Datenbankanwendung

Wintersemester 2014/15

Prof. Dr.-Ing. Sebastian Michel
TU Kaiserslautern

smichel@cs.uni-kl.de

Anmerkungen/Ankündigungen

1. Bemerkung/Klarstellung zur automatischen Erkennung von Endlosrekursion in Folien von Vorlesung 9.
2. Der Zugang zu den Musterlösungen ist nur noch Uni-intern oder via Login möglich. Die Zugangsdaten sind identisch zu den Zugangsdaten für die Beispieldatenbanken.

Der Relationenkalkül

- Bisher: Relationale Algebra, prozedural
- Jetzt: Relationenkalkül, deklarativ
- beide sind gleich mächtig, wenn Relationenkalkül auf sichere Ausdrücke beschränkt ist
- d.h. alle Ausdrücke der relationalen Algebra können auch im sicheren Relationenkalkül ausgedrückt werden und umgekehrt.
- Zwei Varianten:
  - relationaler Tupelkalkül: hatte großen Einfluss auf SQL
  - relationaler Domänenkalkül: hatte großen Einfluss auf "Query-by Example" (QBE)

Der relationale Tupelkalkül

Eine Anfrage im relationalen Tupelkalkül hat die Form

$$\{t|P(t)\}$$

wobei t eine Tupelvariable ist und $P(t)$ ein Prädikat.

$P(t)$ muss erfüllt sein damit t Teil des Ergebnis ist.

Auch in Kombination mit Tupelkonstruktor:

$$\{[t_1.A_1, ..., t_n.A_n]|P(t_1, ..., t_n)\}$$

Beispiele

- C4-Professoren:

$$\{p|p \in Professoren \wedge p.Rang = C4\}$$

- Paare von Professoren (Name) und Assistenten (PersNr):

$$\{[p.Name, a.PersNr]|p \in Professoren \wedge a \in Assistenten$$
$$\wedge p.PersNr = a.Boss\}$$

- Studenten mit mindestens einer Vorlesung von Prof. Curie:

$$\{s|s \in Studenten$$
$$\wedge \exists h \in h\ddot{o}ren(s.MatrNr = h.MatrNr$$
$$\wedge \exists v \in Vorlesungen(h.VorlNr = v.VorlNr$$
$$\wedge \exists p \in Professoren(p.PersNr = v.gelesenVon$$
$$\wedge p.Name = \text{`}Curie\text{`})))\}$$

.... in SQL ... ist es sehr ähnlich:

Studenten mit mindestens einer Vorlesung von Prof. Curie:

```
SELECT s.*
from Studenten s
where exists (
    select h.*
    from hören h
    where h.MatrNr=s.MatrNr and exists (
        select *
        from Vorlesungen v
        where v.VorlNr=h.VorlNr and exists (
            select *
            from Professoren p
            where p.Name='Curie' and p.PersNr=v.gelesenVon)))
```

Allquantor

Wer hat **alle** vierstündigen Vorlesungen gehört?

$$\{s|s \in Studenten \wedge \forall v \in Vorlesungen(v.SWS = 4 \Rightarrow$$
$$\exists h \in hoeren(h.VorlNr = v.VorlNr \wedge h.MatrNr = s.MatrNr))\}$$

Definition des Tupelkalküls

Atome

- $s \in R$, mit s Tupelvariable und R Relationenname
- $s.A \phi t.B$, mit s und t Tupelvariablen, A und B Attributnamen und ϕ Vergleichoperator $(=, \neq, \leq, ...)$
- $s.A \phi c$ mit Konstante c

Formeln

- Alle Atome sind Formeln
- Ist P Formel, so auch $\neg P$ und (P)
- Sind P_1 und P_2 Formeln, so auch $P_1 \wedge P_2$, $P_1 \vee P_2$ und $P_1 \Rightarrow P_2$
- Ist $P(t)$ eine Formel mit freier Variable t, so auch $\forall t \in R(P(t))$ und $\exists t \in R(P(t))$

Sicherheit

- Einschränkung auf Anfragen mit **endlichem** Ergebnis.
- Zum Beispiel ist die Anfrage

$$\{n|\neg(n \in Professoren)\}$$

  nicht sicher.
- Das Ergebnis ist **unendlich**.
- Bedingung: Ergebnis des Ausdrucks muss Teilmenge der Domäne der Formel sein.
- Die **Domäne** einer Formel enthält
  - alle in der Formel vorkommenden Konstanten
  - alle Attributwerte von Relationen, die in der Formel referenziert werden

Der relationale Domänenkalkül

Der Ausdruck des relationalen Domänenkalküls hat die Form

$$\{[v_1, v_2, ..., v_n] | P(v_1, v_2, ..., v_n)\}$$

mit $v_1, ..., v_n$ Domänenvariablen und P ein Prädikat (oder Formel).

