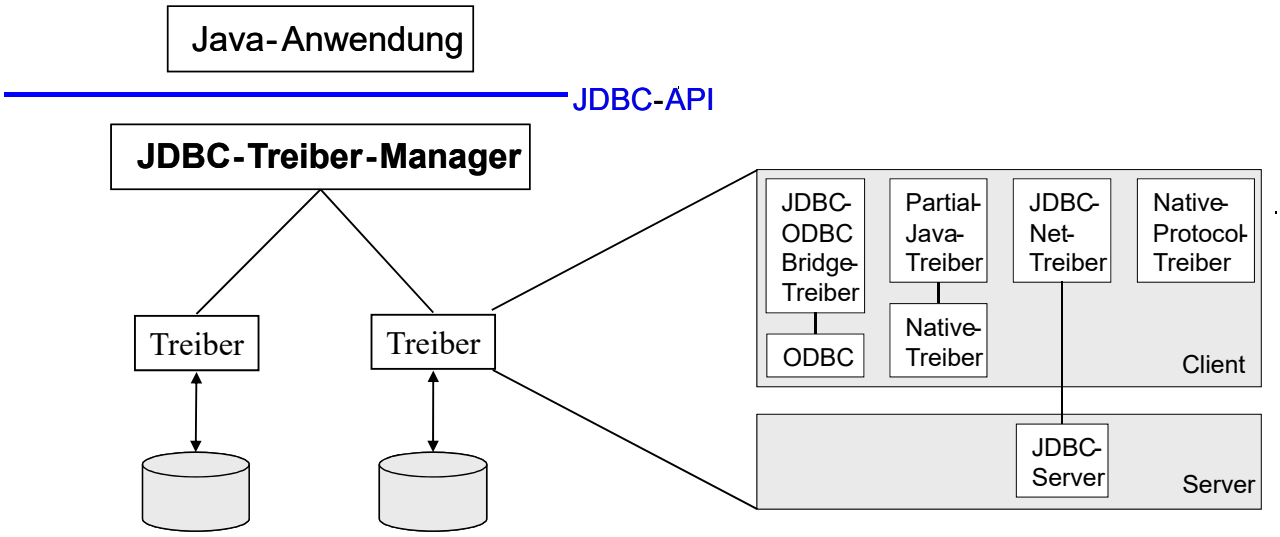
2. DB-Anwendungsprogrammierung (Teil 2)

- JDBC
  - wesentliche JDBC-Operationen
  - Transaktionskontrolle
  - Nutzung von Stored Procedures
- SQLJ
  - Vergleich mit JDBC
  - Iteratorkonzept
- DB-Zugriff mit Skriptsprachen (am Beispiel von PHP)
- Sicherheitsproblem SQL Injection

JDBC (Java Database Connectivity)

- Standardschnittstelle für den Zugriff auf SQL-Datenbanken unter Java
- basiert auf dem SQL/CLI (call-level-interface)
- Grobarchitektur



- durch Auswahl eines anderen JDBC-Treibers kann ein Java-Programm ohne Neuübersetzung auf ein anderes Datenbanksystem zugreifen

JDBC: Grundlegende Vorgehensweise

- Schritt 1: Verbindung aufbauen

```
import java.sql.*;
        ...
Class.forName ( "COM.ibm.db2.jdbc.net.DB2Driver");
Connection con =
        DriverManager.getConnection ("jdbc:db2://host:6789/myDB", "login","pw");
```

- Schritt 2: Erzeugen eines SQL-Statement-Objekts

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

- Schritt 3: Statement-Ausführung

```
ResultSet rs = stmt.executeQuery ("SELECT matrikel FROM student");
```

