Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Übung Datenbanksysteme I

JDBC

Thorsten Papenbrock

Grenzen von SQL

1. Bedingte Anweisungen
   - Erhöhe das Gehalt eines *Arbeitsnehmers* um 2%, falls er eine Belobigung erhalten hat
2. Darstellung von Daten (z.B. Web-Anwendungen)
   - Erstelle eine übersichtliche und schöne Repräsentation für eine Menge von *Produkt*-Tupeln
3. Komplizierte Fragestellungen (z.B. Duplikaterkennung)
   - Finde alle *Kunden*-Tupel, die sich sehr ähnlich sind
4. String-Operationen (z.B. String-Splitting)
   - Teile das Attribut *Name* auf in die Attribute *Vor-*, *Mittel-* und *Nachname* und weise den neuen Attributen anschließend passende Typen zu

Motivation: SQL und Programmiersprachen

- Embedded SQL
  - Kombiniert SQL mit Programmiersprachen
    - ADA, C, C++, Cobol, Fortran, M, Pascal, PL/I, …
  - Bettet SQL beim Preprocessing in den Programmfluss ein
- Stored Procedures
  - Speichern Prozeduren als DBMS Objekte in der Datenbank
  - Können in SQL Ausdrücken aufgerufen werden
- Call-Level-Interface (CLI)
  - Verbindet C mit DBMS
  - Implementiert eine DBMS-spezifische Funktionsbibliothek
  - Benötigt kein Preprocessing
- Java Database Connectivity (JDBC)
  - Funktioniert wie CLI aber für Java

Impedance Mismatch

- Bisher: Generische SQL Schnittstelle
  - Absetzen einzelner, unverknüpfter SQL-Statements
  - Verwendung über Kommandozeile
  - Selten genutzt (→ Datenbank-Administratoren)

- Jetzt: SQL in Programmiersprachen
  - Einbettung in (große) Softwarekomponenten
  - Verwendung aus dem Quellcode heraus
  - Ausgiebig genutzt in allen möglichen Software-Projekten

    ➢ Impedance Mismatch

Impedance Mismatch

- Impedance Mismatch
  = Verwendung unterschiedlicher Datenmodelle
  - Relationales Model (DBMS)
    - Relationen und Attribute
    - Nebenbedingungen
    - Deklarativ
  - Generisches Modell (Programmiersprachen)
    - Pointer, verschachtelte Strukturen und Objekte
    - Schleifen und Verzweigungen
    - i.d.R. Imperativ

➢ Datentransfer zwischen den Modellen ist schwierig!

Impedance Mismatch

- Idee 1: Nutze SQL allein, allerdings …
  - nicht alle Anweisungen ausdrückbar (z.B. nicht "n!")
  - Ausgabe beschränkt auf Relationen (z.B. keine Graphiken)
- Idee 2: Nutze Programmiersprache allein, allerdings …
  - Anweisungen sind meist viel komplexer als SQL-Anfragen
  - Abstraktion von Speicherstrukturen nicht möglich
    - ➢ Physische Datenunabhängigkeit!
  - DBMS sind extrem effizient
- Idee 3: Nutze SQL eingebettet in einer Programmiersprache
  - Programmiersprache ("Host Language") für komplexe Operationen
  - Embedded SQL für effizienten Datenzugriff
  - Explizites Mapping der gelesenen und geschriebenen Daten

Übersicht:

Einbettung von SQL

Embedded SQL	Stored Procedures	OCL und JDBC

Embedded SQL: Übersetzung

Programmiersprache & Embedded SQL

Präprozessor

Programmiersprache & Funktionsaufrufe

Compiler

Ausführbares Programm

DBMS-spezifisch, d.h. Kompiliervorgang muss für jedes DBMS (und möglicherweise auch für verschiedene Versionen desselben DBMS) erneut ausge-Führt werden!