Beispiel:

Matrikelnummer und Namen der Prüflinge von Prof. Russel:

$$\begin{aligned}\{[m, n] | \exists s([m, n, s] \in Studenten \\ \wedge \exists v, p, g([m, v, p, g] \in pruefen \\ \wedge \exists a, r, b([p, a, r, b] \in Professoren \\ \wedge a =' Russel')))\}\end{aligned}$$

Hinweis zu freien Variablen und Quantifizierung, hier und im Tupelkalkül

$$\{[m,n]|\exists s([m,n,s] \in Studenten$$
$$\wedge \exists v,p,g([m,v,p,g] \in pruefen$$
$$\wedge \exists a,r,b([p,a,r,b] \in Professoren$$
$$\wedge a =' Russel')))\}$$

- Wie wir hier sehen wird im innersten Quantor eine Variable a für den Namen eines Professors benutzt.
- Man hätte auch in diesem speziellen Fall den Variablennamen n nochmal in dem Existenzquantor benutzen können, der Logik wegen, da hier kein Konflikt mit dem äußeren "n" des Studentennamens besteht (weil der Scope diesem eine neue Rolle zuteilt)
- Aber ohne dieses neuen Bindens von n und dann mit $\ldots \wedge n =' Russel')$ wäre hier die Suche auf Studenten eingeschränkt worden, die auch $'Russel'$ heißen.

Regel: Vermeiden Sie daher mehrfache Verwendung des gleichen (freien bzw. gebundenen) Parameternamens in unterschiedlichen Scopes

Sicherheit des Domänenkalküls

- Sicherheit ist analog zum Tupelkalkül
- zum Beispiel ist

$$\{[p, n, r, o] | \neg([p, n, r, o] \in Professoren)\}$$

  nicht sicher.
- Ein Ausdruck

$$\{[x_1, x_2, ..., x_n] | P(x_1, x_2, ..., x_3)\}$$

  ist **sicher**, falls folgende drei Bedingungen gelten:

$\{[x_1, x_2, ..., x_n] | P(x_1, x_2, ..., x_3)\}$ sicher falls ....

1. Falls Tupel $[c_1, c_2, ..., c_n]$ mit Konstanten c_i im Ergebnis enthalten ist, so muss jedes c_i $(1 \leq i \leq n)$ in der Domäne von P enthalten sein.
2. Für jede existenz-quantifizierte Teilformel $\exists x(P_1(x))$ muss gelten, dass P_1 nur für Elemente aus der Domäne erfüllbar sein kann - oder evtl. für gar keine.
   Das heisst: Wenn für eine Konstante c das Prädikat $P_1(c)$ erfüllt ist, so muss c in der Domäne von P_1 enthalten sein.
3. Für jede universal-quantifizierte Teilformel $\forall x(P_1(x))$ muss gelten, dass sie dann und nur dann erfüllt ist, wenn $P_1(x)$ für alle Werte der Domäne von P_1 erfüllt ist.
   Das heisst: $P_1(d)$ muss für alle d, die nicht in der Domäne von P_1 enthalten sind, auf jeden Fall erfüllt sein.

Ausdruckskraft

Die drei Sprachen

- relationale Algebra,
- relationaler Tupelkalkül, eingeschränkt auf sichere Ausdrücke
- und relationaler Domänenkalkül, eingeschränkt auf sichere Ausdrücke

... sind **gleich mächtig!**

Datenbank-Zugriff via JDBC

Java Database Connectivity

- bietet Schnittstelle für den Zugriff auf ein DBMS aus Java-Anwendungen

JDBC: Connect und einfache Anfrage

```
1  //registriere geeigneten Treiber (hier fuer Postgresql)
2  Class.forName("org.postgresql.Driver");
3  //erzeuge Verbindung zur Datenbank
4  Connection conn = DriverManager.getConnection(
5       "jdbc:postgresql://localhost/university",
6       "username", "password");
7
8  //erzeuge ein einfaches Statement Objekt
9  Statement stmt = conn.createStatement();
10
11 //mit execute Query koennen nun darauf Anfragen ausgefuehrt
       werden
12 //Ergebnisse in Form eines ResultSet Objekts
13 ResultSet rset = stmt.executeQuery("SELECT p.persnr from
       professoren p");
```

