

12.12.2019

## Übungsblatt 9

### Diskussionsteil (im PS zu lösen; keine Abgabe nötig)

- a) ☐ ★ Begründen Sie warum die Chained I/O Zugriffsmethode im Durchschnitt beim Lesen von mehreren Blöcken schneller ist als die Random I/O Zugriffsmethode.
- b) ☐ ★ (Blockweise Speicherung) Wenn Records variabler Länge verwendet werden, kann bei der Berechnung der Speicheradressen nicht mehr mit konstanten Offsets gearbeitet werden. Was muss stattdessen gemacht werden?
- c) ☐ ★ (Auswirkung der Datenanordnung auf Operationen) Füllen Sie Tabelle 1 aus indem Sie ein + einsetzen, wenn die Operation effizient möglich ist und ein - wenn sie teuer ist (ohne Verwendung von Indizes und Suche nach Primärindex).

	Sortierte Dateien	Unsortierte Dateien
Einfügen		
Suchen		
Löschen		
sortiertes Lesen		

Tabelle 1: Übersicht Datei-Operationen

- d) ☐ ★ Überlegen Sie ob das Einfügen der Werte  $X_1, X_2, \dots, X_n$  und Einfügen derselben Werte in umgekehrter Reihenfolge zum selben B-Baum führen.
- e) ☐ ★ Nehmen Sie an, in der Tabelle film (pagila Datenbank) gibt es jeweils einen B-Baum und einen B<sup>+</sup>-Baum-Index auf das Attribut film\_id<sup>1</sup>. Erstellen Sie eine Abfrage, die vom B-Baum-Index effizienter abgearbeitet werden könnte (im average case) und eine Abfrage, die effizienter vom B<sup>+</sup>-Baum-Index abgearbeitet werden kann. Worin unterscheiden sich diese Queries und wieso haben Sie diese gewählt?
- f) ☐ ★ Woran können Sie die Effizienz der Abfrageverarbeitung messen - beispielsweise um zwei Abfragen zu vergleichen oder den Mehrwert einer weiteren Indexstruktur zu evaluieren?
- g) ☐ ★ Diskutieren Sie, warum moderne Datenbanksysteme einen B<sup>+</sup>-Baum als Default-Indexstruktur anbieten und nicht z.B. Hash-Indizes zum Einsatz kommen.
- h) ☐ ★ Wie viele Vergleiche werden für die Suche eines Eintrags (bzw. des Pointers zum Datensatz) in einem B-Baum der Ordnung  $p$  mit  $n$  Einträgen benötigt? Bitte vervollständigen Sie dazu Tabelle 2.

<sup>1</sup>In der Realität werden in modernen DBMS nur B<sup>+</sup>-Bäume eingesetzt.

	B-Baum	B <sup>+</sup> -Baum
best case		
worst case		

Tabelle 2: B-Baum vs. B<sup>+</sup>-Baum

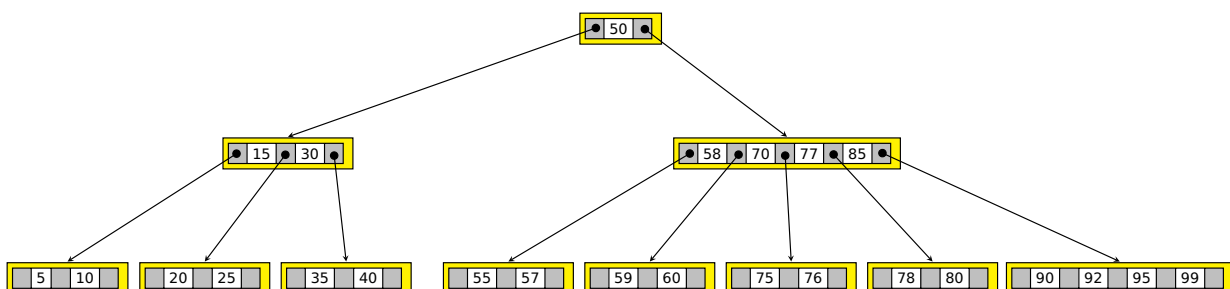
- i) ☐ ★★ Daten sollen in einem B-Baum organisiert werden. Die Größe einer Speicherseite der Festplatte betrage 2048 Bytes, die Größe eines Indexeintrages im B-Baum sei 20 Bytes (inkl. 8 Bytes für linken Teilbaumpointer). Berechnen Sie den für diesen B-Baum die optimale Ordnung.
- j) ☐ ★ In SQL ist es auch möglich, Spalten als “unique” zu kennzeichnen, um zu vermeiden, dass in einer Spalte ein Wert mehrfach auftritt (oder auch in einer Kombination von Spalten). Dies ermöglicht eine zusätzliche Überprüfung zur Datenkonsistenz.
- Unique wird in SQL als Index realisiert und kann z.B. beim Erzeugen einer Tabelle mit `UNIQUE (film_id)` erstellt werden. Welche Indexstruktur bietet sich hier an? Bedenken Sie, dass diese Unique-Bedingung für jedes Insert- und Update-Statement überprüft wird.

