Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Folien basierend auf Thorsten Papenbrock

Übung Datenbanksysteme I
Embedded SQL, Stored Procedures, JDBC

Leon Bornemann
F-2.06, Campus II
Hasso Plattner Institut

Motivation

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 andere Programmiersprachen

1. **Embedded SQL**
   - Integriert SQL in andere Programmiersprachen:
     - ADA, C, C++, Cobol, Fortran, M, Pascal, PL/I, …
   - Bettet SQL beim Preprocessing in den Programmfluss ein
2. **Stored Procedures**
   - Speichern Prozeduren als DBMS Objekte in der Datenbank
   - Können in SQL Ausdrücken aufgerufen werden
3. **DBMS-Funktionsbibliotheken**
   - Verbindet DBMS mit Programmiersprachen
   - Benötigt kein Preprocessing
   - Call-Level-Interface (CLI) für C
   - Java Database Connectivity (JDBC) für Java

Motivation

Impedance Mismatch

- **Bisher: Generische SQL Schnittstelle**
  - Absetzen einzelner, unverknüpfter SQL-Statements
  - Verwendung über Kommandozeile oder DBMS-spezifischer GUIs
  - Selten genutzt (→ Datenbank-Administratoren)
  - Allerdings: SQL ist prinzipiell Turing-vollständig!

- **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)*
    - Primitive: Relationen und Attribute
    - Kontrolle: Nebenbedingungen
    - Modell: Deklarativ
  - *Generisches Modell (Programmiersprachen)*
    - Primitive: Pointer, verschachtelte Strukturen und Objekte
    - Kontrolle: Schleifen und Verzweigungen
    - Model: Imperativ (meistens)

  ➢ Datentransfer zwischen den Modellen notwendig!

Motivation

Impedance Mismatch

- **Idee 1: Nutze SQL allein, allerdings …**
  - nicht alle Anweisungen ausdrückbar (z.B. nicht "n!")
  - Ausgabe beschränkt auf Relationen (z.B. keine Grafiken)
- **Idee 2: Nutze Programmiersprache allein, allerdings …**
  - Anweisungen sind meist viel komplexer als SQL-Anfragen
  - Abstraktion von Speicherstrukturen nicht möglich
    - Verlust der physischen 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

JDBC

DBSI - Übung

Embedded SQL, Stored Procedures, JDBC

Leon Bornemann

Embedded SQL

Übersetzung

Programmiersprache & Embedded SQL

↓

Präprozessor

↓

Programmiersprache & Funktionsaufrufe

↓

Compiler

↓

Ausführbares Programm

Bringt SQL in die Programmiersprache

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

Anfragen ohne Ergebnis

Füge ein Tupel in die Datenbank ein, welches mit Variablen aus dem Programm befüllt wird.

