> 👉 Top Secret 👈

Indizes

I ***a***) *Einführung*

- **Indizes und Performance**:
  - Indizes beschleunigen Abfragen, zB. bei `SELECT`, `UPDATE`, `DELETE`
  - Ohne Index wird ein Full Table Scan durchgeführt – ineffizient für große Tabellen
  - Ein Index sorgt für sortierte Datenstrukturen und schnelleres Suchen
- *Beispiele*:
  - Ein `Primary Key` ***wird immer*** mit einem Index versehen
  - Ein `UNIQUE` Constraint ***wird ebenfalls automatisch*** indexiert
  - Ein `Foreign Key` hingegen ***nicht*** *automatisch*, sollte jedoch manuell indexiert werden

---

I ***b***) Index: *Grundlagen und Analogien*

- **Sortiertes Inhaltsverzeichnis**:
  - ***Ein Index ähnelt dem Inhaltsverzeichnis*** eines Buches

    > Statt das gesamte Buch zu durchsuchen, gibt der Index direkt die relevante Seitennummer an
  - *Vergleich*:

    > **Papier-Telefonbuch**:
    >
    > - Sortiert nach Ort, Nachname, Vorname
    > - Suche nach Vorname erfordert jedoch das Durchsuchen des gesamten Telefonbuchs
    >
    > *Ein Index verbessert Performance ähnlich einem Buchindex*

I ***c***) *Index-Typen*

Grundsyntax für `CREATE INDEX`

```
CREATE INDEX index_name          -- eindeutiger name des zu erstellenden
indexes
```

```
    ON table_name                -- name der tabelle, für die der index
erstellt wird
     ( column1, column2, ... );  -- spalten, auf denen der index basiert
```

- 1) **B-Tree Index**:
  - ***Standardindex*** mit binärer Suche
  - **Balanciert**, um gleich schnelle Zugriffe zu ermöglichen
    - Ideal für Gleichheitsabfragen (=), Bereichsabfragen (> , <) und Sortierungen
    - Gespeichert in einer B-Baum-Struktur für effiziente Suche, Einfügen und Löschen

```
CREATE INDEX idx_emp_salary
  ON employees
     ( salary );
```

- 2) **Bitmap Index**:
  - Geeignet für Spalten ***mit wenigen unterschiedlichen Werten*** (zB. Geschlecht)
  - *Ein Index-Eintrag zeigt auf mehrere Zeilen*
    - Speichern Informationen über die Verteilung von Werten in einer Spalte in komprimierter Form
    - Effizient für Spalten mit wenigen verschiedenen Werten und Gleichheitsabfragen

```
CREATE BITMAP INDEX idx_dept_no
  ON departments
     ( department_no );
```

- 3) **Funktionsbasierter Index**:
  - Index auf einer ***berechneten Spalte oder Funktion*** (zB. `SUM(column)`)
    - Basieren auf einer Funktion, die auf einer oder mehreren Spalten angewendet wird
    - Ermöglichen das Indizieren von berechneten Werten oder komplexen Ausdrücken

```
CREATE INDEX idx_emp_name_upper
  ON employees
     ( UPPER(name) );
```

    - 4) **Clustered Index**:
      - Physisch sortierte Tabellenstruktur, oft in index-organisierten Tabellen verwendet
    - 5) **Bitmap Join Index**:

    - Index einer Tabelle auf Spalten einer anderen Tabelle (zB. zum Verknüpfen von Fremdschlüsseln)

- 6) **Reverse-Key-Indizes**

    - Speichern die Werte in umgekehrter Reihenfolge
    - Effizient für Bereichsabfragen mit großen Wertebereichen

```
CREATE INDEX idx_emp_hire_date_rev
  ON employees
     ( REVERSE(hire_date) );
```

- **7) Weitere Parameter**

    - `TABLESPACE`: Gibt den Tablespace an, in dem der Index gespeichert werden soll
    - `PCTFREE`: Legt den Prozentsatz des freien Raums in jedem Block fest
    - `INITTRANS`: Bestimmt die Anzahl der Transaktionen, die gleichzeitig den Index aktualisieren können
    - `MAXTRANS`: Gibt die maximale Anzahl gleichzeitiger Transaktionen an
    - `STORAGE`: Definiert Speicheroptionen wie Blockgröße und Komprimierung

```
CREATE INDEX idx_order_date_cust_id
  ON orders
     ( order_date, customer_id )
    TABLESPACE idx_tbs
    PCTFREE 10
    INITTRANS 2
    MAXTRANS 20;
```