JDBC: Connect und einfache Anfrage

```
14 //dieses besitzt Metadaten
15 ResultSetMetaData metadata = rset.getMetaData();
16
17 //welche Attribute (Spalten) besitzen die Ergebnis-Tupel?
18 int column_count = metadata.getColumnCount();
19
20 for (int index=1; index<=column_count; index++) {
21       System.out.println("Spalte "+index+" heisst " +
22             metadata.getColumnName(index));
23 }
24
25 //iteriere nun ueber Ergebnisse
26 while (rset.next()) {
27       System.out.println(rset.getString(1));
28 }
```

JDBC Treiber für Postgresql

`http://jdbc.postgresql.org/`

JDBC - wichtige Funktionalitäten

Laden des Treibers

- Kann auf verschiedene Weise erfolgen, z.B. durch explizites Laden mit dem Klassenlader:

```
Class.forName(DriverClassName);
```

Aufbau einer Verbindung

- Connection-Objekt repräsentiert die Verbindung zum DB-Server
- Beim Aufbau werden URL der DB, Benutzername und Passwort aus Strings übergeben (teilweise optional).

```
Connection conn = DriverManager.getConnection(url,
                                              login, password);
```

JDBC - wichtige Funktionalitäten (2)

Anweisungen

- Mit dem Connection-Objekt könne u.a. Metadaten der DB erfragt und Statement-Objekte zum Absetzen von SQL-Anweisngen erzeugt werden
- Erzeugen einer SQL-Anweisung zur direkten (einmaligen) Ausführung

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

- PreparedStatement-Objekt erlaubt das Erzeugen und Vorbereiten von (parametrisierten) SQL-Anweisungen zur wiederholten Ausführung

```
PreparedStatement pstmt = conn.prepareStatement(
        "select * from personal where gehalt >= ?");
```

Schließen von Verbindungen, Statements, usw.

```
stmt.close();
conn.close();
```

JDBC - Anweisungen

Anweisungen (Statements)

- Werden in einem Schritt vorbereitet und ausgeführt

Die Methode executeQuery

- führt die Anfrage aus und liefert Ergebnis zurück

```
Statement stmt = conn.createStatement();
ResultSet rset = stmt.executeQuery(
        "select pnr, name, gehalt
         from personal where gehalt>=40000");
//wir sehen gleich, wie man mit diesem ResultSet arbeitet
```

JDBC - Anweisungen

Die Methode executeUpdate

- werden zur direkten Ausführung von UPDATE-, INSERT-, DELETE- und DDL-Anweisungen benutzt

```
Statement stmt = conn.createStatement();
int n = stmt.executeUpdate(
      "update personal
       set gehalt = gehalt * 1.10
       where gehalt < 20000");

//n enthaelt die Anzahl der aktualisierten Zeilen
```

JDBC - Prepared Anweisungen

PreparedStatement-Objekt

```
PreparedStatement pstmt;
double gehalt = 50000.00;
pstmt = conn.prepareStatement(
    "select * from personal where gehalt >= ?");
```

- Symbol ? markiert hier freie Parameter
- Vor der Ausführung sind dann die Parameter einzusetzen.
- Durch Methoden entsprechend Datentyp, z.B.

```
pstmt.setDouble(1, gehalt);
```

https://docs.oracle.com/javase/8/docs/api/java/sql/PreparedStatement.html

JDBC - Prepared Anweisungen (2)

Ausführen einer Prepared-Anweisung als Anfrage

```
PreparedStatement pstmt;
double gehalt = 50000.00;
pstmt = conn.prepareStatement(
    "select * from personal where gehalt >= ?");
```

Vorbereitung und Ausführung

```
pstmt = con.prepareStatement(
        "delete from personal where name = ?");
pstmt.setString(1, "Maier");

int n = pstmt.executeUpdate();

//Methoden der Prepared-Anweisungen haben keine Argumente
```

JDBC - Ergebnismengen und Cursor

Select-Anfragen und Ergebnisübergabe

- Jede JDBC-Methode, mit der man Anfragen an das DBMS stellen kann, liefert ResultSet-Objekte als Rückgabewert

```
ResultSet rset = stmt.executeQuery(
        "select pnr, name, gehalt
        from personal where gehalt >= " + gehalt);
```

