TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

Übung zur Vorlesung *Einsatz und Realisierung von Datenbanksystemen* im SoSe19

Maximilian {Bandle, Schüle} (i3erdb@in.tum.de)
http://db.in.tum.de/teaching/ss19/impldb/

Blatt Nr. 08

Hausaufgabe 1

Wie ändert sich die Bedeutung des Redo-Log und Undo-Log in Hauptspeicherdatenbanken im Vergleich zu klassischen Datenbanken? Wo werden sie gespeichert?

Da die Daten nicht mehr auf der Festplatte gespeichert werden, müssen sie (falls es keine Snapshots gibt) bei einem Neustart komplett auf Basis des Redo-Logs wiederhergestellt werden. Da nie die Daten einer nicht comitteten Transaktion auf der Platte landen wird die Undo-Log nur benötigt um im Fall eines Abort die Änderungen der Transaktion zurückzusetzen. Außerdem reicht es aus die Redo Log Daten erst beim Commit zu schreiben. Aus diesem Grund sind die Daten der Undo-Log nur zur Laufzeit der Transaktion notwendig und müssen nicht auf die Festplatte geschrieben werden. Während der Wiederherstellung der Datenbank können dann einfach alle Redo Log Einträge wiederhergestellt werden.

Hausaufgabe 2

HyPer schafft 120.000 Transaktionen pro Sekunde. Pro Transaktion werden 120 Byte in die Log geschrieben. Berechnen Sie den benötigten Durchsatz zum Schreiben der Log.

Die Datenbank läuft für einen Monat und stürzt dann ab. Es wurde kein Snapshot erstellt. Berechnen Sie die Recoveryzeit. Gehen Sie davon aus, dass die Recovery durch die Festplatte limitiert ist (100 MiB / s). Wieviel Log Einträge werden pro Sekunde recovert?

Durchsatz = 120.000 * 120 = 14400000 = 13,7 MiB/s.

LogEinträge =120.000*60*60*24*30=311040000000

LogGröße = LogEinträge*120 = 33,95 TiB

RecoveryZeit = 4,12 Tage

RecoveryDurchsatz = 873813 Tx / s.

Hausaufgabe 3

Gegeben eine Tabelle *Produkte* mit folgendem Schema und 10000 Einträgen:

Id (8 Byte) | Name (32 Byte) | Preis (8 Byte) | Anzahl (8 Byte)

Wieviele Daten werden für folgende Queries in die CPU-Caches geladen? Unterscheiden sie jeweils zwischen Row und Column Store.

1. *select * from Produkte*
2. *select Anzahl from Produkte*

Daten können maximal mit Granularität (64 Byte) in den Cache geladen werden. Das heißt, selbst wenn nur auf einen 64 Bit Integer Wert zugegriffen wird, muss ein kompletter 64-Byte Block geladen werden. Mit diesem Hintegrund ergeben sich folgende Ergebnisse:

1. *select * from Produkte*
   a) Row: $10000 * 56 = 560000$ Byte
   b) Column: $10000 * 8 + 10000 * 32 + 10000 * 8 + 10000 * 8 = 560000$ Byte
2. *select Anzahl from Produkte*
   a) Row: $10000 * 56 = 560000$ Byte
   b) Column: $10000 * 8 = 80000$ Byte

Hausaufgabe 4

Rekonstruieren Sie die ursprüngliche SQL-Anfrage aus dem folgenden (Pseudo-)Code eines codegenerierenden Datenbanksystems. Welche Art von Join wurde benutzt? Handelt es sich um Column- oder Row-Store?

```
struct Student { int matrnr; std::string name; int semester; };
struct Hoeren { int matrnr; int vorlnr; };
struct Result { int vorlnr; int a; };

std::vector<Result> compute(std::vector<Student>&ses,std::vector<Hoeren>&hs){
        std::unordered_multimap<int, Hoeren*> h_map;
        std::unordered_map<int, Student*> s_map;
        for (auto &h : hs)
                h_map.insert(std::make_pair(h.matrnr, &h));
        for (auto &s : ses)
                s_map.insert(std::make_pair(s.matrnr, &s));

   // Group by h.vorlnr; avg(s.semester) = sum(s.semester)/count(*)
        std::unordered_map<int, int> count_map;
        std::unordered_map<int, int> sum_map;
        for (auto &h : hs) {
                count_map.insert(std::make_pair(h.vorlnr, 0));
                sum_map.insert(std::make_pair(h.vorlnr, 0));
        }
        for (auto &h : h_map) {
                sum_map[h.second->vorlnr] += s_map[h.first]->semester;
                count_map[h.second->vorlnr]++;
        }
        std::vector<Result> res;
        for (auto &r : sum_map)
                res.push_back({ r.first, r.second / count_map[r.first] });
        return res;
}
```

Lösung: Anfrage gibt Durchschnittssemester pro Vorlesung aus, Hash-Join (Vebesserung wäre Singleton-Join, da `matrnr` unique), Row-Store.

```
select vorlnr, avg(s.semester) from studenten s, hoeren h
where s.matrnr=h.matrnr group by h.vorlnr
```

Hinweis Beantworten Sie die folgenden Anfragen mit Hilfe von SQL und den TPC-H Daten.

Hausaufgabe 5

1. Wie viele Kunden mit einer Bestellung, deren Kommentar (`o_comment`) das Wort „packages“ enthält, gibt es?

```
select count(distinct o_custkey)
from orders where o_comment like '%packages%';
```

2. Was ist die durchschnittliche Anzahl von Dezimalstellen in `l_orderkey`?

```
select avg(char_length(l_orderkey::text))
from lineitem;
```

3. Was sind die Namen aller Kunden und Zulieferer?

```
select c_name from customer
union all
select s_name from supplier
```

Hausaufgabe 6

1. Berechnen Sie für jede Bestellung den Rang (`rank`) nach `o_totalprice`, wobei der höchste Preis dem Rang 1 entspricht.

```
select rank() over (order by o_totalprice desc), o_orderkey from orders;
-- without window function
select (select count(*) + 1 from orders o2
        where o2.o_totalprice > o1.o_totalprice)
    rank, o_orderkey
from orders o1;
```

2. Berechnen Sie für jede Bestellung die wachsende Summe von `o_totalprice` nach Datum (`o_orderdate`). Die Summe soll jedes Jahr neu beginnen.

```
select distinct o_orderdate, sum(o_totalprice) over (
        partition by extract(year from o_orderdate) order by o_orderdate)
from orders;
--
select (
  select sum(o_totalprice)
  from orders o2
  where
    extract(year from o2.o_orderdate) = extract(year from o1.o_orderdate)
   and o2.o_orderdate < o1.o_orderdate)
from orders o1;
```

3. Was ist der Median von `o_totalprice`?

```
select percentile_disc(0.5)
within group (order by o_totalprice) from orders;

-- without window function
with less_then_me as (
 select o1.o_orderkey, count(*) from orders o1, orders o2
 where o2.o_totalprice < o1.o_totalprice group by o1.o_orderkey
), greater_then_me as (
 select o1.o_orderkey, count(*) from orders o1, orders o2
 where o2.o_totalprice > o1.o_totalprice group by o1.o_orderkey
)
select o_totalprice
from orders n, less_then_me l, greater_then_me g
where n.o_orderkey=l.o_orderkey and n.o_orderkey=g.o_orderkey
      and g.count = l.count;
```