Lehrstuhl für Praktische Informatik III
Prof. Dr. Guido Moerkotte
Email: moer@db.informatik.uni-mannheim.de

Datenbanksysteme 1 — 12. Übungsblatt
Herbst-/Wintersemester 2016

Aufgabe 1

Angenommen, beim Lebensmitteldiscounter Aldi werden ca. 10000 Artikel pro Werktag und Filiale verkauft. Die Verkäufe aller Filialen in den letzten drei Jahre seien in einem Data Warehouse gespeichert. Das Data Warehouse hat ein Sternschema, dessen Faktentabelle Tupel enthält, die den Verkauf eines Artikels beschreiben. Ein Datensatz der Faktentabelle verbrauche 32 Bytes auf dem Hintergrundspeicher.

Aufgabe 1 a)

Wieviel Platz belegt die Faktentabelle?

Lösung

Eine kurze Internetrecherche ergibt eine grobe Schätzung für die Untergrenze der Zahl der Filialen: 3500. Gehen wir ausserdem von 303 Werktagen aus.

Also ergibt sich für die Kardinalität der Faktentabelle:

$|Verkauf| = 3 \times 303 \times 10000 \times 3500 = 31.815 \times 10^9$

Bei 32 Bytes pro Datensatz ergibt das also $|Verkauf| \times 32 \times 1024^{-4} = 0,926$ Terabytes an Daten.

Aufgabe 1 b)

Wie lange dauert allein der Datentransfer für einen vollständigen Scan der Faktentabelle, wenn die Festplatte eine Übertragungsrate von 100MB/sec hat?

Lösung

Rund 162 Minuten.

Aufgabe 1 c)

Angenommen, es gibt ca. 100 Produktkategorien. Angenommen, die Zahl der Verkäufe werde pro Produktkategorie und Monat aggregiert. Wie groß sind die materialisierten Aggregate?

Lösung

Angenommen, eine Zahl belegt 4 Byte an Speicher. Dann ergibt sich folgender Speicherverbrauch für die beschriebenen Daten:

$100 \times 36 \times 4B = 14400B$

Das ergibt mit der oben beschriebenen Festplatte eine Scandauer von ca. 0,14 ms.

Angenommen, wir führen auch noch Aggregate für alle Produktei, für jedes Jahr und für alle drei Jahre ein. Dann belegen die Daten

$$(100 + 1) \times (36 + 4) \times 4 = 16160$$

Byte. Das entspricht einer Scan-Zeit von ca. 0.15 ms.

Aufgabe 2

Ein weiteres Beispiel für eine OLAP-Anwendung ist eine Krankenversicherung.

Aufgabe 2 a)

Geben Sie ein Schema für eine Faktentabelle und die Dimensionstabellen Zeit und Arzt an.

Lösung

Die Faktentabelle ist `behandlungen` mit den Attributen `BehandlungsID, Datum, Patient, Arzt, Faktor, Krankenhaus, Behandlungstyp`.

Die Dimensionstabelle `zeit` hat folgende Attribute: `Datum, Tag, Monat, Jahr, Quartal, KW`.

Die Dimensionstabelle für die Ärzte könnte so aussehen: `ID, Name, Vorname, Titel, Adresse, Gebiet, Zulassung`

Aufgabe 2 b)

Formulieren Sie eine Anfrage, mit der die Zahl der Krebsvorsorgeuntersuchungen im Jahr 2010 pro Monat dargestellt wird.

Lösung

```
   select z.monat,count(b.behandlungsid)
from behandlungen b, behandlungstypen t, zeit z
where b.behandlungstyp=t.id and
     b.datum=z.datum and
     z.jahr=2010 and
     t.typ='vorsorge' and
     t.krankheit='krebs'
group by z.monat
```

Aufgabe 2 c)

Geben Sie eine Anfrage an, mit der ein sinnvoller Datenwürfel für Ihr Schema materialisiert werden kann.

