Indexes

🔹 ماذا يعني "Index"؟

- ال **Index** في قواعد البيانات شبيه بصفحة الفهرس في كتاب: بنعمل بنية بيانات خاصة عشان نقدر نلاقي الصفوف بسرعة من غير ما نمسح الجدول كله (Full Table Scan).
- في الغالب الـ DBMS بيستخدم **B-Tree** كهيكل افتراضي، لكن فيه هياكل تانية (Hash، GIN، GiST، Trigram …).

🔹 لماذا نستخدم Index؟ (الفوائد)

1. **تسريع عمليات البحث (SELECT)** — بدل بحث لكل الصفوف، الفهرس يوصلك مباشرة لمعظم النتائج.
2. **تسريع JOINs** — وجود index على أعمدة الربط (FK/PK) يقلّل زمن الربط كثيرًا.
3. **تسريع ORDER BY وGROUP BY** في بعض الحالات عندما يكون الـ index مناسبًا.
4. **فرض قيود UNIQUE** — بعض أنواع الـ index تمنع تكرار القيم.

🔹 أنواع الـ Index الشائعة

1) Primary Index

- يُنشأ تلقائيًا على الـ **PRIMARY KEY**.
- يضمن uniqueness وNOT NULL.

2) Unique Index

- يمنع تكرار القيم في العمود.

```
CREATE UNIQUE INDEX idx_users_email ON users(email);
```

3) Composite (Multi-column) Index

- Index على أكثر من عمود معًا.
- مفيد عندما تبحث عادةً باستخدام أكثر من عمود مرتبًا بنفس الترتيب.

```sql
CREATE INDEX idx_users_lastname_firstname ON users(lastname, firstname);
```

- **ملاحظة مهمة:** Composite index يستفيد فقط إذا كانت شروط WHERE تبدأ بالعمود الأول في الترتيب.

4) Full-Text Index

- مخصص للبحث داخل نص طويل (مثل مقالات، أوصاف).
- في PostgreSQL نستخدم `tsvector` مع **GIN index**.

```sql
CREATE INDEX idx_articles_fts ON articles USING GIN
(to_tsvector('english', body));
```

5) Trigram / pg_trgm

- مفيد للبحث عن أجزاء من النص (`LIKE '%term%'`) أو التشابه (similarity).

```sql
CREATE EXTENSION IF NOT EXISTS pg_trgm;
CREATE INDEX idx_users_name_trgm ON users USING GIN (name gin_trgm_ops);
```

6) Clustered vs Non-Clustered

- ال **Clustered Index**: يعيد ترتيب الصفوف فعليًا وفق ترتيب المفتاح (مثال SQL Server). الجدول الواحد يمكن أن يملك clustered index واحد فقط.
- ال **Non-Clustered Index**: بنية منفصلة تشير إلى صفوف الجدول.
- في PostgreSQL الفهرس الافتراضي B-Tree هو non-clustered، لكن يمكنك `CLUSTER` الجدول يدوياً لتعيد الترتيب.

🔹 تقنيات فنية إضافية (خاصة بـ PostgreSQL)

- ال **GIN / GiST**: لهياكل متقدمة للفهارس على البيانات المعقدة (arrays, jsonb, full-text).
- ال **BRIN**: مناسب للجداول الضخمة جدًا حيث القيم متراصة ترتيبًا (مثال: timestamp متزايد) — خفيف وصغير.

🔹 متى لا أضع Index؟ (سلبيات وقيود)

- الأعمدة التي تحتوي على **قيمتين فقط** (مثلاً boolean مع تكرار كبير) عادة لا تستفيد من index.
- الجداول الصغيرة — Full Table Scan قد يكون أسرع من استخدام الفهرس.

- إذا الجدول يشهد **كتابة كثيفة (INSERT/UPDATE/DELETE)**، كل فهرس إضافي يزيد تكلفة الكتابة لأن الفهارس يحتاج تحديث.
- الفهرس يشغل مساحة (Disk + RAM عند التحميل).

🔹 أمثلة عملية وأوامر شائعة

- **إنشاء Index بسيط:**

```
CREATE INDEX idx_users_username ON users(username);
```

- **إنشاء Index وحذفة:**

```
DROP INDEX idx_users_username; -- Postgres
-- MySQL: DROP INDEX idx_users_username ON users;
```

- **إنشاء Index فريد:**

```
CREATE UNIQUE INDEX uq_users_email ON users(email);
```

- **فحص خطة التنفيذ (EXPLAIN):**

```
EXPLAIN ANALYZE SELECT * FROM users WHERE username = 'ziad';
```

استخدم `EXPLAIN` عشان تشوف هل الاستعلام بيستعمل الفهرس ولا لا.

🔹 نصائح عملية (Best Practices)

- أضف index على الأعمدة المستخدمة كثيرًا في `ORDER BY` , `JOIN` , `WHERE` و `GROUP BY` .
- استخدم **Composite Index** عندما تكون الاستعلامات تستخدم مجموعة أعمدة ثابتة الترتيب.
- راقب أداء الكتابة — لا تفرط في إنشاء فهارس على أي عمود غير ضروري.
- استخدم `EXPLAIN` و(Postgres) `pg_stat_user_indexes` لمراقبة استخدام الفهارس.
- قم بعمليات صيانة دورية: `ANALYZE` , `VACUUM` , و `REINDEX` عند اللزوم.

🔹 خلاصة

- ال **Index = تحسين سرعة القراءة مقابل تكلفة في الكتابة/المساحة.**

- اختيار النوع الصحيح من الفهرس يعتمد على طبيعة البيانات ونوع الاستعلامات.
