

AG Datenbanken und Informationssysteme

Wintersemester 2006 / 2007

Prof. Dr.-Ing. Dr. h. c. Theo Härder
Fachbereich Informatik
Technische Universität Kaiserslautern



<http://www.dvs.informatik.uni-kl.de>

6. Übungsblatt

Für die Übung am Donnerstag, **07. Dezember 2006**,
von 15:30 bis 17:00 Uhr in 13/222.

Machen Sie sich mit JDBC (Java Database Connectivity) vertraut, um die Programmieraufgaben mit Java und JDBC zu bearbeiten. Ergänzend zur Vorlesung finden Sie einen Verweis auf die Online-Dokumentationen von JDBC, wie etwa *JDBC-Technology Guide: Getting Started*, unter den Online-Ressourcen auf der Website zur Vorlesung <http://www.dvs.informatik.uni-kl.de/courses/DBAW>.

Aufgabe 1: Eingebettetes SQL am Beispiel „Lieferung“

Gegeben seien folgende Relationen:

LIEFERANT (LNR, LNAME, STATUS, ORT)

PROJEKT (PNR, PNAME, ORT)

LIEFERUNG (LNR, TNR, PNR, MENGE)

TEIL (TNR, TNAME, FARBE, GEWICHT)

Skizzieren Sie ein Programm mit eingebetteten SQL-Anweisungen, das alle Lieferanten in der Reihenfolge ihrer Lieferantenummer ausgibt. Zu jedem Lieferanten soll direkt eine nach den Projektnummern sortierte Liste aller Projekte ausgegeben werden, die von diesem Lieferanten beliefert werden.

Lösung:

```
#include <stdio>
EXEC SQL INCLUDE sqlca;

main(...) {
    EXEC SQL BEGIN DECLARE SECTION;
        int          cs_lnr;
        char          cs_lname[50];
        char          cs_status[20];
        char          cs_ort[50];
        int          cj_pnr;
        char          cj_pname[50];
        char          cj_ort;

    EXEC SQL END DECLARE SECTION;
    EXEC SQL DECLARE CS CURSOR FOR
        SELECT LNR, NAME, STATUS, ORT
        FROM LIEFERANT
        ORDER BY LNR;
    EXEC SQL DECLARE CJ CURSOR FOR
        SELECT DISTINCT P.PNR, P.NAME, P.ORT
        FROM LIEFERUNG L, PROJEKT P
        WHERE L.PNR = P.PNR AND L.LNR = :cs_lnr
        ORDER BY P.PNR;
    EXEC SQL CONNECT TO testdb;
    EXEC SQL OPEN CS;

    do {
        EXEC SQL FETCH CS INTO :cs_lnr, :cs_lname, :cs_status, :cs_ort;
        if (sqlca.sqlcode != 100) {
            // Ausgabe Lieferant
            ...
            EXEC SQL OPEN CJ;
            do {
                EXEC SQL FETCH CJ INTO :cj_pnr, cj_pname, cj_ort;
                if (sqlca.sqlcode != 100) {
                    // Ausgabe Projekt
                    ...
                }
                else {
                    EXEC SQL CLOSE CJ;
                    break;
                }
            } while (true);
        } else {
            EXEC SQL CLOSE CS;
            break;
        }
    } while (true);
    EXEC SQL DISCONNECT;
}
```

Aufgabe 2: Eingebettetes SQL mit Rekursion, Beispiel „Mitarbeiter-Hierarchie“

In der folgenden DB wird eine Mitarbeiterhierarchie in der üblichen Weise dargestellt:

```
MA (MANR, MANAME, MANAGER, GEHALT)
```

Skizzieren Sie eine Prozedur, die eine Managernummer als Eingabe-Parameter erhält und das Durchschnittsgehalt aller in der Hierarchie (evtl. über mehrere Stufen) untergeordneten Mitarbeiter berechnet.

