Introduction to Project 2

Discussion Session 4

Outline

- Sample RA and RC Queries
- Goal: Query Fakebook Data
- Access to 12 New Tables
- Introduction to JDBC
- Introduction to PreparedStatements
- Project Files
- Example Query
- 9 Tasks
- Execution, Submission, Grading

Ex. 1 – Relational Algebra

Consider the following relational schema for a library (primary keys are underlined)

Member (mid, name)
Book (isbn, title, author, publisher)
Borrow (mid, isbn, date)

Write a relational algebra expression to find the names of members who have borrowed a book written by Robert Caro.

Ex. 1 – Relational Algebra

Consider the following relational schema for a library (primary keys are underlined)

Member (mid, name)
Book (isbn, title, author, publisher)
Borrow (mid, isbn, date)

Write a relational algebra expression to find the names of members who have borrowed a book written by Robert Caro.

$$\pi_{\text{name}}(\sigma_{\text{author = 'Robert Caro'}}(Member \bowtie Book \bowtie Borrow))$$

Ex. 2 – Relational Calculus

Consider the following relational schema (primary keys are underlined, and foreign keys are in italics)

Part(<u>partid</u>, description, price, *suppid*)
Supplier(<u>suppid</u>, suppname, rating, phone number, street, *citycode*)
City(<u>citycode</u>, cityname, country)

Write a relational calculus expression to find the description of all parts that are supplied on "Main Street".

Ex. 2 – Relational Calculus

Consider the following relational schema (primary keys are underlined, and foreign keys are in italics)

Part(<u>partid</u>, description, price, *suppid*)
Supplier(<u>suppid</u>, suppname, rating, phone number, street, *citycode*)
City(<u>citycode</u>, cityname, country)

Write a relational calculus expression to find the description of all parts that are supplied on "Main Street".

$$\{ T \mid \exists\, P \in Part, \exists\, S \in Supplier$$
$$(P.suppid = S.suppid \wedge S.street = \text{Main Street} \wedge$$
$$T.description = P.description) \}$$

Ex. 3 – RA

Find the names of sailors who have reserved all boats.

Sailors

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Reserves

sid	bid	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Boats

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Ex. 3 – RA

Find the names of sailors who have reserved all boats.

$$\rho\ (Tempsids, (\pi_{sid,bid} \text{Reserves}) / (\pi_{bid} Boats))$$

$$\pi_{sname}(Tempsids \bowtie Sailors)$$

Ex. 3 – RA

Find the names of sailors who have reserved all boats.

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (( SELECT B.bid FROM Boats B )
                  EXCEPT
                  (SELECT R.bid
                  FROM Reserves R WHERE
                  R.sid = S.sid ))
```

Project 2 Goal

- In project 1, our focus was on database design
- In project 2, our focus will be:
  - On writing SQL queries, and
  - On embedding SQL queries in Java using JDBC

12 New Tables

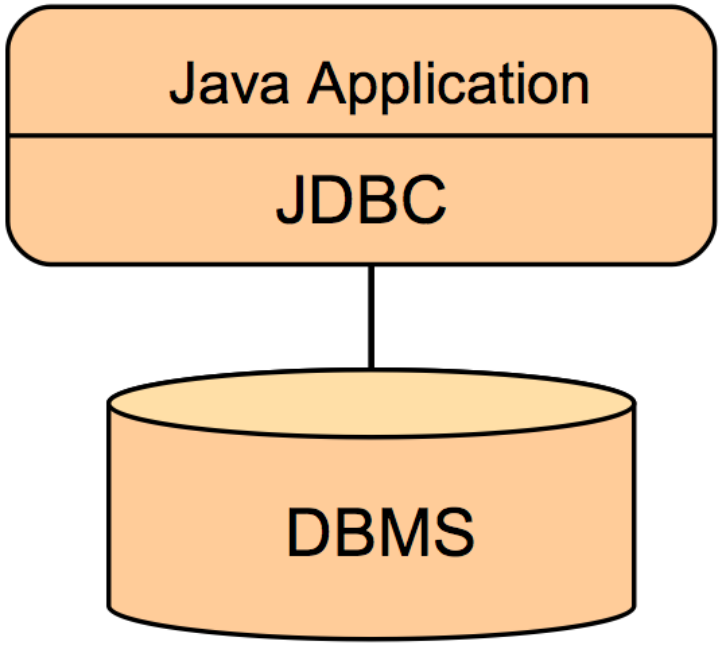
- USERS
- FRIENDS
- CITIES

- Others will be released after the 4 late days of Project 1, since it is based on the solution of Project 1, but you can still start working on it

Accessing Public Data

- <prefix>.<Data Type>_<TABLE_NAME>
- prefix: ethanjyx
- Data Type: “PUBLIC”
- TABLE_NAME: 12 table names
  - Example: ethanjyx.PUBLIC_USERS

Overview



- JDBC: Java API to access a DBMS from an application
- Allows us to:
  - Connect to a DBMS
  - Send queries and update statements to the DBMS
  - Retrieve and process the results received from the database

jdbc:oracle:thin:@forktail.dsc.umich.edu:1521:COURSEDB

JDBC Example

```
String secret = “jfniksan”;
Statement st = null;
ResultSet rst = null;
String query = "SELECT GRADE FROM GRADES WHERE SECRET = " + secret;
try {
        st = conn.createStatement();
        rs = st.executeQuery(query);
        while (rs.next()) {
                String grade = rs.getString(”GRADE");
                System.out.println(grade);
        }
}
catch(SQLException e) {
        System.err.println(e.getMessage());
        throw e;
}
finally {
        if(st != null) st.close();
        if(rst != null) rst.close();
}
```

JDBC Example