- Schritt 4: Iterative Abarbeitung der Ergebnisdatensätze

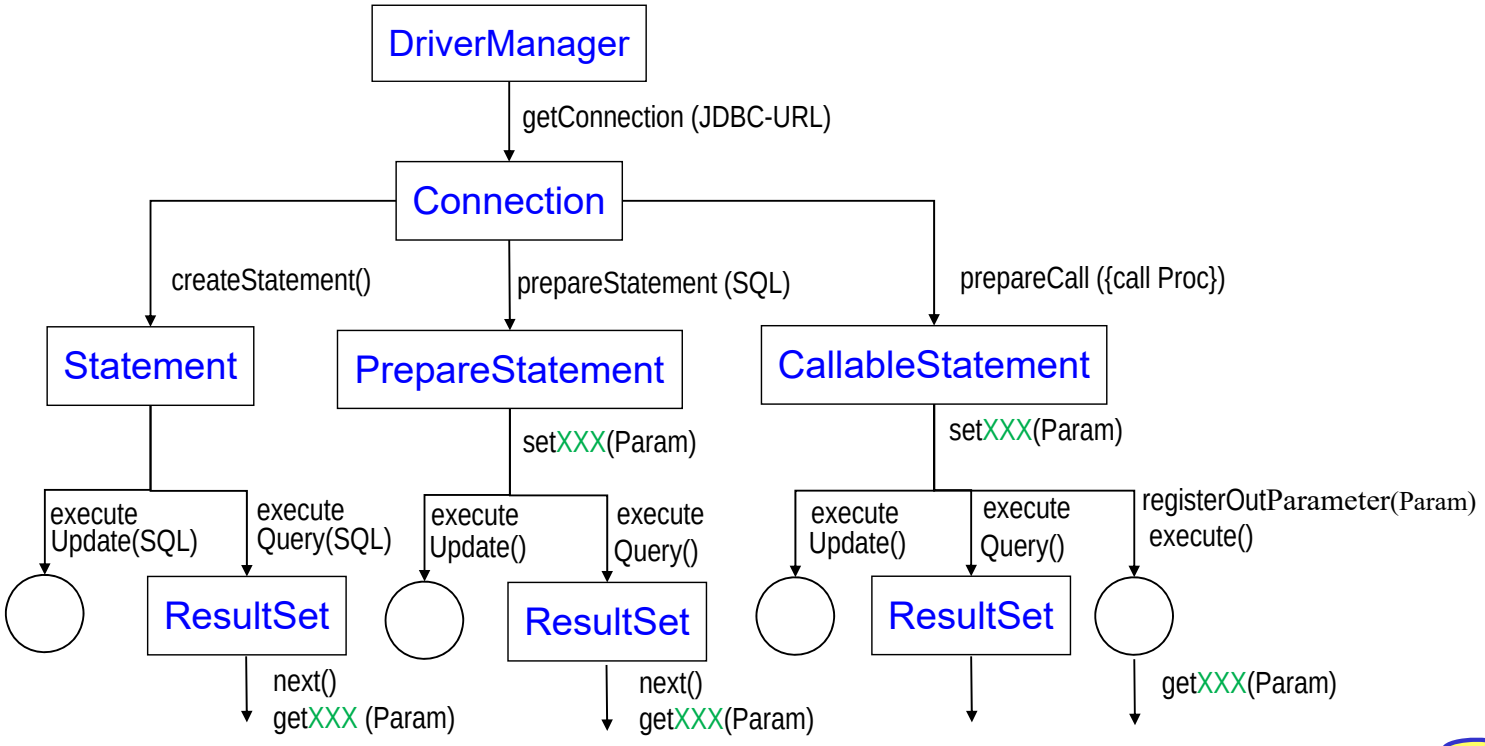
```
while (rs.next())
        System.out.println ("Matrikelnummer: " + rs.getString("matrikel"));
rs.close()
```

- Schritt 5: Schließen der Datenbankverbindung

```
con.close();
```

JDBC-Klassen

- streng typisierte objekt-orientierte API
- Aufrufbeziehungen (Ausschnitt)



