Introductory Databases

Exercise 2

Generic Feedback

Overall this exercise was done very well.

First, let me explain some of the lessons I hope you learned.

1. Making the data constraints explicit (e.g. foreign keys, primary keys, range constraints) means that the DBMS will catch a lot of invalid data and prevent you from having an inconsistent DB.
2. Yes, you can (and should) try to catch a lot of these errors before it gets to the database. So, your Java (or whatever language you use) code should validate any input but if these constraints are also defined in the DB you will catch many more cases – cases that are very hard to reliably catch in your code.
3. So, you need to be aware that exceptions can be generated by your DB code and you should catch and handle these appropriately.
   a. You ought to be able to disable your own data validation to make sure that you can easily test whether this works correctly
4. Having some code that creates and populates a clean database is easy. You can write code that sets up the tables, adds some realistic data and a large amount of synthetic data without too much trouble.
5. If you do this you save yourself a lot of work in the long term.
6. You cannot test your code realistically without a large amount of data. For instance, you won’t see whether your user interaction scales to a large amount of data and you won’t be able to demonstrate how your code scales in terms of time or space.
7. Database operations can be slow. Re-using database connections saves a lot of time. You can also batch operations which will make them run much more quickly.
   a. A related point is that if you are ‘spring cleaning’ your data base then often people remove constraints first and then add them back afterwards – this will dramatically speed things up.
8. Check that your tests give the right answer!
9. If you can generate one report or insert one new value then you can easily extend this to the other cases – but doing it the first time is trickier.

Some positive points
Many of you did a great job and I was impressed by how many had done things such as:

1. Finding on line resources that could be used to populate the databases (e.g. large collections of jokes)
2. Some of you also found and used resources with APIs. This lets you programmatically interact with the resource. Great!
3. Using batches to speed up the population of the database.
4. Building nice UIs
5. And many other things

Problems
There were a few recurrent issues that are worth highlighting.

1. Leaving things too late – don’t expect to move code from one system to another without any problems. It might just be paths or fonts that you need to worry about. Just give yourself enough time to make sure it works properly.
2. It’s convenient for you to use your laptop for everything – but it’s then very easy to do things the wrong way which will only work because all resources are local. Any production system will have database server, web server, client etc. distributed, so make sure it can work like this.
3. Statement objects are usually inefficient and vulnerable to SQL injection. Don’t use them.
4. If you build an SQL string for each user input and then create a new prepared statement for it, then this nullifies the advantages of prepared statements. You should create a parameterised prepared statement and re-use it. This then means that all of the overheads are incurred once and your code is secure.
5. Make sure that you catch and handle exceptions appropriately
   a. ..... And don’t ‘catch Exception’
6. Check that your tests give the right answer! Some of you obviously didn’t!
7. ... and that’s why you need some realistic data so you can see that what you get is sensible.
8. You really must define primary key, foreign key and range constraints.
9. Using a floating point representation for something that is integer is a bad idea. So, for currency you could use the money type or you could represent everything as an integer number of pennies. The problem is that floating point representations are not exact. Yes, you can round values when you output them but there will still be rounding errors. If nothing else this can lead to inconsistencies and open you to salami slicing frauds.
10. Make sure your code is easy to read. This applies to your Java and to your SQL. If you don’t then it’s you that will pay the price because it’s harder to find bugs.

Overall, though, you all did very well.