Informatics II for Engineering Sciences (MSE)

Chapter IV – Databases: SQL

Bernd Brügge Ph.D. , Jan Knobloch, Emitzá Guzmán

Normal forms - Motivation

- **Normalization** of schema by **splitting attributes** into several **relations**
- Splitting has to be without loss of **information**
- Goal:
  - Remove unnecessary redundancy
    ➔ Avoid inconsistency when data is changed
    ➔ Reduce memory consumption
  - Minimize redesign when extending the database structure
  - Allow general queries

Normal forms

- Normal forms sorted by strictness
  - **1st Normal form**: Each attribute has atomary domain

ProfessorLecture					
PersNo	Name	Rank	Room	Title	SWS
2125	Sokrates	C4	226	{5041, Ethics}	4
2125	Sokrates	C4	226	{5049, Maieutics}	2
2125	Sokrates	C4	226	{4052, Logic}	4
…	…	…	…	…	…

Normal forms

- Normal forms sorted by strictness
  - 1st **Normal form**: Each attribute has atomary domain
  - 2nd **Normal form**: No attribute that is not a key is functionally dependent of a subset of any key candidate

ProfessorLecture					
PersNo	Name	Rank	Room	**Title**	SWS
2125	Sokrates	C4	226	Ethics	4
2125	Sokrates	C4	226	Maieutics	2
2125	Sokrates	C4	226	Logic	4
...	...	...	...	...	...

Normal forms

- Normal forms sorted by strictness
  - 1st **Normal form**: Each attribute has atomary domain
  - 2nd **Normal form**: No attribute that is not a key is functionally dependent of a subset of any key candidate
  - 3rd **Normal form**: No attribute that is not a key has a transitive dependency from a key candidate

CD			
CDID	Albumtitle	Singer	Founding year
4711	Not that kind	Anastacia	1999
4712	Wish you were here	Pink Floyd	1964
4713	Freak of Nature	Anastacia	1999
...	...	...	...

Normal forms

- Normal forms sorted by strictness
  - 1st **Normal form**: Each attribute has atomary domain
  - 2nd **Normal form**: No attribute that is not a key is functionally dependent of a subset of any key candidate
  - 3rd **Normal form**: No attribute that is not a key has a transitive dependency from a key candidate
  - Boyce-Codd Normal form
  - 4th Normal form
  - 5th Normal form
- A Normal form fullfills all criteria of the previous normal forms

SQL

- **S**tructured **Q**uery **L**anguage
- Standardized language for
  - Making Queries
  - Data definition (DDL)
  - Data manipulation (DML)
- The language is standardized, however most implementations do not follow the standard entirely

You can try out the queries shown in the lecture on:
http://www.hyper-db.com/interface.html

SQL-Query

```
select   personalNr, name ——— attributes
from professor ——— relations
where  rank = ´C4´;
              |
         condition
```

Professor	
persNo	name
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

- Boolean operators: and, or, not
- Sort result: order by, asc, desc
- Eliminate duplicates: distinct
- Set operators: union, intersect, minus
- Set comparisons: exists, all, in
- Aggregate functions: avg, max, min, count, sum
- Subqueries

SQL-Queries: Syntactical Sugar

```
select *
from Student
where semester > = 1 and semester < = 4;

select *
from Student
where semester between 1 and 4;

select *
from Student
where semester in (1,2,3,4);
```

SQL-Queries: Syntactical Sugar

```
select *
from Student
where name like `T%eophrastos‘;


select distinct s.name
from Course c, participates p, Student s
where s.matrNo = p.matrNo and
      p.courseNo = c.courseNo and
      c.title like `%thik%‘;
```