## Hausaufgabenteil (Zuhause zu lösen; Abgabe nötig)

### Aufgabe 1 (B-Baum Operationen)

[4 Punkte]

Gegeben sei der folgende B-Baum der Ordnung  $p = 5$ .



Führen Sie folgende Operationen jeweils **auf diesem Baum** aus. Zeichnen Sie dabei mindestens 2 Zwischenschritte und das Endergebnis auf und begründen Sie Ihr Vorgehen mit kurzen Kommentaren.

- a) ☐ 1 Punkt Einfügen von 87

**Abgabe**



sheet9\_1a.pdf

- b) ☐ 1.5 Punkte Löschen von 30

**Abgabe**

sheet9\_1b.pdf

c) 1.5 Punkte Löschen von 50**Abgabe**

sheet9\_1c.pdf

**Aufgabe 2 (Indexstrukturen in der Praxis)****[6 Punkte]**

Das Ziel dieser Aufgabe ist zu analysieren, wie Indizes in der Praxis funktionieren, welche Verbesserungen sie mit sich bringen und wie man diese am besten ausnützen kann. Im ersten Schritt werden wir über pgAdmin eine Datenbank aufsetzen und mit Daten aus dem IMDb Dataset<sup>2</sup> befüllen. Diese verwenden wir als Basis für die weiteren Schritte, wo wir verschiedene Abfragen ausführen und vergleichen.

- a) 1 Punkt Navigieren Sie zu <https://www.imdb.com/interfaces/> und verschaffen Sie sich einen Überblick über die Tabellen `title_akas` und `title_ratings` (Attribute, Schema, Datentypen, ...). Erstellen sie über pgAdmin eine neuen Datenbank namens `imdb` und bereiten Sie das Schema dieser Tabelle vor. Achten Sie darauf, korrekte Datentypen zu verwenden. Laden Sie anschließend das Paket `imdb-dataOLAT.zip` herunter und entpacken Sie die tsv-Dateien. Diese Dateien enthalten Daten für die Tabellen `title_akas` und `title_ratings`. Importieren Sie diese Daten in die entsprechende Tabelle. Beachten Sie, dass Sie das beim Import folgendes explizit angeben müssen (weitere Hinweise finden Sie weiter unten):

- NULL-Werte sind durch `\N` gekennzeichnet
- Die Dateien enthalten einen Header
- Spalten sind durch Tabs separiert
- `"` ist das Quote-Zeichen
- `'` müssen "escaped" werden

Führen Sie schließlich folgende Abfrage aus und geben Sie das Ergebnis als csv-Datei ab.

```
1  SELECT COUNT(titleid)
2  FROM    title_akas
3  WHERE   language IS NULL
```

**Abgabe** sheet9\_2a\_create\_db.sql  
 sheet9\_2a\_create\_table\_akas.sql  
 sheet9\_2a\_create\_table\_ratings.sql  
 sheet9\_2a\_query\_result.csv

<sup>2</sup><https://www.imdb.com/interfaces/>

### Hinweis



Falls Sie pgadmin verwenden, können Sie die tsv-Datei importieren indem Sie die Import/Export Funktion verwenden (Rechtsklick auf die gewünschte Tabelle).