- Cursor-Zugriff und Konvertierung der DBMS-Datentypen in passende Java-Datentypen erforderlich
- JDBC-Cursor ist durch die Methode **next()** der Klasse ResultSet implementiert

`https://docs.oracle.com/javase/8/docs/api/java/sql/ResultSet.html`

JDBC - Ergebnismengen und Cursor (2)

Cursor → getInt("pnr") ↓ | getString("name") ↓ | getDouble("gehalt") ↓

↓ next()

123	Maier	23352.00
456	Schulze	34553.00

Zugriff aus Java-Programm

```
while (rset.next()) {
    System.out.print(res.getInt("pnr")+"\t");
    System.out.print(res.getString("name")+"\t");
    System.out.println(res.getString("gehalt"));
}
```

JDBC - Versch. Typen von ResultSets

TYPE_FORWARD_ONLY

- nur Aufruf von **next()** möglich

TYPE_SCROLL_INSENSITIVE

- Scroll-Operationen sind möglich, aber Aktualisierungen der Datenbank verändern ResultSet nach seiner Erstellung nicht

JDBC - Versch. Typen von ResultSets (2)

TYPE_SCROLL_SENSITIVE

- Scroll-Operationen möglich und Änderungen in der Datenbank werden berücksichtigt

ResultSet lässt Änderungen zu oder nicht:

- CONCUR_READ_ONLY
- CONCUR_UPDATABLE

```
Statement stmt = conn.createStatement(
        ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rset = stmt.executeQuery(...);
rset.updateString("name", "Schmitt");
rset.updateRow();
```

JDBC - Zugriff auf Metadaten

Allgemeine Metadaten

- Klasse **DatabaseMetaData** zum Abfragen von DB-Informationen

Informationen über ResultSets

- JDBC bietet die Klasse **ResultSetMetaData**

```
ResultSets rset = stmt.executeQuery("select ...");
ResultSetMetaData rsmd = rset.getMetaData();
```

- Abfragen von Spaltenanzahl, Spaltennamen und deren Typen

```
int anzahlSpalten = rsmd.getColumnCount();
String spaltenName = rsmd.getColumnName(1);
String typeName = rsmd.getColumnTypeName(1);
```

JDBC - Fehlerbehandlung

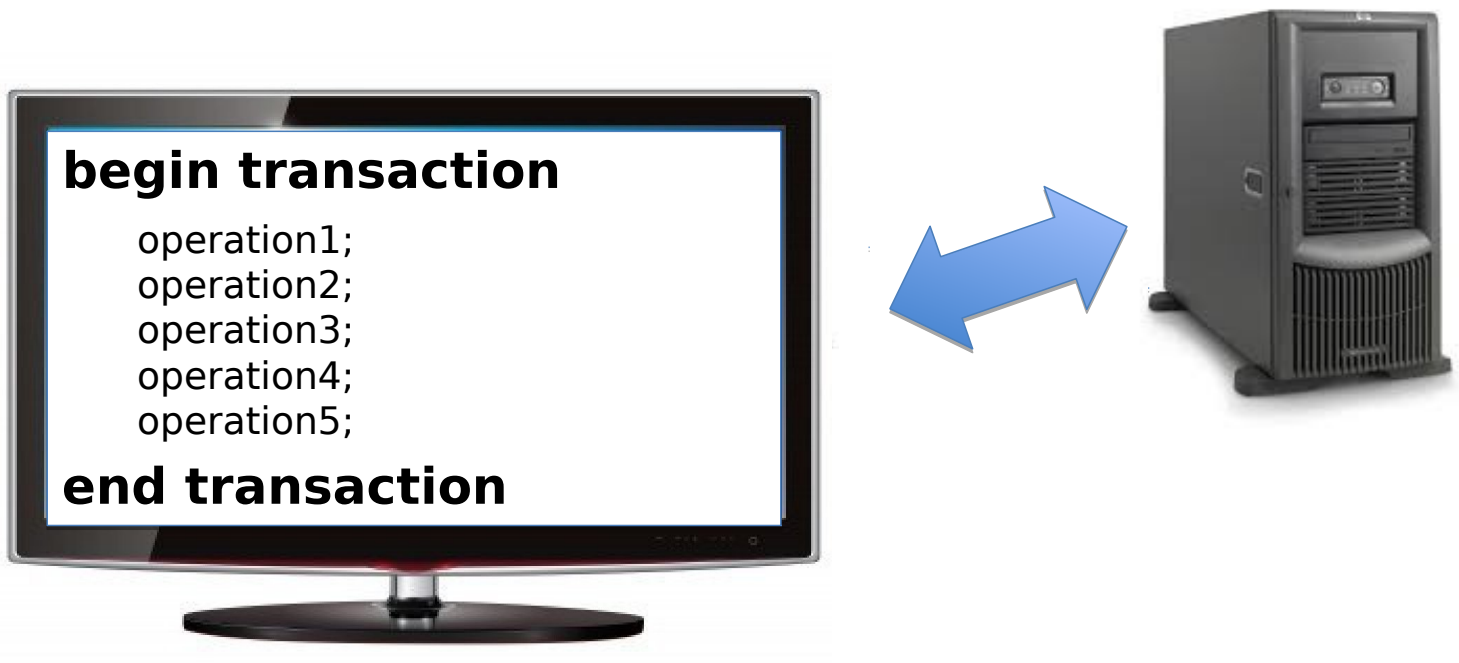
SQLException:

- Spezifikation der Ausnahmen, die eine Methode werfen kann, steht bei ihrer Deklaration (throws Exception)
- Wird Code in einem try-Block ausgeführt, werden im catch-Block Ausnahmen abgefangen.

```
try {
    //code ......
}
catch (SQLException e) {
  System.out.println("Es ist ein Fehler aufgetreten:");
  System.out.println("Msg: "+ e.getMessage());
  System.out.println("SQLState: "+ e.getSQLState());
  System.out.println("ErrorCode: "+ e.getErrorCode());
  //und zum debuggen noch gleich dazu
  e.printStackTrace();
}
```

Anwendungsprogrammierung: Transaktionen

- Nicht nur eine einzelne SQL-Anweisung, sondern ganze Folge davon, je nach Anwendung.
- Eine oder mehrere Anweisungen werden als Transaktion zusammengefasst bzw. betrachtet. Z.B. Abheben von Geld am Geldautomat.



Wiederholung: Transaktion - Klassisches Beispiel

Bei einer typischen Transaktion in einer Bankanwendung:

1. Lese den Kontostand von A in die Variable a: $read(A, a)$;
2. Reduziere den Kontostand um 50 Euro: $a := a - 50$;
3. Schreibe den neuen Kontostand in die Datenbasis: write(A,a);
4. Lese den Kontostand von B in die Variable b: $read(B, b)$;
5. Erhöhe den Kontostand um 50 Euro: $b := b + 50$;
6. Schreibe den neuen Kontostand in die Datenbasis: $write(B, b)$;

Wiederholung: Transaktionen - ACID

Atomicity (Atomarität)

- Alles oder nichts

Consistency

- Konsistenter Zustand der DB $\rightarrow$ konsistenter Zustand

Isolation

- Jede Transaktion hat die DB "für sich allein"

Durability (Dauerhaftigkeit)

- Änderungen erfolgreicher Transaktionen dürfen nie verloren gehen.

Operationen auf Transaktione-Ebene

- **begin of transaction (BOT):** Mit diesem Befehl wird der Beginn einer eine Transaktion darstellende Befehlsfolge gekennzeichnet. Ist implizit, bei Beginn der Befehlssequenz.
- **commit:** Hierdurch wird die Beendigung der Transaktion eingeleitet. Alle Änderungen der Datenbasis werden durch diesen Befehl festgeschrieben, d.h. sie werden dauerhaft in die Datenbank eingebaut.
- **abort:** Dieser Befehl führt zu einem Selbstabbruch der Transaktion. Das Datenbanksystem muss sicherstellen, dass die Datenbasis wieder in den Zustand zurückgesetzt wird, der vor Beginn der Transaktionsausführung existierte.

Wiederholung: Transaktionsverwaltung in SQL

- **commit [work]:** Die in der Transaktion vollzogenen Änderungen werden falls keine Konsistenzverletzung oder andere Probleme aufgedeckt werden festgeschrieben. Das Schlüsselwort work ist optional, d.h. das Transaktionsende kann auch einfach mit commit "befohlen" werden.
- **rollback [work]:** Alle Änderungen sollen zurückgesetzt werden. Anders als der commit-Befehl muss das DBMS die "erfolgreiche" Ausführung eines rollback-Befehls immer garantieren können.