Lösung

```
insert into behandlungswuerfel(typ,krankheit,region,monat,jahr,anzahl)
  select t.typ, t.krankheit, k.region, z.monat, z.jahr, count(b.behandlungsid)
  from behandlungen b, behandlungstypen t, krankenhaus k, zeit z
  where b.behandlungstyp=t.id and
      b.krankenhaus=k.id and
      b.datum=z.datum
  group by cube (z.monat, z.jahr, k.region, t.krankheit, t.typ)
```

Aufgabe 2 d)

Formulieren Sie die Anfrage aus b) so um, dass die materialisierten Aggregate aus c) verwendet werden.

Lösung

Wir können entweder selber über die Regionen pro Monat aggregieren,

```
select monat,sum(anzahl)
from behandlungswuerfel
where
     jahr=2010 and
     typ='vorsorge' and
     krankheit='krebs' and
     monat is not null
group by monat
```

oder noch einfacher direkt die vorhandenen Aggregate aus dem Würfel selektieren:

```
select monat,anzahl
from behandlungswuerfel
where
     jahr=2010 and
     typ='vorsorge' and
     krankheit='krebs' and
     region is null
```

Aufgabe 2 e)

Sei ausserdem ein Index auf dem Datum für die Faktentabelle vorhanden. Am 28. Januar 2010 startete die Versicherung eine Aufklärungskampagne zur Krebsvorsorge.

Wie kann möglichst effizient die Zahl der Krebsvorsorgeuntersuchungen vom 28.1. 2009 bis 31.12. 2009 mit der Zahl für den gleichen Zeitraum des Folgejahres verglichen werden? Formulieren Sie entsprechende SQL-Anfragen.

Lösung

Um eine möglichst schnelle Berechnung zu erzielen, nehmen wir alle Daten von Februar bis Dezember aus dem materialisierten Datenwürfel, und fügen die fehlenden Tupel aus der Faktentabelle hinzu. Dieser Split der Auswertung in zwei Teile lohnt sich, weil wir über den Datumsindex effizient auf alle Tupel in der Faktentabelle zugreifen können, die zwischen 28.1. und 31.1. liegen.

```
with behandvonwuerfel(jahr,anzahl) as
(
     select jahr,sum(anzahl) from behandlungswuerfel
     where jahr>=2009 and jahr<=2010 and monat>1 and
         typ='vorsorge' and krankheit='krebs' and region is null
     group by jahr
),
behandausfakt(jahr,anzahl) as
(
     select jahr,count(behandlungsid)
     from behandlungen b, zeit z, behandlungstypen t
     where b.datum=z.datum and b.behandlungstyp=t.id and
        z.jahr>=2009 and z.jahr<=2010 and
        z.monat=1 and z.tag>=28 and
        t.typ='vorsorge' and t.krankheit='krebs'
     group by jahr
)
select jahr,sum(anzahl)
from (select * from behandausfakt union all select * from behandvonwuerfel)
as t
group by jahr
```

Aufgabe 3

Aufgabe 3 a)

Wofuer steht OLAP und wofuer OLTP?

Lösung

OLTP (On-line Transaction Processing) is characterized by a large number of short on-line transactions (INSERT, UPDATE, DELETE). The main emphasis for OLTP systems is put on very fast query processing, maintaining data integrity in multi-access environments and an effectiveness measured by number of transactions per second. In OLTP database there is detailed and current data, and schema used to store transactional databases is the entity model (usually 3NF).

OLAP (On-line Analytical Processing) is characterized by relatively low volume of transactions. Queries are often very complex and involve aggregations. For OLAP systems a response time is an effectiveness measure. OLAP applications are widely used by Data Mining techniques. In OLAP database there is aggregated, historical

data, stored in multi-dimensional schemas (usually star schema).

Aufgabe 3 b)

Was sind die drei wesentlichen Konzepte von Window Funktionen?

Lösung

Partitionierung: Partitioniert die Argumenttupel gemaess eines oder mehrer Ausdruecke (meist Attributwerte) in unabhaenigige Gruppen.

Sortierung: Erlaubt durch `order-by` in jeder Partition eine Reihenfolge herzustellen.