### Hinweis



Achten Sie darauf, dass die Reihenfolge der Spalten übereinstimmen. Wenn es zu Fehlermeldungen beim Importieren kommt, stellen Sie sicher, dass die Definition der Tabelle korrekt ist oder machen Sie die Datentypen weniger restriktiv.

### Hinweis



Wenn Sie Docker verwenden, können Sie die tsv-Dateien in die imdb-Datenbank importieren, indem Sie folgenden Befehl ausführen:

```
1 cat <filename>.tsv | docker-compose exec -T db psql -U postgres \  
2 -d imdb -c "COPY <tablename>(<attr\_1>, ..., <attr\_n>) \  
3 FROM STDIN CSV DELIMITER E'\t' NULL '\N' QUOTE '\"' ESCAPE '\"' HEADER;"
```

Wobei Sie für <filename> den Pfad und den Namen der tsv-Datei angeben müssen, sowie für <tablename> den Tabellennamen und für <attr\_i> die entsprechenden Attribute angeben müssen.

- b) **1 Punkt** Führen Sie nun folgenden Abfrage aus und notieren Sie die Zeit (in der Query History zu sehen). Schreiben Sie diese Zeit im Format <seconds>s<milliseconds>msec in die Abgabedatei (es gibt hier natürlich nicht *die eine* korrekte Ausführungszeit).

```
1 SELECT MAX(ordering)  
2 FROM title_akas
```

Führen Sie dieselbe Abfrage mit EXPLAIN<sup>3</sup> aus. Wie wird die Abfrage vom DB-System abgearbeitet? Beschreiben Sie was in den einzelnen Schritten des Plans passiert, indem Sie auf die Operationen eingehen (z.B Aggregate, Partial Aggregate, Finalize Aggregate, Gather, Seq Scan, Parallel Seq Scan, Merge ...). Falls Sie pgAdmin verwenden, können Sie sich den Abfrageplan auch grafisch anzeigen lassen (F7).

### Abgabe



- sheet9\_2b\_time.txt
- sheet9\_2b\_explain\_result.txt
- sheet9\_2b\_explain\_description.txt

- c) **1 Punkt** Erstellen Sie mittels CREATE INDEX<sup>4</sup> einen aufsteigend sortierten BTREE-Index auf die Spalte ordering in der title\_akas Relation. Führen Sie folgende Abfrage aus und notieren Sie die Ausführungszeit im Format <seconds>s<milliseconds>msec.

<sup>3</sup><https://www.postgresql.org/docs/11/using-explain.html>

<sup>4</sup><https://www.postgresql.org/docs/11/sql-createindex.html>

```

1  SELECT MAX(ordering)
2  FROM    title_akas

```

Lassen Sie sich wieder mit EXPLAIN den vom Optimierer ermittelten Abfrageplans ausgeben und interpretieren Sie das Ergebnis indem Sie die einzelnen Schritte des Abfrageplans beschreiben. Erklären Sie dabei warum Begriffe wie *Limit*, *Forward*, *Backward*, *Scan*, *using* vorkommen, falls sie vorkommen. Beschreiben Sie wie sich das Erstellen des Index sich auf Ausführungszeit der Abfrage ausgewirkt hat. Führen Sie die Abfrage noch zwei weitere Male aus und achten Sie dabei auf die Ausführungszeit. Was können Sie beobachten? Führen Sie auch das in Ihrer Beschreibung an.

#### Abgabe



sheet9\_2c\_create\_index.sql  
 sheet9\_2c\_time.txt  
 sheet9\_2c\_explain\_output.txt  
 sheet9\_2c\_explain\_description.txt

d) **1 Punkt** Man spricht von zusammengesetzten (compound oder composite) Indizes, wenn der Index mehrere Spalten beinhaltet. Damit werden im B-Baum nicht einzelne Werte, sondern Tupel indiziert (z.B. (region, titleid)). Erstellen Sie einen zusammengesetzten B-Baum-Index über die Spalten *region* und *titleid* – in dieser Reihenfolge und beide Attribute aufsteigend sortiert. Folgende Abfragen sind gegeben:

1) Eine Abfrage, die beide Spalten *region* und *titleid* einschränkt

```

1  SELECT *
2  FROM    title_akas
3  WHERE   titleid = 'tt6996876'
4  AND     region = 'DE';

```

2) Eine Abfrage, die nur *titleid* einschränkt

```

1  SELECT *
2  FROM    title_akas
3  WHERE   titleid = 'tt6996876';

```

3) Eine Abfrage, die nur *region* einschränkt

```

1  SELECT *
2  FROM    title_akas
3  WHERE   region = 'DE';

```

4) Eine Abfrage, die nach *region* und *titleid* aufsteigend sortiert

```

1  SELECT *
2  FROM    title_akas
3  ORDER BY region ASC, titleid ASC

```

5) Eine Abfrage, die nach *region* aufsteigend und nach *titleid* absteigend sortiert

```

1  SELECT  *
2  FROM    title_akas
3  ORDER BY region ASC, titleid DESC

```

6) Eine Abfrage, die nach region und titleid absteigend sortiert

```

1  SELECT  *
2  FROM    title_akas
3  ORDER BY region DESC, titleid DESC

```

Betrachten Sie jeweils das Ergebnis des EXPLAIN-Aufrufs dazu. Welche Unterschiede in Ausführungszeit und Abarbeitungsstrategie bemerken Sie? Beschreiben Sie für welche Abfragen der zusammengesetzte Index Vorteile bringt und für welche nicht. Falls der zusammengesetzte Index bei einer Abfrage nicht verwendet wird, begründen Sie warum das (wahrscheinlich) so ist.

#### Abgabe



sheet9\_2d\_create\_compound\_index.sql  
 sheet9\_2d\_interpretation.txt

e) 2 Punkte Betrachten Sie die zwei folgenden Varianten einer SQL-Abfrage:

```

1  WITH languagecounts AS (
2    SELECT  titleid, COUNT(language) AS no_languages
3    FROM    title_akas
4    GROUP BY titleid
5  )
6  SELECT    tconst, no_languages
7  FROM      title_ratings
8  INNER JOIN languagecounts
9  ON        title_ratings.tconst = languagecounts.titleid
10 WHERE     no_languages > (
11   SELECT   AVG(no_languages)
12   FROM     languagecounts
13 )

```

–

```

1  SELECT    tconst, COUNT(language) AS no_languages
2  FROM      title_akas
3  INNER JOIN title_ratings
4  ON        title_ratings.tconst = title_akas.titleid
5  GROUP BY  tconst
6  HAVING     COUNT(language) > (
7    SELECT   AVG(languagecount)
8  FROM (

```




```
9      SELECT    COUNT(language) AS languagecount
10     FROM      title_akas
11     GROUP BY   titleid
12   ) AS tmp
13   )
```

Wir möchten diese Queries gerne näher analysieren und inspizieren, wie genau das DBMS diese Abfrage abarbeitet. Der JSON-Output von EXPLAIN gibt uns hierzu ausführliche Informationen, die wir im Folgenden vergleichen möchten. Beantworten Sie dazu folgende Fragen:

- 1) Welche Abfrage ist schneller?
- 2) Welche Schritte werden für die Abarbeitung der WITH-Abfrage durchgeführt? Erklären Sie die wichtigsten Operationen. Welche Schritte sind zeitlich gesehen die Bottlenecks der WITH-Abfrage?
- 3) Welche Schritte werden für die Abarbeitung der HAVING-Abfrage durchgeführt? Erklären Sie die wichtigsten Operationen. Welche Schritte sind zeitlich gesehen die Bottlenecks der HAVING-Abfrage?

#### Abgabe



-  sheet9\_2e\_answer\_1.txt
-  sheet9\_2e\_answer\_2.txt
-  sheet9\_2e\_answer\_3.txt

**Wichtig:** Laden Sie bitte Ihre Lösung in OLAT hoch und geben Sie mittels der Ankreuzliste auch unbedingt an, welche Aufgaben Sie gelöst haben. Die Deadline dafür läuft am Vortag des Proseminars um 23:59 (Mitternacht) ab.