SQL-Queries: Syntactical Sugar

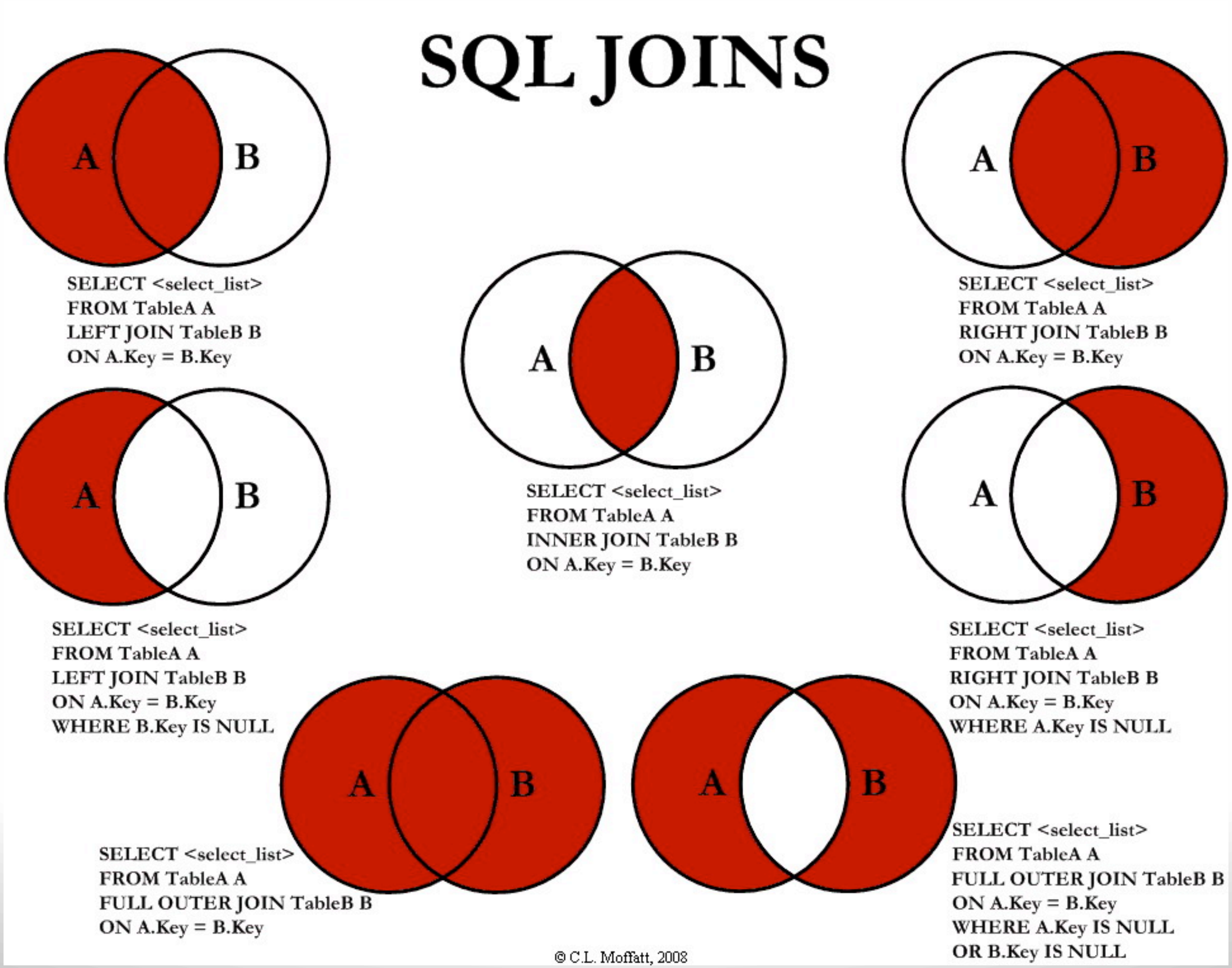
Place holder: "%" ; "_"

- "%" for any symbols or no symbol
- "_" for exactly one symbol

```
select *
from Student
where name like `T%eophrastos‘;

select distinct s.name
from Course c, participates p, Student s
where s.matrNo = p.matrNo and
      p.courseNo = c.courseNo and
      c.title like `%thik%‘;
```

SQL-Queries: Joins



SQL JOINS

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

© C.L. Moffatt, 2008

SQL

- **S**tructured **Q**uery **L**anguage
- Standardized language for
  - Making Queries
  - Data definition (DDL)
  - Data manipulation (DML)
- The language is standardized, however most implementations do not follow the standard entirely

You can try out the queries shown in the lecture on: http://www.hyper-db.com/interface.html

SQL-Queries: Schema definition - Create database

create table Student

(matrNo **integer primary key**,

name **varchar**(30) **not null,**

semester **integer check** semester)

Student		
matrNo	name	semester

SQL-Queries: Schema definition - Create database with Composite Primary Key

```
Create table participates (
    matrNo int(11) not null,
    courseNo int(11) not null,
    primary key (matrNo, courseNo)
)
```

