Database and Artificial Intelligence Group

Datenbanksysteme (VU 4.0, 184.686)
Übungsteil, WS 2011/2012

Beispiel 3

Die folgenden Aufgabenstellungen basieren auf der Datenbank, welche Sie für Beispiel 2 des Übungsteils zu erstellen hatten. Sie können entweder auf Ihrer Datenbank weiterarbeiten, oder die Musterlösung des 2. Beispiels verwenden, welche auf der LVA-Homepage zum Download bereitsteht (allerdings erst nach den Abgabegesprächen zu Beispiel 2). Stellen Sie allerdings in jedem Fall in Ihrem Abgabe-Verzeichnis auf bordo.dbai.tuwien.ac.at die Create-, Insert- und Drop-Befehle jener Datenbank bereit, auf der Sie arbeiten.

Stellen Sie sicher, dass die Daten, mit denen Sie testen, der Spezifikation entsprechen (d. h. Kardinalitäten berücksichtigen usw.). Sie müssen keine Fälle berücksichtigen, die der Spezifikation widersprechen.

Lösen Sie die folgenden Probleme mittels SQL:

1. Geben Sie die Namen ALLER Anwendungen aus, die jeweilige Gesamtanzahl der Downloads von allen Versionen der Anwendung zusammen genommen, sowie dem Umsatz durch die Käufe der jeweiligen Anwendung. Sortieren Sie die Ausgabe absteigend nach der Anzahl der Downloads.
2. Geben Sie die Namen jener Anwendungen aus, für die die Anzahl der Autoren mit einer auf ".com" endenden E-Mail-Adresse maximal ist. Geben Sie zugleich diese Anzahl aus.
3. Geben Sie die Liste aller Anwendungsnamen aus, zusammen mit dem Namen irgendeines Autors der jeweiligen Anwendung. (Welcher gewählt wird, ist egal.) Jede Anwendung soll also genau einmal in der Liste aufscheinen.
4. Wählen Sie per Hand eine Kategorie aus, die mindestens einer anderen Kategorie untergeordnet ist. Schreiben Sie eine Anfrage, die diese Kategorie ausgibt, sowie rekursiv alle übergeordneten Kategorien. Geben Sie für jede Kategorie den Namen und die Beschreibung aus. Wenn eine Kategorie keine Beschreibung besitzt, weil sie auf der obersten Ebene der Hierarchie liegt, geben Sie "keine Beschreibung" aus. (Beachten Sie, dass dazu möglicherweise Typumwandlungen notwendig sind.) Passen Sie die Tupel in Ihrer Datenbank so an, dass es zu der von Ihnen ausgewählten Kategorie mindestens zwei Ebenen übergeordneter Kategorien gibt. Achten Sie darauf, dass Ihre Daten keinen Zyklus enthalten und dass Ihre Anfrage allgemein formuliert ist (d. h. für beliebige Datenbankausprägungen funktioniert).
5. Schreiben Sie Befehle zum Erzeugen und Löschen einer View ("benutzer_statistik_view"), welche für JEDEN Benutzer ("name")
   - die Anzahl der Anwendungen ausgibt, von denen er irgendwelche Versionen heruntergeladen hat, (Hinweis: Das Schlüsselwort für Duplikatelimination kann auch in Aggregatfunktionen verwendet werden.)
   - die Anzahl der Anwendungen ausgibt, die er gekauft hat,
   - sowie das Verhältnis von gekauften zu heruntergeladenen Anwendungen in Prozent (0 wenn nichts heruntergeladen wurde) als ganze Zahl ausgibt.

   Sortieren Sie das Ergebnis nach der Anzahl der heruntergeladenen Anwendungen.

Lösen Sie die folgenden Probleme mittels PL/pgSQL:

6. Schreiben Sie einen Trigger, der vor dem Eintragen in die Tabelle "Download" überprüft, ob der Download überhaupt stattfinden darf. Ein Download darf genau dann stattfinden, wenn die Anwendung gratis ist, oder wenn der Benutzer die Anwendung gekauft hat (und zwar vor dem Download). Das Eintragen soll außerdem fehlschlagen, wenn die Version der Plattform, für die die

Anwendung heruntergeladen wird, gar nicht von der jeweiligen Version der Anwendung unterstützt wird. Im Fall eines Fehlschlags wegen eines nicht vorher erfolgten Kaufs soll die Exception "nicht berechtigt" geworfen werden; falls die Plattform nicht unterstützt wird, "nicht unterstuetzt".

7. Schreiben Sie eine Prozedur **f_erhoehter_preis** mit einem Anwendungsnamen als Parameter, die den Preis der Anwendung
   - um 30% erhöht zurückgibt, wenn die durchschnittliche Bewertung 5 beträgt,
   - um 20% erhöht zurückgibt, wenn die durchschnittliche Bewertung mindestens 4 und unter 5 ist,
   - um 15% erhöht zurückgibt, wenn die durchschnittliche Bewertung mindestens 3 und unter 4 ist,
   - um 10% erhöht zurückgibt, wenn die durchschnittliche Bewertung mindestens 2 und unter 3 ist,
   - um 5% erhöht zurückgibt, wenn die durchschnittliche Bewertung mindestens 1 und unter 2 ist, und
   - ansonsten unverändert zurückgibt.