Framing: Restriktion der Tupel einer Partition, auf die sich die Window-Funktion bezieht.

Aufgabe 3 c)

Beschreiben was folgende SQL Anfrage vermutlich ausdrueckt und markieren sie welche Teile der Anfrage welche Konzepte von Window-Funktionen implementieren:

```
    select Ort, Zeit, Wert, abs(Wert - (avg(Wert) over w))
from Messungen
window w as (
partition by Ort
order by Zeit
range between 5 preceeding and 5 following
```

Lösung

Fuer jeden Wert der an einem bestimmten Ort zu einer bestimmten Zeit gemessen wurde, wird die absolute Abweichung vom Mittelwert der 5 naechst vorher bis zu den 5 naechst folgenden Tupeln gebildet.

Partitionierung: `partition by Ort`
Sortierung: `order by Zeit`
Framing: `range between 5 preceeding and 5 following`

Beachte, man muss ein `window` nicht explizit in eine variable (`window w as`)auslagern. Die Definition des `window` kann auch direkt auf `over` folgen.

Aufgabe 3 d)

Du willst heute Abend feiern gehen. Zur Auswahl stehen verschiedene Clubs. Entscheidende Faktoren fuer die Wahl des Clubs sind der Eintrittspreis und der Preis fuer einen Tequila Shot. Die Clubs sind in einer Relation `Club` mit folgendem Schema gespeichert:

`Name`	`Eintrittspreis`	`Shotpreis`

Finde alle Clubs die von keinem anderem Club dominiert werden, d.h. alle Clubs zu denen es keinen Club gibt der sowohl einen guenstigeren Eintrittspreis als auch einen guenstigeren Shotpreis hat.
(In anderen Kontexten spraeche man auch davon alle pareto-optimalen Clubs zu finden)

Lösung

```
   select C.Name
from Club C
skyline of C.Eintrittspreis min, C.Shotpreis min
```

Aufgabe 4

Aus der Vorlesung ist Ihnen das objekt-relationale Uni-Schema bekannt. Dabei wurden Prüfungen in einer geschachtelten Tabelle innerhalb der Studententabelle gespeichert.

Revidieren Sie diesen Ansatz und definieren Sie in Oracle-Syntax eine alleinstehende Tabelle, die alle Prüfungsdaten speichert (objekt-relational). Befüllen Sie die Tabelle beispielhaft mit der Prüfung vom Student Carnap, welcher am 15. Juli 1998 mit Professor Russel stattfand.

Lösung

```
CREATE OR REPLACE TYPE ProfessorenTyp AS OBJECT (
  PersNr NUMBER,
  Name VARCHAR(20),
  Rang CHAR(2),
  Raum Number
)
/


CREATE TYPE VorlesungenTyp
/

CREATE TYPE VorlesungsListenTyp AS TABLE OF REF VorlesungenTyp
/

CREATE OR REPLACE TYPE VorlesungenTyp AS OBJECT (
  VorlNr NUMBER,
  TITEL VARCHAR(20),
  SWS NUMBER,
  gelesenVon REF ProfessorenTyp,
  Voraussetzungen VorlesungsListenTyp
)
/

CREATE OR REPLACE TYPE StudentenTyp AS OBJECT (
  MatrNr NUMBER,
  Name VARCHAR(20),
  Semester NUMBER,
```

```
  hoert VorlesungsListenTyp
)
/

CREATE OR REPLACE TYPE PruefungenTyp AS OBJECT (
  Pruefling REF StudentenTyp,
  Inhalt REF VorlesungenTyp,
  Pruefer REF ProfessorenTyp,
  Note DECIMAL(3,2),
  Datum Date
)
/


CREATE TABLE ProfessorenTab OF ProfessorenTyp (PersNr PRIMARY KEY);

CREATE TABLE StudentenTab OF StudentenTyp (MatrNr PRIMARY KEY)
    NESTED TABLE hoert STORE AS BelegungsTab;

CREATE TABLE PuefungenTab OF PruefungenTyp;

CREATE TABLE VorlesungenTab OF VorlesungenTyp
    NESTED TABLE Voraussetzungen STORE AS VorgaengerTab;


INSERT INTO ProfessorenTab VALUES (2126, 'Russel', 'C4', 232);
INSERT INTO ProfessorenTab VALUES (2127, 'Kopernikus', 'C3', 310);

INSERT INTO VorlesungenTab
  SELECT 5001, 'Grundzuege', 4, REF(p), VorlesungsListenTyp()
  FROM ProfessorenTab p

INSERT INTO StudentenTab VALUES(28106, 'Carnap', 3, VorlesungsListenTyp());
  WHERE Name = 'Kopernikus';

INSERT INTO TABLE
  (SELECT s.hoert
    from StudentenTab s
    where s.MatrNR = 28106)
    select  REF(v)
    from VorlesungenTab v
    where v.VorlNr = 5001;

INSERT INTO PruefungenTab (Pruefling, Inhalt, Pruefer, Note, Datum)
VALUES ( (select ref(s)
          from StudentenTab s
          where Name = 'Carnap'),
          ( select  ref(v)
          from VorlesungenTab v
```

```
        where Titel = 'Grundzuege'),
        ( select  ref(p)
        from ProfessorenTab p
        where Name = 'Russel'), 1.0, DATE '1998-7-15');


select  p.pruefling.name, p.inhalt. titel ,  p.pruefer.name, p.note, p.datum
from pruefungentab p;
```

