TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

# Übung zur Vorlesung *Einsatz und Realisierung von Datenbanken* im SoSe23

Alice Rey, Maximilian Bandle, Michael Jungmair (i3erdb@in.tum.de)
http://db.in.tum.de/teaching/ss23/impldb/

**Blatt Nr. 03**

**Hinweise** Die Datalogaufgaben können auf `https://souffle.db.in.tum.de/` getestet werden. Auf der Seite kann unter *examples* ein entsprechender Datensatz geladen werden. Die neuen IDB Regeln sollten am Ende der EDB definiert und dann im Query-Eingabefeld abgefragt werden.

Zusätzlich zu der in der Vorlesung vorgestellten Syntax hier noch eine Kurzübersicht der Vergleichsoperatoren: $X < Y, Y > X$ (kleiner, größer), $X <= Y, X >= Y$ (kleiner oder gleich, größer oder gleich), $X = Y, X! = Y$ (gleich, ungleich), $!pred(X, Y)$ (existiert nicht pred(X,Y)).

## Hausaufgabe 1

Die AMU (Alexander-Maximilians-Universität) hat eine Datenbank mit den Durchschnittsnoten aller Studenten mit Name.

Schema: {[Name | Durchschnittsnote]}

Der einzige Schutzmechanismus dieser Datenbank ist, dass immer mindestens 3 Tupel aggregiert werden. Als Ausgabe sind nur `COUNT` und `AVG` zulässig.

1. Beschreibe eine Methode, um herauszufinden, was die Note des schlechtesten Studenten der AMU ist
   1. Finde den gesamten Durchschnitt und die Gesamtanzahl:
      ```
      SELECT AVG(Durchschnittsnote), COUNT(*) FROM Noten
      ```
   2. Finde eine Anfrage die alle außer dem schlechtesten zurück gibt (lange ausprobieren).
      ```
      SELECT AVG(Durchschnittsnote), COUNT(*) FROM Noten
      WHERE Durchschnittsnote < $NoteSchlechtester
      ```
   3. Der Grenzwert ist der schlechteste.
2. Max will Alex' Durchschnittsnote herausfinden. Dazu stellt er folgende Anfragen mit Ergebnis:

   ```
   SELECT AVG(Durchschnittsnote), COUNT(*) FROM Noten => (2,5 ; 10.000)
   SELECT AVG(Durchschnittsnote), COUNT(*) FROM Noten WHERE Name != 'Alex'
   => (2,5001 ; 9.999)
   ```

   Kann er aus den Ergebnissen Alex' Note berechnen? Wenn ja, wie, wenn nein, wieso nicht?

   Ja, er kann: NoteAlex = 10.000 * 2,5 - 9.999 * 2,5001 = 1,5

## Hausaufgabe 2

Die Prüfungs-Datenbank der AMU wurde leider von einem unachtsamen Programmierer geschrieben. Es gibt ein Formular, in dem man nach seinen Klausurnoten suchen kann, allerdings wird die Benutzereingabe nicht geprüft.

Schema: Prüfung: {[Vorlesung, Note, Matrikelnummer]}

Benutzte Anfrage:

```
SELECT *
FROM Prüfung
WHERE Matrikelnummer='{MatrikelNr}' AND Vorlesung='{Benutzereingabe}'
```

- Benutzereingabe: ist die Benutzereingabe.
- MatrikelNr: wird automatisch mit deiner Matrikelnummer befüllt.

Schreibe eine Benutzereingabe, mit der du alle deine Noten auf 1.0 setzen kannst.

**Lösung:**

```
egal'; UPDATE Prüfung SET Note=1.0 WHERE Matrikelnummer='03670815;
```

## Hausaufgabe 3

Sie wollen der AMU helfen, ihre Datenbank sicherer zu machen, und finden bei einer kurzen Suche im Internet *Prepared Statements* und *Input Sanitization*.

1. Erklären Sie kurz beide Methoden, besonders deren Unterschiede in der Behandlung von bösartigen Eingaben.

   **Lösung:**

   **Input Sanitization** ersetzt oder entfernt Sonderzeichen, sodass die Zeichenkette sicher an die Datenbank übergegeben werden kann. Damit wird der Angriff unschädlich gemacht, da z.B. Anführungszeichen ersetzt werden.

