Einsatz und Realisierung von Datenbanksystemen

ERDB Übungsleitung

Alice Rey, Maximilian Reif, Tobias Götz

i3erdb@in.tum.de

Folien erstellt von Maximilian Bandle & Alexander Beischl

Organisatorisches
Disclaimer

Die Folien werden von der Übungsleitung allen Tutoren zur Verfügung gestellt.

Sollte es Unstimmigkeiten zu den Vorlesungsfolien von Prof. Kemper geben, so sind die Folien aus der Vorlesung ausschlaggebend.

Falls Ihr einen Fehler oder eine Unstimmigkeit findet, schreibt an i3erdb@in.tum.de mit Angabe der Foliennummer.

Sicherheitsaspekte

Sicherheitsaspekte
SQL-Injections

- Hinter nahezu allen WebApps stehen Datenbanksysteme
- Aus den eingegebenen Parametern werden SQL-Anfragen generiert
- Den Eingaben darf niemals getraut werden, da sie SQL-Statements enthalten können
- Nutzer könnte auf dem Server Code ausführen

Sicherheitsaspekte
SQL-Injections

Name	Passwort	Geheimnis
Max	MaxIstToll!	Lässt Übungen halten
Alex	MaxIstDoof;)	Ist fleißig und schreibt MA ;)
Hacker	54321	Hat große Pläne

Beispiel WebApp

Name: Max

Passwort: MaxIstToll!

Dein Geheimnis ist: *Hält keine Übung mehr*

Sicherheitsaspekte
SQL-Injections

Beispiel WebApp

Name $name

Passwort $passwort

Dein Geheimnis ist: $geheimnis

Code der WebApp

```
$geheimnis = SELECT Geheimnis FROM Geheimnisse
  WHERE Name = '$name' AND Passwort = '$passwort';

print “Dein Geheimnis ist: “ + query($geheimnis)
```

Sicherheitsaspekte
SQL-Injections

Beispiel WebApp

Name: Max

Passwort: MaxIstToll!

Dein Geheimnis ist: *Lässt Übungen halten*

Code der WebApp

```
$geheimnis = SELECT Geheimnis FROM Geheimnisse
  WHERE Name = 'Max' AND Passwort = 'MaxIstToll!';

print “Dein Geheimnis ist: “ + query($geheimnis)
```

Sicherheitsaspekte
SQL-Injections

Beispiel WebApp

Name	Alex
Passwort	???' OR 'x'='x

Dein Geheimnis ist: *Lässt Übungen halten*
Ist fleißig und schreibt MA ;)
Hat große Pläne

Code der WebApp

```
$geheimnis = SELECT Geheimnis FROM Geheimnisse
  WHERE Name = 'Alex' AND Passwort = '???' OR 'x'='x';

print “Dein Geheimnis ist: “ + query($geheimnis)
```

Hinweis:
AND bindet stärker als OR, deshalb werden hier alle Geheimnisse ausgegeben.

Sicherheitsaspekte
SQL-Injections

Beispiel WebApp

Name: Alex

Passwort: ???'; DELETE FROM Geheimnisse WHERE 'x' = 'x

Dein Geheimnis ist: ERROR

Code der WebApp

```
$geheimnis = SELECT Geheimnis FROM Geheimnisse
  WHERE Name = 'Alex' AND Passwort = '???';
  DELETE FROM Geheimnisse WHERE 'x' = 'x';

print “Dein Gehemnis ist: “ + query($geheimnis)
```

Aufgabe 1

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.

Aufgabe 2

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.
2. Beschreiben Sie Vorteile von *Prepared Statements*, die über Sicherheit hinaus reichen.
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;
}
```

Sicherheitsaspekte
Input Sanitization

Beispiel WebApp

Name: Alex

Passwort: ???“ OR “x“=“x

Dein Geheimnis ist: Kein Ergebnis

sanitize löscht bzw. ersetzt alle Zeichen die aus einer Zeichenkette “ausbrechen” können.

sanitize(???“ OR “x“=“x) =>
“??? OR x=x“

Code der WebApp

