Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Datenbanksysteme I
Übung: JDBC

Jana Bauckmann

Wo reicht SQL alleine nicht?

- Web-Anwendungen
  - Daten übersichtlich und schön präsentieren
- Komplizierte Fragestellungen
  - Sind sich 2 Tupel ähnlich? ➔ Duplikaterkennung
- String-Operationen
  - Mache aus einem Attribut **`Name`** die Attribute **`Vorname`**, **`Mittelname`**, **`Nachname`** und parse die Werte entsprechend.
- Bedingte Anweisungen
  - Wenn ein Arbeitnehmer eine Belobigung bekommen hat, gib ihm 2 % mehr Geld, sonst nur 1,5 %.

SQL mit einer Programmiersprache verbinden

- Embedded SQL
  - Kombiniert SQL mit Programmiersprachen
    - ADA, C, C++, Cobol, Fortran, M, Pascal, PL/I, …
  - Einbettung von SQL durch Preprocessing
- Stored procedures / PSM
  - Speicherung von Prozeduren als DBMS Objekte
  - Aufruf aus SQL Ausdrücken
- Call-level-interface (CLI)
  - Verbindet C mit DBMS
  - Spezielle Funktionsbibliothek
  - Spart das Preprocessing
- Java Database Connectivity (JDBC)
  - Wie CLI aber für Java
  - Relevant für die Übung

SQL vs. Programmiersprachen

Bisher

- Generische SQL Schnittstelle
- Kommandozeile
- Von allen DBMS angeboten
- Selten genutzt
  - Datenbankadmins

Jetzt

- SQL Ausdrücke in größeren Softwarekomponenten
  - Anwendungen
  - DB Tools
- Verwendet aus einer anderen Programmiersprache heraus

Impedance Mismatch

Die Datenmodelle von DBMS und Programmiersprachen unterscheiden sich sehr.

- Programmiersprachen
  - Integer, String, Real, …
  - Pointer
  - Arrays
  - Insbesondere: Keine Mengen
- Relationales DBMS
  - Relationen und Attribute
  - Keine Pointer
  - Keine Schleifen
  - Keine Verzweigungen
  - Keine komplexen Strukturen

Datentransfer zwischen beiden Modellen ist also schwierig.

Impedance Mismatch

- Alternative 1: Nur Programmiersprache verwenden
  - Aber: SQL sehr nützliche und einfache (very high-level) Sprache
  - Aber: Programmierer sollen nichts über Speicherstruktur wissen
    - Physische Datenunabhängigkeit!
  - Aber: DBMS sehr effizient
- Alternative 2: Nur SQL verwenden
  - Aber: In Basis-SQL nicht alles ausdrückbar
    - Z.B. n!
  - Aber: Ausgabe sind immer nur Relationen
    - Und z.B. nicht Graphiken
- Alternative 3: Einbettung von SQL in eine Programmiersprache
  - Programmiersprache: „*Host language*“ („Wirtssprache“)
  - SQL: Embedded SQL (eingebettetes SQL)

Überblick

- Embedded SQL
- Stored procedures / PSM
- Call-level-interface (CLI) und Java Database Connectivity (JDBC)

Embedded SQL - Ablauf

Programmiersprache +
Embedded SQL

Präprozessor

Programmiersprache +
Funktionsaufrufe

Compiler

Ausführbares Programm

DBMS-spezifisch,
d.h. muss für jedes DBMS
neu kompiliert werden

Embedded SQL

- Host-Variablen deklarieren
  - ```
    EXEC SQL BEGIN DECLARE SECTION;
            char studioName[50], studioAdr[256];
            char SQLSTATE[6];
    EXEC SQL END DECLARE SECTION;
    ```
- „einfache“ Statements
  - ```
    EXEC SQL INSERT INTO Studio(Name, Adresse)
                    VALUES (:studioName, :studioAdr);
    ```
  - Klappt für jeden SQL Ausdruck, der kein Ergebnis produziert.
    - Wegen *Impedance Mismatch*
    - `INSERT, DELETE, UPDATE, CREATE, DROP, …`