participates	
matrNo	courseNo

SQL-Queries: Data integrity

Foreign Key

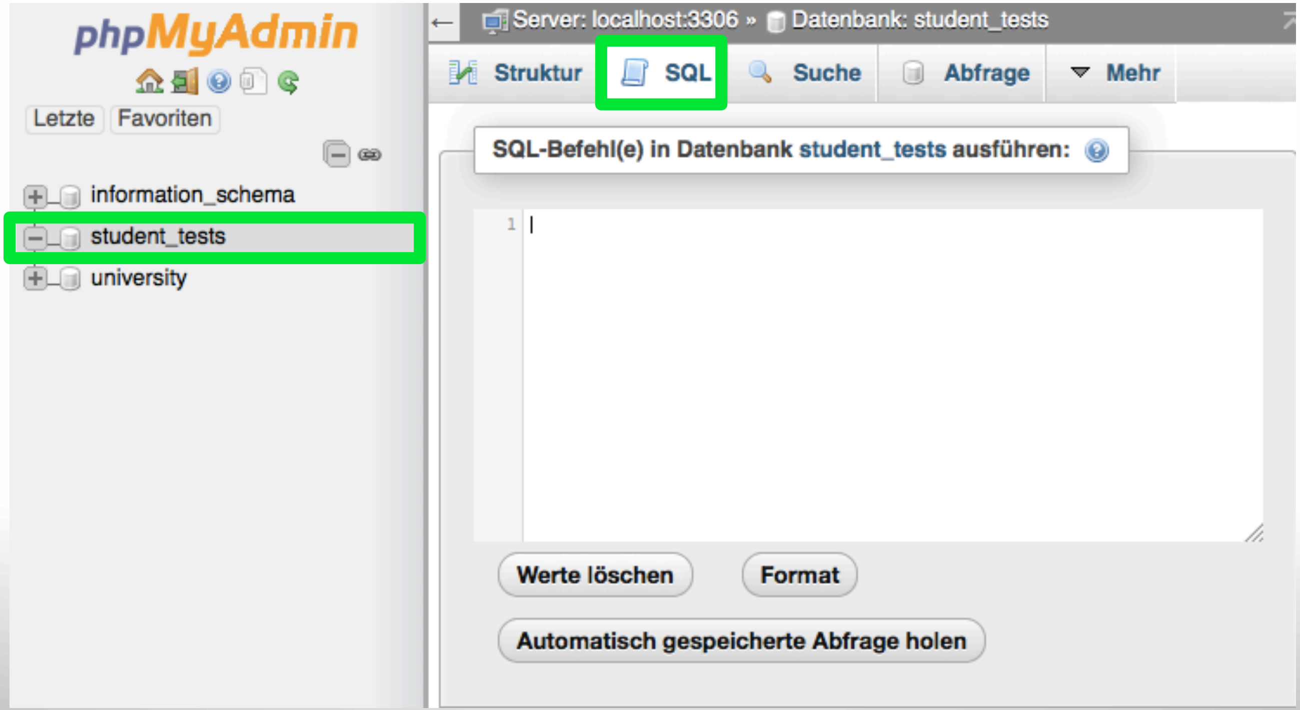
- Point towards a tuple in another relation
  e.g. taughtBy in Course relates to the primary key in the relation Professor
- Referential integrity: Foreign keys need to refer to another existing tuple or contain a null value

Course			
courseNo	title	sws	taughtBy

```
create table Vorlesungen
    (courseNo       integer primary key,
    title           varchar(30),
    sws             integer,
    taughtBy        integer references Professoren on delete
                    set null);
```

Exercise 1

- Open https://infeedbruegge.in.tum.de/mse in browser
  - Username: mse
  - Password: mse
- Open the database **student_tests** and switch to **SQL** tab


phpMyAdmin
Letzte
Favoriten
information_schema
student_tests
university
Server: localhost:3306 » Datenbank: student_tests
Struktur
SQL
Suche
Abfrage
Mehr
SQL-Befehl(e) in Datenbank student_tests ausführen:
Werte löschen
Format
Automatisch gespeicherte Abfrage holen

Exercise 1 – Create a table…

- Named "*YourName*Contacts" where you replace *YourName* with your actual name
- With the following attributes:
  - contactID, integer, not null
  - firstName, varchar, length 25, not null
  - lastName, varchar, length 25, not null
  - phoneNumber, varchar, length 25, not null
- With composite primary key, consisting of
  - contactID
  - phoneNumber

Exercise 1 - Solution

```
create table BarbaraReichartContacts (
    contactID integer not null,
    firstname varchar(25) not null,
    lastname varchar(25) not null,
    phoneNumber varchar(25) not null,
    primary key (contactID, phoneNumber)
);
```

SQL

- **S**tructured **Q**uery **L**anguage
- Standardized language for
  - Making Queries
  - Data definition (DDL)
  - Data manipulation (DML)
- The language is standardized, however most implementations do not follow the standard entirely

You can try out the queries shown in the lecture on:
http://www.hyper-db.com/interface.html

SQL – Changing database state

Insert tuples

```
insert into participates
        select matrNo, courseNo
        from Student, Course
        where title = `Logic` ;

insert into Studenten (matrNo, name)
        values (28121,  `Archimedes`);
```

Student		
matrNo	name	semester
⋮	⋮	⋮
29120	Theophrastos	2
29555	Feuerbach	2
28121	Archimedes	-

SQL – Changing database state

Delete Tuples

delete Student

where semester > 13;

Change Tuples

update Student

set semester= semester + 1;

Exercise 2 – Insert Contacts

- Open the table "*YourName*Contacts"
- Add the following entries:

YourNameContacts			
contactID	firstName	lastName	phoneNumber
1	Jan	Knobloch	0176/173 143 21
1	Jan	Knobloch	0175/223 333 44
2	Emitza	Guzman	089/289 18213
4	Bernd	Brügge	
3	Barbara	Reichart	089/289 18213
3	Alfons	Reichart	089/289 18213

- You cannot create some of the entries due to constraints, which ones? What is the constraint that hinders creation?