Lösung:

Die nun zunächst folgende Lösung ist **falsch!** (Warum?)

```
PROCEDURE AVG_GEHALT ( IN_MANR : INTEGER) : REAL;

VAR count_ma : INTEGER; (* globale Var. zum Zaehlen aller Mitarbeiter *)
    geh_summe : REAL;

PROCEDURE GEHALT_SUMME ( IN_MANR : INTEGER ) : REAL;
(*berechnet die Gehaltssumme aller in_manr untergebenen Mitarbeiter*)
VAR manr : INTEGER;
    geh, sum_geh : REAL;

EXEC SQL DECLARE C0 CURSOR FOR
    SELECT MANR, GEHALT FROM MA
    WHERE MANAGER = :IN_MANR;

BEGIN
    EXEC SQL OPEN C0;
    sum_geh := 0.0;
    EXEC SQL FETCH C0 INTO :manr, :geh;
    WHILE SQLSTATUS # NOTFOUND DO
        sum_geh := sum_geh + geh;
        INC(count_ma);
        sum_geh := sum_geh + GEHALT_SUMME(manr);
        EXEC SQL FETCH C0 INTO manr, geh;
    END (*WHILE*);
    EXEC SQL CLOSE C0;
    RETURN sum_geh;
END GEHALT_SUMME;

BEGIN
    count_ma := 0;
    geh_summe := GEHALT_SUMME(IN_MANR);
    IF count_ma > 0 THEN
        RETURN ( geh_summe / count_ma)
    ELSE
        RETURN 0.0;
    END;
END AVG_GEHALT;
```

KORREKT ist folgende Lösung:

```

PROCEDURE AVG_GEHALT ( IN_MANR : INTEGER) : REAL;

VAR count_ma : INTEGER; (* Zaehlen aller Mitarbeiter *)
    geh_summe : REAL; (* Summieren der Einzelgehälter *)
    PNR_Stack : INT_Stack; (* Stack-Var. vom abstrakten Datentyp
                            INT_Stack (INTEGER-Werte in Form eines
                            Stacks organisiert und mit den üblichen
                            Funktionen verwaltet) zum Speichern der
                            noch abzuarbeitenden Mitarbeiter *)

    akt_manr : INTEGER;

PROCEDURE GEHALT_TEILSUMME ( IN_MANR : INTEGER );
(* berechnet die Gehaltssumme aller IN_MANR direkt untergebenen
Mitarbeiter und speichert deren PNR im PNR_Stack *)
VAR manr : INTEGER;
    geh : REAL;

EXEC SQL DECLARE C0 CURSOR FOR
    SELECT MANR, GEHALT FROM MA
    WHERE MANAGER = :IN_MANR;

BEGIN
    EXEC SQL OPEN C0;
    EXEC SQL FETCH C0 INTO :manr, :geh;
    WHILE SQLSTATUS # NOTFOUND DO
        PUSH(PNR_Stack, manr);
        geh_summe := geh_summe + geh;
        INC(count_ma);
        EXEC SQL FETCH C0 INTO :manr, :geh;
    END (*WHILE*);
    EXEC SQL CLOSE C0;
    RETURN;
END GEHALT_TEILSUMME;

BEGIN
count_ma := 0;
geh_sum := 0;
INIT (PNR_Stack);
PUSH (PNR_Stack, IN_MANR);
WHILE NOT EMPTY (PNR_Stack) DO
    akt_manr := POP (PNR_Stack);
    GEHALT_TEILSUMME (akt_manr);
END;
IF count_ma > 0 THEN
    RETURN ( geh_summe / count_ma)
ELSE
    RETURN 0.0;
END;
END AVG_GEHALT;

```

Aufgabe 3: JDBC

Gegeben sei eine DB, die folgende Relationen enthält:

LIEFERANT (LNR, LNAME, STATUS, ORT)

PROJEKT (PNR, PNAME, ORT)

LIEFERUNG (LNR, TNR, PNR, MENGE)

TEIL (TNR, TNAME, FARBE, GEWICHT)

Schreiben Sie mit Hilfe von JDBC ein Java-Programm, das alle Lieferanten in der Reihenfolge ihrer Lieferantenummer ausgibt. Zu jedem Lieferanten sollen unmittelbar alle die von diesem Lieferanten belieferten Projekte in der Reihenfolge ihrer Projektnummer aufgelistet werden. Dabei soll weiterhin zu jedem Projekt eine nach den Teilenummern sortierte Liste aller Teile ausgegeben werden, die für dieses Projekt vom jeweiligen Lieferanten geliefert werden. Ihr Programm soll zwei Eingabeparameter erhalten: Benutzername und Passwort, die für die DB-Verbindung benötigt werden.

