Pizza Project: database, running SystemTest, domain objects

Using createdb.sql and dropdb.sql on pe07.cs.umb.edu or other Linux server

On pe07, we will have system programs sqlplus for Oracle and mysql for Mysql giving us convenient access to our databases.

To use createdb.sql for Oracle: cd to the pizza1/database directory and sqlplus user/user@//dbs3.cs.umb.edu/dbs3 < createdb.sql

or cd to the pizza1/database directory, log in to sqlplus and do SQL> @ createdb

For mysql access from topcat, using createdb.sql:
mysql -u user -D userdb -p < createdb.sql
or
mysql -u user -D userdb -p mysql> source createdb.sql

Running scripts for H2

• Our third database, H2, does not have a system program like sqlplus or mysql.
• However, it does have a program called RunScript inside its h2.jar, which is in lib-for-ant, a subdirectory of the database directory.
• After cd pizza1/database:
• Our third database, H2, does not have a system program like sqlplus or mysql.
• However, it does have a program called RunScript inside its h2.jar, which is in lib-for-ant, a subdirectory of the database directory.
• After cd pizza1/database:
• Use jar tf lib-for-ant/h2.jar to see RunScript listed as in package org.h2.tools.
• We can run createdb.sql with h2.jar as follows: put it on the classpath, execute that class:
  java -cp lib-for-ant/h2.jar org.h2.tools.RunScript
• This outputs help for that program, so we can compose the full command we need, with –url, other args:
• You can see the JDBC url for H2 given here as an argument, and the user "test".
• This long command is the basis of the shell script runH2Script.sh (or .cmd for Windows)
• The shell script runH2Script.sh contains:
  java -cp lib-for-ant/h2.jar org.h2.tools.RunScript –url jdbc:h2:~/test –user test –script $1 –showResults

The "$1" here allows us to use this shell file for all three scripts, for example dropdb.sql can be run by
runH2Script.sh dropdb.sql (just drop the .sh for Windows)

Automating our work: H2 case

• Similarly, runH2Script.cmd for Windows has "$1" in it instead of $1, for the same capability.
• These scripts for H2 work at home just as well as on pe07.
• Note that you will end up having two different H2 databases, one on pe07 (using a file in your login directory) and another at home (using a file in your home directory on that system).
• Another option is to use ant: see its build.xml in the database directory, but don’t worry about it if you don’t know ant. We have scripts for all cases…

Running DB scripts at home

Database Tables

Look at createdb.sql found in the database directory of the pizza1 project.

Because of database standardization (SQL92, entry level), the same file runs on Oracle, mysql, and h2.

[But care needs to be made not to use product-specific extensions such as Oracle’s varchar2 data type or mysql’s auto_increment]

Database application life cycle: create tables, use app, which queries, inserts, updates, deletes rows. Schema changes are rare and usually not done by the app itself.

Best practice: have SQL scripts to create the database, also drop the database, so it’s easy to reinit as needed.

We have createdb.sql and dropdb.sql. Also showdb.sql.

How to run these?

Running scripts for H2

• Our third database, H2, does not have a system program like sqlplus or mysql.
• However, it does have a program called RunScript inside its h2.jar, which is in lib-for-ant, a subdirectory of the database directory.
• After cd pizza1/database:
• Use jar tf lib-for-ant/h2.jar to see RunScript listed as in package org.h2.tools.
• We can run createdb.sql with h2.jar as follows: put it on the classpath, execute that class:
  java -cp lib-for-ant/h2.jar org.h2.tools.RunScript
• This outputs help for that program, so we can compose the full command we need, with –url, other args:

• You can see the JDBC url for H2 given here as an argument, and the user “test”.
• This long command is the basis of the shell script runH2Script.sh (or .cmd for Windows)
• The shell script runH2Script.sh contains:
  java -cp lib-for-ant/h2.jar org.h2.tools.RunScript –url jdbc:h2:~/test –user test –script $1 –showResults

The "$1" here allows us to use this shell file for all three scripts, for example dropdb.sql can be run by
runH2Script.sh dropdb.sql (just drop the .sh for Windows)
Automating script-running for Oracle, mysql

It turns out that the RunScript program from the H2 jar is a completely general utility to use JDBC to run scripts on databases so we have scripts to take advantage of this. For oracle and mysql, we need username and password and server's hostname to make the JDBC connection, and of course we want to automate that. See DatabaseSetup.html for details on setting up environment variables ORACLE_USER, etc., to specify the user and password as well as elements of the DB url.

runOracleScript showdb.sql [etc.] works by JDBC from anywhere to Oracle on dbs3, using ORACLE_USER env var, etc.
runMysqlScript showdb.sql [etc.] works by JDBC from anywhere to Mysql on pe07, using MYSQL_USER env var, etc.

Summary of ways to run createdb.sql, etc.

You have ways to load and drop all four databases you have for this course: use ant or...