Homework #9 - SQL

- Write the following queries for the **university** database:
  - Select minimal **and** maximal semester of all students
  - Select all students that take a class taught by Sokrates. Be sure to eliminate possible duplicates (e.g. when Susan visits two of Sokrates classes)
- Edit tables in **student_tests**
  - Create a new table that contains phone number and type (e. g. mobile, work, private, …). You can store the type as a varchar
  - Make sure phone number is a foreign key
  - Add 5 tuples, then remove one of them

- Test your queries in https://infeedbruegge.in.tum.de/mse
  - Username: mse
  - Password: mse

Informatics II for Engineering Sciences (MSE)

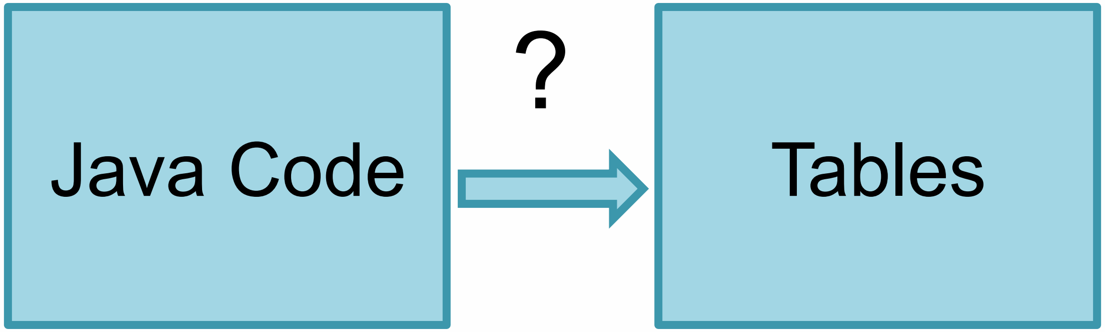
Chapter III – Using Databases from Java

17/07/15

Bernd Brügge Ph.D. , Jan Knobloch, Emitzá Guzmán

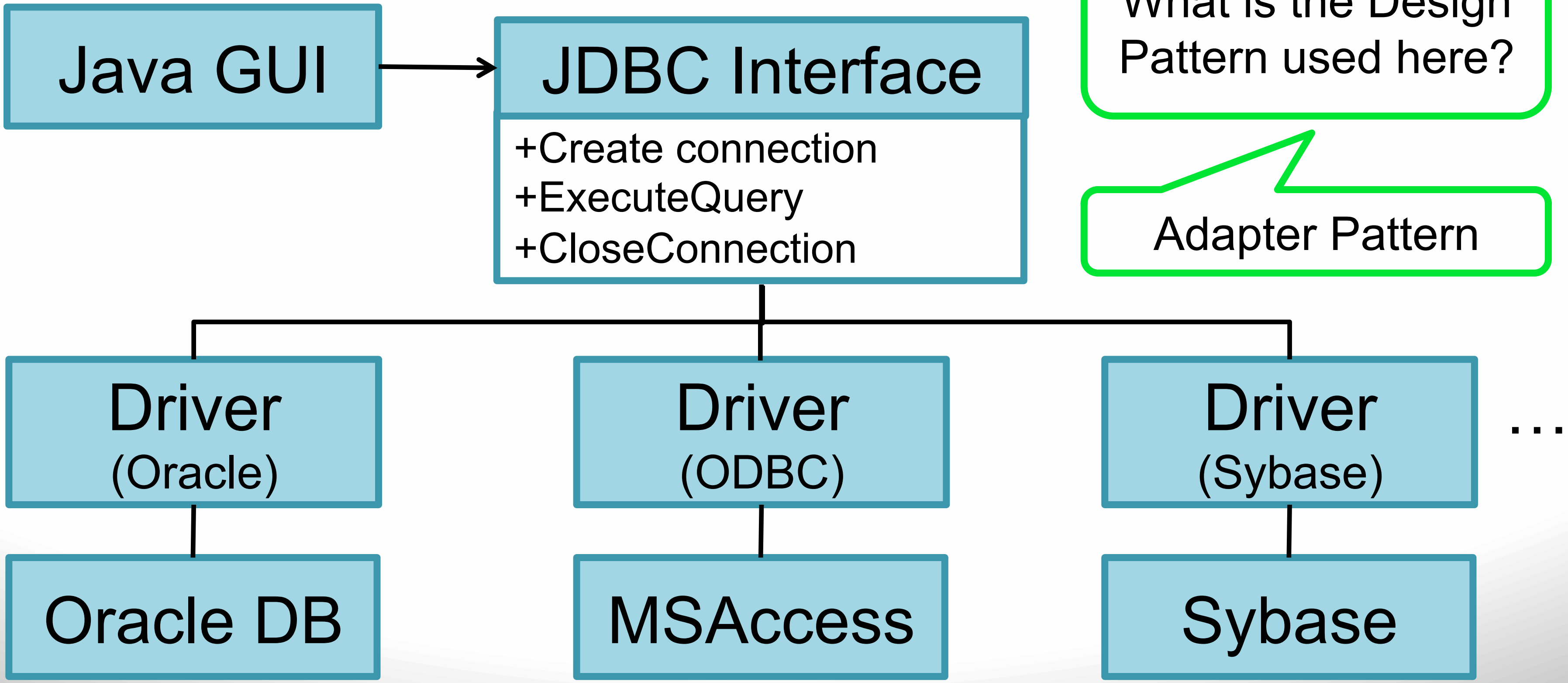
Use Database in Java Code

- We showed you how to map UML to Java
- We showed you how to map UML to Tables
- How do we connect Java Code with Tables?