   **Prepared Statement** bereitet eine SQL-Anfrage mit Platzhaltern vor. Damit wird zwischen der Struktur der Anfrage und der Eingabe unterschieden. Die Eingabe wird nachträglich in den Platzhalter eingefügt. Damit kann die Struktur der Anfrage nicht mehr verändert werden und der Angriff schlägt fehl.
2. Beschreiben Sie Vorteile von *Prepared Statements*, die über Sicherheit hinaus reichen.

   **Lösung:** Das Vorbereiten der Anfrage kann der Datenbank erlauben, die Anfrage als Template vorzubereiten, in das nur noch an bestimmten Stellen Eingaben eingefügt werden müssen. Dadurch können Anfragen, die das gleiche Template benutzen, mit kürzerer Latenz ausgeführt werden.
3. Nachfolgend sehen Sie die Funktion `exec_unsafe()`, die das Formular serverseitig aufruft, um die Klausurnote abzufragen. Ersetzen Sie diese durch eine Funktion `exec_prep()`, die mittels eines *Prepared Statements* auf die Datenbank zugreift.

```
#include <pqxx/pqxx>
#include <iostream>
#include <string>
pqxx::result exec_unsafe(pqxx::connection& conn, int matrnr, std::string vorl){
  std::string q = "SELECT␣*␣FROM␣pruefung␣WHERE␣matrikelnummer=",
    q2 = "AND␣vorlesung='", q3 ="'";
  pqxx::work tx{conn, ""}; // Begin of transaction
  pqxx::result r(tx.exec(q + std::to_string(matrnr) + q2 + vorl + q3));
  tx.commit(); // Commit transaction
  return r;
}
int main(int argc, char* argv[]){
  pqxx::connection conn;
```

```
  auto r = exec_unsafe(conn, 123, "Grundzuege");
  for(auto row: r)
    std::cout << row["note"] << std::endl;
  return 0;
}
```

**Lösung:**

```
#include <pqxx/pqxx>
#include <iostream>
#include <string>
pqxx::result exec_prep(pqxx::connection& conn, int matrnr, std::string vorl){
  pqxx::work tx{conn, ""}; // Begin of transaction
  auto r = tx.prepared("getGrade")(matrnr)(vorl).exec();
  tx.commit(); // Commit transaction
  return r;
}
int main(int argc, char* argv[]){
  pqxx::connection conn;
  conn.prepare("getGrade",
    "SELECT␣*␣FROM␣pruefung␣WHERE␣matrikelnummer=$1␣AND␣vorlesung=$2");
  auto r = exec_prep(conn, 123, "Grundzuege");
  for(auto row: r)
    std::cout << row["note"] << std::endl;
  return 0;
}
```

## Hausaufgabe 4

Bob hat ein Vorlesungsverzeichnis für die Universität programmiert und unter `http://db.in.tum.de/~schuele/sql_verzeichnis.html` online gestellt.

Um die Suche zu erleichtern, kann die Anzahl der SWS durch einen Parameter eingeschränkt werden. Finden Sie einen speziell präparierten Parameter, bei dessen Eingabe statt der Vorlesungen die Liste der Studenten ausgegeben wird. Die Datenbank folgt dem bekannten Universitätsschema.

Bob erfährt von der Sicherheitslücke und schlägt vor, die bekannten Tabellen einmalig mit zufälligen Namen umzubenennen, so seien sie nicht zu finden. Würde diese *Sicherheitsmaßnahme* helfen?

- Injection: `0 union all select name, matrnr, semester from studenten`
- Nein, da z.B. mit `0 union select tablename,1,1 from pg_tables` eine Liste der Datenbanken ausgegeben werden kann.

## Hausaufgabe 5

Sie haben die User-Tabelle zweier Pizzalieferanten ausgelesen, jedoch scheinen die Passwörter uncharakteristisch kompliziert zu sein. Das von Ihnen erhaltene Resultat ist das folgende:

```
id          name        password
----------  ----------  ----------------------------------------
1           luigi       4d75e8db6a4b6205d0a95854d634c27a
2           mario       fe78ea401158dd5847c4090b8bb22477e510febf
```

- Was könnte der Grund für diese hexadezimalen, 32 bzw. 40 Stellen langen Passwörter sein?
- Können Sie trotzdem den Klartext finden?

