Indizes & Laufzeitkomplexität

1.)

Folgende Abfragen sollen beschleunigt werden.

> *Erstellen Sie - für jede Abfrage einzeln betrachtet - einen oder mehrere Index/izes, so dass jede Abfrage für sich bestmöglich abgearbeitet werden kann.*

a)

```
SELECT * FROM emp WHERE ENAME =: some_name;
```

```
CREATE INDEX emp_ename ON emp(ename);
```

b)

```
SELECT * FROM emp WHERE job=:sel_job AND sal > :min_sal ORDER BY hiredate DESC;
```

```
CREATE INDEX idx_emp_job_sal_hiredate ON emp(job, sal, hiredate   DESC);
```

c)

```
SELECT * FROM emp WHERE (ROUND(sal/1000)*1000) = 2000;
```

- (alle Mitarbeiter, deren Verdienst auf ganze Tausender gerundet 2000 beträgt)
- `(ROUND(sal/1000)*1000)` verhindert die direkte Verwendung eines einfachen Index auf `sal` .
- **Function based Index** :

```
CREATE INDEX emp_rd_sal_idx ON emp(ROUND(sal / 1000) * 1000);
```

- **2.)**

Welche Zugriffsarten werden für folgende Queries verwendet?

(begründen Sie)

> *Gehen Sie davon aus, dass die abgefragten Tabellen eine große Zahl von Zeilen enthalten und dass die gegebenen Indexe nur für den jeweiligen Unterpunkt der Aufgabe existieren.*
>
> (DDL/DML zu Tabelle inventories siehe DataWarehouse.zip)

- **a)**

```
SELECT COUNT(*) FROM table;
```

  - **i)**

    ### *Ohne Primary-Key, ohne Index* :

    - FULL TABLE SCAN

      > ***Dies ist die langsamste Methode, insbesondere bei großen Tabellen.***

  - **ii)**

    ### *Ohne Primary-Key, beliebiger Index* :

    - Da im Index ***keine*** `NULL` ***-Werte enthalten*** sind muss entweder die Query
      > *zB:* (.. `WHERE xyz IS NOT NULL, WHERE xyz > 10` ..) - **`NULL` -Werte ausschließen**,

      oder die indizierte Spalte **`NOT NULL`** sein.
    - Sonst **kann der Index nicht verwendet werden**, da eventuelle `NULL` -Werte übersehen würden.

  - **iii)**

    ### *Ohne Primary-Key, beliebiger `NOT NULL` Index* :

    - FULL INDEX SCAN

- **iv)**

  Mit Primary Key

  - `FULL INDEX SCAN`

- **b)**

```
CREATE INDEX inventories_product_idx ON inventories(product_id);
SELECT SUM(quantity) FROM inventories WHERE product_id = 210;
```

`INDEX RANGE SCAN + TABLE ACCESS BY ROWID`

- (da quantity nicht im Index enthalten ist)

- **c)**

```
CREATE INDEX inventories_quantity_idx ON inventories(quantity);
CREATE INDEX inventories_product_idx ON inventories(product_id);
SELECT SUM(quantity) FROM inventories WHERE product_id = 210;
```

`INDEX RANGE SCAN + TABLE ACCESS BY ROWID`

- ***wie b.)***
  (der zusätzliche Index auf quantity kann nicht benützt werden)

- **d)**

```
CREATE INDEX inventories_prod_quant_idx ON inventories(product_id, quantity);
SELECT SUM(quantity) FROM inventories WHERE product_id = 210;
```

`INDEX RANGE SCAN`

- (Da sämtliche Daten im Index enthalten sind ***entfällt der Table Access By Rowid***)

- **e)**

```
CREATE INDEX inventories_quant_prod_idx ON inventories(warehouse_id, product_id);
SELECT COUNT(product_id) FROM inventories WHERE product_id = 210;
```

Entweder `FAST FULL SCAN` ***oder*** `SKIP SCAN .`

3.)

Welcher Komplexitätsklasse gehörden folgende Operationen an:

- **a)**

```
SELECT * FROM emp WHERE sal > 1000;
```

(ohne index)

- Das Datenbanksystem muss die gesamte Tabelle durchsuchen, um Zeilen mit sal > 1000 zu finden.
  - $O(n)$
    > Lineare Zeitkomplexität

- **b)**

```
SELECT ename FROM emp ORDER BY ename;
```

(kein Index)

- Das Datenbanksystem muss alle Zeilen basierend auf der Spalte ename sortieren.
  - $O(n \log n)$
    > Sortieralgorithmen haben typischerweise diese Komplexität

- **c)**

```
SELECT e.ename, e.sal, (SELECT COUNT(*) FROM emp e2 WHERE e2.sal < e.sal) FROM emp e;
```

(ohne index auf `sal`)

- Für jede Zeile in der äußeren Abfrage muss die innere Abfrage die gesamte Tabelle durchsuchen, um Zeilen mit einem niedrigeren Gehalt zu zählen.
  - $O(n^2)$
    > Quadratische Zeitkomplexität, da die innere Abfrage für jede Zeile der äußeren Abfrage ausgeführt wird

> ### *Spezialfall*
>
> - **d)**
>
> ```
> SELECT e.ename, e.sal, ( SELECT COUNT(*) FROM emp e2 WHERE e2.sal < e.sal ) FROM emp > e;
> ```
>
> *(mit index auf `sal`)*
>
> - Die innere Abfrage kann mithilfe des Index effizient die Anzahl der Zeilen mit > einem niedrigeren Gehalt ermitteln.
> - $O(n^2)$
>
> *dominierende Faktor ist die Sortierung der Ergebnisse der äußeren Abfrage*