Oracle SQL

By Dhandapani Yedappalli Krishnamurthi Sep 20, 2025

Part 1: Database & DBMS Fundamentals

---

1. Introduction to Database

Definition:

- A **database** is an organized collection of data, stored and retrieved electronically.
- Example: Banking database → tables for customers, accounts, transactions.

Illustration:
📂 Database = Digital cupboard →

- **Table** = Shelf
- **Row** = File
- **Column** = Label

---

2. Introduction to DBMS

DBMS (Database Management System):

- A software that helps store, retrieve, and manage data.
- Examples: Oracle, MySQL, SQL Server, PostgreSQL.

Advantages:
✔ Data consistency
✔ Security

✔ Backup & Recovery
✔ Multi-user support

---

3. Characteristics of DBMS

- **Data Abstraction** (Users don't need to know low-level storage)
- **Data Independence** (Schema changes don't affect apps)
- **Concurrent Access** (Multiple users safely access data)
- **Security** (User access control)
- **Integrity Constraints** (Rules to ensure correct data)

---

4. DBMS Models

- **Hierarchical Model** – Tree structure (Parent → Child)
- **Network Model** – Records linked by relationships
- **Relational Model** – Tables with rows & columns (most common)

📌 **Oracle uses Relational Model (RDBMS).**

---

5. Relational DBMS (RDBMS)

- Stores data in **tables (relations)**.
- Each **row (tuple)** = a record.
- Each **column (attribute)** = a property.

Example in Oracle SQL:

```
CREATE TABLE Customers (
    Cust_ID   NUMBER PRIMARY KEY,
    Cust_Name VARCHAR2(50),
    City      VARCHAR2(30)
);
```

Insert Data:

```
INSERT INTO Customers VALUES (1, 'Anita',
'Chennai');
INSERT INTO Customers VALUES (2, 'Rahul',
'Bangalore');
```

Query Data:

```
SELECT * FROM Customers;
```

Cust_ID	Cust_Name	City
1	Anita	Chennai
2	Rahul	Bangalore

---

6. Data Integrity

Integrity ensures data correctness & consistency.
Types:

- **Entity Integrity**: Primary Key must be unique & not NULL.
- **Referential Integrity**: Foreign Key must reference existing record.
- **Domain Integrity**: Values must be valid for column datatype.

Example (Oracle SQL):

```
CREATE TABLE Orders (
    Order_ID  NUMBER PRIMARY KEY,
    Cust_ID   NUMBER REFERENCES Customers(Cust_ID),
    Amount    NUMBER CHECK (Amount > 0)
);
```

👉 Here:

- `Order_ID` ensures **entity integrity**.
- `Cust_ID` ensures **referential integrity**.
- `CHECK (Amount > 0)` ensures **domain integrity**.

---

7. Security in Database

- **Authentication** (Login with username/password)
- **Authorization** (Grant/Revoke rights)
- **Encryption** (Storing sensitive data securely)

Example (Oracle SQL):

```
CREATE USER trainee IDENTIFIED BY pass123;
GRANT CONNECT, RESOURCE TO trainee;
REVOKE RESOURCE FROM trainee;
```

---

8. Normalization & Codd's Rules

- **Normalization**: Process of organizing data to avoid redundancy.
- **Codd's Rules**: 12 golden rules for a **"fully relational" DBMS** (e.g., data should be stored in tables, not files).

👉 Example:

- **Unnormalized Data**:

```
Cust_ID | Cust_Name | Orders
1       | Anita     | O101, O102
```

- **Normalized Data**:

```
Customers: Cust_ID | Cust_Name
Orders:    Order_ID | Cust_ID
```

---

9. First Normal Form (1NF)

- No **repeating groups**.
- Each column must hold **atomic values**.