- Wie können Sie das Passwort sicherer speichern?
- Wie können Sie für diese Art von Passwortspeicherung Bruteforce-Attacken erschweren?
- Das erste Passwort wurde MD5 gehasht, das zweite SHA-1.
- Webrecherche nach dem Hash, es gibt sog. Rainbow-Tables in denen zahlreiche MD5- und SHA-1-Hashes vorberechnet sind.
- MD5 mehrfach anwenden, besser: Einen Salt verwenden, beispielsweise das Passwort zusammen mit dem Erstellungsdatum des Accounts oder dem Accountnamen hashen.
- Eine bessere Hashfunktion verwenden (MD5 und SHA1 werden nicht mehr empfohlen), die mehr Rechenzeit benötigt. Genauso wichtig, den User zwingen, komplexere Passwörter zu benutzen, damit Wörterbuchangriffe ineffizient werden.
- Alternativ hilft eine Key-Derivation-Function (KDF), ein Passwort mittels einer Pseudozufallsfunktion zu strecken.

**Gruppenaufgabe 6**

Sie fangen die folgende, mit RSA verschlüsselte Nachricht ab: 13. Sie kennen den öffentlichen Schlüssel (3,15). Wie lautet die Nachricht im Klartext? Geben Sie die komplette Herleitung an.
Alles laut Wikipedia `https://de.wikipedia.org/wiki/RSA-Kryptosystem`:

- Öffentlicher Schlüssel (e,N)
- Privater Schlüssel: (d,N)

$$N = p * q$$

mit $p$ und $q$ sehr großen Primzahlen. $e$, der sog. Verschlüsselungsexponent wird als teilerfremde Zahl zu $\phi(N)$ gewählt, wobei gilt $1 < e < \phi(N)$. $\phi(N)$ ist hierbei definiert als $\phi(N) = (p-1) * (q-1)$. $d$, der sog. Entschlüsselungsexponent, ist gerade das multiplikative Inverse von $e$ bezüglich des Moduls $\phi(N)$. Die Berechnung erfolgt mittels erweiterten euklidschen Algorithmus.
Die Entschlüsselung einer verschlüsselten Nachricht $C$ zu ihrem Klartext $K$ erfolgt mittels der Formel

$$K = C^d \mod N.$$

Aus der Angabe wissen wir:

- $N = 15$
- $e = 3$
- $C = 13$

Wir müssen also zunächst $d$ berechnen. Dies wäre einfach, wenn wir $\phi(N)$ wüssten. Hierzu ist die Primfaktorzerlegung von $N$ nötig. Dies ist für die Zahl 15 äußerst einfach, es gilt $N = 5 * 3$. Damit ist $\phi(N) = 4 * 2 = 8$. Die Lösung der Kongruenz

$$e * d \equiv 1 \ mod \phi(N)$$

bzw. im konkreten Fall

$$3 * d \equiv 1 \ mod 8$$

können wir raten, indem wir alle im Bezug auf 8 teilerfremden Zahlen $z$ betrachten, für die gilt: $1 < z < 8$.
Für $z = 3$ gilt $3 * 3 \equiv 1 \ mod 8$, womit $d = 3$ ist.
Wir entschlüsseln nun die Nachricht:

$$K = 13^3 \mod 15 = 7.$$

Der Klartext $K$ ist also 7.

## Hausaufgabe 7

Ein bekannter Anwendungsbereich des RSA-Kryptosystems ist das Verschlüsseln und Signieren von E-Mails. Die beiden Standards sind S/MIME und OpenPGP. Für ersteres benötigen Sie ein Zertifikat, ausgestellt von einer Zertifizierungsstelle, wie sie die TUM für alle E-Mail-Adressen ausgibt: `https://wiki.rbg.tum.de/Informatik/Helpdesk/Mail#A_2.2_Zertifikat_f_195_188r_nicht_in.tum.de_Adresse`

Ein Schlüsselpaar für OpenPGP können Sie sich jederzeit selbst und für jede E-Mail-Adresse zulegen. Beschäftigen Sie sich näher mit OpenPGP, damit Sie Ihre E-Mails verschlüsseln können und schicken Sie Ihrem Tutor eine verschlüsselte und signierte E-Mail. Ihr Tutor erklärt Ihnen während der Übungsstunde, an welche Adresse Sie Ihre E-Mail schicken sollen und wo Sie den entsprechenden öffentlichen Schlüssel erhalten. Dafür erhalten Sie einen Bonuspunkt.

