SQL Commands Reference: PostgreSQL vs Oracle

This document compares **PostgreSQL** and **Oracle** SQL commands side-by-side, with usage examples and notes on differences.

---

1. CREATE TABLE

Postgres:

```
CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    salary NUMERIC(10,2)
);
```

Oracle:

```
CREATE TABLE employees (
    id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    name VARCHAR2(100),
    salary NUMBER(10,2)
);
```

Difference: Postgres uses `SERIAL` or `GENERATED AS IDENTITY`. Oracle (12c+) supports `IDENTITY`, older versions require a `SEQUENCE`.

---

2. INSERT

Postgres:

```
INSERT INTO employees (name, salary) VALUES ('Khushi', 50000.00);
```

Oracle:

```
INSERT INTO employees (name, salary) VALUES ('Khushi', 50000.00);
```

Difference: Same syntax. In older Oracle, you may need a sequence for IDs.

---

3. SELECT

Postgres:

```
SELECT * FROM employees;
SELECT name, salary FROM employees WHERE salary > 30000;
```

Oracle:

```
SELECT * FROM employees;
SELECT name, salary FROM employees WHERE salary > 30000;
```

Difference: Row limiting differs → Postgres uses `LIMIT`, Oracle uses `FETCH FIRST ... ROWS ONLY`.

---

4. UPDATE

Postgres:

```
UPDATE employees SET salary = 60000 WHERE name = 'Khushi';
```

Oracle:

```
UPDATE employees SET salary = 60000 WHERE name = 'Khushi';
```

Difference: None.

---

5. DELETE

Postgres:

```
DELETE FROM employees WHERE name = 'Khushi';
```

Oracle:

```
DELETE FROM employees WHERE name = 'Khushi';
```

Difference: None.

---

6. Constraints

- **Primary Key**

```
-- Postgres
id SERIAL PRIMARY KEY

-- Oracle
id NUMBER PRIMARY KEY
```

- **Foreign Key**

```
-- Postgres
FOREIGN KEY (dept_id) REFERENCES department(id)

-- Oracle
FOREIGN KEY (dept_id) REFERENCES department(id)
```

- **Unique**

```
UNIQUE (email)
```

- **Not Null**

```
name VARCHAR(100) NOT NULL
```

- **Check**

```
CHECK (salary > 0)
```

Difference: Syntax is the same.

---

7. Indexes

Postgres:

```
CREATE INDEX idx_emp_name ON employees(name);
```

Oracle:

```
CREATE INDEX idx_emp_name ON employees(name);
```

Use: Indexes speed up searches but slow down inserts/updates.

---

8. Joins

Postgres & Oracle (same syntax):

```
-- Inner Join
SELECT e.name, d.dept_name
FROM employees e
INNER JOIN department d ON e.dept_id = d.id;

-- Left Join
SELECT e.name, d.dept_name
FROM employees e
LEFT JOIN department d ON e.dept_id = d.id;

-- Right Join
SELECT e.name, d.dept_name
FROM employees e
RIGHT JOIN department d ON e.dept_id = d.id;

-- Full Join
SELECT e.name, d.dept_name
FROM employees e
FULL JOIN department d ON e.dept_id = d.id;
```

Difference: Same in both.

9. Aggregate Functions

Postgres & Oracle (same syntax):

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

10. Functions

- **Built-in Example**

```
-- Postgres
SELECT NOW();

-- Oracle
SELECT SYSDATE FROM dual;
```

- **User-defined Example**

```
-- Postgres
CREATE FUNCTION add_num(a INT, b INT) RETURNS INT AS $$
BEGIN
  RETURN a + b;
END; $$ LANGUAGE plpgsql;

-- Oracle
CREATE OR REPLACE FUNCTION add_num(a IN NUMBER, b IN NUMBER)
RETURN NUMBER AS
BEGIN
  RETURN a + b;
END;
```

Difference: Postgres uses `plpgsql`. Oracle uses `PL/SQL`.

11. Views

Postgres:

```
CREATE VIEW emp_view AS
SELECT name, salary FROM employees WHERE salary > 30000;
```

Oracle:

```
CREATE VIEW emp_view AS
SELECT name, salary FROM employees WHERE salary > 30000;
```

Difference: Same.

---

12. Transactions & ACID

Postgres:

```
BEGIN;
UPDATE employees SET salary = salary + 1000 WHERE id = 1;
COMMIT;
ROLLBACK; -- undo if needed
```

Oracle:

```
BEGIN
  UPDATE employees SET salary = salary + 1000 WHERE id = 1;
END;
COMMIT;
ROLLBACK;
```

ACID Properties: Atomicity, Consistency, Isolation, Durability.

---

13. Users & Permissions

Postgres:

```sql
CREATE USER khushi WITH PASSWORD 'pass123';
CREATE ROLE devs;
GRANT devs TO khushi;
GRANT SELECT, INSERT ON employees TO khushi;
REVOKE INSERT ON employees FROM khushi;
```

Oracle:

```sql
CREATE USER khushi IDENTIFIED BY pass123;
CREATE ROLE devs;
GRANT devs TO khushi;
GRANT SELECT, INSERT ON employees TO khushi;
REVOKE INSERT ON employees FROM khushi;
```

Difference: Very similar, but Oracle requires `CREATE SESSION` privilege for a new user to log in.

✅Summary

- **Core SQL (CREATE, INSERT, SELECT, UPDATE, DELETE, JOINs, aggregates)** → almost identical.
- **Differences**: Data types (`VARCHAR` vs `VARCHAR2`), auto-increment (`SERIAL` vs SEQUENCE/IDENTITY), limiting rows, date/time functions, procedural languages (`plpgsql` vs PL/SQL).