Anfragen ohne Ergebnis

1. Host-Variablen deklarieren

```
EXEC SQL BEGIN DECLARE SECTION;
    char studioName[50], studioAdr[256];
    char SQLSTATE[6];
EXEC SQL END DECLARE SECTION;
```

Hostvariablen mit Doppelpunkt

2. Anfrage ohne Ergebnis

```
EXEC SQL INSERT INTO Studio(Name, Adresse)
         VALUES (:studioName, :studioAdr);
```

- Verfahren für jeden SQL-Ausdruck, der kein Ergebnis liefert:
  `INSERT`, `DELETE`, `UPDATE`, `CREATE`, `DROP`, …

Anfragen ein Ergebnis-Tupel

1. Host-Variablen deklarieren

```
EXEC SQL BEGIN DECLARE SECTION;
    char studioName[50], studioAdr[256];
    char SQLSTATE[6];
EXEC SQL END DECLARE SECTION;
```

2. Anfrage ein Ergebnis-Tupel

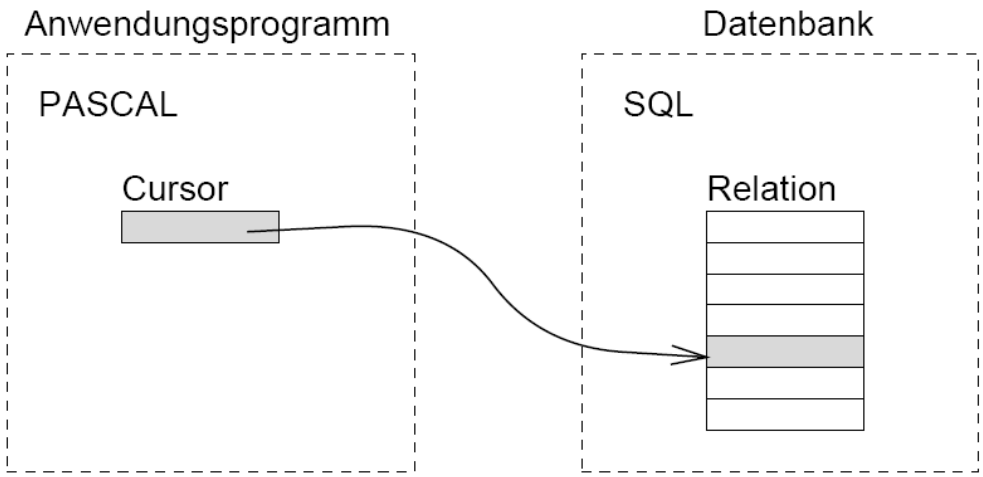
```
EXEC SQL SELECT Name, Adresse
         INTO :studioName, :studioAdr
         FROM Studio
         WHERE id = 310;
```

Anfragen viele Ergebnis-Tupel

1. Host-Variablen deklarieren

```
EXEC SQL BEGIN DECLARE SECTION;
    char studioName[50], studioAdr[256];
    char SQLSTATE[6];
EXEC SQL END DECLARE SECTION;
```

2. Anfrage viele Ergebnis-Tupel



Embedded SQL:
Anfragen viele Ergebnis-Tupel

```
void printGehaltsBereiche() {
   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 und Cursor weitersetzen

≡ NO DATA

Cursor schließen

Übersicht:

Einbettung von SQL

Embedded SQL	Stored Procedures	OCL und JDBC

Konzept und Umsetzung

- Persistant Stored Modules (PSM):
  - „Gespeicherte Prozeduren“, engl. *Stored Procedures*
  - Speichern Prozeduren als Datenbankelemente
  - Mischen SQL und Programmiersprache
  - Können in regulären SQL Ausdrücken verwendet werden

```
CREATE PROCEDURE <Name>(<Parameter in/out>)
       <Lokale Variablendeklarationen>
       <Body der Prozedur>;
```