Interfacing Java to Relational Database: JDBC

- Standardized interface to connect relational databases with Java



What is the Design Pattern used here?

Adapter Pattern

JDBC: Create Connection

```
Statement statement = null;
Connection connection = null;
try {
   Class.forName("org.h2.Driver");
   connection = DriverManager.getConnection("jdbc:h2:~/
      info2", "sa", "");
   statement = connection.createStatement();
} catch (Exception e) {
   System.err.println("Error: " + e);
   System.exit(-1);
}
```

Load Driver

Open connection using:

- Database url
- User name
- Password

Create statement

Exception handling

JDBC: Execute query

```
try {
   ResultSet resultSet = statement.executeQuery("select
      avg(semester) from Student");
   resultSet.next();
   System.out.println("Average age: " +
      resultSet.getDouble(1));
   resultSet.close();
} catch (SQLException se) {
   System.out.println("Error: " + se);
}
```

Execute Query and store results in ResultSet

Get value in first column

JDBC: Iterate result set

```
try {
   ResultSet resultSet = statement.executeQuery("select
      name, room from Professor where rank = 'C4'");
   System.out.println("C4-Professors:");
   while (resultSet.next()) {
      System.out.println(resultSet.getString("Name") +
         " " + resultSet.getInt("Room"));
   }
   resultSet.close();
} catch (SQLException se) {
   System.out.println("Error: " + se);
}
```

Execute Query and store results in ResultSet

Iterate over resultSet

JDBC: Close connection

```
try {
   statement.close();
   connection.close();
} catch (SQLException e) {
   System.out.println("Error when closing DB connection: "
      + e);
}
```

Close statement and connection

Exercise 3: JDBC

Get started:

- Download the project from moodle
- Run the DatabaseCreator
- You can look at the database either by
  - Double-clicking on the h2-1.4.179.jar (Only if you associated .jar files with Java)
  - Run the jar from the terminal using java -jar h2*.jar

Task:

- Formulate an SQL query for each of the tasks below
- Execute the query using java code
- Log the result

Queries:

- Count the number of C4 professors.
- Select all students with less than 8 semesters

JDBC: Prepared Statements

- Creates a Statement, which is easy to reuse with different parameters
  e.g. Finding several students by name
- Faster execution of batches
- Checks parameters, hence it helps avert SQL injections

JDBC: Prepared Statements

```
String statementString = "update contact set
firstname = ? where contactID = ?";
```

String containing query

? is placeholder for concrete values

```
PreparedStatement updateStatement =
con.prepareStatement(statementString );
```

Create PreparedStatement

```
updateStatement.setString(1, "Dieter");
updateStatement.setInt(2, 5);
```