- for Oracle on dbs3 and mysql on pe07: log in to pe07, run sqlplus or mysql as described on slide 3, or use the runOracleScript or runMysqlScript from anywhere that has the needed environment variables set up.
- for H2 at pe07: runH2Script on pe07
- for H2 at home: runH2Script at home

Use of environment variables: this is a commonly-used way to conveniently handle properties that vary across systems. We have ORACLE_SITE set to dbs3.cs.umb.edu on pe07 but set to localhost on home systems, so the JDBC url is ${ORACLE_SITE}:1521:dbs3 in general, which evaluates to "localhost:1521:dbs3" at home (for tunnel to dbs3.cs.umb.edu), and "dbs3.cs.umb.edu:1521:dbs3" at cs.umb.edu.

FYI: ant as DB tool

- Note that the job of creating the database is not part of Maven's expertise, which is building software using libraries. It's a perfect job for ant, since ant is command-driven, more general than maven, and has a useful "sql" ant task that knows how to use JDBC to talk to any database.
- The database directory has build.xml, ant's config file
- In build.xml, see targets show-oradb, drop-oradb, etc. You need the environment variables MYSQL_USER, MYSQL_PW, etc. for this.
- In the database directory, use "ant load-oradb" or "ant load-mysqldb" or "ant load-hsqldb", etc.

Looking at createdb.sql:

- Tables menu_toppings and menu_sizes just list possible toppings and sizes: ones on the menu presented to students.
- Note their UNIQUE constraints, so "Pepperoni" is listed only once here.
- Table pizza_orders holds orders. Orders can have multiple toppings, but only one size.
- To attach multiple toppings to one pizza order, there is a pizza_topping table with a FK to orders for the order this row is for. This is a N-1 (many-to-one) relationship.
- Note that pizza_topping does not have a UNIQUE constraint on topping_name. That allows multiple orders to have the same topping—see examples soon.

Design details of pizza orders and toppings

- Each order has its own rows in pizza_size and pizza_toppings to describe its details. That way, we can delete an order and its details without disturbing any other order.
- There is no FK from the pizza_topping topping_name to the menu_topping topping_name. This allows a topping to be deleted from the menu when it is no longer available. The program will use a topping_name from menu_toppings to create a particular topping for a new pizza.

Example of the pizza Database

All but one of the tables have "id" columns that have artificial values generated by the program, and id used as the PK.
Understanding the DB

From these 3 tables, we see that there are two pizza orders, order 1 with two toppings, Onions and Pepperoni, and size small, and order 2 with one topping, Pepperoni, and size large.

Order 1 is PREPARING (status=1), for room 5, and order 2 is BAKED (status = 2), for room 3. Both were made on day 1.

These orders were created with the help of menu_toppings and menu_sizes. Here are possible contents for them...

Menu tables: choices for new pizza orders

Intro to pizza1’s SystemTest

- SystemTest is an “end to end” test, or and “integration test”. It tests the whole system by calling the top layer, like an ordinary app does.
- Compare this to a “unit test”, that tests a single class.
- Full testing of a system involves both kinds of tests.
- Before running SystemTest, we are expected to load the database with createdb.sql.

• SystemTest execution: involves two pizza orders, and certain sequence of actions on them, including admin actions.

- But first, it reinitializes the DB, adds one menu size and one menu topping.
- Because of this, all runs of SystemTest end up with the same database contents, i.e., it’s reproducible, as a good test should be.

Running SystemTest on H2, Oracle

Build the system: cd pizza1; mvn clean package
Load the database: cd database; runH2Script createdb.sql (Windows)
on Mac/Linux: chmod +x *.sh, then runH2Script.sh createdb.sql
Run the jar: cd ..; runOnH2 SystemTest (Windows)
on Mac/Linux: chmod +x *.sh, then runOnH2.sh SystemTest

Running it with Oracle: Need environment vars ORACLE_SITE, etc., see DatabaseSetup.html

Mvn clean package if needed (not needed if just did above commands)
Load the database: cd database; runOracleScript createdb.sql (add .sh for Mac/Linux)
Run the jar: cd ..; runOnOracle SystemTest

SystemTest Actions

First pizza: for room 5
1. Ordered (status = 1, preparing)
2. Admin said pizza ready (status=2, baked)
3. Student received it (status =3, finished)

Second pizza: for room 1
1. Ordered (status=1)
2. Day ended, status=3

Database contents after whole run...

DB After SystemTest run
One more table: pizza_sys_tab

This table controls id values for inserts (Oracle doesn't support auto_increment like MySQL), and holds the current day number.

<table>
<thead>
<tr>
<th>next_menu_topping_id</th>
<th>next_menu_size_id</th>
<th>next_order_id</th>
<th>next_pizza_topping_id</th>
<th>next_pizza_size_id</th>
<th>current_day</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

During run of System Test:
• Insert one menu_topping, one menu_size, 2 pizza orders, 2 order toppings, 2 order sizes,
• Finish day, so current_day becomes 2.