JDBC: Beispiel 1

- Beispiel: Füge alle Matrikelnummern aus Tabelle ‘Student’ in eine Tabelle ‘Statistik’ ein

```
import java.sql.*;
 ...
public void copyStudents() {
 try {
   Class.forName („COM.ibm.db2.jdbc.net.DB2Driver“);// lade JDBC-Treiber (zur Laufzeit)
 } catch (ClassNotFoundException e) {// Fehlerbehandlung}
 try {
   String url = "jdbc:db2://host:6789/myDB"// spezifiziert JDBC-Treiber, Verbindungsdaten
   Connection con = DriverManager.getConnection(url, "login", "password");
   Statement stmt = con.createStatement();// Ausführen von Queries mit Statement-Objekt
   PreparedStatement pStmt = con.prepareStatement("INSERT INTO statistik (matrikel)
                                                   VALUES (?)");
                                // Prepared-Stmts für wiederholte Ausführung

   ResultSet rs = stmt.executeQuery("SELECT matrikel FROM student");// führe Query aus

   while (rs.next()) {                    // lese die Ergebnisdatensätze aus
      String matrikel = rs.getString(1);  // lese aktuellen Ergebnisdatensatz
      pStmt.setString (1, matrikel);      // setze Parameter der Insert-Anweisung
      pStmt.executeUpdate();              // führe Insert-Operation aus
   }
   con.close();
 } catch (SQLException e) { // Fehlerbehandlung}
}
```


JDBC: Transaktionskontrolle

- **Transaktionskontrolle** durch Methodenaufrufe der Klasse `Connection`
  - `setAutoCommit`: Ein-/Abschalten des Autocommit-Modus (jedes Statement ist eigene Transaktion)
    - Default: true
  - `setReadOnly`: Festlegung ob lesende oder ändernde Transaktion
  - `setTransactionIsolation`: Festlegung der Synchronisationsanforderungen (None, Read Uncommitted, Read Committed, Repeatable Read, Serializable)
  - `commit` bzw. `rollback`: erfolgreiches Transaktionsende bzw. Transaktionsabbruch

- Beispiel

```
try {
   con.setAutoCommit (false);
   // einige Änderungsbefehle, z.B. Inserts
   con.commit ();
} catch (SQLException e) {
   try { con.rollback (); } catch (SQLException e2) {}
} finally {
   try { con.setAutoCommit (true); } catch (SQLException e3) {}
}
```


Gespeicherte Prozeduren in Java

- Erstellen einer Stored Procedure (z. B. als Java-Methode)

```
public static void kontoBuchung(int konto,
                        java.math.BigDecimal betrag,
                           java.math.BigDecimal[] kontostandNeu)
 throws SQLException {
   Connection con = DriverManager.getConnection
                        ("jdbc:default:connection");
                              // Nutzung der aktuellen Verbindung
   PreparedStatement pStmt1 = con.prepareStatement(
          "UPDATE account SET balance = balance + ?
                WHERE account_# = ?");
   pStmt1.setBigDecimal( 1, betrag);
   pStmt1.setInt( 2, konto);
   pStmt1.executeUpdate();
   PreparedStatement pStmt2 = con.prepareStatement(
          "SELECT balance FROM account WHERE account_# = ?");
   pStmt2.setInt( 1, konto);
   ResultSet rs = pStmt2.executeQuery();
   if (rs.next()) kontostandNeu[0] = rs.getBigDecimal(1);
   pStmt1.close(); pStmt2.close(); con.close();
   return;
}
```

Gespeicherte Prozeduren in Java (2)

- Deklaration der Prozedur im Datenbanksystem mittels SQL

```
CREATE PROCEDURE KontoBuchung(   IN konto INTEGER,
                                 IN betrag DECIMAL (15,2),
                              OUT kontostandNeu DECIMAL (15,2))
 LANGUAGE java
 PARAMETER STYLE java
 EXTERNAL NAME 'myjar:KontoClass.kontoBuchung'
                // Java-Archiv myjar enthält Methode
```

Gespeicherte Prozeduren in Java (3)

- Aufruf einer Stored Procedure in Java

```
public void ueberweisung(Connection con, int konto1, int konto2,
java.math.BigDecimal betrag)
throws SQLException {
  con.setAutoCommit (false);
  CallableStatement cStmt = con.prepareCall("{call KontoBuchung (?, ?, ?)}");
  cStmt.registerOutParameter(3, java.sql.Types.DECIMAL);
  cStmt.setInt(1, konto1);
  cStmt.setBigDecimal(2, betrag.negate());
  cStmt.executeUpdate();
  java.math.BigDecimal newBetrag = cStmt.getBigDecimal(3);
  System.out.println("Neuer Betrag: " + konto1 + " " + newBetrag.toString());
  cStmt.setInt(1, konto2);
  cStmt.setBigDecimal(2, betrag);
  cStmt.executeUpdate();
  newBetrag = cStmt.getBigDecimal(3);
  System.out.println("Neuer Betrag: " + konto2 + " " + newBetrag.toString());
  cStmt.close();
  con.commit ();
  return;
}
```