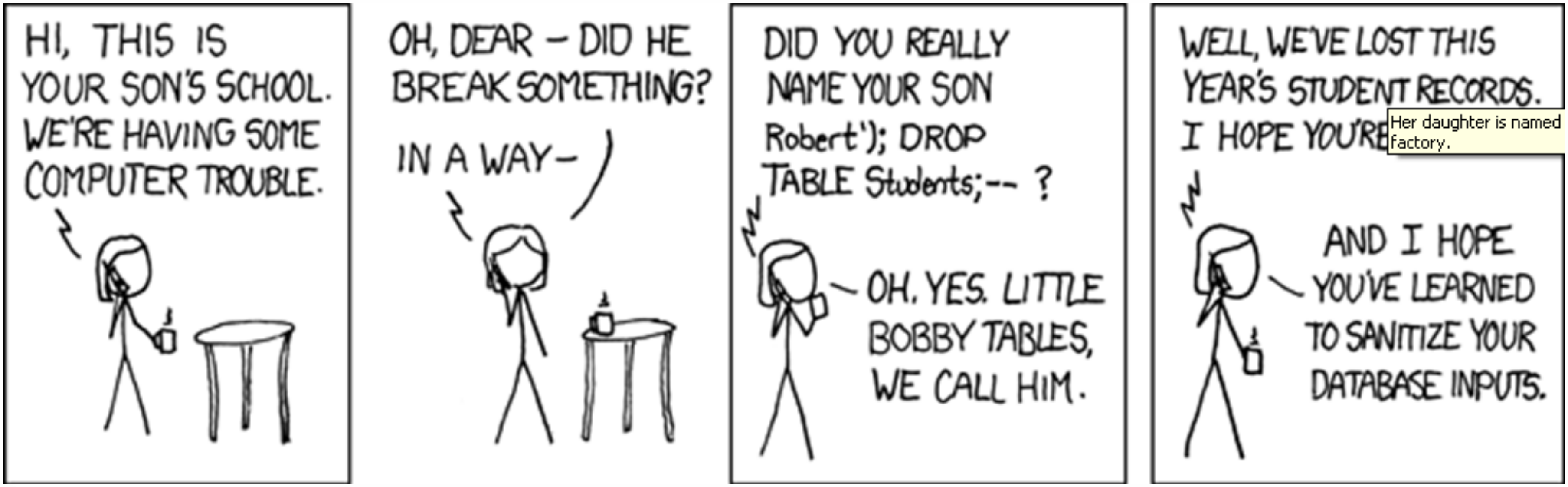
Replace placeholders with values

Index of placeholder

Value

JDBC: Prepared Statements

User Input is always a potential security hazard!


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE
Her daughter is named factory.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

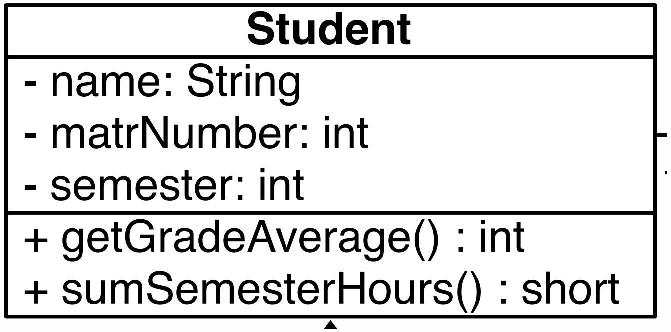
Drop table removes a table

ORM (Object-Relational-Mapping)

- Directly maps objects to a relational database → Virtual object database

- Advantage:
  - Save code
  - Work on objects (higher abstraction)
- Disadvantage:
  - Potential performance hit due to high level of abstraction
  - Might create poorly designed databases

ORM Mapping for one class: Entity



Student

- name: String
- matrNumber: int
- semester: int

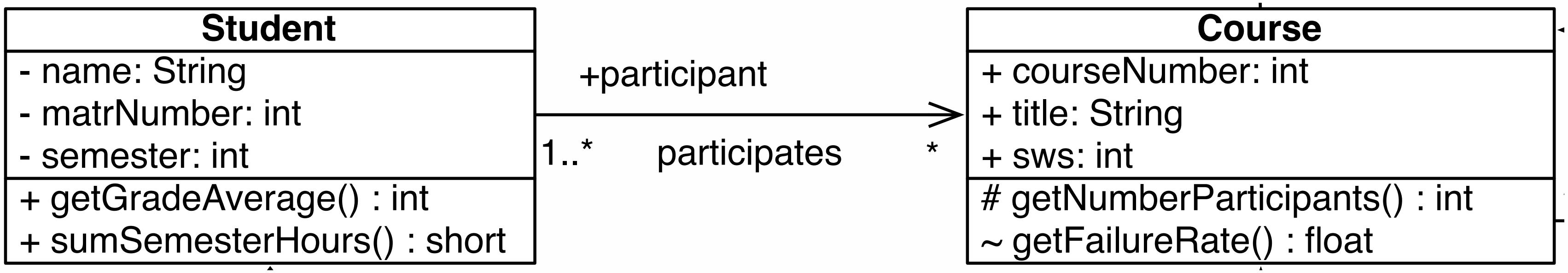
+ getGradeAverage() : int
+ sumSemesterHours() : short

Mapping can easily done using **annotations**

```
@Entity
public class Student {
   @Id
   @GeneratedValue(strategy=GenerationType.AUTO)
   private int matrNumber;
   …
}
```

ORM Mapping of relations between classes

- Also done using annotations: @OneToOne, @ManyToOne, @ManyToMany



Student

- name: String
- matrNumber: int
- semester: int

+ getGradeAverage() : int
+ sumSemesterHours() : short

+participant

1..*

participates

*

Course

+ courseNumber: int
+ title: String
+ sws: int

# getNumberParticipants() : int
~ getFailureRate() : float

```
@Entity
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int matrNumber;

    @ManyToMany(fetch=FetchType.LAZY)
    private List<Course> courses;
    …
}
```

ORM: Entity Manager

Create new Student

```
EntityManager entityManager =

entityManagerFactory.createEntityManager();
entityManager.getTransaction().begin();
entityManager.persist( new Student(12354,
“Donald”, 12);
entityManager.getTransaction().commit();
entityManager.close();
```