Wiederholung: Der "non-repeatable read" Fehler

Abhängigkeit von anderen Updates (**non-repeatable read**)

Transaktion T_1	Transaktion T_2
	select sum(Kontostand) **from** Konten
update Konten **set** Kontostand=42000 **where** kontoId=12345	
	select sum(Kontostand) **from** Konten

Wiederholung: Das "Phantomproblem"

Abhängigkeit von neuen/gelöschten Tupeln (**Phantomproblem**)

Transaktion T_1	Transaktion T_2
	select sum(Kontostand) **from** Konten
insert into Konten **values** (C,1000,...)	
	select sum(Kontostand) **from** Konten

JDBC - Transaktionen

Transaktionen

- Bei der Erzeugung eines Connection-Objekts ist (in der Regel) als Default der Modus **autocommit** eingestellt. D.h. nach jeder Aktion wird ein Commit ausgeführt.
- Um Transaktionen als Folgen von Anweisungen abwickeln zu können, ist dieser Modus auszuschalten.

```
conn.setAutoCommit(false);
```

- Für eine Transaktion können sogenannte Konsistenzstufen (isolation levels) wie TRANSACTION_SERIALIZEABLE, TRANSACTION_REPEATABLE_READ usw. eingestellt werden.

```
conn.setTransactionIsolation(
    Connection.TRANSACTION_SERIALIZABLE);
```

JDBC - Transaktionen (2)

Beendigung oder Zurücksetzung

```
conn.commit();
```

bzw.

```
conn.rollback();      //oder: conn.rollback(savepoint)
```

Sicherungspunkte (Savepoints)

```
Savepoint sp = conn.setSavepoint();  //bzw. mit Namen
Savepoint namedSp = conn.setSavepoint("mySavePoint");
```

Programm kann mit mehreren DBMS verbunden sein

- Selektives Beenden/Zurücksetzen von Transaktionen pro DBMS
- Kein global atomares Commit möglich

JDBC - Transaktionen: Beispiel

http://www.tutorialspoint.com/jdbc/jdbc-transactions.htm

```
try{
   //Assume a valid connection object conn
   conn.setAutoCommit(false);
   Statement stmt = conn.createStatement();
   String SQL = "INSERT INTO Employees  " +
           "VALUES (106, 20, 'Rita', 'Tez')";
   stmt.executeUpdate(SQL);
   //Submit a malformed SQL statement that breaks
   String SQL = "INSERTED IN Employees  " +
                "VALUES (107, 22,  'Sita', 'Singh')";

   stmt.executeUpdate(SQL);
   // If there is no error.
   conn.commit();
}catch(SQLException se){
   // If there is any error.
   conn.rollback();
}
```

JDBC - Transaktionen: Konsistenzsstufen

Fields

Modifier and Type	Field and Description
static int	**TRANSACTION_NONE** A constant indicating that transactions are not supported.
static int	**TRANSACTION_READ_COMMITTED** A constant indicating that dirty reads are prevented; non-repeatable reads and phantom reads can occur.
static int	**TRANSACTION_READ_UNCOMMITTED** A constant indicating that dirty reads, non-repeatable reads and phantom reads can occur.
static int	**TRANSACTION_REPEATABLE_READ** A constant indicating that dirty reads and non-repeatable reads are prevented; phantom reads can occur.
static int	**TRANSACTION_SERIALIZABLE** A constant indicating that dirty reads, non-repeatable reads and phantom reads are prevented.

Default in Postgresql-JDBC ist TRANSACTION_READ_COMMITTED

`http://docs.oracle.com/javase/8/docs/api/java/sql/Connection.html`

Anmerkung: SQL Injections

SQL Anfragen wird in Anwendung erstellt, wobei id eine Benutzereingabe ist

```
....”SELECT author, subject, text ” +
      ”FROM artikel WHERE ID=” + id
```

Aufruf z.B. durch Webserver `http://webserver/cgi-bin/find.cgi?ID=42`

SQL Injection zum Ausspähen von Daten

`http://webserver/cgi-bin/find.cgi?ID=42+UNION+SELECT+`
`login,+password,+’x’+FROM+user`

Führt zur SQL Anweisung:

select author, subject, text **from** artikel
where ID=42 **union select** login, password, ’x’ **from** user;

Und andere Fälle bis hin zum Einschleußen von beliebigen Code auf Rechner + öffnen einer Shell (abh. von DBMS)

Hilfe u.a. durch Benutzen von PreparedStatements

Übersicht unter: `http://de.wikipedia.org/wiki/SQL-Injection`