HW 7

Priya Rao Chagaleti

- Overview of HW 7
- Implementation details
- Quick demo
- Hints and common mistakes

Overview of HW 7

Setup

- Download the startup code from the homework page
- Untar it: tar –xvzf filename
- Modify dbconn.properties and save

IMPORTANT NOTE: Don’t forget to uncomment the lines after modifying!

From your terminal…

- Include the required JDBC jar files

- This tells Java how to connect to a SQL Azure database server, and needs to be in your CLASSPATH

- Compile the starter code
- Run the executable

Alternately, Eclipse setup also possible…

What you need to do – Part 1

- Goal: Create your customer database
- Turnin: the setup file which populates all the database tables
- Useful SQL commands: CREATE TABLE and INSERT
- Schema for the tables:

Customer: **id** (integer), a **login**, a **password**, a **first name**, a **last name**

Plan. **plan id** (integer), a **name**, the **maximum number** of rentals allowed, the **monthly fee**

Rental: **customer id**, **movie id** (from IMDB), a **status** that can be open, or closed, the **date and time** the movie was checked out, to distinguish multiple rentals of the same movie by the same customer.

When a customer first rents a movie, then you create an *open* entry in Rentals; when he returns it you update it to *closed* (you never delete it).

Additional relationships

- Each customer has exactly one rental plan.
- Each rental refers to exactly one customer.

IMPORTANT!

Create a clustered index on each table before you can insert values.

Execute setup.sql from m01rrgdwg2.database.windows.net

What you need to do – Part 2

- Goal: Write the Java application that your customers will use
- Turnin: Query.java
- Steps:
  - Make authentication logic work in the starter code.
  - Complete the provided IMDB movie search function, fixing a security flaw in it along the way.
  - Write a new, faster version of the search function.
  - Implement the remaining functions in Query.java to read and write customer data from your database.

What you need to do – Part 2a

Goal: Search command should return:

1. all movies whose title matches the string, case-insensitively
2. their director(s)
3. their actor(s)
4. an indication of whether the movie is available for rental (remember that you can rent the movie to only one customer at a time), or whether the movie is already rented by this customer (some customers forget; be nice to them), or whether it is unavailable (rented by someone else).

Good news: 1 and 2 are done for you!

What you need to do – Part 2b

Goal: Push the join to the database engine

How:

- Write a function called fastsearch, by using joins (or outer joins? you need to determine that!) computed in the database engine, instead of the dependent joins computed in Java by the search function
- Fastsearch should return:
  - the movie information (id, title, year)
  - its actors
  - its director

What you need to do – Part 2c

Goal: Complete the following transactions and don't let the user cheat:

1. The "login" transaction
2. The "print customer info" transaction
3. The "plan" transaction
4. The "rent" transaction
5. The "return" transaction

Concretely:

(a) when a customer requests to rent a movie, you may need to deny this request

(b) when a customer selects a "lower" plan (with fewer allowed movies), you may also need to deny this request (why?).

You can implement denying in many ways, but we strongly recommend using the SQL ROLLBACK statement.

What you need to do – Part 2d

- Goal: Avoid SQL injection
- Where: search and fastsearch

Implementation details

How to include SQL statements in Java

- Declare the SQL statement as a string in Query.java
- Also declare a corresponding preparedstatement
- Add the sql query in prepareStatements() by conn.prepareStatement(…)
- Execute the the preparedStatement by ".executeQuery" to get a "ResultSet" object
- Traverse the result set to get the required output

Good start to JDBC:
http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html

Useful functions

- While creating SQL strings - The ? denotes a parameter which will be substituted later, specify by *Preparedstatement*.setInt(i, parameter-you-want-to-replace-with)
- *Preparedstatement*. clearParameter to start afresh
- Retrieve the required attributes from your result set by .getString(i) or .getInt(i)
- *PreparedStatement*.executeUpdate() for insert/update statements where the query does not return a ResultSet

Common mistakes and useful hints ahead…

Customer Database Design

- If we have a primary key in the table, the clustered indices are created automatically.
- Minimum 8 tuples means 8 tuples in total not 8 tuples for each table.
- Make sure the statements in setup.sql are in the right order.
- The date field in rental database should be a valid SQL date type that includes time.

Java customer application

- Fastsearch is faster but it is for single word only.
- Print customerinfo = at least the customer name and the number of movies the user can still rent.

FastSearch explanation 1

- In fast search, you should really execute three queries only (forget about movie availability):
  - the first query should compute the movie metadata for all movies that match the keyword search,
  - the second query should find the directors for all movies that match the keyword search,
  - the third one should similarly find the actors for all the movies that match the keyword search.

  Execute each of these three queries separately. You then need to merge the results of the three queries *in* the Java code. The merge will be easier if you sort the results of the three queries.
- There is also a way to actually merge all of this info in a single SQL query but don't worry about that because it's similarly easy to write a very expensive single SQL query. Best to try writing three queries exactly.

FastSearch explanation 2

- "If you take a close look at the "search" method, you will see that the "search" method is iterating over all the mids returned when you search for the movie. Then, for each of the mids, it will issue 2 queries to the database: getting the actors for that mid, getting the director(s) for that mid. This results in number_of_mids * 2 queries when we do the "search" method, which is expensive. The idea of "fastsearch" is that we want to reduce the queries being issued to the database. This will give you only a few queries (way less than that of "search" method.) Therefore, it will be faster in the sense that we don't need to connect to the database and run a lot of queries."