Lösung:

```
import java.sql.*;

public class ShowSuppliers {

    public static void main (String [] argv) {

        String url = "jdbc:db2:demoDB"; // URL für die JDBC-Ver-
        bindung
        String user, passwd;
        Connection conn;
        Statement stmt;
        PreparedStatement pstmt1, pstmt2;
        ResultSet rs1, rs2, rs3;
        String lnr, pnr;

        // Pruefe Eingabeparameter fuer die DB-Verbindung
        if (argv.length != 2) {
            System.out.println ("Falsche Parameterangaben!");
            System.exit (-1);
        }

        user = argv[0];
        passwd = argv[1];

        try {
            // Lade JDBC-Treiber und erzeuge Verbindungsobjekt
            Class.forName ("COM.ibm.db2.jdbc.app.DB2Driver");
            conn = DriverManager.getConnection (url, user, passwd);

            // Erzeuge PreparedStatement-Objekte
            pstmt1 = conn.prepareStatement
                ("SELECT DISTINCT P.pnr, P.pname, P.ort " +
                 "FROM lieferung L, projekt P " +
                 "WHERE L.pnr = P.pnr AND L.lnr = ? " +
                 "ORDER BY P.pnr");
```

```

pstmt2 = conn.prepareStatement
("SELECT T.tnr, T.tname, T.farbe, T.gewicht, L.menge "+
"FROM lieferung L, teil T " +
"WHERE L.lnr = ? AND L.pnr = ? AND L.tnr = T.tnr " +
"ORDER BY T.tnr");

// Liste Lieferanten
System.out.println ("Lieferantenliste mit Projekten:");
stmt = conn.createStatement ();
rs1 = stmt.executeQuery ("SELECT * FROM lieferant ORDER BY
lnr");

while (rs1.next ()) {
    lnr = rs1.getString ("LNR");
    System.out.println (lnr + ", " + rs1.getString ("LNA-
ME") + ", " + rs1.getInt ("STATUS") + ", " + rs1.getString ("ORT"));

    pstmt1.setString (1, lnr);
    rs2 = pstmt1.executeQuery ();

    // Liste Projekte
    while (rs2.next ()) {
        pnr = rs2.getString ("PNR");
        System.out.println (" " + pnr + ", " + rs2.get-
String ("PNAME") + ", " + rs2.getString ("ORT"));

        pstmt2.setString (1, lnr);
        pstmt2.setString (2, pnr);
        rs3 = pstmt2.executeQuery ();

        // Liste Teile
        while (rs3.next ()) {
            System.out.println (" " + rs3.getString
("TNR") + ", " + rs3.getString ("TNAME") + ", " + rs3.getString ("FAR-
BE") + ", " + rs3.getFloat ("GEWICHT") + ", " + rs3.getInt ("MENGE"));
        } // while rs3
    } // while rs2
} // while rs1

// SchlieÙe Ressourcen
stmt.close ();
pstmt1.close ();
pstmt2.close ();
conn.close ();
} // try
catch (ClassNotFoundException e) {
    System.out.println ("ClassNotFoundException beim Laden von
JDBC-Driver: " + e.getMessage ());
}
catch (SQLException e) {
    System.out.println ("SQLException: " + e.getMessage ());
}
} // main
} // class

```

Aufgabe 4: JDBC und Metadaten

Gegeben sei eine nicht leere relationale Datenbank. Schreiben Sie mit Hilfe von JDBC ein Java-Programm, das für einen als Eingabeparameter eingegebenen Basisrelationennamen zunächst überprüft, ob die Relation in der Datenbank existiert. Ist die Relation vorhanden, so soll die zugehörige vollständige SQL-Anweisung für die Erzeugung der entsprechenden Relation generiert und ausgegeben werden. Die Ausgabe muss den Relationennamen und -attribute mit dem jeweiligen Datentyp und eventueller Längenspezifikation enthalten. Betrachten Sie dabei nur Attribute vom Typ VARCHAR, INTEGER und DECIMAL. Neben einem Relationennamen soll das Programm zwei weitere Eingabeparameter berücksichtigen: Benutzername und Passwort, die für die Verbindung zur Datenbank benötigt werden.