1. Host-Variablen deklarieren

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

2. Anfrage ohne Ergebnis

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

Hostvariablen mit Doppelpunkt

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

Embedded SQL

Anfragen mit einem Ergebnis-Tupel

Lese die Werte eines Tupels in Variablen des Programms ein

1. Host-Variablen deklarieren

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

2. Anfrage mit einem Ergebnis-Tupel

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

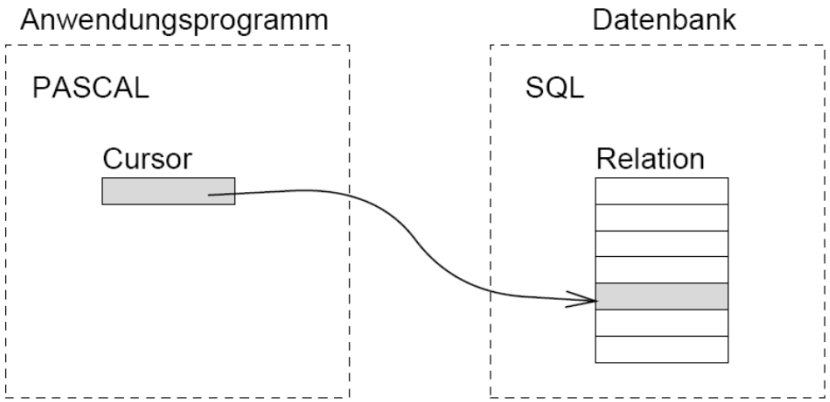
Embedded SQL

Anfragen mit vielen Ergebnis-Tupeln

1. Host-Variablen deklarieren

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

2. Anfrage mit vielen Ergebnis-Tupeln



Embedded SQL

Anfragen mit vielen Ergebnis-Tupeln

```
void printGehaltsBereiche() {

                Stellen = 0: Anzahl Manager = 2
                Stellen = 1: Anzahl Manager = 0
                Stellen = 2: Anzahl Manager = 2
                Stellen = 3: Anzahl Manager = 12
                Stellen = 4: Anzahl Manager = 39
                Stellen = 5: Anzahl Manager = 43
                Stellen = 6: Anzahl Manager = 31
                Stellen = 7: Anzahl Manager = 25
                Stellen = 8: Anzahl Manager = 6
                Stellen = 9: Anzahl Manager = 2
                Stellen = 10: Anzahl Manager = 0
                Stellen = 11: Anzahl Manager = 0
                Stellen = 12: Anzahl Manager = 2
                Stellen = 13: Anzahl Manager = 1
                Stellen = 14: Anzahl Manager = 1

}
```

Gib für jede Dezimalstellenanzahl von 0 bis 14 die Anzahl an Managern mit einem Gehalt dieser Stellen aus

Embedded SQL

Anfragen mit vielen Ergebnis-Tupeln

```
void printGehaltsBereiche() {

   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 < 15) counts[stellen]++;
   }

   EXEC SQL CLOSE managerCursor;

   for (i = 0; i < 15; i++)
        printf(„Stellen = %d: Anzahl Manager = %d\n", i, counts[i]);
}
```

Wird als Nebeneffekt befüllt

Cursor deklarieren und öffnen

Tupel abrufen und Cursor weitersetzen

≡ NO DATA

Cursor schließen

Embedded SQL

Anfragen mit vielen Ergebnis-Tupeln

```
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 < 15) counts[stellen]++;
   }

   EXEC SQL CLOSE managerCursor;

   for (i = 0; i < 15; i++)
        printf(„Stellen = %d: Anzahl Manager = %d\n", i, counts[i]);
}
```

