database
research group

Datenbanken II

Wintersemester 2019/20

Praktische Aufgabe 2

Der Inhalt dieser übung wird im *1. Quiz* überprüft.

---

1 Dateiorganisation in *PostgreSQL*

Auch wenn *PostgreSQL* sehr mächtig erscheint, werden alle Daten in normalen Dateien im Dateisystem abgelegt. In diesem Teil der Übung untersuchen wir, wie diese Daten physisch gespeichert und organisiert werden, beziehungsweise, wie wieder darauf zugegriffen wird.

Wie bereits erwähnt werden alle Daten in Dateien abgelegt. Diese werden im **data directory** (Datenverzeichnis) von *PostgreSQL* gespeichert. Der Pfad des Datenverzeichnisses kann mit `SHOW DATA_DIRECTORY;` in der *PostgreSQL* Konsole auszulesen werden.

Beispiel 1 *Abfrage des Datenverzeichnisses.*

```
SHOW DATA_DIRECTORY;

/var/lib/postgresql/11/main
(1 row)
```

Für jede Datenbank exisitiert im Datenverzeichnis ein Unterverzeichnis *base/*, welches alle Tabellen und Daten der jeweiligen Datenbank enthält. Um ein Verzeichnis einer Datenbank zuordnen zu können, ist der Name des Verzeichnisses gleich der **OID** der Datenbank. OID steht für *Object Identifier*, dieser ist eindeutig für eine Datenbank und kann in `pg_database` mittels `SELECT oid, datname FROM pg_database;` ausgelesen werden. `pg_database` ist ein Katalog von *PostgreSQL*, der Informationen von allen existierenden Datenbanken enthält.

Beispiel 2 *OID Anfrage für alle Datenbanken.*

```
SELECT oid, datname FROM pg_database;

  oid  |   datname
-------+-------------
 12403 | postgres
 17447 | db2
```

Da wir nun wissen, wo Datenbanken im Dateisystem abgelegt sind, wollen wir herausfinden, wo die Daten einer spezifischen Tabelle liegen. Grundsätzlich wird jede Tabelle in *PostgreSQL* in einer, oder mehreren Dateien abgelegt. Diese Dateien sind standardmäßig auf eine Größe von einem Gigabyte limitiert. Um den Pfad zu den Dateien einer Tabelle zu bekommen, kann folgender Befehl verwendet werden (`SELECT pg_relation_filepath('<Tabellenname>');`). Diese Anfrage gibt den Pfad zur Tabelle relativ zum Datenverzeichnis zurück.

Beispiel 3 *Relativer Pfad zur Datei der Tabelle 'names' in der Datenbank 'db2'.*

```
SELECT pg_relation_filepath('names');

 pg_relation_filepath
----------------------
   base/17447/27741
```

Werden aufgrund der Größe mehrere Dateien für eine Tabelle benötigt, werden bei den Namen der zusätzlichen Dateien die Suffixe .1, .2, etc. angehängt. Diese Dateien enthalten sämtliche Daten einer Tabelle und sind in 8 Kilobyte große **Pages** (Seiten) unterteilt.

Beispiel 4 *Gegeben sei eine Tabelle (12345) der Größe* 1.5 *GB, was* 196608 *Seiten entspricht. Da nur* 1 *GB pro Tabellendatei erlaubt ist, werden zwei Dateien benötigt. Somit sind die Tabelle wie folgt im Speicher strukturiert:*

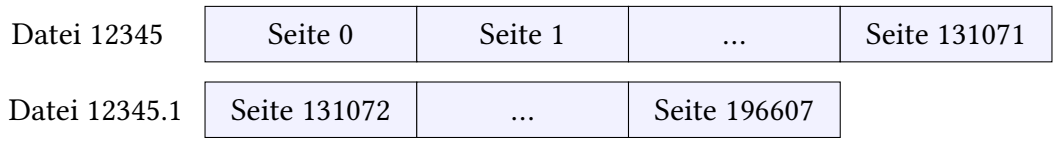

Abbildung 1: Dateiorganisation der Tabelle in Beispiel 4.