```
$geheimnis = SELECT Geheimnis FROM Geheimnisse
  WHERE Name     = sanitize(Alex) AND
        Passwort = sanitize(???“ OR “x“=“x);

print “Dein Geheimnis ist: “ + query($geheimnis)
```

Sicherheitsaspekte
Stored Procedures

Beispiel WebApp

Name: Alex

Passwort: ???“ OR “x“=“x

Dein Geheimnis ist: Kein Ergebnis

prepare bereitet die Anfrage vor und legt bei den Platzhalten den Datentyp fest. Die Struktur kann sich nicht mehr ändern!

queryPrepare ersetzt die Platzhalter als String. Damit ist die bösartige Eingabe un-wirksam, da sie die Struktur nicht ändert .

Code der WebApp

```
$geheimnis = prepare(SELECT Geheimnis FROM Geheimnisse
  WHERE Name = $1 AND Passwort = $2;)

print “Dein Geheimnis ist: “ +
      queryPrepare($geheimnis, Alex, ???“ OR “x“=“x)
```

Aufgabe 3

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 ein 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 *Sicherheitsmaßname* helfen?

Aufgabe 4

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

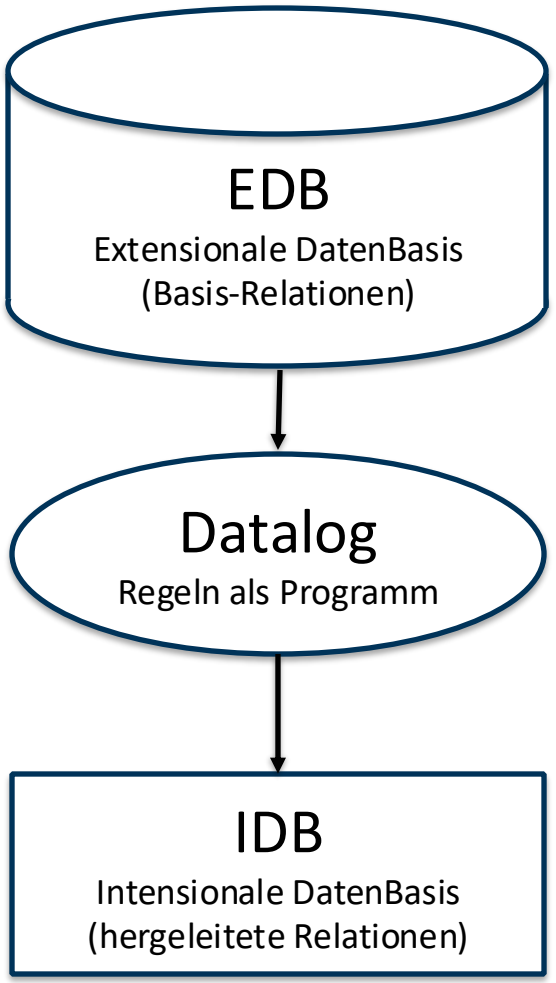
- Was könnte der Grund für diese hexadezimalen, 32 bzw. 40 Stellen langen Passwörter sein?
- Können Sie trotzdem den Klartext finden?
- Wie können Sie das Passwort sicherer speichern?
- Wie können Sie für diese Art von Passwortspeicherung Bruteforce-Attacken erschweren?

Deduktive Datenbanken

Deduktive Datenbanken
Einführung

- EDB/Faktenbasis ist die Menge der Relationen
- Deduktion durch Datalog (Data + Prolog -> Datalog)
- Die IDB entsteht durch Anwenden der Datalog-Regeln auf die EDB

➞ Erzeugt weitere Menge von Relationen



Deduktive Datenbanken
Regeln

Deklarationen:

.decl vorlesungen(VorlNr: number, Titel: symbol, SWS: number, PersNr: number)

.decl professoren(PersNr: number, Name: symbol, Rang: symbol, Raum: number)

.decl sokratesVL(Titel: symbol, SWS: number)

Basisrelationen:

vorlesungen(5001,"grundzuege",4,2137).

professoren(2125,"sokrates","c4",226).

Regelerzeugung und Join:

sokVL(T,S) :- vorlesungen(_,T,S,P), professoren(P, "sokrates",_,_), S>2.

Deduktive Datenbanken
Syntax

Variablen: Mögliche Typen: **symbol**, **number**, **unsigned**, **float**

Relationen:

- **Deklaration**: .decl relation(feld1: symbol,feld2: number,feld3: float)
- **Fakten**: relation("Wert1", 2, 3.0).
- **Regeln**: relation(Param1, …) :- Ausdruck. *Punkt immer als Abschluss*

Logische Verknüpfung:

Komma => Und

Semikolon oder Regel mehrfach definieren => Oder

Prädikate: <, =, >, <=, >=, !=

Negation: !relation(Param1, …)

Ausgabe: .output relation

Aufgabe 5

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)
   ```