Wird als Nebeneffekt befüllt

Cursor deklarieren und öffnen

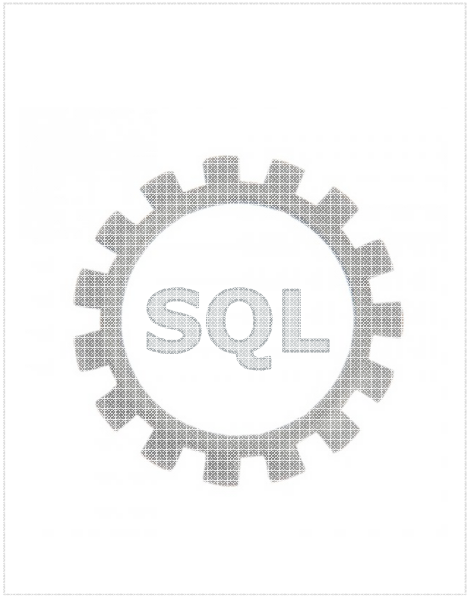
Tupel abrufen und Cursor weitersetzen

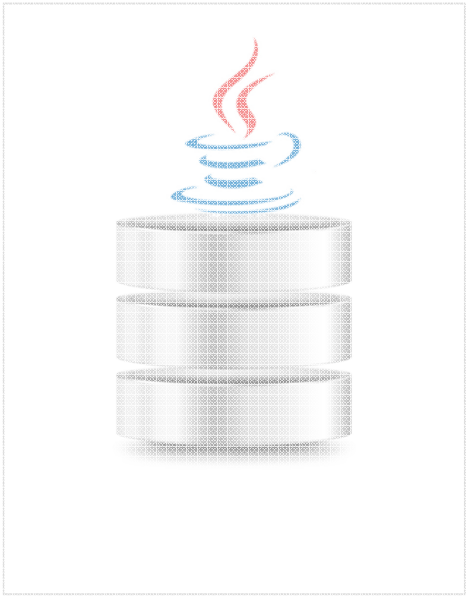
≡ NO DATA

Cursor schließen

Übersicht

Einbettung von SQL





Embedded SQL

Stored Procedures

JDBC

DBSI - Übung

Embedded SQL, Stored Procedures, JDBC

Leon Bornemann

Stored Procedures

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

Bringt Programmiersprache zu SQL

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