```
CREATE FUNCTION <Name>(<Parameter in>) RETURNS <Typ>
       <Lokale Variablendeklarationen>
       <Body der Prozedur>;
```

Stored Procedures / PSM:

Beispiel: Mittelwert und Varianz

```
CREATE PROCEDURE MeanVar( IN studioName 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 = studioName;
   DECLARE neueLaenge INTEGER;
   DECLARE filmAnzahl INTEGER;

   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;
```

Parameter in/out

Lokale Variablen-deklarationen

Body der Prozedur

Externe Definition

- Stored Procedures können externen Code verwenden
- Code muss dazu in bestimmtem Verzeichnis liegen
- Beispiel:
  - Datenbank: DB2
  - Externe Sprache: Java

```
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;
```

Übersicht:

Einbettung von SQL

Embedded SQL	Stored Procedures	OCL und JDBC

CLI und JDBC:
Übersetzung

Programmiersprache &
Embedded SQL

Präprozessor

Programmiersprache &
Funktionsaufrufe

Compiler

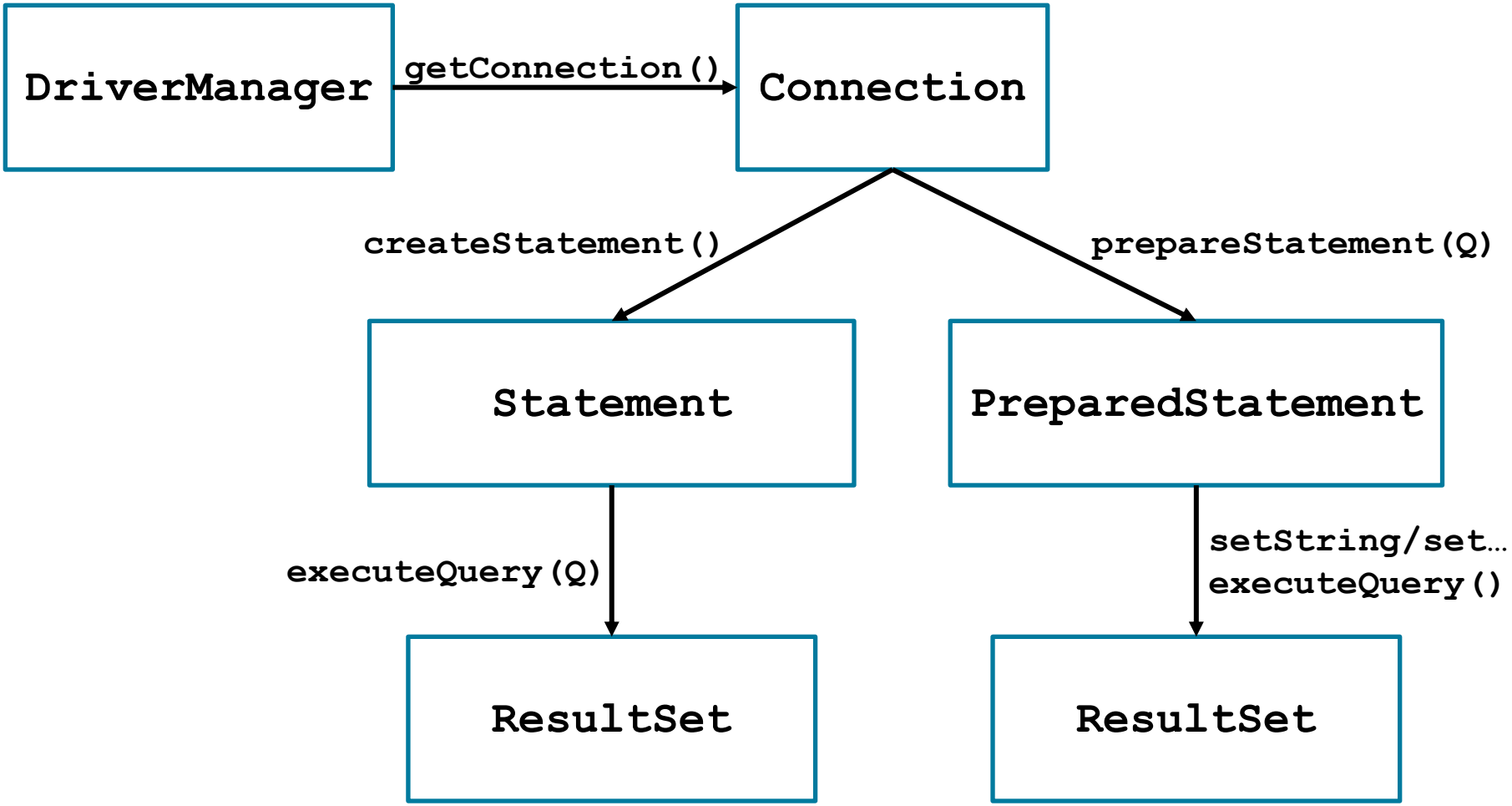
SQL-Bibliothek (Treiber)
vom DBMS-Hersteller

Ausführbares Programm

Konzept

- DBMS-Spezifische Funktionsbibliotheken:
  - Ermöglichen …
    - Programmieren in einer Programmiersprache (Wirtssprache)
    - Einbettung von Datenbankanfragen in SQL
  - Bieten spezielle Funktionen für den Datenbankzugriff
  - Umgehen den Präprozessor
    - Kompiliertes Ergebnis ist aber gleich!
- Call-Level-Interface (CLI)
  - Verbindet C mit DBMS
  - Adaptiert von ODBC (Open Database Connectivity)
- Java Database Connectivity (JDBC)
  - Verbindet Java mit DBMS
  - Nutzt Objektorientierung im Gegensatz zu CLI

JDBC: Überblick



Erste Schritte

1. DBMS-spezifischen Treiber einbinden
   - JDBC-Bibliothek in den Classpath einbinden

2. Verbindung zur Datenbank aufbauen