SystemTest input file test.dat

File test.dat, input to SystemTest: tests both student and admin actions
ai admin initializes system (after "ant load-oradb" for example)
ss 5 student order from room 5
ss 5 student order from room 5
ss 5 status of orders from room 5 (answer— once PREPARING)
ss 5 status of orders from room 5 (answer—one, now BAKED)
ss 5 student receives the room-5 order (acknowledges its receipt)
ss 5 status of orders from room 5 (answer—one FINISHED)
ss admin report on in-progress orders (one PREPARING)
aip admin advances day (so pizza for room 1 is now FINISHED)
ss admin report on in-progress orders (none)

Notes on E-R design, cont.

Chen Notation There are several styles for E-R diagrams. The most classic and common in textbooks is the Chen notation, which puts a diamond on each relationship connection.
Murach uses something in between Chen and UML.
See pg. 663 for an example.
He would use
1:* to label PizzaOrder-Topping
1:1 to label PizzaOrder-PizzaSize
Domain Objects Again

Now that we understand how pizza orders are represented in the database, where they are “persistent”, i.e., have relatively long lifetimes, we turn back to their fleeting representations in memory while being involved in processing a request from the user.

Remember the layers. We were just looking at the database at the bottom. The DAO layer converts database data to Java objects that are used by the business programmers of the service layer, the heart of the application.

Our domain classes are POJOs, or Java Beans.

Simple POJO – plain old Java object -- has “properties”, each of which has a getter and / or a setter

Example POJO: PizzaOrder

The pair of public methods in PizzaOrder.java:

```java
public int getId()
public void setId(int id)
```

Defines “id” as an int property of PizzaOrder as a Java Bean or POJO.

- In fact, we could just have one of these two methods and still say id is a property.
- Full-fledged Java Beans also are expected to have a no-args constructor and implement Serializable. But POJOs don’t really need these.
- See pp. 174-175 in Murach on Java Beans.
- Also see Java tutorial at http://docs.oracle.com/javase/tutorial/javabeans

Relationships between POJOs

In PizzaOrder we see:

```java
public Set<PizzaTopping> getToppings() {…}
```

So we say there is a property of type Set<PizzaTopping>. Each PizzaOrder object has a Set of PizzaToppings. Each PizzaOrder object has a PizzaSize object describing its size.

Example:

```java
public boolean containsPizzaTopping(PizzaTopping topping)
```

But this is not of the right form to define a POJO “property”

```java
public PizzaSize getSize()
```

This defines a PizzaSize property. Each PizzaOrder has a PizzaSize object describing its size.

What does it mean to “navigate” between objects?

- It means to follow an object reference from one object to another
- For example, the PizzaOrder object has a field of type PizzaSize, so it has an object ref from PizzaOrder to PizzaSize. You can see this by looking at the object graph diagram.
- Thus we can navigate from PizzaOrder to its PizzaSize, and also from PizzaOrder to its various PizzaToppings.
- This setup allows navigation from a PizzaOrder to its PizzaSize, but there is no reference in a PizzaSize to allow navigation from a PizzaSize object to its PizzaOrder. Therefore, the arrow goes from PizzaOrder to PizzaSize, but not vice versa.
- Similarly, there is no reference in PizzaTopping back to PizzaOrder.
- We only implement navigation refs that we might actually use. We never need to get back from a PizzaSize to a PizzaOrder because we always have the related PizzaOrder on hand. So we keep PizzaSize and PizzaTopping really simple.

PizzaOrder object graph

At runtime, a PizzaOrder object has various simple properties such as room number, a PizzaSize object specifying its size, and a Set of PizzaTopping objects for its toppings.

But Set is just an interface — what is its object?

A Set can be implemented by a HashSet or a TreeSet, and these are real objects containing possibly multiple references (shown by arrows) to element objects, like this:

```
PizzaOrder ---------> SetImplementation ------------> PizzaTopping
|                     |                              |
| V                    | V                              |
| PizzaSize            | PizzaTopping
```

Note: It is essential that you understand object graphs and how to draw them. With pencil and paper, you can use ovals around the class names and proper arrows between them.

UML Class diagram (FYI)

The arrows show that navigation is possible from PizzaOrder to PizzaSize and the various PizzaToppings. See http://en.wikipedia.org/wiki/Class_diagram
Database relationships vs. object relationships

- We see directionality in object relationships, since object references go from one object to another.
- But relational database relationships are bidirectional, or really direction-agnostic.
- Different queries can be centered on one table, reach out with joins, etc. to other tables.
- So we can go to the database and find all the orders using Pepperoni, for example.
- (NoSQL databases like MongoDB don't have this property. A pizza order in MongoDB would typically be all in one record, including the set of toppings.)