Conditions & Filtering

🔹 Introduction

عند كتابة **SQL queries**، جزء كبير من القوة بييجي من قدرات **التصفية (Filtering)** وفرض الشروط على الصفوف باستخدام الـ **WHERE clause**، pattern matching، واستخدامات `IN` و `BETWEEN` ، وإرشادات للأداء.

✅ WHERE clause

- **الوصف:** تُستخدم لتحديد الشروط اللي يجب أن تتحقق لكي يتم إرجاع السطر ضمن نتيجة الـ `SELECT` (أو لتحديد الصفوف المتأثرة في `UPDATE` / `DELETE`).

التركيب العام:

```
SELECT columns
FROM table
WHERE condition;
```

مثال بسيط:

```
SELECT *
FROM users
WHERE age >= 18;
```

- هذا يرجع كل المستخدمين الذين عمرهم >= 18.

ملاحظة مهمة:

- إذا لم تذكر `WHERE` في `UPDATE` أو `DELETE` فسيتم تعديل/حذف كل الصفوف — احذر.

✅ Logical operators: AND, OR, NOT

- ال **AND** → كل الشروط المرتبطة بـ `AND` يجب أن تكون صحيحة.
- ال **OR** → يكفي أن يتحقق أحد الشروط.
- ال **NOT** → ينفي الشرط.

أولوية التنفيذ:

1. NOT (أعلى أسبقية)
2. AND
3. OR (أدنى أسبقية)

مثال:

```
-- جلب المستخدمين الذين عمرهم بين 18 و 30 ومنطقتهم 'Cairo'
SELECT * FROM users
WHERE age BETWEEN 18 AND 30
  AND city = 'Cairo';

-- استخدام NOT
SELECT * FROM products
WHERE NOT (price > 100); -- نفس معنى price <= 100
```

استخدام الأقواس: دائماً استعمل `()` عندما تكون الشروط مركبة لتوضيح النية ومنع الغموض:

```
-- الفرق: هل نريد (A AND B) OR C أم A AND (B OR C) ؟
SELECT * FROM t
WHERE (A AND B) OR C;
```

✅ Pattern matching: LIKE & wildcards

- ال **LIKE** يستخدم للمطابقة النمطية البسيطة على النصوص.
- ال **Wildcards الشائعة:**
  - `%` → أي سلسلة (صفر أو أكثر من الأحرف)
  - `_` → حرف واحد فقط

أمثلة:

```
-- يبدأ بـ 'Ab'
SELECT * FROM users WHERE name LIKE 'Ab%';

-- ينتهي بـ 'son'
SELECT * FROM users WHERE name LIKE '%son';

-- يحتوي على 'admin' في أي مكان
SELECT * FROM users WHERE name LIKE '%admin%';
```

```
-- مثلا : حرف ثالث يمكن يكون أي حرف 'Jo_' يطابق 'Jon' و 'Joe'
SELECT * FROM users WHERE username LIKE 'Jo_';
```

الحساسية لحالة الأحرف (Case Sensitivity):

- في **PostgreSQL**: `LIKE` حسّاس لحالة الأحرف حسب collation، لكن يوجد `ILIKE` لعدم الحساسية (case-insensitive):

```
SELECT * FROM users WHERE name ILIKE '%ali%'; -- يطابق 'Ali' أو 'ALI' أو 'ali'
```

- في **MySQL** الافتراضي `LIKE` عادة غير حساس لحالة الأحرف على collations الشائعة.

هروب wildcard: لو تريد تبحث عن `%` أو `_` حرفيًا، استعمل `ESCAPE` أو تابع قواعد الـ RDBMS:

```
SELECT * FROM texts WHERE content LIKE '%50\%%' ESCAPE '\'; -- يبحث عن '50%'
```

أداء الـ LIKE:

- `LIKE 'prefix%'`
  - يمكن استغلال الفهرس (index) عادةً → أداء جيد.
- `LIKE '%substring%'`
  - **لا يستغل الفهرس** ويؤدي إلى مسح جدولي (full table scan). للحالات المعقدة استخدم Full-Text Search أو أدوات بحث متخصصة.

✅Using IN and BETWEEN

IN

- **الوصف:** للتحقق إذا كانت قيمة العمود موجودة ضمن قائمة من القيم.
- **مثال:**

```
SELECT * FROM orders
WHERE status IN ('pending', 'processing', 'shipped');
```

- **IN مع Subquery:**

```
SELECT * FROM users
WHERE id IN (SELECT user_id FROM orders WHERE amount > 1000);
```

- ال **NOT IN**: انتبه عند وجود `NULL` في القائمة أو في subquery — يمكن أن يؤدي إلى سلوك غير متوقع (النتيجة قد تكون فارغة). استخدم `NOT EXISTS` كبديل آمن في هذه الحالة.