Hinweis:

Verwenden Sie die `getMetaData`-Methode des `Connection`-Objekts für den Zugriff auf das `DatabaseMetaData`-Objekt, mit dessen Methoden Sie dann die benötigten Metadaten abfragen.

Lösung:

```
import java.sql.*;

public class GenSQLStmt {

    public static void main (String [] argv) {

        String url = "..."; // URL für die JDBC-Verbindung
        String user, passwd;
        Connection conn;
        ResultSet rs;
        DatabaseMetaData dbmd;
        String tableName, catalogName, schemaName;
        int columnType;
        String sqlString;
        int i, columnSize;
        String [] tableTypes = {"TABLE"};

        // Pruefe Eingabeparameter fuer die DB-Verbindung
        if (argv.length != 3) {
            System.out.println ("Falsche Parameterangaben!");
            System.exit (-1);
        }

        tableName = argv[0];
        user = argv[1];
        passwd = argv[2];
        catalogName = "..."; // Üblicherweise ist dies der DB-Name
        schemaName = user; // Üblicherweise ist dies der Benutzer-Name

        try {
            // Lade JDBC-Treiber und erzeuge Verbindungsobjekt
            Class.forName ("..."); // JDBC-Treiber
            conn = DriverManager.getConnection (url, user, passwd);

            // Hole DB-Metadaten
            dbmd = conn.getMetaData ();

            // Pruefe, ob die Basisrelation in der DB existiert.
            rs = dbmd.getTables (catalogName, schemaName, tableName,
                                tableTypes);
            if (rs.next ()) {
```

```

System.out.println ("Relation \"" + tableName + "\"
                    existiert.");
sqlString = "CREATE TABLE " + tableName + " (\n";
rs.close();

// Hole alle Attributdaten
rs = dbmd.getColumns (catalogName, schemaName,
                    tableName, "%");

// Iteriere ueber die Attributobjekte in rs
i = 1;
while (rs.next () ) {

    // Hole Attributnamen
    if (i > 1) { sqlString = sqlString + ",\n"; }
    sqlString += " " + rs.getString("COLUMN_NAME") +

" ";

    // Hole Datentyp und -laenge, falls noetig
    columnType = rs.getInt ("DATA_TYPE");
    columnSize = rs.getInt ("COLUMN_SIZE");

    if (columnType == java.sql.Types.VARCHAR) {
        sqlString += "VARCHAR(" + columnSize + ")";
    }
    else if (columnType == java.sql.Types.INTEGER) {
        sqlString += "INTEGER";
    }
    else if (columnType == java.sql.Types.DECIMAL) {
        sqlString += "DECIMAL(" + columnSize + "," +
            rs.getInt ("DECIMAL_DIGITS") + ")";
    }
    i++;
} // while
rs.close();

sqlString = sqlString + "\n );";
System.out.println (sqlString);
} /if
else {
    System.out.println ("Relation \"" + tableName + "\"
                        existiert nicht.");
}

// SchlieÙe Ressourcen
stmt.close();
conn.close ();
}
catch (ClassNotFoundException e) {
    System.out.println ("ClassNotFoundException beim Laden von
JDBC-Driver: " + e.getMessage ());
}
catch (SQLException e) {
    System.out.println ("SQLException: " + e.getMessage());
}
}
}
}

```


Aufgabe 5: Abbildung von Sicht Operationen auf Tabellen (Basisrelationen)

In dieser Aufgabe betrachten wir die Abbildung von SELECT-Anweisungen auf Views. Dazu haben wir in der Vorlesung die schrittweise Ersetzung von View-Definitionen auf Basisrelationen betrachtet.

