# SQL-Oracle Cheat Sheet

By Dhandapani Yedappalli Krishnamurthi Sep 22, 2025

## Oracle SQL Cheatsheet (A–Z)

---

### 🅰️ Aggregate Functions

```
SELECT COUNT(*), SUM(salary), AVG(salary), MIN(salary),
MAX(salary)
FROM Employees;
```

- `COUNT` – number of rows
- `SUM` – total
- `AVG` – average
- `MIN/MAX` – smallest/largest

---

### 🅱️ Basic Data Types

- `NUMBER(p,s)` → numeric (precision, scale)
- `VARCHAR2(size)` → variable length string
- `CHAR(size)` → fixed length string
- `DATE` → date & time

- `CLOB/BLOB` → large objects

---

## 🅲 Constraints

```
CREATE TABLE Orders (
   Order_ID   NUMBER PRIMARY KEY,
   Cust_ID    NUMBER REFERENCES Customers(Cust_ID),
   Amount     NUMBER CHECK (Amount > 0),
   Email      VARCHAR2(50) UNIQUE,
   Status     VARCHAR2(10) DEFAULT 'PENDING'
);
```

- `PRIMARY KEY`, `FOREIGN KEY`, `CHECK`, `NOT NULL`, `UNIQUE`, `DEFAULT`

---

## 🅳 DDL (Data Definition Language)

```
CREATE TABLE Employees (Emp_ID NUMBER PRIMARY KEY, Name
VARCHAR2(50));
ALTER TABLE Employees ADD Salary NUMBER;
DROP TABLE Employees;
TRUNCATE TABLE Employees; -- fast delete, no rollback
```

---

## 🅴 Expressions & Operators

- **Comparison**: `=`, `<`, `>`, `<=`, `>=`, `<>`
- **Logical**: `AND`, `OR`, `NOT`
- **Arithmetic**: `+`, `-`, `*`, `/`, `%`
- **Special**: `BETWEEN`, `IN`, `LIKE`, `IS NULL`

---

## 🅵 Functions

```
-- String
SELECT UPPER('oracle'), LOWER('SQL'), LENGTH('Test'),
SUBSTR('Oracle',1,3) FROM dual;

-- Date
SELECT SYSDATE, ADD_MONTHS(SYSDATE, 3),
MONTHS_BETWEEN(SYSDATE, DATE '2024-01-01') FROM dual;

-- Conversion
SELECT TO_DATE('20-SEP-25','DD-MON-RR'),
TO_CHAR(SYSDATE,'YYYY-MM-DD'), TO_NUMBER('100') FROM dual;

-- Math
SELECT ROUND(123.456,2), TRUNC(123.456,1), MOD(10,3) FROM
dual;
```

---

## 🅶 GROUP BY & HAVING

```
SELECT Dept_ID, COUNT(*)
FROM Employees
GROUP BY Dept_ID
HAVING COUNT(*) > 5;
```

---

## 🅷 Hints (Optimizer)

```
SELECT /*+ INDEX(Employees idx_emp_name) */ Name FROM
Employees;
```

---

## 🅸 Indexes

```
CREATE INDEX idx_emp_name ON Employees(Name);
DROP INDEX idx_emp_name;
```

---

## 🅹 Joins

```
-- Inner Join
SELECT e.Name, d.Dept_Name
FROM Employees e
JOIN Departments d ON e.Dept_ID = d.Dept_ID;

-- Left Outer Join
SELECT e.Name, d.Dept_Name
FROM Employees e LEFT JOIN Departments d ON e.Dept_ID =
d.Dept_ID;

-- Self Join
SELECT e1.Name, e2.Name AS Manager
FROM Employees e1 JOIN Employees e2 ON e1.Manager_ID =
e2.Emp_ID;
```

---

## 🅺 Keys

- `PRIMARY KEY` → unique + not null
- `FOREIGN KEY` → ensures referential integrity
- `CANDIDATE KEY` → possible unique identifiers
- `COMPOSITE KEY` → multiple columns as primary key

