# SQL Statements (DDL & DML)

---

## 🔹 1) DDL — Data Definition Language

### ✅ CREATE TABLE

ينشئ **Table** جديد ويحدد **Columns** و **Constraints**.

**مثال بسيط:**

```
CREATE TABLE users (
  id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  username VARCHAR(50) UNIQUE NOT NULL,
  password TEXT NOT NULL,
  age INT,
  created_at TIMESTAMP DEFAULT NOW()
);
```

**مثال بمفتاح أجنبي (Foreign Key) وقيود (CHECK):**

```
CREATE TABLE orders (
  id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  user_id INT NOT NULL,
  amount NUMERIC(10,2) NOT NULL CHECK (amount >= 0),
  status VARCHAR(20) NOT NULL DEFAULT 'pending',
  CONSTRAINT fk_orders_users
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

> **ملاحظة:** في PostgreSQL يمكنك استخدام `SERIAL` أو الصيغة الحديثة `GENERATED AS IDENTITY` لإنشاء أعداد تلقائية.
>
> ملاحظة: استخدام **GENERATED ... AS IDENTITY** هو الأسلوب الحديث بدل **SERIAL**.

---

### ✅ ALTER TABLE

يعدل بنية جدول موجود.

**إضافة عمود:**

```
ALTER TABLE users
ADD email VARCHAR(120);
```

**إعادة تسمية عمود:**

```
ALTER TABLE users
RENAME COLUMN username TO user_name;
```

**تغيير نوع عمود (مع تحويل القيم USING):**

```
ALTER TABLE users
ALTER COLUMN age TYPE SMALLINT USING age::SMALLINT;
```

**تعيين قيمة افتراضية/إزالتها:**

```
ALTER TABLE users
ALTER COLUMN created_at SET DEFAULT NOW();

ALTER TABLE users
ALTER COLUMN created_at DROP DEFAULT;
```

**حذف عمود:**

```
ALTER TABLE users
DROP COLUMN email;
```

**إضافة قيد UNIQUE بعد الإنشاء:**

```
ALTER TABLE users
ADD CONSTRAINT uq_users_user_name UNIQUE (user_name);
```

---

## ✅ DROP TABLE

يحذف الجدول بالكامل.

**حذف مباشر:**

```sql
DROP TABLE orders;
```

**حذف إن وُجد (لتجنب الخطأ):**

```sql
DROP TABLE IF EXISTS orders;
```

**الحذف مع الكيانات المعتمدة (CASCADE):**

```sql
DROP TABLE users CASCADE; -- يحذف أي FKs/Views معتمدة
```

> احذر استخدام **CASCADE** لأنه قد يحذف كيانات مرتبطة.

---

## 🔹 2) DML — Data Manipulation Language

### ✅ INSERT INTO

يضيف **Rows** جديدة.

**إدراج صف واحد:**

```sql
INSERT INTO users (user_name, password, age)
VALUES ('ziad', 'secret', 22);
```

**إدراج عدة صفوف:**

```sql
INSERT INTO users (user_name, password, age)
VALUES
  ('ali', 'p1', 25),
  ('mona', 'p2', 28),
  ('sara', 'p3', 30);
```

**ال INSERT مع RETURNING (ترجّع القيم المولدة):**

```sql
INSERT INTO orders (user_id, amount, status)
VALUES (1, 199.99, 'paid')
RETURNING id, created_at;
```

**ال INSERT من SELECT:**

```sql
INSERT INTO orders (user_id, amount, status)
SELECT id, 0, 'pending' FROM users WHERE age >= 30;
```

---

## ✅ UPDATE

يعدل بيانات موجودة.

**تحديث بسيط بشرط WHERE:**

```sql
UPDATE users
SET age = age + 1
WHERE user_name = 'ziad';
```

**ال UPDATE مع RETURNING:**

```sql
UPDATE orders
SET status = 'shipped'
WHERE id = 10
RETURNING id, status;
```

**ال UPDATE باستخدام JOIN (بـ FROM):**

```sql
-- مثال: زوّد amount 10% لكل طلبات مستخدمين أعمارهم > 25
UPDATE orders o
SET amount = o.amount * 1.10
FROM users u
WHERE o.user_id = u.id
  AND u.age > 25;
```

---

## ✅ DELETE

يحذف Rows.

**حذف بشرط:**

```sql
DELETE FROM users
WHERE age < 18;
```

ال **DELETE** مع **RETURNING** لمعرفة ما تم حذفه:

```
DELETE FROM orders
WHERE status = 'pending'
RETURNING id, user_id;
```

ال **TRUNCATE** لحذف كل الصفوف بسرعة (بدون **WHERE**):

```
TRUNCATE TABLE orders RESTART IDENTITY; -- يُصفّر عدّاد الـ IDENTITY
```

---

## ✅ SELECT

يجلب البيانات.

**أساسيات:**

```
SELECT id, user_name, age
FROM users
WHERE age >= 20
ORDER BY age DESC
LIMIT 5 OFFSET 0;
```

ال **DISTINCT** (إرجاع قيم فريدة فقط):

```
SELECT DISTINCT status
FROM orders;
```

تجميع **Aggregation + GROUP BY + HAVING**:

```
SELECT user_id, COUNT(*) AS orders_count, SUM(amount) AS total_amount
FROM orders
GROUP BY user_id
HAVING SUM(amount) > 500;
```

ال **JOIN** بين جدولين:

```
SELECT u.user_name, o.id AS order_id, o.amount, o.status
FROM users u
JOIN orders o ON o.user_id = u.id
```

```
WHERE o.status = 'paid'
ORDER BY o.amount DESC;
```

---

## 🔹 نصائح سريعة

- دايمًا اختم الـ **Statement** بـ `;` .
- استخدم **WHERE** بحذر في **UPDATE/DELETE** لتجنب التعديل/الحذف الشامل.
- في PostgreSQL، الأسماء غير المقتبسة تتحول **lowercase** تلقائيًا؛ لو استخدمت اقتباس مزدوج `"Name"` ستصبح **Case Sensitive**.
- استخدم **RETURNING** للاستعلام عن الصفوف التي تم التأثير عليها بدون `SELECT` إضافي.

---