
DELETE FROM employees WHERE id = 2;
ROLLBACK TO SAVEPOINT before_update;
COMMIT;
```

DQL – Data Query Language

Retrieves data from the database.

- **SELECT**: Used to query data

Example:

```
SELECT name, department FROM employees WHERE department = 'IT';
```

---

2. SQL Data Types — In-Depth Guide

Data types specify the kind of data that each column in a table can hold. Choosing the right data type is crucial for data integrity, performance, and storage efficiency.

Numeric Types

- **INT / INTEGER**: Whole numbers, typically 4 bytes.
  Example: `age INT`
- **SMALLINT / BIGINT**: Smaller/larger ranges of integers.
- **DECIMAL(p, s)**: Fixed-point numbers with precision (`p`) and scale (`s`).
  Example: `salary DECIMAL(10, 2)` (up to 10 digits, 2 after decimal)
- **NUMERIC**: Synonym for `DECIMAL` in many databases.
- **FLOAT / REAL / DOUBLE PRECISION**: Approximate, floating-point numbers.

Character/String Types

- **CHAR(n)**: Fixed-length string (padded with spaces if shorter).
- **VARCHAR(n)**: Variable-length string, up to `n` characters.
- **TEXT**: Large variable-length string (size limit depends on DBMS).

Date and Time Types

- **DATE**: Stores year, month, and day.
- **TIME**: Stores time of day (hours, minutes, seconds).
- **TIMESTAMP**: Date and time, often with time zone support.
- **INTERVAL**: Represents a span of time (PostgreSQL).

Boolean Type

- **BOOLEAN**: Stores `TRUE`, `FALSE`, or `NULL`.

Auto-Increment/Serial Types

- **SERIAL**: Auto-incrementing integer (PostgreSQL).
- **IDENTITY**: Standard SQL auto-increment (supported in some DBMS).
- **AUTO_INCREMENT**: MySQL auto-incrementing integer.

Other Types

- **BLOB / BYTEA**: Binary data (images, files).
- **UUID**: Universally unique identifier.
- **ENUM**: Set of predefined values (MySQL, PostgreSQL).

Example Table Definition

```
CREATE TABLE orders (
    order_id SERIAL PRIMARY KEY,
    customer_name VARCHAR(50) NOT NULL,
    order_date DATE DEFAULT CURRENT_DATE,
    total_amount DECIMAL(10, 2),
    is_paid BOOLEAN DEFAULT FALSE
);
```

Tips

- Use the smallest data type that fits your needs for better performance.
- Always specify length for `CHAR` and `VARCHAR`.
- Use `NOT NULL` to enforce required fields.
- Use `DEFAULT` to set automatic values for new rows.
- Check your database documentation for supported types and their limits.

---

3. Basic Clauses: SELECT, FROM, WHERE — In-Depth Guide

The `SELECT`, `FROM`, and `WHERE` clauses form the foundation of nearly every SQL query. Mastering their usage is essential for effective data retrieval and manipulation.

SELECT Clause

The `SELECT` clause specifies which columns or expressions you want to retrieve from the database. It can include:

- **Column names**: Retrieve specific columns.
- **Expressions**: Perform calculations or transformations (e.g., `salary * 12`).
- **Functions**: Use built-in functions (e.g., `UPPER(name)`, `COUNT(*)`).
- **Aliases**: Rename columns or expressions for clarity using `AS`.

Examples:

```
-- Select specific columns
SELECT name, salary FROM employees;

-- Select all columns
SELECT * FROM employees;

-- Use expressions and aliases
SELECT name, salary * 12 AS yearly_salary FROM employees;

-- Use functions
SELECT UPPER(name) AS uppercase_name FROM employees;
```

Tips:

- Use only the columns you need for better performance.
- Aliases (`AS`) make results more readable, especially for calculated fields.

FROM Clause

The `FROM` clause identifies the table(s) from which to retrieve data. It can reference:

- **Single table**: The most basic usage.

- **Multiple tables**: For joins (covered in advanced topics).
- **Subqueries**: Use a query as a virtual table.

Examples:

```
-- Select from a single table
SELECT name FROM employees;