Hostvariablen mit Doppelpunkt

Embedded SQL
Anfrage mit Anfrageergebnis

- Nur ein Tupel: Direkte Bindung an gemeinsame Variablen
  - EXEC SQL SELECT Name, Adresse
    INTO **:studioName, :studioAdr**
    FROM **Studio** WHERE id = 310;

- Mehrere Tupel: Cursor

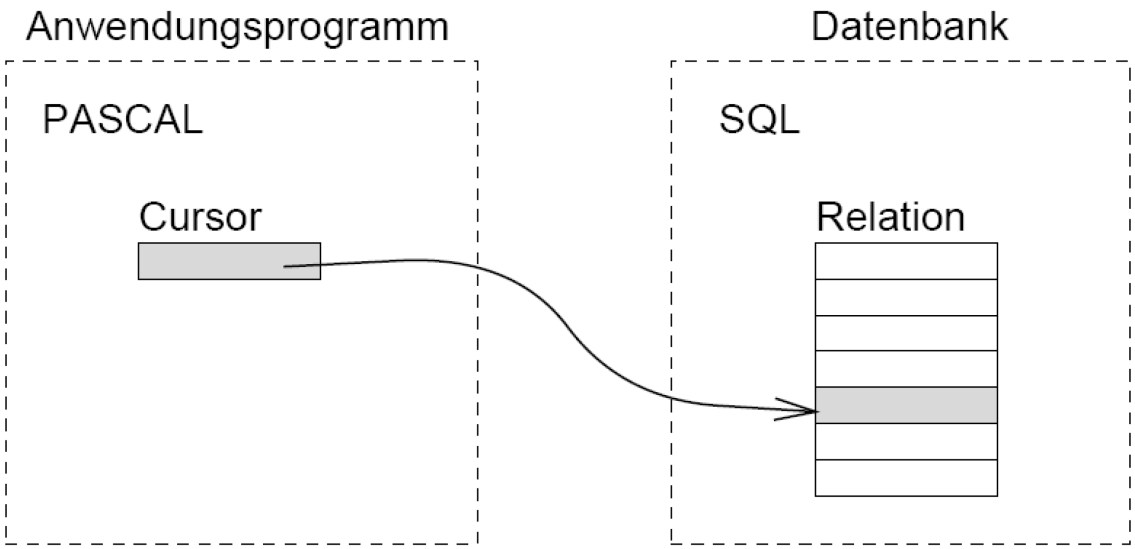


Abbildung: Kai-Uwe Sattler (TU Ilmenau)

Embedded SQL
Cursor – Beispiel


```
void gehaltsBereiche() {
    int i, stellen, counts[15];
    EXEC SQL BEGIN DECLARE SECTION;
        int gehalt; char SQLSTATE[6];
    EXEC SQL END DECLARE SECTION;

    EXEC SQL DECLARE managerCursor CURSOR FOR
        SELECT Gehalt FROM Manager;
    EXEC SQL OPEN managerCursor;

    for(i=0; i < 15; i++) counts[i] = 0;
    while(1) {
        EXEC SQL FETCH FROM managerCursor INTO :gehalt;
        if(strcmp(SQLSTATE,"02000")) break;
        stellen = 1;
        while((gehalt /= 10) > 0) stellen++;
        if(stellen <= 14) counts[stellen]++;
        }
    EXEC SQL CLOSE managerCursor;
    for (i=0; i<15; i++)
        printf("Stellen = %d: Anzahl Manager = %d\n",
            i, counts[i]);
}
```

Cursor deklarieren und öffnen

Tupel abrufen

≡ NO DATA

Cursor schließen

Überblick

- Embedded SQL
- Stored procedures / PSM
- Call-level-interface (CLI) und Java Database Connectivity (JDBC)

PSM / Stored Procedures

- PSM: Persistent, Stored Modules
  - „Gespeicherte Prozeduren“
  - *Stored Procedures*
- Speicherung von Prozeduren als Datenbankelemente
- Mischung aus SQL und Programmiersprache
- Prozeduren können in SQL Anfragen oder anderen SQL Ausdrücken verwendet werden.
- CREATE PROCEDURE <name> (<parameter>)
  lokale Variablendeklarationen
  body der Prozedur;
- CREATE FUNCTION <name> (<parameter>) RETURNS <Typ>
  lokale Variablendeklarationen
  body der Funktion;