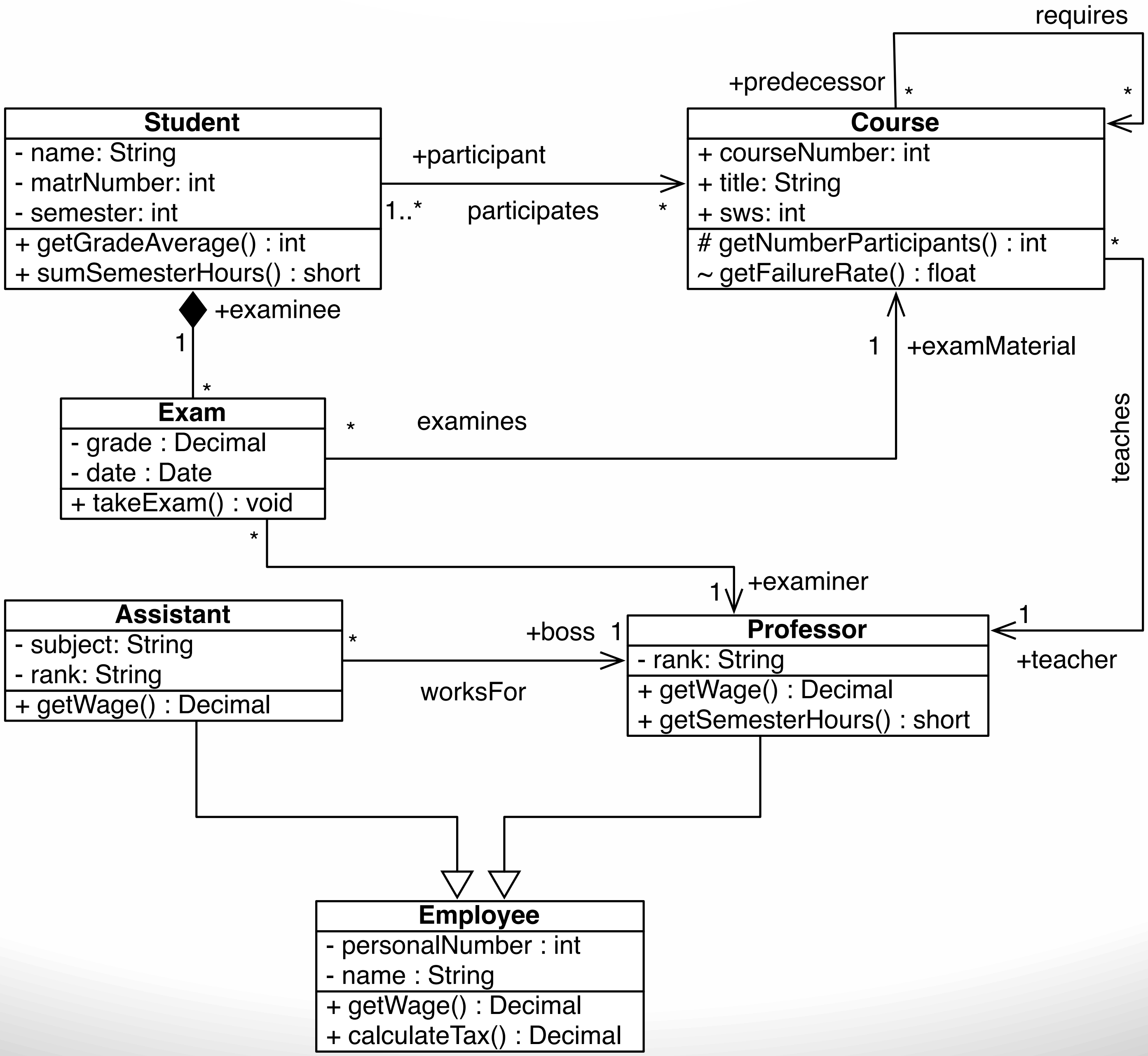
ORM: Entity Manager

Find a student

```
entityManager =
   entityManagerFactory.createEntityManager();
entityManager.getTransaction().begin();
List<Student> result = entityManager.createQuery
( "from
   Student", Student.class ).getResultList();
for ( Student student : result ) {
  System.out.println( ”Student (" + student.get
  Name() + ”));
}
entityManager.getTransaction().commit();
entityManager.close();
```

Exercise 4:

- Write the professor class
- Add annotations to support ORM (@Entity, @Id, @OneToOne, @OneToMany, @ManyToMany)
- Assume that all other classes are provided
- Professor should have personalNumber, name, and rank