Um die Anzahl an Seiten einer Tabelle in *PostgreSQL* auszulesen, kann man die Größe der Tabelle durch die Seitengröße dividieren, siehe Beispiel 5.

Beispiel 5 *Anzahl der Seiten einer Tabelle in* PostgreSQL *auslesen.*

```
SELECT pg_relation_size('names') /
  current_setting('block_size')::int AS nr_of_pages;

 nr_of_pages
-------------
      160171
(1 row)
```

Jede einzelne Seite ist als sogenannte **Slotted Page** organisiert (siehe Abbildung 2): (1) Der **Header** (Kopf) beinhaltet Metadaten über die Seite, z.B., eine Checksum oder Start und Ende des freien Speichers (free space). (2) Der **Row Offset** erlaubt mittels Zeiger den Zugriff auf bestimmte Zeilen, sodass der n-te Zeiger auf die n-te Zeile verweißt. (3) Der **Free Space** kennzeichnet unverbrauchten Speicher innerhalb dieser Seite. (4) Die **Rows** (Zeilen) enthalten die Daten der jeweiligen Zeile.

Abbildung 2: Grafische Darstellung des Seitenformats.

Beispiel 6 *Gegeben sei eine Tabelle mit drei Zeilen. Die Slotted Page der daraus resultierende Seite ist wie folgt strukturiert:*

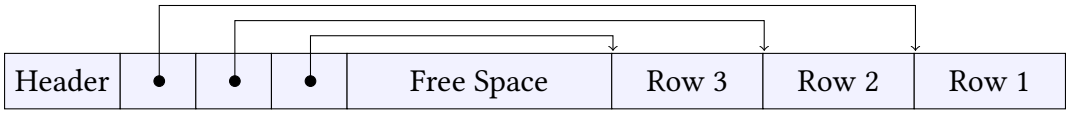

Abbildung 3: Seitenstruktur der Tabelle in Beispiel 6.

Jede Zeile der Tabelle ist in einer einzelnen Seite gespeichert (sofern sie nicht zu groß ist, dies wird in dieser Übung jedoch nicht behandelt). Auffällig ist, dass die erste Zeile am Ende der Seite gespeichert ist. Wenn eine neue Zeile gespeichert werden soll, wird der neue Zeiger am Ende des Row Offsets und die neuen Daten zu Beginn des Rows Bereich alloziert.

Das Format einer einzelnen Zeile ist einfach gehalten und ist in Abbildung 4 skizziert. Neben einem **Header** (Kopf), der Metadaten enthält, wird jedes **Field** (Attribut) nacheinander gespeichert. Ein Attribut kann entweder eine fixe oder variable Größe haben.

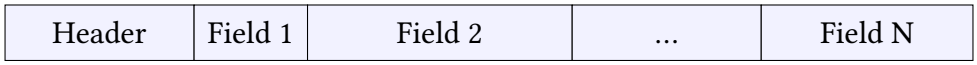

Abbildung 4: Grafische Darstellung der des Zeilenformats.

Im letzten Schritt betrachten wir, wie sich eine Seite verhält, wenn sich ihr Inhalt verändert (zum Beispiel, wenn eine Zeile gelöscht wird). Dazu legen wir eine Tabelle `boats`, exakt wie in Beispiel 3 von Übungsblatt 1, an. Verwenden Sie dazu folgende Befehle:

Beispiel 7 *Anlegen der Beispieltabelle* `boats`.

```
CREATE TABLE boats(bid INT, bname VARCHAR(20));
INSERT INTO boats VALUES(1, 'Alpha');
INSERT INTO boats VALUES(2, 'Pi');
INSERT INTO boats VALUES(3, 'Epsilon');
```

Da die Tabelle nur wenige Daten enthält, passen alle Zeilen auf eine Seite. Um Informationen über diese Seite zu bekommen, muss zuerst eine Erweitung von *PostgreSQL* mittels `CREATE EXTENSION pageinspect;` aktiviert werden. Anschließend kann der Befehl in Beispiel 8 verwendet werden um eine Seite zu analysieren. **lp** ist der Index einer Zeile, **lp_off** ist die Adresse der dazugehörigen Daten, **ctid** ist die ID der Zeile (bestehend aus Seitennummer und dem Index der Zeile) und den Daten selbst.