Geben seien die folgenden Relationen:

```
STUDENT(MatrNr, FbNr, Name, Vorname, PruefOrd)
MITARBEITER(PersNr, FbNr, Name, Vorname, Gehalt)
PROFESSOR(PersNr, Arbeitsgruppe)
FACHBEREICH(FbNr, Name, Dekan)
```

mit:

```
STUDENT.FbNr ist Fremdschlüssel auf FACHBEREICH.FbNr,
MITARBEITER.FbNr ist Fremdschlüssel auf FACHBEREICH.FbNr,
PROFESSOR.PersNr ist Fremdschlüssel auf MITARBEITER.PersNr,
FACHBEREICH.Dekan ist Fremdschlüssel auf PROFESSOR.PersNr,
MITARBEITER-PROFESSOR-Hierarchie ist abgebildet durch vertikale Partitionierung
```

und die folgenden Sichtdefinitionen:

```
CREATE VIEW StudFbPO96 (MatrNr, Name, Vorname, FbNr, FbName, PO) AS
  SELECT s.MatrNr, s.Name, s.Vorname, f.FbNr, f.Name, s.PruefOrd
  FROM STUDENT s, FACHBEREICH f
  WHERE s.FbNr = f.FbNr
  AND PruefOrd = 'PO96';
```

```
CREATE VIEW GehaltFB (FbNr, FBName, GehSum) AS
  SELECT f.FbNr, f.Name, sum(m.Gehalt)
  FROM MITARBEITER m, FACHBEREICH f
  WHERE m.FbNr = f.FbNr
  GROUP BY f.FbNr, f.Name;
```

```
CREATE VIEW GehaltProfFB (FbNr, FBName, GehSum) AS
  SELECT f.FbNr, f.Name, sum(m.Gehalt)
  FROM MITARBEITER m, FACHBEREICH f, PROFESSOR p
  WHERE m.FbNr = f.FbNr AND m.PersNr = p.PersNr
  GROUP BY f.FbNr, f.Name;
```

Formulieren Sie für die folgenden Aufgaben jeweils SQL-Anfragen (unter bestmöglichem Einsatz der vordefinierten Views) und geben sie für jede dieser Anfragen an, wie die schrittweise Abbildung auf Basisrelationen durchgeführt wird. Welche Anfragen lassen sich nicht so leicht abbilden? Welchen Zusatzmechanismus muss ein Datenbanksystem bereitstellen, damit alle Anfragen beantwortet werden können. Wie sehen ihre Antworten auf die zuvor gestellten Fragen aus, wenn alle Sichten materialisiert abgespeichert werden.

- Finde für alle Studenten die nach der Prüfungsordnung von 1996 studieren die Professoren, die sie unterrichten.
- Welcher Dekan verdient am meisten?
- In welchem Fachbereich verdienen die Professoren zusammen am meisten Geld?
- In welchem Fachbereich verdienen die Mitarbeiter ohne Berücksichtigung der Professorengelälter am meisten Geld?
- Wieviel verdienen die Professoren des Fachbereichs 'Informatik' im Mittel.

Lösung:

- a) Finde für alle Studenten die nach der Prüfungsordnung von 1996 studieren die Professoren, die sie unterrichten.

```
SELECT DISTINCT p.Name
FROM StudFbPO96 s, MITARBEITER m, FACHBEREICH f, PROFESSOR p
WHERE s.FbNr = m.FbNr AND m.FbNr = f.FbNr AND m.PersNr = p.PersNr;
```

wird ersetzt durch:

```
SELECT DISTINCT p.Name
FROM STUDENT s, FACHBEREICH fNeu, MITARBEITER m, FACHBEREICH f,
PROFESSOR p
WHERE s.FbNr = m.FbNr AND m.FbNr = f.FbNr AND m.PersNr = p.PersNr
AND s.FbNr = fNeu.FbNr AND PruefOrd = 'PO96';
```

- b) Welcher Dekan verdient am meisten?

Es lässt sich keine vordefinierte View sinnvoll einsetzen.

```
SELECT p.PersNr
FROM MITARBEITER m, PROFESSOR p, FACHBEREICH f
WHERE m.PersNr = p.PersNr AND p.PersNr = f.Dekan
HAVING gehalt = max(gehalt)
```

- c) In welchem Fachbereich verdienen die Professoren zusammen am meisten Geld?

```
SELECT gFB.FbNr
FROM GehaltProfFB gFB
HAVING GehSum = max(GehSum);
```

```
SELECT f.FbNr,
FROM MITARBEITER m, FACHBEREICH f, PROFESSOR p
WHERE m.FbNr = f.FbNr AND m.PersNr = p.PersNr
GROUP BY f.FbNr, f.Name;
HAVING GehSum = max(sum(m.Gehalt));
```

Diese Art der Umformung ist semantisch nicht korrekt, weil sich die Aggregatfunktion (max) auch auf die GROUP-BY Definition bezieht.