SQLJ

- Eingebettetes SQL (Embedded SQL) für Java
- direkte Einbettung von SQL-Anweisungen in Java-Code
  - Präprozessor um SQLJ-Programme in Java-Quelltext zu transformieren
- Vorteile
  - Syntax- und Typprüfung zur Übersetzungszeit
  - Vor-Übersetzung (Performance)
  - einfacher/kompakter als JDBC
  - streng typisierte Iteratoren (Cursor-Konzept)

SQLJ (2)

- eingebettete SQL-Anweisungen: #sql [[<context>]] { <SQL-Anweisung> }
  - beginnen mit **#sql** und können mehrere Zeilen umfassen
  - können Default-Verbindung oder explizite Verbindung verwenden
  - können Variablen der Programmiersprache (:x) bzw. Ausdrücke (:y + :z) enthalten

- Vergleich SQLJ – JDBC (1-Tupel-Select)

SQLJ

```
#sql [con]{ SELECT name INTO :name
            FROM student WHERE matrikel = :mat};
```

JDBC

```
java.sql.PreparedStatement ps =
      con.prepareStatement („SELECT name “+
            „FROM student WHERE matrikel = ?“);
ps.setString (1, mat);
java.sql.ResultSet rs = ps.executeQuery();
rs.next()
name= rs.getString(1);
rs.close;
```

SQLJ (3)

- Iteratoren zur Realisierung eines Cursor-Konzepts
  - eigene Iterator-Klassen
  - benannte Iteratoren: Zugriff auf Spalten des Ergebnisses über Methode mit dem Spaltennamen
  - Positionsiteratoren: Iteratordefinition nur mit Datentypen; Ergebnisabruf mit *FETCH*-Anweisung ; *endFetch()* zeigt an, ob Ende der Ergebnismenge erreicht
- Vorgehensweise (benannte Iteratoren)
  1. Definition Iterator-Klasse

```
#sql public iterator IK (String a1, String a2);
```

  2. Zuweisung mengenwertiges Select-Ergebnis an Iterator-Objekt

```
IK io;
#sql io = { SELECT a1, a2 FROM …};
```

  3. Satzweiser Abruf der Ergebnisse

```
while  io.next() {  …;
     String s1 = io.a1();
     String s2 = io.a2(); … }
```

  4. Schließen Iterator-Objekt

```
io.close();
```

SQLJ (4)

- Beispiel: Füge alle Matrikelnummern aus Tabelle ‘Student’ in eine Tabelle ‘Statistik’ ein.

```
import java.sql.*;
import sqlj.runtime.ref.DefaultContext;
...
public void copyStudents() {
String drvClass=„…“;
Class.forName(drvClass);

String url = “jdbc:db2://host:6789/myDB”
Connection con=DriverManager.getConnection
        (url, “login”, “password”);

  // erzeuge Verbindungskontext
DefaultContext ctx =
           new DefaultContext(con);
  // definiere Kontext als Standard-Kontext
DefaultContext.setDefaultContext(ctx);
```

```
// 1. deklariere Iterator-Klasse
#sql public iterator
        MatrikelIter (String matrikel);

// 2. erzeuge Iterator-Objekt;
 // Zuordnung und Aufruf der SQL-Anfrage
MatrikelIter mIter;
#sql mIter=
    {SELECT matrikel FROM student};

// 3. Abruf Ergebniswerte
while (mIter.next()) {
 #sql {INSERT INTO statistik (matrikel)
     VALUES ( mIter.matrikel()) };
 }
// 4. Schließe Iterator
mIter.close();
```

DB-Anbindung mit Skriptsprachen

- grundlegendes Vorgehen
  1. Aufbau einer Verbindung zum Datenbank-Server
  2. Auswahl einer Datenbank
  3. Interaktion mit Datenbank über SQL
  4. Verarbeitung und Präsentation von Ergebnissen