-- Select from a subquery (derived table)
SELECT avg_salary FROM (SELECT AVG(salary) AS avg_salary FROM employees) AS stats;
```

Tips:

- Always specify the correct table name.
- Use table aliases for clarity, especially in complex queries.

WHERE Clause

The `WHERE` clause filters rows based on specified conditions. Only rows that satisfy the condition(s) are returned.

Supported Operators:

- **Comparison**: `=`, `<>`, `!=`, `<`, `>`, `<=`, `>=`
- **Logical**: `AND`, `OR`, `NOT`
- **Pattern matching**: `LIKE`, `ILIKE` (case-insensitive in PostgreSQL)
- **Set membership**: `IN`, `NOT IN`
- **Range**: `BETWEEN ... AND ...`
- **NULL checks**: `IS NULL`, `IS NOT NULL`

Examples:

```
-- Simple condition
SELECT name FROM employees WHERE department = 'IT';

-- Multiple conditions
SELECT name FROM employees WHERE department = 'IT' AND salary > 50000;

-- Pattern matching
SELECT name FROM employees WHERE name LIKE 'A%';

-- Set membership
SELECT name FROM employees WHERE department IN ('HR', 'Finance');

-- Range
SELECT name FROM employees WHERE salary BETWEEN 40000 AND 60000;

-- NULL check
SELECT name FROM employees WHERE manager_id IS NULL;
```

Tips:

- Use parentheses to group conditions and control logical precedence.
- Avoid using functions on columns in `WHERE` if possible, as it may prevent index usage and slow down queries.
- Always use parameterized queries in applications to prevent SQL injection.

Advanced Usage

- **Subqueries in WHERE**: Filter based on results from another query.

```
SELECT name FROM employees WHERE department_id IN (SELECT id FROM
departments WHERE location = 'NY');
```

- **Expressions and Calculations**: Use arithmetic or string operations in conditions.

```
SELECT name FROM employees WHERE (salary * 1.1) > 60000;
```

- **Filtering on Derived Columns**: Use subqueries or Common Table Expressions (CTEs) if you need to filter on calculated columns.

---

4. Filtering Data: AND, OR, NOT, IN, BETWEEN, LIKE

Used to apply complex conditions to filter query results.

- `AND`: Both conditions must be true
- `OR`: At least one condition must be true
- `NOT`: Negates the condition
- `IN`: Matches any value from a list
- `BETWEEN`: Value falls within range (inclusive)
- `LIKE`: Pattern matching using `%` (wildcard) and `_`

Examples:

```
SELECT * FROM employees WHERE department = 'IT' AND salary > 50000;
SELECT * FROM employees WHERE department = 'HR' OR department = 'Finance';
SELECT * FROM employees WHERE NOT department = 'HR';
SELECT * FROM employees WHERE department IN ('HR', 'Finance');
SELECT * FROM employees WHERE salary BETWEEN 40000 AND 60000;
SELECT * FROM employees WHERE name LIKE 'A%';
```

Tips:

- Combine filters using parentheses for clarity

5. Sorting Results: ORDER BY

Sorts results based on one or more columns.

- **ASC**: Ascending (default)
- **DESC**: Descending

Syntax:

```
SELECT column1 FROM table ORDER BY column1 ASC|DESC;
```

Examples:

```
SELECT name, salary FROM employees ORDER BY salary DESC;
SELECT * FROM employees ORDER BY department ASC, salary DESC;
```

Tips:

- Some databases allow `NULLS FIRST` or `NULLS LAST`

---

6. Aliases: AS

Aliases rename columns or tables for clarity.

Syntax:

```
SELECT column AS alias FROM table;
```

Column Alias:

```
SELECT name AS employee_name, salary * 12 AS yearly_salary FROM employees;
```

Table Alias:

```
SELECT e.name, d.name FROM employees AS e JOIN departments AS d ON e.department_id
= d.id;
```

Note: `AS` is optional but improves readability.

---

7. Handling NULLs: IS NULL, COALESCE, IFNULL

NULL represents unknown or missing values.

- `IS NULL` / `IS NOT NULL`: Check for NULLs
- `COALESCE()`: Returns first non-null value
- `IFNULL()` / `NVL()`: Returns fallback if NULL (MySQL/Oracle)

Examples:

```
SELECT * FROM employees WHERE manager_id IS NULL;
SELECT name, COALESCE(manager_id, 'No Manager') AS reporting_to FROM employees;
SELECT name, IFNULL(manager_id, 'No Manager') AS reporting_to FROM employees;
```

Tips:

- `= NULL` will not work — use `IS NULL`
- Use `COALESCE()` for defaulting display/report values

Would you like to continue with advanced topics like Joins, Aggregations, or Constraints?