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

```
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

Erstelle eine Funktion, die den Durchschnitt und die Varianz der Filmlänge von Filmen eines Studios ermittelt

Unterstützte Programmiersprachen sind abhängig vom DBMS

HPI Hasso Plattner Institut

Stored Procedures

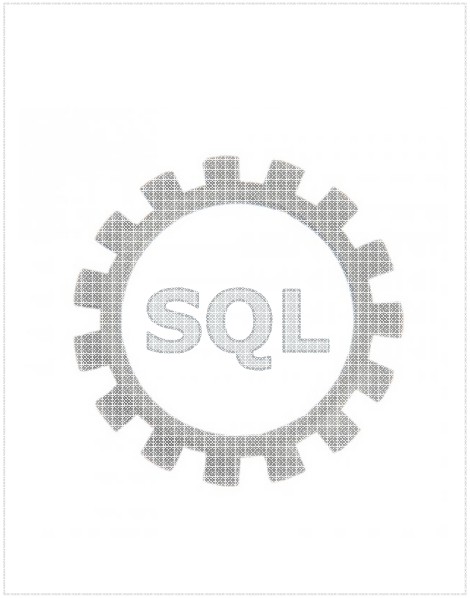
Externe Definition

- 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

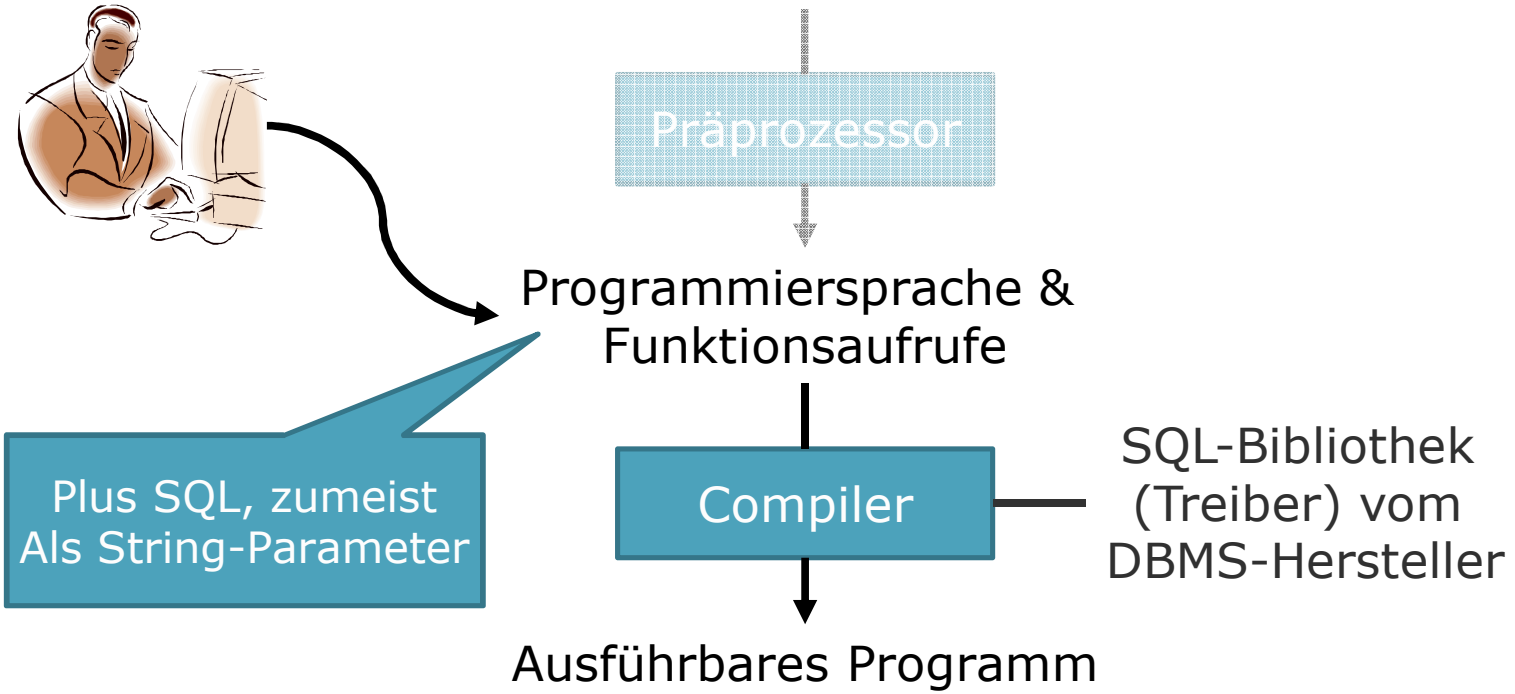


Embedded SQL

Stored Procedures