Beispiel – Cursor und Schleifen

- Gegeben ein Studioname, berechne Mittelwert und Varianz der Filmlängen des Studios
-
```
CREATE PROCEDURE MeanVar(
            IN s CHAR[15],
            OUT mittelwert REAL,
            OUT varianz REAL
    )
    DECLARE Not_Found CONDITION FOR SQLSTATE '02000';
    DECLARE FilmCursor CURSOR FOR
            SELECT Laenge FROM Filme WHERE StudioName=s;
    DECLARE neueLaenge INTEGER;
    DECLARE filmAnzahl INTEGER;
```

Beispiel – Cursor und Schleifen

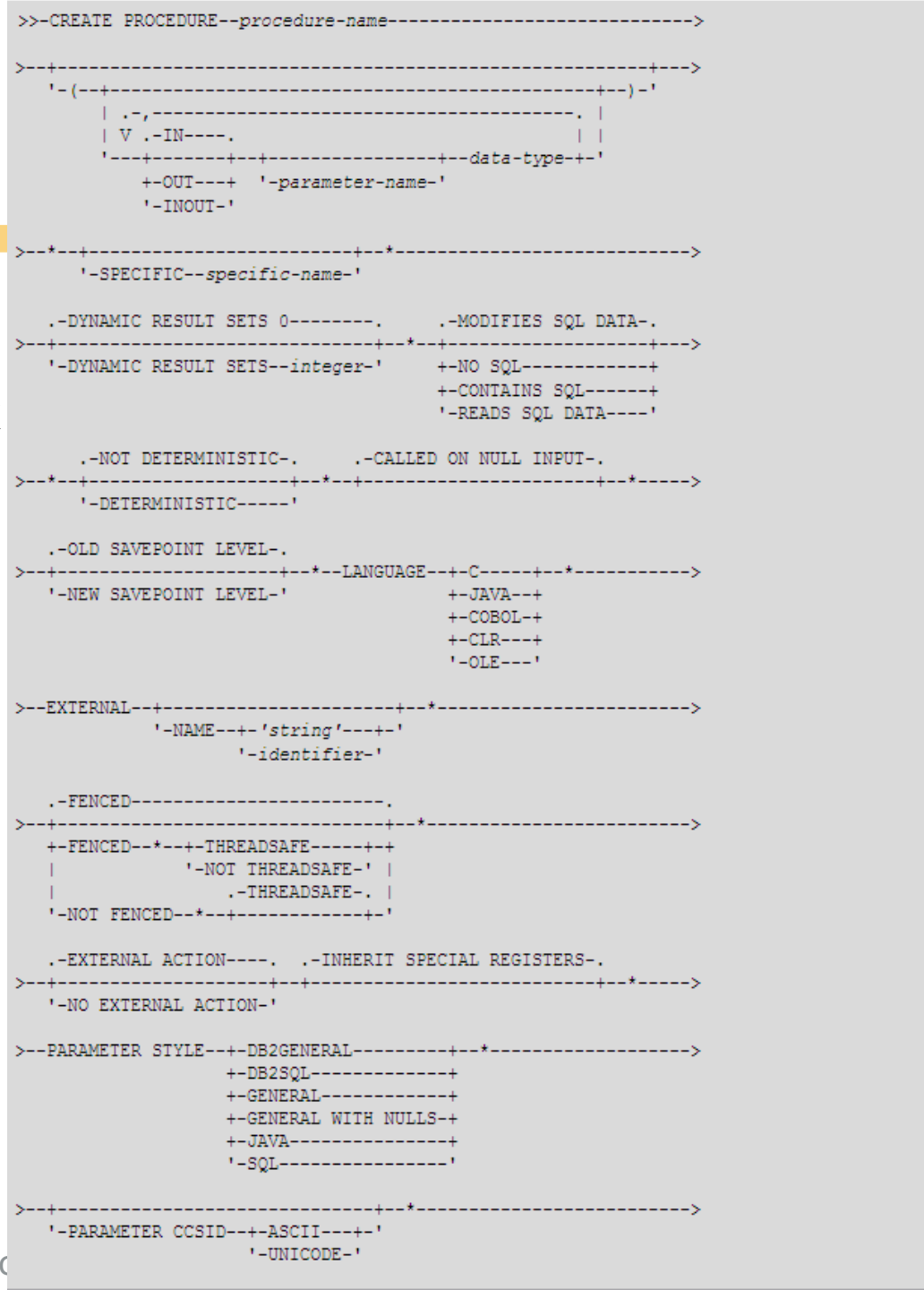

```
BEGIN
    SET mittelwert = 0.0;
    SET varianz = 0.0;
    SET filmAnzahl=0;
    OPEN FilmCursor;
    FilmLoop: LOOP
            FETCH FilmCursor INTO neueLaenge;
            IF Not_Found THEN LEAVE FilmLoop END IF;
            SET filmAnzahl = filmAnzahl + 1;
            SET mittelwert = mittelwert + neueLaenge ;
            SET varianz = varianz + neueLaenge * neueLaenge;
    END LOOP;
    CLOSE FilmCursor;
    SET mittelwert = mittelwert / filmAnzahl;
    SET varianz = varianz/filmAnzahl - mittelwert *
                                              mittelwert;
END;
```

