Lösungen zu den Übungsaufgaben aus Kapitel 7

Edwin Schicker

Lösung zu Aufgabe 1

Der kostenbasierte Optimizer benötigt zusätzlich umfangreiche Statistiken zu den einzelnen Relationen. Diese Statistiken sollten immer aktuell sein. Dadurch kann der Optimizer aber die einzelnen Befehle wesentlich besser optimieren.

Lösung zu Aufgabe 2

Die Restriktion erfolgt vor dem Verbund. Der Befehl lautet nun:

```
SELECT A.Bezeichnung, AP.Anzahl, Datum
FROM Auftrag NATURAL INNER JOIN Auftragsposten AP
    INNER JOIN (SELECT Nr FROM KUNDE WHERE Name='Fahrrad Shop') ON Nr=Kundnr
    INNER JOIN Artikel A ON Anr=Artnr
```

Lösung zu Aufgabe 3

Ein eindeutiger Index ist für Namen in der Regel zu einschränkend.

```
CREATE INDEX IBezeichng ON Artikel (Bezeichnung) ;
```

Lösung zu Aufgabe 4

```
CREATE INDEX Suche ON Personal (Einsatzort, Name) ;
```

Lösung zu Aufgabe 5

Vorteil 1: Zugriff gezielt nur auf kleine Relationen (z.B. Monat Juni 2013)

Vorteil 2: Parallele Zugriffe bei großen Suchvorgängen.

Hash-Partitionierung: Vorteil 1 entfällt, aber Vorteil 2 bleibt. Insbesondere liefert die Hashpartitionierung eine sehr gleichmäßige Aufteilung.

Lösung zu Aufgabe 6

Die Referenzpartitionierung ermöglicht es, eine Relation nach einem Kriterium zu partitionieren, das in dieser Relation als Attribut nicht existiert. Beispiel: Partitionierung der Relation *Auftragsposten* nach dem Datum.

In SQL Server und MySQL müsste in Relation *Auftragsposten* das Attribut *Datum* hinzugefügt werden. Dann wäre eine Partitionierung möglich. Die Relation wäre aber nicht mehr in der dritten Normalform.

Lösung zu Aufgabe 7

Einsatz in Datenbanken, wo nur selten Änderungen stattfinden, die Performance aber sehr wichtig ist. Dies trifft vor allem auf Data Warehouses zu. Hier wird nicht geändert, die Abfragen sind gleichzeitig extrem komplex.

Lösung zu Aufgabe 8

Mit der Restriktion wird die Relation *Kunde* von 6 auf einen Eintrag reduziert. Alle folgenden Joins liefern entsprechend kleinere Zwischenrelationen. Bei großen Relationen wirkt sich dies enorm aus.

Lösung zu Aufgabe 9

Sind die betroffenen Relationen bereits sortiert, so müssen diese nur noch zusammen gemischt werden. Dieses Mischen ist relativ performant, ein Merge-Join ist daher dann üblich. Liegen beide zu verknüpfenden Attribute als Indexe vor, so gilt diese Aussage ebenfalls. SQL Server legt daher auch für Fremdschlüssel meist Indexe automatisch an.

Lösung zu Aufgabe 10

Eine Gruppierung ist eine Zusammenfassung und setzt in der Regel eine vorherige Sortierung voraus. Diese Sortierungen sind sehr aufwendig, ein Index kann hier etwas Unterstützung leisten. Eine Group-By-Klausel macht aber erst Sinn, wenn auch Aggregatfunktionen verwendet werden. In diesen Fällen muss aber dann doch auf die Daten zugegriffen werden. Letztlich muss die gesamte Relation in den Arbeitsspeicher geladen werden.

Lösung zu Aufgabe 11

```
CREATE PROCEDURE Preisnachlass( nachlass Numeric ) AS
BEGIN
```

```
  IF nachlass <= 10 THEN
    UPDATE Artikel
    SET Preis = Preis*(100-nachlass)/100, Netto = Netto*(100-nachlass)/100,
          Steuer = Steuer*(100-nachlass)/100;
    UPDATE Auftragsposten
    SET Gesamtpreis = Gesamtpreis*(100-nachlass)/100;
  END IF;
EXCEPTION WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE ('Fehler in der Prozedur Preisnachlass');
END;
```

Lösung zu Aufgabe 12

```
$sql =    "Select Nr, Name FROM Lieferant
            Where Nr IN (SELECT Liefnr FROM Lieferung WHERE Anr = ?" ;
$stmt = $conn->prepare($sql);
foreach ($artnr in $liste)                  // $liste enthalte die gesuchten Artikel
{  echo "<p>Artikelnummer: $artnr</p><p>Lieferanten:</p>";
  $stmt->bindParam(1, $artnr);
  $stmt->execute( ) ;
  while ($row = $stmt->fetch( ) )
    echo "<p>Lieferantnr: $row[NR]; Name: $row[NAME]</p>";
}
```