JDBC

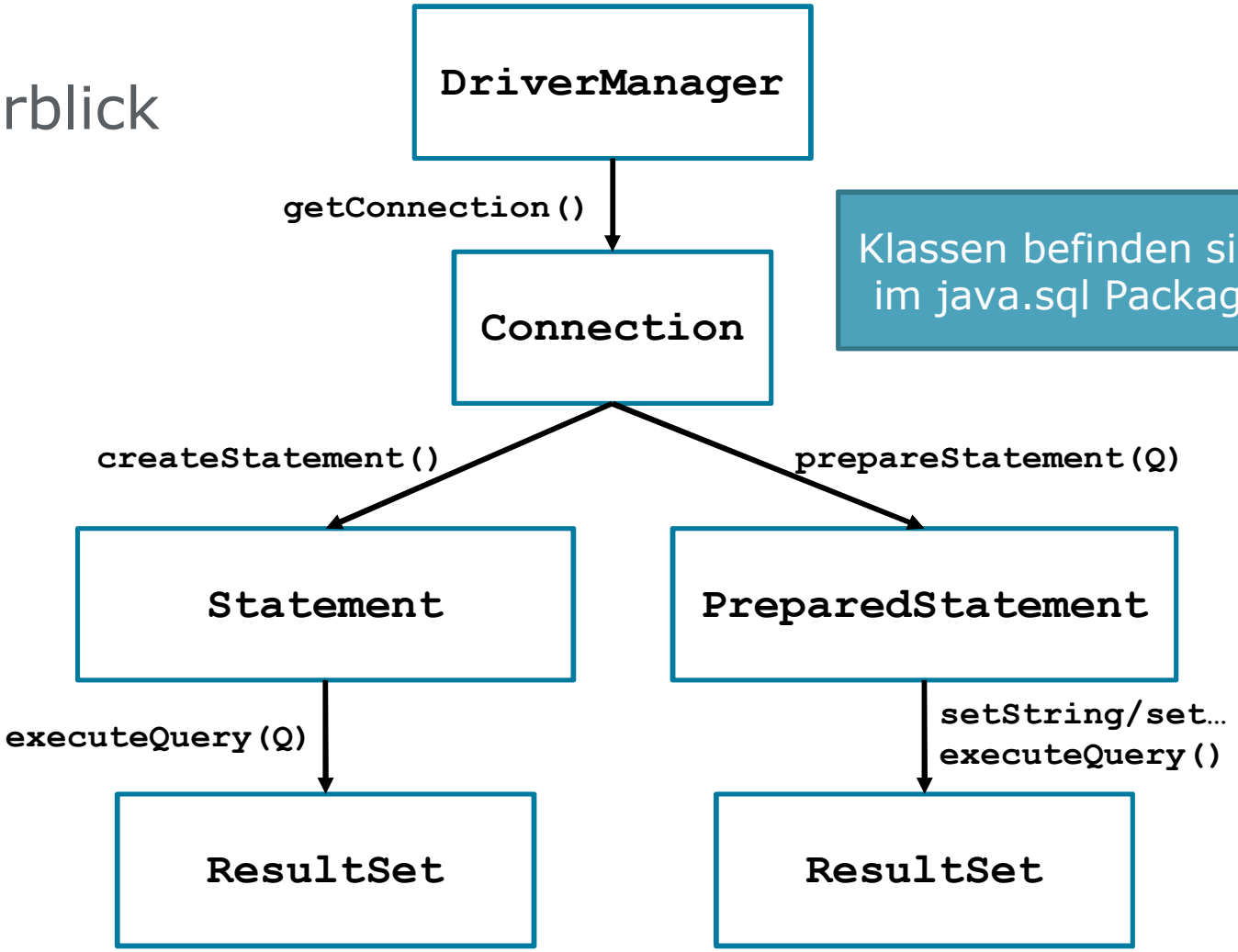
Übersetzung



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

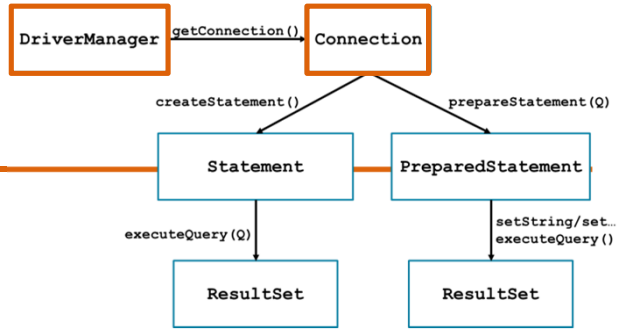
JDBC
Überblick



Klassen befinden sich im java.sql Package

JDBC

Erste Schritte



1. **DBMS-spezifischen Treiber einbinden**
   - JDBC-Bibliothek in den Classpath einbinden
2. **Driver explizit laden**
   - Class.forName(„org.postgresql.Driver“);

For Pre JDBC 4.0 Drivers

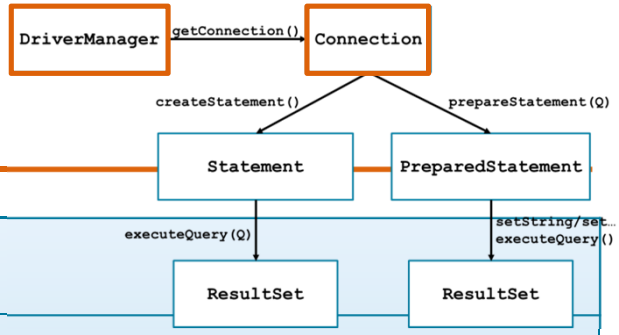
3. **Verbindung zur Datenbank aufbauen**