Externe Stored Procedures

- Stored Procedures basierend auf einem Stück Code
- Beispiel für DB2

```
CREATE PROCEDURE PARTS_ON_HAND
    (IN PARTNUM INTEGER,
     OUT COST DECIMAL(7,2),
     OUT QUANTITY INTEGER)
EXTERNAL NAME 'parts.onhand'
LANGUAGE JAVA
PARAMETER STYLE JAVA
```

- Code muss in bestimmtem Verzeichnis liegen
- Schnittstelle vorgegeben



Überblick

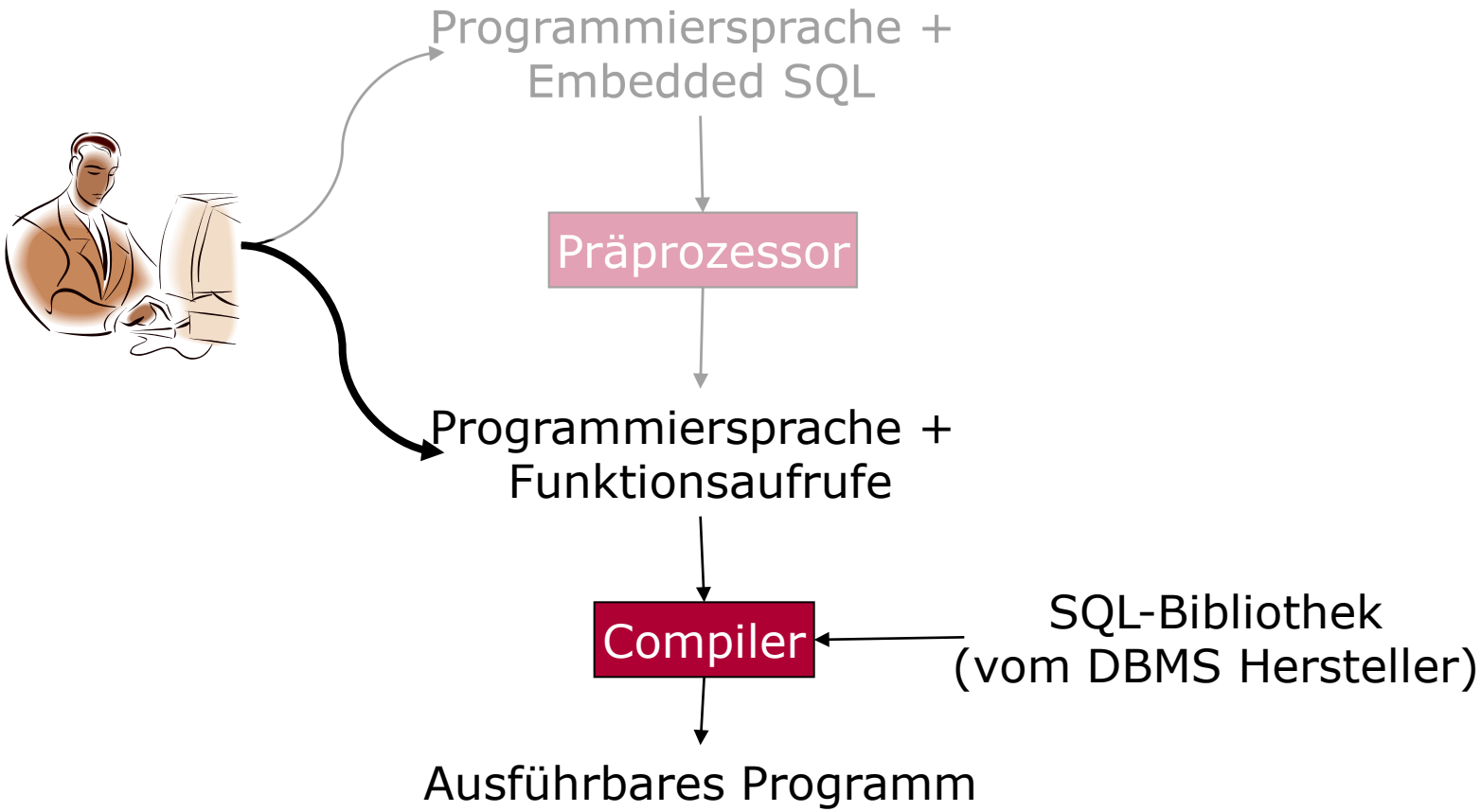
- Embedded SQL
- Stored procedures / PSM
- Call-level-interface (CLI) und Java Database Connectivity (JDBC)