   Wenn die Anwendung nicht bewertet wurde, soll ihr unveränderter Preis zurückgegeben werden.
8. Schreiben Sie eine Prozedur **f_anwendungen_verteuern**, die für alle Anwendungen mit mehr als einem Autor den Preis auf das jeweilige Ergebnis der Funktion "f_erhoehter_preis" setzt und Anwendungsnamen, alten und neuen Preis als Hinweise ausgibt, wenn der neue Preis sich vom alten Preis unterscheidet. Andernfalls soll keine Änderung und keine Ausgabe für die jeweilige Anwendung erfolgen.
9. Schreiben Sie eine Funktion **f_benutzer_loeschen**, die alle Benutzer löscht, die nichts gekauft und seit mindestens einem Jahr nichts heruntergeladen haben. Es soll die Anzahl der gelöschten Benutzer zurückgegeben werden und für jeden gelöschten Benutzer ein Hinweis mit dem Benutzernamen ausgegeben werden.
10. Überlegen Sie sich eine sinnvolle Testabdeckung für die PL/pgSQL-Programmteile laut Punkt 6 - 9, z.B.: Erweiterung der Testdaten vom 2. Übungsbeispiel, Aufruf der zu testenden PL/pgSQL-Programmteile mit entsprechenden Ausgaben, so dass sich die erfolgreiche Durchführung der Tests überprüfen lässt. Stellen Sie in Ihrem Abgabe-Verzeichnis die SQL-Dateien mit den zusätzlichen INSERT-Befehlen und den "Testtreibern" in der Datei `test.sql` bereit. Sie müssen in der Lage sein, diese SQL-Dateien und PL/pgSQL-Dateien im Rahmen des Abgabegesprächs ablaufen zu lassen.

Lösen Sie folgende Probleme mittels Java und JDBC:

Sie können die folgende Vorlage verwenden: AppStoreVerwaltung.java

11. Schreiben Sie eine Java Klasse **AppStoreVerwaltung** mit folgenden Methoden:
    1. Eine Methode **"dbConnect()"**, die eine JDBC-Verbindung zur Datenbank herstellt und AUTOCOMMIT ausschaltet. Anm.: Sie benötigen kein Passwort um sich auf Ihre Datenbank auf der Bordo zu verbinden. Beim lokalen Testen auf der Bordo verwenden Sie localhost als host-Parameter.
    2. Eine Methode **"druckeAnwendungen()"**, die ALLE Anwendungen (Name, Preis) sortiert nach dem Anwendungsnamen ausgibt, zusammen mit dem erhöhten Preis und der Anzahl der Downloads aller Versionen der jeweiligen Anwendung zusammengenommen. Verwenden Sie zum Ermitteln des erhöhten Preises die Funktion **"f_erhoehter_preis"** aus Punkt 7. Die Ausgabe soll als Comma-Separated List erfolgen, d. h.: eine Bildschirm-Zeile pro Zeile der Tabelle; die Spaltenwerte werden einfach nacheinander ausgegeben und mittels "," getrennt.
    3. Eine Methode **"erhoehePreise()"**, die exakt die Funktionalität der PL/pgSQL-Prozedur von Punkt 8 bereitstellt (aber nichts ausgibt). Beachten Sie bei der Erstellung dieser Methode folgende Punkte:
       - Verwenden Sie in **"erhoehePreise()"** eine SELECT-Abfrage, um die zu verteuernden Anwendungen auszuwählen, über die iteriert werden soll. Verwenden Sie Prepared Statements.
       - **"erhoehePreise()"** soll die Funktion **"f_erhoehter_preis"** aus Punkt 7 verwenden. Ansonsten dürfen keine weiteren Benutzer-definierten Funktionen oder Prozeduren verwendet werden. Insbesonders darf nicht einfach die Prozedur aus Punkt 8 aufgerufen werden.
       - Stellen Sie durch entsprechende Transaktionsverwaltungskommandos sicher, dass die Methode **"erhoehePreise()"** entweder die vorgesehene Änderung erfolgreich durchführt (und festschreibt) oder die Datenbank unverändert lässt.
       - Vermeiden Sie in der Java-Methode die Verwendung von "*" in SELECT-Anweisungen.

Listen Sie statt dessen in jeder SELECT Anweisung explizit die Spalten auf, die Sie auslesen (oder verändern) möchten, z. B.: In der Uni-DB würden Sie "SELECT MatrNr, Name, Semester FROM Studenten" statt "SELECT * FROM Studenten" schreiben. Und wenn das Semester in der Programmlogik keine Rolle spielt, würden Sie "SELECT MatrNr, Name FROM Studenten" schreiben.