I ***d***) *Inserts und Constraints*

- ***Performance von `INSERT` mit Index***:
    - Bei jedem `INSERT` prüft die Datenbank, ob ein Datensatz mit dem gleichen `Primary Key` existiert
    - **Ohne** Index muss die **gesamte Tabelle durchsucht** werden
    - **Mit** Index erfolgt die Prüfung wesentlich **schneller**
- ***Strategien für große Datenimporte***:
    - **Constraints deaktivieren**:

        - Lade alle Daten ohne Überprüfung der Constraints
        - Schalte die Constraints danach wieder ein

> - *Dies ist effizienter, da nicht jede Zeile einzeln geprüft wird*

- **Index deaktivieren**:

> - Lade die Daten in die Tabelle, ohne den Index zu aktualisieren
> - Aktiviere den Index erst nach dem Laden

I ***e***) *Updates und Indizes*

- **Effekte von Indizes bei** `UPDATE`:
  - Ein *`UPDATE` auf einer indizierten Spalte muss sowohl die Tabelle als auch den Index aktualisieren*
  - ***Schreiboperationen*** auf Spalten mit vielen Indizes ***sind daher langsamer***
  - ***Indizes lohnen sich vor allem bei häufigen Leseoperationen***

- **Taktik**:
  - Vor Datenänderungen Indizes deaktivieren und anschließend wieder aktivieren, um Performance zu steigern

---

I ***f***) *Sortierung und Platz*

- Sortierung in Tabellen:
  - **Ohne Index** sind Tabellen **unsortiert**
  - Ein Index sorgt für eine logische Sortierung, physisch bleibt die Tabelle unsortiert

- Speicherplatz und Fragmentierung:
  - Variablenlängenfelder (zB. `VARCHAR`) können mehr oder weniger Platz benötigen
  - **Lücken** entstehen durch gelöschte oder geänderte Datensätze
  - Effiziente Nutzung des Speicherplatzes hängt von der Datenstruktur ab

---

I ***g***) *Abfrageoptimierung*

- Effizienz durch Index Only Scan:
  - Manche Abfragen können direkt über den Index beantwortet werden, ohne auf die Tabelle zuzugreifen
  - *Beispiel*: `COUNT(*)` auf einer indizierten Spalte

- Funktionsbasierte Indizes:
  - Nützlich bei Abfragen mit Funktionen wie `AVG(column)`:

```
CREATE INDEX idx_abs_value ON table (AVG(column));
```

- Verbessern die Performance bei komplexen Abfragen

- Probleme durch Funktionen:
  - Funktionen wie `TO_NUMBER` verhindern die Nutzung eines Index

    > **Lösung**: Einen ***funktionsbasierten Index*** erstellen oder die Abfrage umschreiben

---

I ***h***) *Datenbanken ohne Indizes*

- Vollständige Tabellenscans:
  - Ohne Index werden bei jeder Abfrage alle Zeilen durchsucht

- Abfrageszenarien ohne Index:
  - *Beispiel*: Abfragen mit `YEAR(date_column)` zwingen die Datenbank zu einem Full Table Scan

---

I ***h***) *Indizes in Szenarien*

- 1. Update einer Tabelle mit einem Index:
  - Wenn auf eine indizierte Spalte zugegriffen wird, müssen auch alle Indizes aktualisiert werden
  - *Beispiel*:

    ```
    UPDATE employees SET lastname = 'Smithson' WHERE lastname =
    'Smith';
    ```

- 2. Hinzufügen eines Index für seltene Abfragen:
  - Wenn eine Tabelle selten gelesen, aber oft beschrieben wird, ist die Abwägung wichtig
  - *Beispiel*: Jahresbilanzprüfung, wo Abfragen schnell erfolgen sollen

- 3. Index auf zusammengesetzten Schlüsseln:
  - *Beispiel*: Index auf `(lastname, firstname)`

    > Die Suche nach `lastname` ist ***schnell***, die Suche nur nach `firstname` hingegen ***ineffizient***
