Inoffizielle Lösung der Datenbankklausur vom 18.02.2014

Joshua Gleitze, Roman Langrehr

3. August 2016

Aufgabe 1

	Richtig	Falsch
Das Cursor-Konzept ist sowohl Bestandteil von JDBC als auch von Embedded SQL.	x	
Mit JDBC werden Fehler im SQL-Statement stets erst zur Laufzeit erkannt.	x	
JDBC standardisiert unter anderem, wie man durch das Anfrageergebnis navigiert.		
JDBC hat die Eigenschaft, dass auf dem Client kein DBMS-spezifischer Code laufen muss.		x
JDBC bietet die Möglichkeit, auf Schema-Informationen zu einem Anfrageergebnis zuzugreifen.	x	
Embedded SQL bietet die Möglichkeit, schemaspezifische Fehler im SQL-Statament zum Übersetzungszeitpunkt zu erkennen.	x	
Embedded SQL bietet die Möglichkeit, zum Übersetzungszeitpunkt zu erkennen, dass der Typ einer Variablen in der Host-Sprache und der eines Attributs der Relation nicht übereinstimmen.	x	
Gespeicherte Prozeduren bieten die Möglichkeit zu erkennen, dass der Typ einer Variablen in einer Prozedur und der eines Attributs der Relation nicht übereinstimmen.	x?	
Ein Vorübersetzer für Embedded SQL übersetzt eingebettete Statements nach SQL.		x
Ein Vorübersetzer für Embedded SQL übersetzt die eingebetteten Statements in Stored-Procedure-Aufrufe.		x

Wenn man mit einer JPA (Java Persistence API) arbeitet, bleibt dem Anwendungsentwickler das Formulieren deklarativer Anfragen (in SQL bzw. SQL-artigen Sprachen) erspart.		
Gespeicherte Prozeduren können die üblichen aus imperativen Programmiersprachen bekannten Kontrollstrukturen enthalten.	x	
Gespeicherte Prozeduren können SQL-Anfragen enthalten.	x	
Herkömmliche eindimensionale Indexstrukturen können die Auswertung mehrdimensionaler Bereichsanfragen beschleunigen.		x
Der kd-Baum ist nicht balanciert.	x	
Die Ausschnitte des Datenraums, die Geschwisterknoten im kd-Baum enstsprechen, sind immer überlappungsfrei.	x	
Das (in der Vorlesung vorgestellte) optimale Verfahren zum Finden des nächsten Nachbarn mit Hilfe räumlicher Indexstrukturen ist ein Beispiel für Breitensuche.		x
Das Ergebnis des DBSCAN-Algorithmus ist unabhängig von der Reihenfolge, in der die Objekte betrachtet werden.		x
Die Komplexität von DBSCAN ist logarithmisch in der Anzahl der Datenobjekte.		x
Die Zahl der Cluster, die der DBSCAN-Algorithmus zurückliefert, ist ein Parameter dieses Algorithmus.		x
In hochdimensionalen Merkmalsräumen wächst der erwartete Abstand zweier Datenobjekte (jedes für sich zufällig gewählt, Datenobjekte gleichverteilt) mit der Anzahl der Datenobjekte.		x
In hochdimensionalen Merkmalsräumen wächst die Anzahl der Blätter einer räumlichen Indexstruktur mit der Anzahl der Dimensionen.		

Aufgabe 3

a)

```
DELETE FROM Pizzazutat
WHERE p_id = 182;

DELETE FROM Pizzazutat
```

```
WHERE p_id = 1
AND z_id =1;

INSERT INTO Pizzazutat
VALUES(1, 1);

ALTER TABLE Pizzazutat
ADD FOREIGN KEY (p_id) REFERENCES Pizza(id),
ADD FOREIGN KEY (z_id) REFERENCES Zutaten(id),
ADD UNIQUE (p_id, z_id);
```

b)

```
SELECT name FROM (
        SELECT z_id, COUNT(*) AS cp
        FROM Pizzazutat
        WHERE z_id <> 1
        AND z_id <> 2
        GROUP BY z_id
        HAVING NOT EXISTS (
                SELECT COUNT(*)
                FROM Pizzazutat
                WHERE z_id <> 1
                AND z_id <> 2
                GROUP BY z_id
                HAVING COUNT(*) > cp
        )
) ids JOIN Zutat
ON Zutat.id = ids.z_id
```

c)

```
CREATE VIEW Pizzapreis AS
SELECT name, (SUM(kosten) + 2) * 1.19 AS preis
FROM (
        SELECT kosten, p_id
        FROM Pizzazutat
        JOIN Zutat
        ON Pizzazutat.z_id = Zutat.id
) zt
JOIN Pizza
ON Pizza.id = zt.p_id
GROUP BY Pizza.id;
```

d)

```
SELECT x.name, y.name, z.name
```

```
FROM Zutat x JOIN Zutat y JOIN Zutat z
WHERE x.kosten + y.kosten + z.kosten < 6
AND NOT EXISTS (
        SELECT *
        FROM Pizzazutat px JOIN Pizzazutat py JOIN Pizzazutat pz
        WHERE px.z_id = x.id
        AND py.z_id = y.id
        AND pz.z_id = z.id
        AND px.p_id = py.p_id
        AND px.p_id = pz.p_id
)
AND x.id < y.id
AND y.id < z.id
```

e)

```
CREATE OR REPLACE PROCEDURE bestellePizza(x VARCHAR2, n NUMBER) IS verfueg
        BEGIN
        --berechne, wie viele Pizzen des Typs x hergestellt werden können
        SELECT MIN(vorrat) INTO verfuegbar
        FROM Pizza JOIN (
                SELECT p_id, vorrat
                FROM Pizzazutat JOIN Zutat
                ON Pizzazutat.z_id = Zutat.id
        ) zz
        ON Pizza.id = zz.p_id
        WHERE Pizza.name = x;
        IF n <= verfuegbar THEN
                --passe vorrätige Mengen an
                UPDATE Zutat
                SET vorrat = vorrat - n
                WHERE EXISTS (
                        SELECT *
                        FROM Pizza JOIN Pizzazutat
                        ON Pizza.id = Pizzazutat.p_id
                        WHERE Pizza.name = x
                        AND Pizzazutat.z_id = Zutat.id
                );
        ELSE
                DBMS_OUTPUT.PUT_LINE('Mi␣dispiace!');
        END IF;
END;
```

f)

```
ALTER TABLE Pizzazutat
ADD anzahl NUMBER NOT NULL DEFAULT 1;
```