```
String URL = "jdbc:postgresql://<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"

JDBC

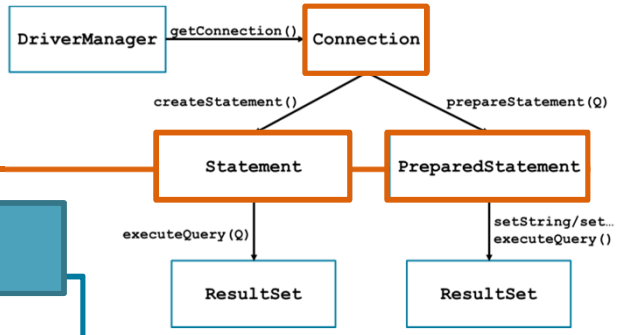
DBMS-URLs



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}

JDBC

Statements



Statement

Noch ohne SQL-Anfrage

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

Prepared Statement

Für häufige SQL-Anfragen

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

SQL Ausdrücke ausführen

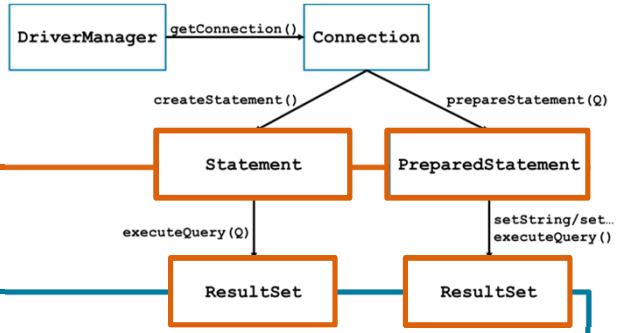
```
1. ResultSet rs = stmt.executeQuery(<Anfrage>);
2. ResultSet rs = pstmt.executeQuery();
3. stmt.executeUpdate(UpdateAnfrage)
4. pstmt.executeUpdate()
```

ResultSet als Rückgabewert

Ohne Rückgabewert

JDBC

Beispiele



Liste alle Manager-Gehälter:

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

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

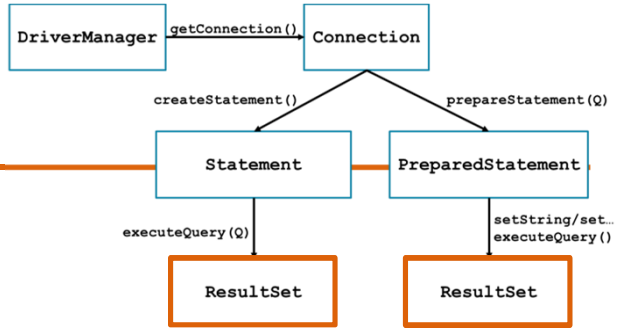
Füge neues Schauspieler-Tupel ein:

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

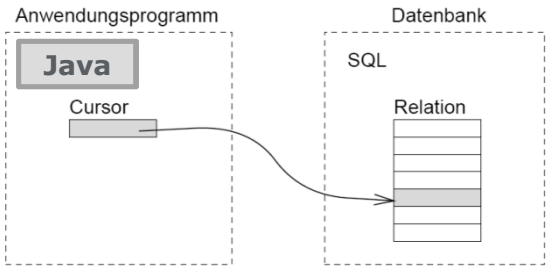
```
2.  PreparedStatement pstmt = con.prepareStatement(
                       "INSERT INTO spielt_in VALUES(" +
                       "'Star Wars', 1979, 'Harrison Ford'");
    pstmt.executeUpdate();
```

JDBC

ResultSet: Cursor



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

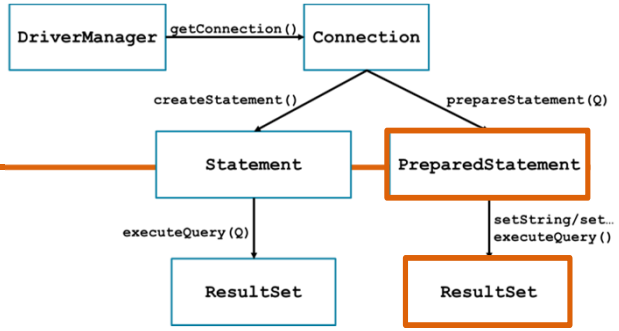


- Anwendungsbeispiel:

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

JDBC

Paramater



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

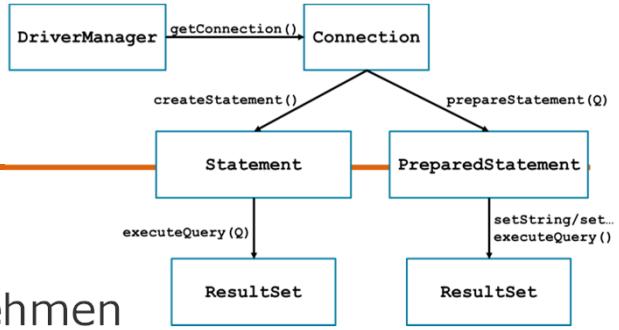
- Beispiel: Einfügen eines neuen Studios

```
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();
```