```
String URL = "jdbc:db2://<server>:<port>/<db_name>";
String name = "<username>";
String pw = "<password>";
Connection con = DriverManager.getConnection(URL, name, pw);
```

   - URL ist DBMS- und Datenbank-spezifisch
   - URL-Pattern: "jdbc:subprotocol:datasource"

DBMS-URLs

Vendor	DBMS-URL
DB2	jdbc:db2://{host}[:{port}]/{dbname}
Derby	jdbc:derby://server[:port]/databaseName[;URLAttributes=value[;...]]
HSQLDB	jdbc:hsqldb[:{dbname}][:hsql://{host}[/{port}]]
MS SQL-Server	jdbc:microsoft:sqlserver://{host}[:{port}][;DatabaseName={dbname}]
MySQL	jdbc:mysql://{host}[:{port}]/[{dbname}]
Oracle	jdbc:oracle:thin:@{host}:{port}:{dbname}
PostgreSQL	jdbc:postgresql://[{host}[:{port}]]/{dbname}

Überblick

Statement

Noch ohne SQL-Anfrage

```
Statement stmt = con.createStatement();
```

Prepared Statement

Für häufige SQL-Anfragen

```
PreparedStatement pstmt = con.prepareStatement(<Anfrage>);
```

SQL Ausdrücke ausführen

1. `ResultSet rs = stmt.executeQuery(<Anfrage>);`
2. `ResultSet rs = pstmt.executeQuery();`
3. `stmt.executeUpdate(UpdateAnfrage)`
4. `pstmt.executeUpdate()`

ResultSet als Rückgabewert

Ohne Rückgabewert

Beispiele

Liste alle Manager-Gehälter:

```
1. Statement stmt = con.createStatement();
   ResultSet rs = stmt.executeQuery("SELECT Gehalt FROM Manager");
```