- DB-Anbindungsmöglichkeiten am Beispiel von PHP
  - spezielle Module je nach DBS: MySQL, MS SQL, PostgreSQL, ...
  - allgemeinere Bibliotheken/Module für DBS-unabhängigen Zugriff
    - Beispiele: **PDO**, PEAR::DB
    - Vorteil: DBS kann ausgetauscht werden, Implementierung bleibt
  - einfache Realisierung von Web-Anwendungen

PHP – Variablen und Arrays

- Variablen und Datentypen
  - Variablennamen beginnen mit **$:** *$name*, *$address*, *$city*, *$zip_code*
  - keine Deklaration von Variablen
  - Variablentyp wird automatisch zugewiesen / angepasst

    $name = “Ernie“; (String)
    $zip_code = 04103; (Integer → ganze Zahl)
    $price = 1.99; (Double → Gleitkommazahl)

- Arrays
  - indexiert: Zugriff auf Inhalt über numerischen Index

    $zahlen = array(1, 2, 3, 4);
    $zahlen[3] = 0;
  - assoziativ: Zugriff über Stringschlüssel

    $address[“street“] = “Augustusplatz 10“;
    $address[“zip_code“] = 04109;
    $address[“city“] = “Leipzig“;
  - Kombination möglich (mehrdimensionale Arrays), z.B. *$addresses*[4][“zip_code“]

PHP – DB Anbindung (Beispiele)

- MySQL-spezifisch

```
$con = mysqli_connect(“host“, “user“, “password“);
mysqli_select_db($con, “myDB“);

$result = mysqli_query($con, “SELECT * FROM Employees WHERE EmployeeID = 1“);
...
```

- generisch via PDO (PHP Data Objects)

```
$module = “mysql“;          //$module = “mssql“;
$con = new PDO (“$module:host=myHost; dbname=myDB“, “user“, “password“);

$result = $con->query (“SELECT * FROM Employees WHERE EmployeeID = 1“);
...
```

Realisierung von Webanwendungen

- PHP-Code kann direkt in Webseiten eingebettet werden
  - PHP Server generiert dynamisch HTML
  - Übergabe an den Web-Server → Weiterleitung an Web-Browser
  - besonders effizient bei der Erzeugung und Auswertung von HTML-Formularen

- Beispiel

<?php echo "Diesen Text bitte fett darstellen !"; ?>

generiert →

Diesen Text bitte fett darstellen !

Browser interpretiert /zeigt →

Diesen Text bitte fett darstellen !

PHP Web-Beispiel (1)

- Beispiel: webbasierte Verwaltung von Mitarbeitern
- 1. Anwendungsfall: Einfügen eines neuen Mitarbeiters

Eingabemaske

```
<form method="post" action="insert_employee.php">
    Name:<br/>
    <input type="text" name="full_name" /><br/>

    Address:<br/>
    <input type="text" name="address" /><br/>

    Zip Code:<br/>
    <input type="text" name="zip_code" /><br/>

    City:<br/>
    <input type="text" name="city" /><br/>

    <input type="submit" name="add_employee" value="Add employee"/>
</form>
```

PHP Web-Beispiel (2)

- Verarbeitung der Daten in PHP
  - Variablen aus Formular sind über globales assoziatives PHP-Array $_POST verfügbar:
    <input type='text' name='city'/> + Manhattan/New York
    → $_POST['city'] enthält Wert „Manhattan/New York"
  - Verbindung zum DB-Server erstellen
  - Anwendungsfall abarbeiten

```
<?php
  $con = new PDO("$mysql:host=myHost; dbname=myDB", "user", "password");


  if(isset($_POST["add_employee"])) {
    $full_name= $_POST["full_name"];
    $address = $_POST['address"];
    $zip_code = $_POST["zip_code"];
    $city= $_POST["city"];

    $sql = "INSERT INTO Employees(Name, Address, ZipCode, City)
             VALUES($full_name, $address, $zip_code, $city)";

    $result = $con->exec($sql);
    if ($result == 0 ) {
        echo "Error adding new employee !";
    } else {
        echo "New employee sucessfully inserted !";
    }
  }
?>
```