---

## 🅻 LIKE (Pattern Matching)

```
SELECT * FROM Employees WHERE Name LIKE 'A%';   -- starts
with A
SELECT * FROM Employees WHERE Name LIKE '%n';   -- ends with
n
SELECT * FROM Employees WHERE Name LIKE '_a%';  -- second
letter is a
```

---

## 🅼 MERGE (UPSERT)

```
MERGE INTO Employees e
USING New_Employees n
ON (e.Emp_ID = n.Emp_ID)
WHEN MATCHED THEN UPDATE SET e.Salary = n.Salary
WHEN NOT MATCHED THEN INSERT (Emp_ID, Name, Salary) VALUES
(n.Emp_ID, n.Name, n.Salary);
```

---

## 🅽 NULL Handling

```
SELECT NVL(commission, 0) FROM Employees;
SELECT COALESCE(commission, bonus, 0) FROM Employees;
```

---

## 🅾 ORDER BY

```
SELECT * FROM Employees ORDER BY Salary DESC;
```

---

## 🅿 Privileges

```
GRANT SELECT, INSERT ON Employees TO trainee;
REVOKE INSERT ON Employees FROM trainee;
```

---

## 🅀 Queries (Subqueries)

```
-- Single-row subquery
SELECT * FROM Employees WHERE Salary > (SELECT AVG(Salary)
FROM Employees);

-- Multi-row subquery
SELECT * FROM Employees WHERE Dept_ID IN (SELECT Dept_ID
FROM Departments WHERE Location = 'Chennai');

-- Correlated subquery
SELECT e.Name FROM Employees e
WHERE Salary > (SELECT AVG(Salary) FROM Employees WHERE
Dept_ID = e.Dept_ID);
```

---

## 🅁 ROLLUP & CUBE

```
SELECT Dept_ID, Job_ID, SUM(Salary)
FROM Employees
GROUP BY ROLLUP(Dept_ID, Job_ID);

SELECT Dept_ID, Job_ID, SUM(Salary)
FROM Employees
GROUP BY CUBE(Dept_ID, Job_ID);
```

---

## 🅂 Set Operations

```
SELECT Emp_ID FROM Employees
UNION
```

```
SELECT Emp_ID FROM Managers;

SELECT Emp_ID FROM Employees
INTERSECT
SELECT Emp_ID FROM Managers;

SELECT Emp_ID FROM Employees
MINUS
SELECT Emp_ID FROM Managers;
```

---

## 🅣 Transactions

```
BEGIN;
UPDATE Employees SET Salary = Salary + 500 WHERE Dept_ID =
10;
SAVEPOINT step1;
DELETE FROM Employees WHERE Dept_ID = 20;
ROLLBACK TO step1;  -- undo delete but keep salary update
COMMIT;             -- save permanently
```

---

## 🅤 UPDATE

```
UPDATE Employees SET Salary = Salary * 1.10 WHERE Dept_ID =
101;
```

---

## 🅥 Views

```
CREATE VIEW HighSalaryEmployees AS
SELECT Name, Salary FROM Employees WHERE Salary > 50000;
```

```
DROP VIEW HighSalaryEmployees;
```

---

## 🅆 WITH (CTE – Common Table Expression)

```
WITH Dept_Salary AS (
  SELECT Dept_ID, AVG(Salary) AS avg_sal
  FROM Employees
  GROUP BY Dept_ID
)
SELECT e.Name, e.Salary, d.avg_sal
FROM Employees e JOIN Dept_Salary d ON e.Dept_ID =
d.Dept_ID;
```

---

## 🅇 XML Functions (Oracle specific)

```
SELECT XMLELEMENT("employee", Name, Salary) FROM Employees;
```

---

## 🅈 Your Session Info

```
SHOW USER;
SELECT USER, SYSDATE FROM dual;
```

---

## 🅉 Zones & Time Functions

```
SELECT CURRENT_DATE, CURRENT_TIMESTAMP, SYSTIMESTAMP FROM
dual;
```

---

# Quick Oracle-Specific Notes

- Oracle uses **`dual` table** for test queries.
- **`VARCHAR2`** is preferred over `VARCHAR`.
- `ROWNUM` and `FETCH FIRST n ROWS ONLY` for limiting rows.
- Use `DBMS_XPLAN.DISPLAY` to view execution plans.