JDBC

Zusammenfassung



1. JDBC-Bibliothek einbinden
   - DBMS-JAR in den Classpath aufnehmen
2. Verbindung zur Datenbank aufbauen
   - Ggf. Driver-Klasse explizit laden
   - **`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, dann 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;
```

In eigener (fremder) Sache

- Unser Masterprojekt benötigt Unterstützung.
- Thema: Vandalism Detection in Wikipedia Tables
  - Wie sieht sowas aus?

Vandalism in Tables

Revision as of 18:07, 24 November 2009

aaaaaaaaaaaa	aaaaaaaaaaaa	hhhhhhhhhhh
gggggggggggg	hhhhhhhhhhhh	gggggggggggggg
!!!!!!!!!!!!!!!!	!!!!!!!!!!	!!!!!!!!!!!!!!!!

{{!!!!!!!!!!!!!!1

Scull redirects here. The boney structure of an animal's head is the "skull".

Ethnic groups	Population (million)	Percentage	
Javanese	95.217	40.22	Bengkulu, East Java, East Kalir South Sumatra, Yogyakarta
Sundanese	36.701	15.5	Banten, I like cookies
Batak	8.466	3.58	North Sumatra
Other Sulawesi ethnic groups	7.634	3.22	Sulawesi

Vandalism in Tables

Career statistics

Year	Team	G-S	Passing Att.-Comp.	Yards	Pct.	TD	Int.	Long	Sa L
2001	San Diego	1-0	27-15	221	.556	1	0	40	2
2002	San Diego	16-16	526-320	3,284	.608	17	16	52	24
2003	San Diego	11-11	356-205	2,108	.576	11	15	68	21
2004	San Diego	15-15	400-262	3,159	.655	27	7	79	18
2005	San Diego	16-16	500-323	3,576	.646	24	15	54	27
2006	New Orleans	16-16	554-356	4,418	.643	26	11	86	18-105	96.2
2007	New Orleans	16-16	652-440Template:NFLN/R	4,423	.675	28	18	58	16-109	89.4
2008	New Orleans	16-16	635-413	5,069	.650	34	17	84	13-92	96.2
2009	New Orleans	10-10	320-218	500,000,000	.681	27	9	63	13-84	105.8
Totals		114-113	3,970-2,552	29,004	.643	190	108	86	152-1114	90.7
Postseason		3-3	123-78	916	.634	5	2	88	8-46	92.7

Other substances

The following table is a list of a variety of substances ordered by increasing vapor pressure.

Substance	Vapor Pressure (SI units)	Vapor Pressure (bar)	Vapor Pressure (mmHg)	Temperature
Tungsten	100 Pa	0.001	0.75	3203 °C
Ethylene glycol	0.5 kPa	0.005	3.75	20 °C
Propanol	2.4 kPa	0.024	18.0	20 °C
Water	2009 kPa	0.024	18.0	20 °C

DBSI - Übung

Embedded SQL, Stored Procedures, JDBC

Leon Bornemann

Chart **34**

In eigener (fremder) Sache

- Unser Masterprojekt benötigt Unterstützung.
- Thema: Vandalism Detection in Wikipedia Tables
  - Wie sieht sowas aus?
- Was muss getan werden?
  - Manuelle Annotation von vielen Daten
  - Konkrete Frage: Ist eine Revision Vandalismus oder nicht?
- Wann?
  - Donnerstag 14.06. ab 17:00
- Wo?
  - Bei uns im F-Gebäude
  - Raum wird noch festgelegt

Warum solltet ihr teilnehmen?

- Spaß
- Ewige Dankbarkeit der Masterstudenten :)
- Socializing
  - Einblick in Forschungsarbeit
  - Einblick in Masterprojekte
  - Direkter Kontakt zu den Masterst…ten
- Weitere Gründe:

Free Pizza

Segway Probefahrten

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Übung Datenbanksysteme I

Embedded SQL, Stored Procedures, JDBC

Leon Bornemann
F-2.06, Campus II
Hasso Plattner Institut