PHP Web-Beispiel (3)

- 2. Anwendungsfall: Auflisten aller Mitarbeiter (mit Prepared Statement)

```
<html>
  <body>
<?php
    $con = new PDO("$mysql:host=myHost; dbname=myDB", "user", "password");

    $sql= "SELECT * FROM Employees WHERE CITY LIKE ?";
    $stmt= con->prepare($sql);
    $stmt->execute(array("%New York%"));

    $rows= $stmt->fetchAll(PDO::FETCH_ASSOC);
    foreach($rows as $row) {
?>
        <h2>Employee # <?php echo $row["EmployeeID"]?></h2><br/><br/>
        <b>Name:</b>   <?php echo $row["Name"]?> <br/>
        <b>Address:</b><?php echo $row ["Address"]?> <br/>
        <b>ZIP/City:</b><?php echo $row["ZipCode"] . "," . $row["City"]?><br/><br/>
<?php
    }
?>
  </body>
</html>
```

Employee #4711

Name: Ernie
Address: Sesame Street 123
ZIP/City: 10123 Manhattan/New York

Employee #4712

Name: Bert
Address: Sesame Street 125
ZIP/City: 10123 Manhattan/New York

SQL Injection

- Dynamische SQL-Statements können für unberechtigte Zugriffe missbraucht werden
- Beispiel-Aufruf
  - http://company.com/employee.php?ID=1
- SQL-Anweisung ist Zeichenkette, die ggf. manipuliert werden kann
  - Verallgemeinerung der Where-Klausel, um Daten auszuspähen
  - Verkettung mehrerer Statements, z.B. um Daten zu ändern/löschen
- mögliche Abhilfen
  - Nutzung 2-stufiger SQL-Aufrufe mit Vorbereiten von (parametrisierten) Statements fester Struktur
  - Vergabe restriktiver Zugriffsrechte zum Ausschluss von Änderungen oder Zugriff auf sensitive Attribute (Kontoangaben etc.)

JDBC-Beispiel Gehaltsänderung

- Verwendung eines Eingabeparameters

```
import java.sql.*;
 ...
public void main (string[] args){
 try {
   Class.forName („COM.ibm.db2.jdbc.net.DB2Driver“);// lade JDBC-Treiber (zur Laufzeit)
 } catch (ClassNotFoundException e) {// Fehlerbehandlung}
 try {
   String url = "jdbc:db2://host:6789/myDB2"// spezifiziert JDBC-Treiber, Verbindungsdaten
   Connection con = DriverManager.getConnection(url, "login", "password");

   Statement stmt = con.createStatement();

   boolean success = stmt.execute ("UPDATE PERS SET Gehalt=Gehalt*2.0 WHERE PNR="+args[0])

   con.close();
 } catch (SQLException e) { // Fehlerbehandlung}
}
```

args[0]=„35"

PNR	NAME	GEHALT
34	Mey	32000
35	Schultz	42500
37	Abel	41000

→

PNR	NAME	GEHALT
34	Mey	32000
35	Schultz	**85000**
37	Abel	41000

Bsp: Parametrisierung ermöglicht Manipulation

- Verwendung eines Eingabeparameters

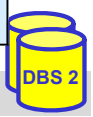
```
import java.sql.*;
 ...
public void main (string[] args){
 try {
   Class.forName („COM.ibm.db2.jdbc.net.DB2Driver“);// lade JDBC-Treiber (zur Laufzeit)
 } catch (ClassNotFoundException e) {// Fehlerbehandlung}
 try {
   String url = "jdbc:db2://host:6789/myDB2"// spezifiziert JDBC-Treiber, Verbindungsdaten
   Connection con = DriverManager.getConnection(url, "login", "password");

   Statement stmt = con.createStatement();

   boolean success = stmt.execute ("UPDATE PERS SET Gehalt=Gehalt*2.0 WHERE PNR="+args[0])

   con.close();
 } catch (SQLException e) { // Fehlerbehandlung}
}
```

args[0]=
„35 OR Gehalt<100000"