```
2. PreparedStatement managerStmt = con.prepareStatement(
                                        "SELECT Gehalt FROM Manager");
   ResultSet rs = pstmt.executeQuery();
```

Füge neues Schauspieler-Tupel ein:

```
1. Statement stmt = con.createStatement();
   stmt.executeUpdate("INSERT INTO spielt_in VALUES(" +
                      "'Star Wars', 1979, 'Harrison Ford'");
```

```
2. PreparedStatement managerStmt = con.prepareStatement(
                      "INSERT INTO spielt_in VALUES(" +
                      "'Star Wars', 1979, 'Harrison Ford'");
   pstmt.executeUpdate();
```

Cursor - ResultSet

- Methoden des ResultSet
  - **`next()`**
    - liefert nächstes Tupel
    - liefert **`FALSE`**, falls kein weiteres Tupel vorhanden
  - **`getString(i)`**
    - Liefert Wert des i-ten Attributs
    - **`getInt(i)`**, **`getFloat(i)`** usw.
    - Alternativ: **`getString("<AttributName>")`**
  - Anwendungsbeispiel:

```
while(gehaelter.next()){
        gehalt = gehaelter.getInt(1);
        // Operationen auf gehalt
}
```

Parameter

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

- Beispiel: Einfügen eines neuen Studios

```
String studioName = "Pixar Animation Studios";
String studioAdr = "Emeryville, Vereinigte Staaten";
PreparedStatement studioStmt = con.prepareStatement(
       "INSERT INTO Studio(Name, Adresse) VALUES(?, ?)");

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

Zusammenfassung

1. JDBC-Bibliothek einbinden
   - DBMS-spezifische JAR in den Classpath aufnehmen
2. Verbindung zur Datenbank aufbauen
   - Connection über **`DriverManager`** herstellen
3. SQL-Anfragen ausführen
   - **`Statements`** oder **`PreparedStatements`** nutzen
4. Ergebnis der SQL-Anfragen abfragen
   - **`ResultSet`** auswerten

Exkurs 1: Foreign Keys

Schema:

- Lieferant (LieferantID, LieferantName, Adresse)
- Produkt (ProduktID, ProduktName, Einkaufspreis, LieferantID)

„Wir erlauben nur Produkte in unserer Datenbank, zu denen auch ein Lieferant existiert, und sollte ein Lieferant aussteigen, wollen wir auch dessen Produkte löschen."

Wie muss der geforderte Foreign Key definiert werden?

```
ALTER TABLE Produkt ADD CONSTRAINT ProduktFremdschlüssel
FOREIGN KEY (LieferantID) REFERENCES Lieferant (LieferantID)
ON DELETE CASCADE;
```

Exkurs 2: Trigger

Schema:

- Lieferant (LieferantID, LieferantName, Adresse)
- Produkt (ProduktID, ProduktName, Einkaufspreis, LieferantID)

„Sobald das letzte Produkt eines Lieferanten gelöscht wird, soll auch der entsprechende Lieferant gelöscht werden."

Wie muss der entsprechende Trigger definiert werden?

```
CREATE TRIGGER LöscheLieferant AFTER DELETE ON Produkt
REFERENCING OLD AS gelöschtesProdukt
FOR EACH ROW
WHEN (0 = (SELECT COUNT(*)  FROM Produkt
                WHERE LieferantID = gelöschtesProdukt.LieferantID))
DELETE FROM Lieferant WHERE LieferantID = gelöschtesProdukt.LieferantID;
```

Übung:
Allgemeine Informationen

- Übung 3: SQL
  - kann nächste Woche Freitag morgen (9-12 Uhr) bei mir im Büro abgeholt werden
  - korrekter Datenimport diesmal nicht bewertet, bei der nächsten Übung aber schon
- Übung 4: JDBC
  - Abgabe als Aufgabe 5!
  - neben den pdf-Lösungen auch den Quellcode mit hochladen