CLI: Call-Level-Interface – Idee

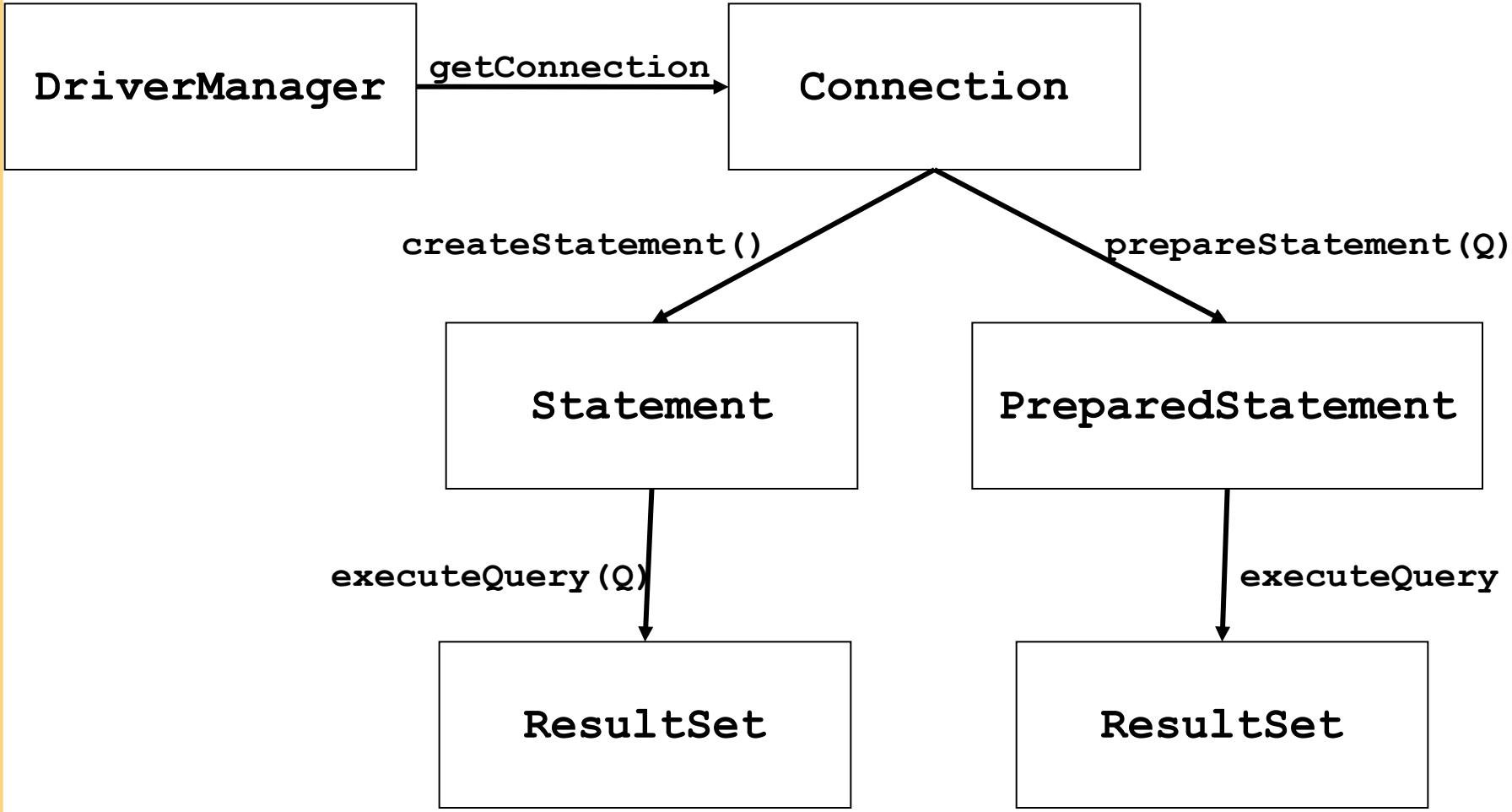
- Programmieren in einer Programmiersprache (Wirtssprache)
- Verwendung spezieller Funktionen für den Datenbankzugriff
  - Funktionsbibliothek
- Umgehung des Präprozessors
  - Kompiliertes Ergebnis ist gleich