BETWEEN

- **الوصف:** للاختبار إن القيمة داخل نطاق متضمن (inclusive)
- **الصيغة:** `column BETWEEN low AND high` تعادل `column >= low AND column <= high`.

مثال:

```
SELECT * FROM products
WHERE price BETWEEN 100 AND 200; -- يشمل 100 و 200
```

- **ملاحظة:** `BETWEEN` يعمل على التواريخ أيضاً:

```
SELECT * FROM events
WHERE event_date BETWEEN '2025-01-01' AND '2025-12-31';
```

✅ NULL handling (جزء مهم مرتبط بالفلترة)

- ال `NULL` يعني "قيمة غير معروفة" — لا تساوي أي قيمة، ولا حتى نفسها.
- لا تكتب `WHERE column = NULL` — هذا خاطئ.
- استخدم:
- `IS NULL` أو `IS NOT NULL`

أمثلة:

```
SELECT * FROM users WHERE deleted_at IS NULL; -- لم يتم حذفهم
SELECT * FROM profiles WHERE bio IS NOT NULL; -- لديهم bio
```

التفاعل مع NOT IN و NULL:

- لو استعملت `NOT IN (subquery)` والـ subquery يعيد قيمة NULL، فالميزة قد ترجع فارغة. الحل: استعمل `NOT EXISTS` أو فلتر NULL داخل الـ subquery.

✅ Advanced pattern options (Postgres examples)

- **ILIKE**: case-insensitive LIKE.
- **SIMILAR TO**: نمط regex بسيط.

- **POSIX regex (`~`, `~*`)**:
  - `~` = regex case-sensitive
  - `~*` = regex case-insensitive

أمثلة:

```
SELECT * FROM users WHERE email ~ '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'; -- regex
SELECT * FROM users WHERE name ILIKE '%omar%';
```

🔹 Performance tips (Sargability & Index usage)

- اجعل الشروط **sargable** لكي يستفيد من الـ index: يعني لا تلف العمود بدالة في الـ WHERE.
  - سيء: `'WHERE LOWER(name) = 'ali` → يمنع استخدام index على `name` (إلا لو عندك index خاص بالـ (lower(name)).
  - أفضل: `'WHERE name ILIKE 'ali` في Postgres أو أنشئ index على التعبير.
- استخدم `EXPLAIN` أو `EXPLAIN ANALYZE` لمعرفة خطة التنفيذ.
- لتصفية نصوص كاملة أو كلمات داخلية استخدم Full-Text Search (مثل `tsvector` / `tsquery` في Postgres) بدلاً من `'%LIKE '%term` .

🔹 أمثلة مركّبة شاملة

مثال 1 — استعلام مركّب:

```
SELECT u.id, u.name, o.total
FROM users u
JOIN orders o ON o.user_id = u.id
WHERE u.country = 'EG'
  AND (o.status = 'paid' OR o.status = 'shipped')
  AND o.created_at BETWEEN '2025-01-01' AND '2025-06-30'
  AND u.email IS NOT NULL
ORDER BY o.created_at DESC;
```

مثال 2 — استخدام IN مع Subquery آمن:

```
SELECT * FROM products p
WHERE p.id NOT IN (
  SELECT product_id FROM discontinued_products WHERE product_id IS NOT
```

```
NULL
);
-- أو الأفضل باستخدام NOT EXISTS:
SELECT * FROM products p
WHERE NOT EXISTS (
  SELECT 1 FROM discontinued_products d WHERE d.product_id = p.id
);
```

🔹 الخلاصة

- ال `WHERE` هي البوابة لتصفية الصفوف — استخدمها بحذر مع `UPDATE/DELETE` .
- استعمل `()` لتنظيم الشروط المركبة وتأمين الأولويات.
- ال `LIKE` جيد للبحث النصي البسيط؛ لكن `LIKE '%term%'` بطيء — فكر في Full-Text Search.
- ال `IN` مريح لكن احذر من `NULL` في القوائم/ `NOT EXISTS` — subqueries أكثر أمانًا.
- ال `BETWEEN` مفيدة للتواريخ والنطاقات الرقمية.
- راقب الأداء باستخدام `EXPLAIN` واهتم بأن تكون شروطك sargable عندما تحتاج سرعة.