OpenPGP: `https://de.wikipedia.org/wiki/OpenPGP`
Enigmail für Thunderbird: `https://www.enigmail.net`

## Hausaufgabe 8

Gegeben sei die folgende Segler-Boots-Reservierung-Datenbank:

```
.decl segler(sid: number, sname: symbol, einstufung: number, alter: number)
.decl boot(bid: number, bname: symbol, farbe: symbol)
.decl reservierung(sid: number, bid: number, datum: number)
```

Beantworten Sie die folgenden Anfragen in Datalog und testen Sie unter (`http://souffle.db.in.tum.de/`, Examples => Segler-Boots-Reservierung):

1. Geben Sie die Farben aller Boote, die von 'Lubber' reserviert wurden, aus.

```
.decl lubber_farbe(farbe: symbol)
lubber_farbe(F) :- segler(SID,"Lubber",_,_), reservierung(SID,BID,_),
                   boot(BID, _, F).
.output lubber_farbe
```

2. Geben Sie alle Segler aus, die eine Einstufung von mindestens 8 oder das Boot 103 reserviert haben.

```
.decl a2(sid: number, name: symbol)
a2(SID,N) :- segler(SID,N,R,_), R>=8.
a2(SID,N) :- segler(SID,N,_,_), reservierung(SID,103,_).
.output a2
```

3. Geben Sie die Namen aller Segler aus, die mindestens zwei Boote reserviert haben.

```
.decl doppelBoot(name: symbol)
doppelBoot(S) :- segler(SID,S,_,_), reservierung(SID,BIDA,_),
                 reservierung(SID,BIDB,_), BIDA!=BIDB .
```

4. Geben Sie alle Segler aus, die noch nie ein rotes Boot reserviert haben.

```
.decl rotReserviert(sid: number)
rotReserviert(SID) :- segler(SID,_,_,_), reservierung(SID,BID,_),
                      boot(BID,_,"red").
.decl nichtRot(sid: number, name: symbol)
nichtRot(SID,S) :- segler(SID,S,_,_), !rotReserviert(SID).
.output nichtRot
```

5. Geben Sie alle Segler aus, die mehr als 20 Jahre alt sind und kein rotes Boot reserviert haben.

```
.decl rotReserviert(sid: number)
rotReserviert(SID) :- segler(SID,_,_,_), reservierung(SID,BID,_),
                      boot(BID,_,"red").
.decl nichtRotAlt(sid: number, segler: symbol, alter: number)
nichtRotAlt(SID,S,A) :- segler(SID,S,_,A), A>20, !rotReserviert(SID).
.output nichtRotAlt
```

6. Geben Sie die Ids der Segler aus, deren Einstufung besser als die eines Seglers mit Namen 'Horatio' ist.

```
.decl nichtSchlecht(sid: number)
nichtSchlecht(SID) :- segler(SID,_,R,_), segler(_,"Horatio",RH,_), R > RH.
.output nichtSchlecht
```

7. Geben Sie die Ids der Segler aus, deren Einstufung besser als die aller Segler mit Namen 'Horatio' ist.

```
.decl dochSchlecht(sid: number)
dochSchlecht(SID) :- segler(SID,_,R,_), segler(_,"Horatio",RH,_), R<=RH.
.decl nochBesser(sid: number)
nochBesser(SID) :- segler(SID,_,_,_), !dochSchlecht(SID).
.output nochBesser
```

8. Geben Sie den Namen und Alter des ältesten Seglers aus.

```
.decl junger(sid: number)
junger(SID) :- segler(SID,_,_,A), segler(_,_,_,AO), A<AO.
.decl alter(name: symbol, alter: number)
alter(S,A) :- segler(SID,S,_,A), !junger(SID).
.output alter
```