- SQL/CLI
  - Adaptiert von ODBC (Open Database Connectivity)
  - Für die Programmiersprache C
- JDBC (Java Database Connectivity)
  - Gleiches Ziel wie SQL/CLI
  - Java statt C als Programmiersprache
  - Unterschiede aufgrund der Objektorientierung von Java

CLI / JDBC - Ablauf



JDBC Überblick



Erste Schritte

- Treiber für das DBMS einbinden
  - DBMS spezifisch
  - Wird jeweils mit DBMS mitgeliefert
- Verbindung zur Datenbank aufbauen
  - `Connection con = DriverManager.getConnection(URL, name, password);`
  - URL ist DBMS und Datenbank-spezifisch
    - "jdbc:subprotocol:datasource";
  - `Connection con = DriverManager.getConnection("jdbc:db2://<server>:<port>/<db_name>", "<username>", "<password>");`

Ausdrücke in JDBC

- Statement
  - `Statement stmt = createStatement();`
  - Noch keiner SQL-Anfrage assoziiert
- PreparedStatement
  - `PreparedStatement pstmt = prepareStatement(Anfrage);`
  - Falls Anfrage öfters ausgeführt wird.
- SQL Ausdrücke ausführen (4 Varianten)
  1. `ResultSet rs = stmt.executeQuery(Anfrage);`
     - ResultSet als Rückgabewert
  2. `ResultSet rs = pstmt.executeQuery();`
     - Für PreparedStatements
     - ResultSet als Rückgabewert
  3. `excuteUpdate(UpdateAnfrage)`
     - Kein Rückgabewert
  4. `excuteUpdate()`
     - Für PreparedStatements
     - Kein Rückgabewert

JDBC – Beispiel

- Gegeben die Connection **con**
- Anfrage: SELECT Gehalt FROM Manager; (2 Varianten der Ausführung)

1. ```
   Statement managerStmt = con.createStatement();
   ResultSet gehaelter = managerStmt.executeQuery(
                         „SELECT Gehalt FROM Manager“);
   ```
2. ```
   PreparedStatement managerStmt = con.prepareStatement(
                         „SELECT Gehalt FROM Manager“);
   ResultSet gehaelter = managerStmt.executeQuery();
   ```