Beispiel 8 *Auslesen der Seiteninformationen der Tabelle* `boats` *(Seitennummer ist* 0*).*

```
SELECT lp, lp_off, t_ctid, t_data
  FROM heap_page_items(get_raw_page('boats', 0));

 lp | lp_off | t_ctid |            t_data
----+--------+--------+------------------------------
  1 |   8152 | (0,1)  | \x010000000d416c706861
  2 |   8120 | (0,2)  | \x02000000075069
  3 |   8080 | (0,3)  | \x03000000011457073696c6f6e
```

Was passiert nun, wenn wir eine Zeile aus dieser Seite löschen? Um dies herauszufinden, entfernen wir das Boot `Pi` aus der Tabelle. Dazu verwenden wir folgenden *SQL* Befehl: `DELETE FROM boats WHERE bname='Pi';`. Anschließend verwenden wir den Befehl aus Beispiel 8, um die Seite erneut zu analysieren. Trotz des Entfernens einer Zeile sieht die Seite unverändert aus. *PostgreSQL* markiert die Zeile intern als gelöscht, gibt die Zeile jedoch nicht gleich frei. Um den Speicher wieder freizugeben, kann der Befehl `VACUUM` verwendet werden. Für unsere Tabelle `boats` funktioniert dies wie folgt: `VACUUM boats;`. Verwendet man nun den Befehl aus Beispiel 8, sieht man eine veränderte Seite, die in Beispiel 9 dargestellt ist.

Beispiel 9 *Auslesen der Seiteninformationen der Tabelle* `boats` *nach dem Löschen einer Zeile und* `VACUUM`*.*

```
SELECT lp, lp_off, t_ctid, t_data
  FROM heap_page_items(get_raw_page('boats', 0));

 lp | lp_off | t_ctid |            t_data
----+--------+--------+------------------------------
  1 |   8152 | (0,1)  | \x010000000d416c706861
  2 |      0 |        |
  3 |   8112 | (0,3)  | \x03000000011457073696c6f6e
```

Der Pointer bleibt zur weiteren Verwendung erhalten, der Speicher wird jedoch freigegeben. Dies wird ersichtlich, wenn Sie mit `INSERT INTO boats VALUES (4, 'Xi');` eine neue Zeile in die Datenbank einfügen. Die resultierende Seite ist in Beispiel 10 zu sehen.

Beispiel 10 *Wiederverwendung der ID eines Zeigers einer gelöschten Zeile.*

```
SELECT lp, lp_off, t_ctid, t_data
  FROM heap_page_items(get_raw_page('boats', 0));

 lp | lp_off | t_ctid |            t_data
----+--------+--------+------------------------------
  1 |   8152 | (0,1)  | \x010000000d416c706861
  2 |   8080 | (0,2)  | \x04000000075869
  3 |   8112 | (0,3)  | \x0300000011457073696c6f6e
```

Zusammenfassend ist jede Zeile (je nach Größe gemeinsam mit weiteren Zeilen) auf einer 8 Kilobyte großen Seite gespeichert. Jede Seite ist wiederum Teil einer 1 Gigabyte großen Datei. Diese Datei (je nach Größe gemeinsam mit weiteren Dateien) beinhaltet die gesamten Daten einer Tabelle.

2 Zusätzliche Informationen

Das Kapitel zur Dateiorganisation in *PostgreSQL* basiert auf den Inhalt der folgenden Websites:

- Rachid Belaid:
  `http://rachbelaid.com/introduction-to-postgres-physical-storage/`
- malisper:
  `https://malisper.me/the-file-layout-of-postgres-tables/`
- $< 8k >$data:
  `https://www.8kdata.com/blog/postgresql-page-layout/`
- PostgreSQL Dokumentation:
  `https://www.postgresql.org/docs/current/storage.html`
- citusdata:
  `https://www.citusdata.com/blog/2016/03/30/five-ways-to-paginate/`