PNR	NAME	GEHALT
34	Mey	32000
35	Schultz	42500
37	Abel	41000

→

PNR	NAME	GEHALT
34	Mey	
35	Schultz	
37	Abel	

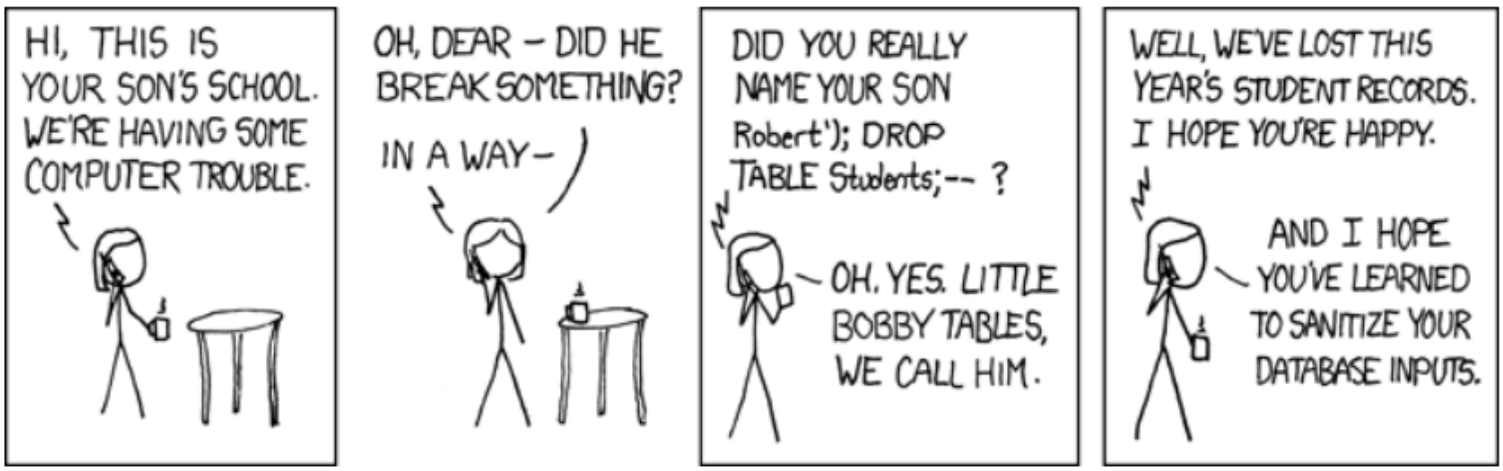


Bsp.: Abhilfemöglichkeit

- Nutzung von Prepared-Statement ermöglicht Abhilfe

```
import java.sql.*;
 ...
public void main (string[] args){
 try {
   Class.forName („COM.ibm.db2.jdbc.net.DB2Driver“);// lade JDBC-Treiber (zur Laufzeit)
 } catch (ClassNotFoundException e) {// Fehlerbehandlung}
 try {
   String url = "jdbc:db2://host:6789/myDB2"// spezifiziert JDBC-Treiber, Verbindungsdaten
   Connection con = DriverManager.getConnection(url, "login", "password");

        PreparedStatement pStmt = con.prepareStatement("UPDATE PERS
                                                SET Gehalt=Gehalt*2.0 WHERE PNR=?"
        pStmt.setInt (1, 35);    // setze Parameter der Update-Anweisung
        pStmt.executeUpdate();

        pStmt.setString (1, args[0]);
        pStmt.executeUpdate();

   con.close();
 } catch (SQLException e) { // Fehlerbehandlung}
}
```

args[0]= „35 OR Gehalt<100000"



URL: https://xkcd.com/327/

Zusammenfassung

- JDBC: Standardansatz für DB-Zugriff mit Java
  - Call-Level-Interface (erfordert keinen Präcompiler)
  - Verwendung von dynamischem SQL
  - Transaktionen mit mehreren DB-Operationen erfordern Abschalten von Autocommit
- SQLJ: Embedded SQL für Java
  - Nutzung von Iteratoren
- DB-Zugriff in Skriptsprachen wie PHP ähnelt CLI
  - flexible und schnelle Realsierungen
  - einfache Erstellung von Webanwendungen
- SQL-Injections zu verhindern
  - Nutzung von Prepared-Statements mit Parametern