- Updates: Schauspieler einfügen
  - ```
    Statement spielerStmt = con.createStatement();
    spielerStmt.executeUpdate(
            "INSERT INTO spielt_in VALUES(" +
            "'Star Wars', 1979, 'Harrison Ford'");
    ```
    ```

Cursor in JDBC (ResultSet)

- Methoden des ResultSet
  - **`next()`**
    - liefert nächstes Tupel
    - FALSE falls kein weiteres Tupel vorhanden
  - **`getString(i)`**
    - Liefert Wert des i-ten Attributs
    - **`getInt(i)`**, **`getFloat(i)`** usw.
  - **`while(gehaelter.next()){`**

```
        gehalt = gehaelter.getInt(1);
        // gehalt ausgeben o.ä.
}
```

Parameter übergeben

- Mittels **`PreparedStatement`**
- Fragezeichen als Platzhalter für Parameter
  - Bindung mittels **`setString(i, v)`**, **`setInt(i, v)`** usw.

-
```
PreparedStatement studioStmt = con.prepareStatement(
      "INSERT INTO Studio(Name, Adresse) VALUES(?, ?)");

// Werte für studioName und studioAdr bestimmen
…

studioStmt.setString(1, studioName);
studioStmt.setString(2, studioAdr);
studioStmt.executeUpdate();
```

Zusammenfassung: Arbeiten mit JDBC

- Erstellen einer Java-Klasse
  - Datenbankverbindung herstellen
    - über den DriverManager
  - SQL-Anfragen ausführen
    - als Statements oder PreparedStatements
  - Ergebnisse von SQL-Anfragen
    - ResultSets
- Kompilieren des Codes
  - Einbinden der JDBC-Bibliotheken über den classpath
- Ausführen des Programms

Aufgabe 1 – Foreign Keys

- Schema
  - `Lieferant (LieferantenID, LieferantenName)`
  - `Produktkatalog(ProduktID, ProduktName, Einkaufspreis, LieferantenID)` mit LieferantenID FK zu Lieferant
- Wir wollen nur Produkte in unserer Datenbank, zu denen auch ein Lieferant existiert. Und sollte ein Lieferant aussteigen, wollen wir auch dessen Produkte aus dem Produktkatalog löschen.
- Wie muss der Foreign Key lauten?

Aufgabe 1 – Foreign Keys

- Schema
  - `Lieferant (LieferantenID, LieferantenName)`
  - `Produktkatalog(ProduktID, ProduktName, Einkaufspreis, LieferantenID)` mit LieferantenID FK zu Lieferant

- Wir wollen nur Produkte in unserer Datenbank, zu denen auch ein Lieferant existiert. Und sollte ein Lieferant aussteigen, wollen wir auch dessen Produkte aus dem Produktkatalog löschen.
- Wie muss der Foreign Key lauten?

- ALTER TABLE Produktkatalog ADD FOREIGN KEY (LieferantenID) REFERENCES Lieferant ON DELETE CASCADE

Aufgabe 2 – Trigger

- Schema
  - `Lieferant (LieferantenID, LieferantenName)`
  - `Produktkatalog(ProduktID, ProduktName, Einkaufspreis, LieferantenID)` mit LieferantenID FK zu Lieferant
- Sobald wir in unserem Produktkatalog das letzte Produkt eines Lieferanten löschen, wollen wir auch den Lieferanten löschen.
- Was muss der Trigger tun?

Aufgabe 2 – Trigger

- Schema
  - Lieferant (LieferantenID, LieferantenName)
  - Produktkatalog(ProduktID, ProduktName, Einkaufspreis, LieferantenID)

```
CREATE TRIGGER deleteLieferant AFTER DELETE ON Produktkatalog
REFERENCING old as deletedProd
FOR EACH ROW
WHEN (0 = (SELECT count(*)
              FROM Produktkatalog
              WHERE LieferantenID = deletedProd.LieferantenID))
DELETE FROM Lieferant WHERE LieferantenID = deletedProd.LieferantenID
```