Aufgabe 5

Erstellen Sie einen komplexen Attribut-Typ der Informationen über Gebrauchtwagen wie Kilometerstand und Verbrauch modelliert. Leiten Sie von dem Obertyp zwei Untertypen ab, einen für den US-Markt und einen weiteren für den DE-Markt. Versehen Sie Ihren Attribut-Typ mit einer Funktion, die den Verbrauch als Anzahl der verbrauchten Liter je 100 km ausgibt. Gehen Sie davon aus, dass für den US-Markt der Verbrauch als Anzahl von Meilen je verbrauchter Gallone gespeichert wird.

Definieren Sie für die Attribute *Verbrauch* und *Kilometerstand* jeweils einen distinct type. Nutzen Sie bei Ihrer Implementierung die DB2 Syntax.

Lösung

```
create distinct type mileage_type as float with comparisons;
create distinct type gas_mileage_type as float with comparisons;

create type car_info as (
        country varchar(20),
        str_mileage  varchar(5),
        str_gas_mileage  varchar(5))
        not final
        not instantiable
        mode DB2SQL
                method get_mileage()
                returns mileage_type
                language SQL
                deterministic
                reads sql data
                no external action,
                method get_gas_mileage()
                returns gas_mileage_type
                language SQL
                deterministic
                reads sql data
                no external action;

create type car_info_de under car_info as (
        num_mileage mileage_type,
```

```
        num_gas_mileage gas_mileage_type,
        last_tuev_au date)
        final
        instantiable
        mode DB2SQL
                overriding method get_mileage() returns mileage_type,
                overriding method get_gas_mileage() returns gas_mileage_type;


create method get_mileage() for car_info_de
        return self .. num_mileage;

create method get_gas_mileage() for car_info_de
        return self .. num_gas_mileage;

create type car_info_us under car_info as (
        num_mileage mileage_type,
        num_gas_mileage gas_mileage_type)
        final
        instantiable
        mode DB2SQL
                overriding method get_mileage() returns mileage_type,
                overriding method get_gas_mileage() returns gas_mileage_type;

create function convert_mileage_us(m mileage_type) returns mileage_type
return (cast(m as float) * 1.6093);

create function convert_gas_mileage_us(gm gas_mileage_type) returns gas_mileage_type
return (cast(gm as float) / cast(2.352 as float));

create method get_mileage() for car_info_us
        return convert_mileage_us( self .. num_mileage);

create method get_gas_mileage() for car_info_us
        return convert_gas_mileage_us( self .. num_gas_mileage);
```