```
String secret = “jfniksan”;
Statement st = null;
ResultSet rst = null;
String query = "SELECT GRADE FROM GRADES WHERE SECRET = " + secret;
try {
        st = conn.createStatement();
        rs = st.executeQuery(query);
        while (rs.next()) {
                String grade = rs.getString(”GRADE");
                System.out.println(grade);
        }
}
catch(SQLException e) {
        System.err.println(e.getMessage());
        throw e;
}
finally {
        if(st != null) st.close();
        if(rst != null) rst.close();
}
```

may throw exception, and jump into catch{} directly

Catches thrown exception, print out error and propagate the exception

code in finally{} always runs in the end, ensures the closing of resources

What if input is changed?

```
String secret = “jfniksan or 1 = 1”;
Statement st = null;
ResultSet rst = null;
String query = "SELECT GRADE FROM GRADES WHERE SECRET = " + secret;
try {
        st = conn.createStatement();
        rs = st.executeQuery(query);
        while (rs.next()) {
                String grade = rs.getString(”GRADE");
                System.out.println(grade);
        }
}
catch(SQLException e) {
        System.err.println(e.getMessage());
        throw e;
}
finally {
        if(st != null) st.close();
        if(rst != null) rst.close();
}
```

It returns all the grades

PreparedStatement

```
String secret = “jfniksan OR 1 = 1”;
PreparedStatement st = null;
ResultSet rst = null;
String query = "SELECT GRADE FROM GRADES WHERE SECRET = ?”;
try {
      st = prepareStatement(query);
      st.setString(1, secret);
      rs = st.executeQuery();
      while (rs.next()) {
            String grade = rs.getString(”GRADE ");
            System.out.println(grade);
      }
}
catch(SQLException e) {
      System.err.println(e.getMessage());
      throw e;
}
finally {
      if(st != null) st.close();
      if(rst != null) rst.close();}
```

Set the parameter, index starts at 1 instead of 0

This way, the OR statement never runs, it becomes part of the parameter

PreparedStatement

- Another main benefit: it is pre-compiled
- Different inputs just need to get inserted as parameters

- Ordinary Statement: have to compile the statement every time it runs

Project Files

- TestFakebookOracle.java
  - Main function for running the program
  - Fill in *oracleUserName* and *password* --- your task
- FakebookOracle.java
  - **DO NOT MODIFY**
  - Data structures and abstract function declarations
- MyFakebookOracle.java
  - Subclass of FakebookOracle
  - Function implementations --- your task

Table Names

```
// DO NOT modify this constructor
public MyFakebookOracle(String u, Connection c) {
    super();
    String dataType = u;
    oracleConnection = c;
    // You will use the following tables in your Java code
    cityTableName = prefix+dataType+"_CITIES";
    userTableName = prefix+dataType+"_USERS";
```

Query 0

- Computes month of birth information
  - Month in which most friends were born
  - Month in which fewest friends were born
  - Retrieve the names of friends for the above two months

Query 0 (Part 1)

It is long. Please use an editor to go through the code.

Sample Queries

- Query1 (10 points): Find information about names

  The next query asks you to find information about user names, including (1) The longest last name, (2) The shortest last name, and (3) The most common last name.

  The following code snippet illustrates the data structures that should be constructed. However, it is up to you to add your own JDBC query to answer the question correctly.

- Query2 (10 points): Find "popular" friends

  The next query asks you to find information about all users who have strictly more than 80 friends in the network. Again, you will place your results into the provided data structures. The sample code in MyFakebookOracle.java illustrates how to do this.

Execution Using Eclipse

Demo?

- You can get Eclipse Classic at: http://www.eclipse.org/downloads/packages/eclipse-classic-371/indigosr1
- Create a project called "project2"
- Inside the project, create a package called "project2"
- Add 3 java files to the package
- Go to 'Project Settings' then 'Java Build Path', then click on the 'Libraries' tab, then 'Add External JAR' – add ojdbc.jar
- Note: Oracle JDBC Driver (ojdbc6.jar) requires Java JDK 1.6 (CAEN has JDK 1.6, check it by "java -version")

Execution From Command Line

Demo?

- Create a directory called “project2”
- Add 3 Java source files in this directory.
- Compile
  - javac project2/FakebookOracle.java project2/MyFakebookOracle.java project2/TestFakebookOracle.java
- Run:
  - java -cp "/path/to/ojdbc6.jar:” project2/TestFakebookOracle
  - /path/to/ is the path to the directory of ojdbc6.jar file
  - Don’t forget the quotation marks

Submission

- Turn in your code via CTools
- Submit only one file:
  - MyFakebookOracle.java
  - It should contain all of your code for queries 0-9
  - It should run with an unmodified version of FakebookOracle.java
  - (Refer to the spec for the detailed requirement)

Grading

- 9 Query tasks – 90 points
  - The queries will be auto graded.
  - You have the output from PUBLIC dataset for debugging purposes.
  - We will also test your queries on a second dataset, which is hidden from you.
  - 50% points for each dataset.
- Java/SQL programming style – 10 points
  - Use PreparedStatements
  - Follow try-catch-finally block
  - Try to do most of work inside the SQL not in Java.
    - (This will be actually easier for you to implement)
    - Check Ctools for hints