Student
- name: String
- matrNumber: int
- semester: int

+ getGradeAverage() : int
+ sumSemesterHours() : short

Course

+ courseNumber: int
+ title: String
+ sws: int

# getNumberParticipants() : int

~ getFailureRate() : float

Exam
- grade : Decimal
- date : Date

+ takeExam() : void

Assistant
- subject: String
- rank: String

+ getWage() : Decimal

Professor
- rank: String

+ getWage() : Decimal
+ getSemesterHours() : short

Employee
- personalNumber : int
- name : String

+ getWage() : Decimal
+ calculateTax() : Decimal

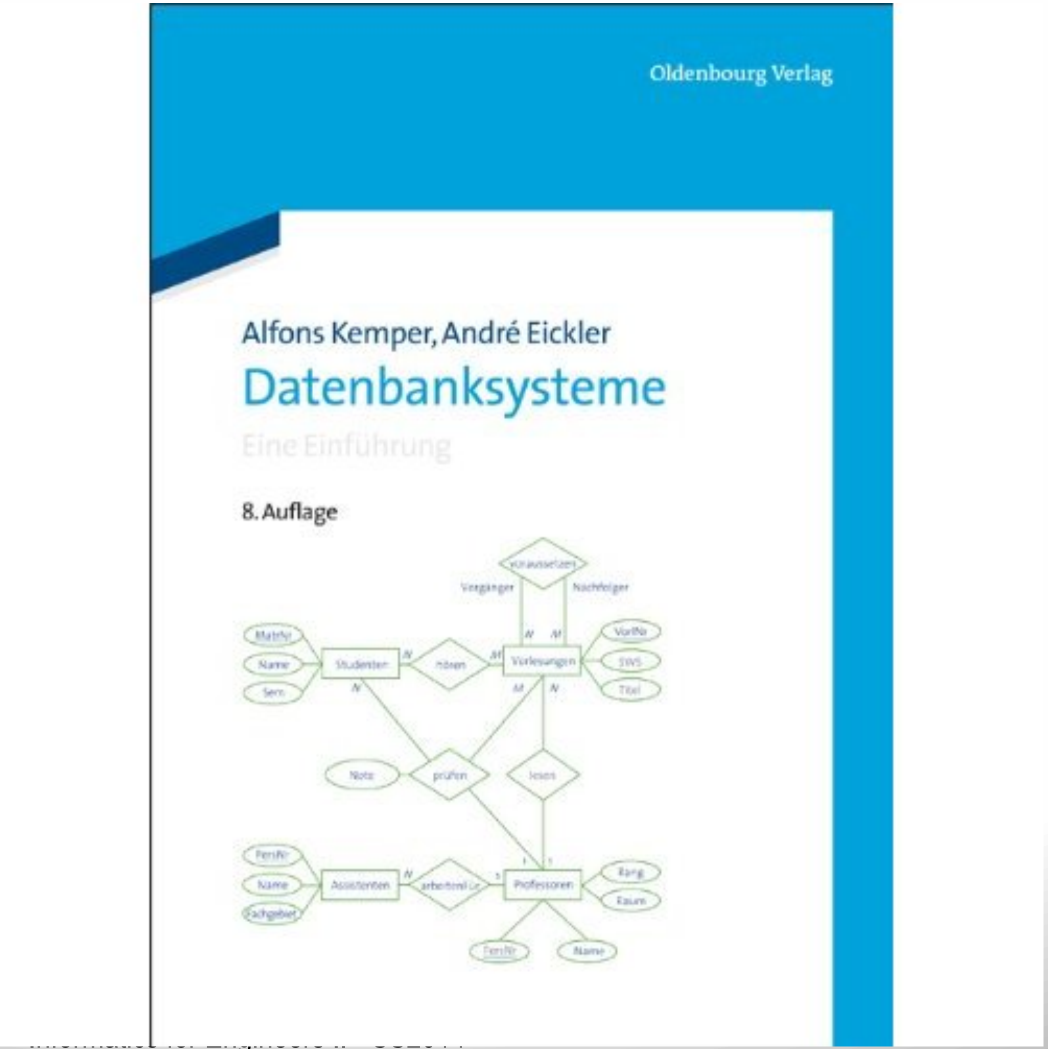
NoSQL (Not only SQL)

- Motivation (Why NoSQL)
  - Scalability
- Solution: Horizontal Scaling
- Major use case: Interactive web applications
  - Examples:
    - Social Networks (Facebook, Twitter, Digg)
    - Auction Platforms (eBay)
- Problems
  - Only weak support for consistency
  - Lack of standardized interfaces
  - Only low-level query languages

Overview

- Databases have several goals:
  - Avoid redundancy and inconsistency
  - Rich access to data
  - Synchronize concurrent data access
  - Avoid loss of data
  - Enforcing integrity rules
  - Security and Privacy
- There exist several types of databases, most important
  - Relational database
  - Objects database
  - NoSQL database
- Relational databases
  - Consist of tables
  - Tables are defined using a schema, which includes data types and integrity information
  - **SQL** is a language that allows to create, alter and query a relational database
- ORM
  - Maps objects into a relational database automatically
  - Usually implemented using annotations or subclassing
- NoSQL
  - Attempt to deal with performance issues in huge real-time web applications
  - Often sacrifice consistency to achieve higher performance

Bibliography



Oldenbourg Verlag

Alfons Kemper, André Eickler

Datenbanksysteme

Eine Einführung

8. Auflage