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

   ```
   .decl a2(sid: number, name: symbol)
   ```

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

   ```
   .decl doppelBoot(name: symbol)
   ```

Aufgabe 5

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):

4. Geben Sie alle Segler aus, die noch nie ein rotes Boot reserviert haben.
5. Geben Sie alle Segler aus, die mehr als 20 Jahre alt sind und kein rotes Boot reserviert haben.
6. Geben Sie die Ids der Segler aus, deren Einstufung besser als die eines Seglers mit Namen 'Horatio' ist.
7. Geben Sie die Ids der Segler aus, deren Einstufung besser als die aller Segler mit Namen 'Horatio' ist.
8. Geben Sie den Namen und Alter des ältesten Seglers aus.

Deduktive Datenbanken
Rekursion

Datenbasis: direkt(Start, Ziel, Linie)

Ziel: indirekt(Start, Ziel, Stops)

1. Basisfall => Fülle die Relation mit Anfangswerten

```
indirekt( Start, Ziel, Stops ) :- direkt( Start, Ziel, _), Stops = 0.
```

2. Rekursion => Nutze die Relation selbst und erweitere sie

```
indirekt( Start, Ziel, StopsNeu ) :-
        indirekt( Start, Station, Stops ),
        direkt( Station, Ziel, _),
        StopsNeu = Stops + 1.
```

Aufgabe 6

KindEltern		
Vater	Mutter	Kind
Zeus	Leto	Apollon
Zeus	Leto	Artemis
Kronos	Rheia	Hades
Zeus	Maia	Hermes
Koios	Phoebe	Leto
Atlas	Pleione	Maia
Kronos	Rheia	Poseidon
Kronos	Rheia	Zeus

Gegeben sei die nachfolgende *KindEltern*-Ausprägung für den Stammbaum-Ausschnitt der griechischen Götter und Helden:

Formulieren Sie folgende Anfragen in Datalog und testen Sie unter (`http://souffle.db.in.tum.de/`):

a) Bestimmen Sie alle Geschwisterpaare.

b) Ermitteln Sie Paare von Cousins und Cousinen beliebigen Grades. Die Definition finden Sie auf Wikipedia.

c) Geben Sie alle Verwandtschaftspaare an. Überlegen Sie sich eine geeignete Definition von Verwandtschaft und setzen Sie diese in Datalog um.

d) Bestimmen Sie alle Nachfahren von Kronos. Formulieren Sie die Anfrage auch in SQL, so dass sie unter PostgreSQL ausführbar ist (online testen unter: `http://sqlfiddle.com` mit der Datenbank PostgreSQL statt MySQL, das Schema Textfeld können sie leer lassen, müssen aber trotzdem auf 'Build Schema' drücken). Sie können die Daten als Common Table Expression definieren und dann nutzen:

Aufgabe 6

```
WITH RECURSIVE
kindEltern(vater,mutter,kind) as (
  VALUES
   ('Zeus', 'Leto', 'Apollon'),
   ('Zeus', 'Leto', 'Artemis'),
   ('Kronos', 'Rheia', 'Hades'),
   ('Zeus', 'Maia', 'Hermes'),
   ('Koios', 'Phoebe', 'Leto'),
   ('Atlas', 'Pleione', 'Maia'),
   ('Kronos', 'Rheia', 'Poseidon'),
   ('Kronos', 'Rheia', 'Zeus')
),
parent(eltern,kind) as (
  select vater, kind from kindEltern UNION
  select mutter, kind from kindEltern
)
select * from parent where eltern='Zeus'
```

Fragen?