Example:

```
-- ❌ Bad (multi-valued Orders)
Cust_ID | Cust_Name | Orders
1       | Anita     | O101, O102

-- ✅ Good (split into atomic rows)
Cust_ID | Cust_Name | Order_ID
1       | Anita     | O101
1       | Anita     | O102
```

---

10. Second Normal Form (2NF)

- Table should be in 1NF.
- No **partial dependency** (non-key attribute depending on part of composite key).

---

11. Third Normal Form (3NF)

- Table in 2NF.
- No **transitive dependency** (non-key depending on another non-key).

How SQL Works Under the Hood

When you run a SQL query like:

```
SELECT Cust_Name
FROM Customers
WHERE City = 'Chennai';
```

Oracle (or any RDBMS) goes through these steps:

---

1. SQL Parsing

- SQL is **parsed** by the SQL engine.
- Checks
    - **syntax** (`SELECT` written correctly?),
    - **semantics** (does `Customers` table exist? does `Cust_Name` column exist?).
- If invalid → you get an **error** immediately.

Oracle stores parsed SQL in the Shared Pool (library cache).

- If the same query comes again → it reuses the **execution plan** (saves time).

👉 Example:

```
SELECT * FROM Customers;  -- parsed and cached
```

If you run it again, Oracle won't re-parse.

---

2. Query Optimization

- Oracle checks **multiple ways** to run the query.
- Chooses the **best execution plan** using the **Cost-Based Optimizer (CBO)**.
- It considers:
    - Indexes (use them or not?)
    - Table size
    - Joins order
    - Statistics

Example:

```
EXPLAIN PLAN FOR
SELECT Cust_Name FROM Customers WHERE City = 'Chennai';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

This shows **how Oracle plans to run your query** (Index Scan, Full Table Scan, etc.).

---

3. Row Source Generation (Execution Plan)

- Oracle **creates a plan** step by step:
  - **Full Table Scan** (check all rows)
  - or **Index Scan** (jump directly to matching rows).

Example: If the City column has an index, Oracle may use **Index Range Scan** instead of scanning the whole table.

---

4. Query Execution

- Oracle actually **runs the chosen plan**.
- Retrieves data blocks from **Buffer Cache** (memory).
- If not in cache → fetch from **disk (datafiles)**.

👉 Oracle works in units called **blocks** (usually 8 KB).

---

5. Fetching Results

- Results are fetched row by row.
- Oracle sends results to the client in **batches (arrays)**, not one row at a time (to optimize performance).

Example: SQL*Plus might fetch 15 rows per batch by default.

Behind the Scenes Components

- **Parser** → breaks SQL into tokens.
- **Optimizer** → decides best plan.
- **Row Source Generator** → prepares step-by-step plan.
- **Query Executor** → retrieves & returns rows.
- **Buffer Cache** → memory area to reduce disk reads.

Example Walkthrough (with Index)

```
-- Assume index on City
SELECT Cust_Name FROM Customers WHERE City = 'Chennai';
```

Under the hood:

1. Parser validates SQL.
2. Optimizer checks → "Should I scan whole Customers table or use index on City?"
3. Finds that **index on City** is efficient.
4. Execution plan: **Index Range Scan → Table Access by RowID**.
5. Fetches matching rows from buffer cache / disk.
6. Sends results to user.

Analogy

Think of it like a **restaurant order system** :

- You (client) → give the **order** (SQL query).
- Waiter (parser) → checks if order is valid.
- Chef (optimizer) → decides the best way to cook it.
- Kitchen (executor) → prepares the dish.
- Waiter brings results (rows) to you.

Performance Tips

- **Indexes** help optimizer choose faster paths.
- **Statistics** (ANALYZE TABLE or DBMS_STATS) help optimizer know table sizes.
- **Bind variables** (`:param`) let Oracle reuse execution plans.
- **Avoid SELECT *** → fetch only needed columns.
- Use **EXPLAIN PLAN** and **AUTOTRACE** to see what's happening.

✅ In short:
SQL → Parse → Optimize → Generate Plan → Execute → Fetch results.