4. Eine Methode **"entferneDownloads()"**, die alle Einträge in der Tabelle "Downloads" entfernt, die mindestens ein Jahr alt sind. Die Methode soll auf der Konsole ausgeben, wieviele Datensätze gelöscht wurden.
5. Eine Methode **"dbDisconnect()"**, die die bestehende JDBC-Verbindung wieder schließt.
6. Überlegen Sie sich eine sinnvolle Testabdeckung der Java-Klasse "AppStoreVerwaltung". Legen Sie dazu eine weitere Methode **"testAppStoreVerwaltung()"** an, die die anderen Methoden aufruft und entsprechende Ausgaben erzeugt, so dass sich die erfolgreiche Durchführung der Tests überprüfen lässt. Sie müssen in der Lage sein, diese Tests im Rahmen des Abgabegesprächs ablaufen zu lassen.

Für die Abgabe beim Tutor bereitzustellen

- Bringen Sie bitte Ihren **Studentenausweis** zur Abgabe mit. Eine Abgabe ohne Ausweis ist nicht möglich.
- Stellen Sie in Ihrem Abgabeverzeichnis auf bordo eine **Listing-Datei** mit dem Namen listing.txt bereit, die Sie beim Testen der SQL- und PL/pgSQL-Dateien (wie in Beispiel 2 des Übungsteils) erzeugt haben.
- In Summe sind also folgende **5 Dateien** zu erstellen:
  - sql-teil.sql (Lösung zu Punkt 1 - 5)
  - plpgsql-teil.sql (Lösung zu Punkt 6 - 9)
  - test.sql (Testabdeckung zu Punkt 6 - 9)
  - AppStoreVerwaltung.java (Lösung zu Punkt 11)
  - listing.txt (mittels \o Befehl erzeugt)
- Achten Sie weiters darauf, dass sich in Ihrem Abgabe-Verzeichnis auch die Create- Insert- und Drop-Befehle jener Datenbank befinden, welche Sie für die Lösung der Beispiele verwendet haben. Vergessen Sie nicht, die DROP-Datei dahingehend anzupassen, dass sie nun zusätzlich auch Befehle enthält, mit der sich alle in Punkt 6 - 9 erzeugten Datenbankobjekte löschen lassen.
- Die Beispiele müssen bis zum Abgabetermin am 4.12.2011 um Mitternacht auf unserem Server (bordo.dbai.tuwien.ac.at) im Unterverzeichnis `beispiel3` verfügbar sein (die Dateien werden automatisch abgesammelt und den Tutoren zur Verfügung gestellt). Es sind keine Ausdrucke erforderlich.
- Wir erwarten von Ihnen eigenständige Lösungen. Plagiate werden **nicht** akzeptiert und mit 0 Punkten bewertet.
- Stellen Sie sicher, dass alle geforderten Programmteile (d.h.: sql-teil.sql für Punkt 1 - 5, plpgsql-teil.sql und test.sql für Punkt 6 - 9 sowie AppStoreVerwaltung.java für Punkt 11) unter PostgreSQL 8.4 auf dem Server bordo.dbai.tuwien.ac.at fehlerlos laufen. Sie müssen in der Lage sein, diese Dateien im Rahmen des Abgabegesprächs ablaufen zu lassen.

Anmerkungen

plpgsql does not exist

Sollten Sie auf unserem Server folgenden Fehler bekommen, melden Sie sich unter dbs@dbai.tuwien.ac.at:

```
psql:a_create.sql:21: ERROR: language "plpgsql" does not exist HINT: Use CREATE LANGUAGE to load the language into the database.
```

Falls Sie auf Ihrer eigenen DB arbeiten müssen Sie die plpgsql Sprache "erlauben". Dies geschieht mit `createlang plpgsql` auf der Kommandozeile (oder mit CREATE LANGUAGE, siehe Postgres Dokumentation).

Classpath

Der Classpath ist standardmäßig auf den JDBC Treiber und das aktuelle Verzeichnis gesetzt. D. h.

```
CLASSPATH=/usr/share/pgsql/postgresql-<VERSION>.jar:.
```

Sollten Sie eine Fehlermeldung wie `Exception in thread "main" java.lang.NoClassDefFoundError:` bekommen, müssen Sie den Classpath an Ihre Verzeichnisstruktur anpassen.

Bewertung

Für das Übungsbeispiel 3 werden **maximal 15 Punkte** vergeben. Im Rahmen des Abgabesprächs wird nicht nur die Korrektheit der Lösungen sondern auch (und vor allem) das **Verständnis** überprüft. Für die einzelnen Aufgaben erhalten Sie die maximal möglichen Punkte nur dann, wenn die Lösung richtig ist **und** wenn Sie in der Lage sind, diese Lösung zu erklären.

Die Verteilung der Punkte erfolgt nach folgendem Schlüssel:

- Datei sql-teil.sql: max. 5 Punkte
- Datei plpgsql-teil.sql und test.sql: max. 5 Punkte
- Datei AppStoreVerwaltung.java: max. 5 Punkte
- Datei listing.txt: Vorhandensein für das Erreichen der vollen Punktezahl notwendig