Eine korrekte Umformung ist:

```
SELECT gFB.FbNr
FROM (SELECT f.FbNr, f.Name, sum(m.Gehalt)
FROM MITARBEITER m, FACHBEREICH f
WHERE m.FbNr = f.FbNr
GROUP BY f.FbNr, f.Name) gFB
HAVING GehSum = max(GehSum);
```

- d) In welchem Fachbereich verdienen die Mitarbeiter ohne Berücksichtigung der Professorengelälter am meisten Geld?

```
SELECT gFB.FbNr
FROM GehaltFB gFB, GehaltProfFB gpFB
WHERE gFB.FbNr = gpFB.FbNr
HAVING (gFB.GehSum - gpFB.GehSum) = max(gFB.GehSum - gpFB.GehSum)
```

```

SELECT gFB.FbNr
FROM MITARBEITER gFBm, FACHBEREICH gFBf
     MITARBEITER gpFBm, FACHBEREICH gpFBf, PROFESSOR gpFBp
WHERE gFBm.FbNr = gFBf.FbNr
     AND gpFBm.FbNr = gpFBf.FbNr AND gpFBm.PersNr = gpFBp.PersNr
     AND gFBf.FbNr = gpFBf.FbNr
GROUP BY gFBf.FbNr
HAVING (sum(gFBm.Gehalt) - sum(gpFBm.Gehalt)) =
       max(sum(gFBm.Gehalt) - sum(gpFBm.Gehalt))

```

Eine solche Umformung ist auch hier, aus dem gleichen Grund wie bei c), nicht möglich.

Eine korrekte Umformung ist:

```

SELECT gFB.FbNr
FROM (SELECT f.FbNr, f.Name, sum(m.Gehalt)
      FROM MITARBEITER m, FACHBEREICH f
      WHERE m.FbNr = f.FbNr
      GROUP BY f.FbNr, f.Name) gFB,
     (SELECT f.FbNr, f.Name, sum(m.Gehalt)
      FROM MITARBEITER m, FACHBEREICH f, PROFESSOR p
      WHERE m.FbNr = f.FbNr AND m.PersNr = p.PersNr
      GROUP BY f.FbNr, f.Name) gpFB
WHERE gFB.FbNr = gpFB.FbNr
HAVING (gFB.GehSum - gpFB.GehSum) = max(gFB.GehSum - gpFB.GehSum)

```

e) Wieviel verdienen die Professoren des Fachbereichs 'Informatik' im Mittel.

```

SELECT avg(gpFB.GehSum)
FROM GehaltProfFB gpFB, FACHBEREICH f
WHERE gpFB.FbNr = f.FbNr AND f.Name = 'Informatik'

SELECT avg(sum(gpFBm.Gehalt))
FROM MITARBEITER gpFBm, FACHBEREICH gpFBf, PROFESSOR gpFBp,
     FACHBEREICH f
WHERE gpFBm.FbNr = gpFBf.FbNr AND gpFBm.PersNr = gpFBp.PersNr
     AND gpFBf.FbNr = f.FbNr AND f.Name = 'Informatik'
GROUP BY f.FbNr;

```

ist semantisch nicht korrekt.

Korrekt wäre hingegen:

```

SELECT avg(gpFB.GehSum)
FROM (SELECT f.FbNr, f.Name, sum(m.Gehalt)
      FROM MITARBEITER m, FACHBEREICH f, PROFESSOR p
      WHERE m.FbNr = f.FbNr AND m.PersNr = p.PersNr
      GROUP BY f.FbNr, f.Name) gpFB,
     FACHBEREICH f
WHERE gpFB.FbNr = f.FbNr AND f.Name = 'Informatik'

```

Für c), d), e) lassen sich keine Äquivalenten Umformungen auf Basisrelationen ableiten. Die Datenbank muss intern temporäre Tabellen unterstützen um die Anfragen semantisch korrekt durchführen zu können. Wären die Views materialisiert, müssten keine Werte zwischengespeichert werden.