

SQL Assignment 8 – part 2 [15 points] – due November 8 at 11:59PM

OBJECTIVE: Practice database tuning, i.e. given a database and a workload, tune the database to make the workload faster.

STARTER CODE: Download the archive code-sql8.zip from KSOL. This archive contains the following files:

- DatabaseGenerator.java
- TestQueries.java
- dbconn.config
- import-database.sql

RUN THE STARTER CODE

1. Edit `dbconn.config` to update your username and password.
2. Run the following commands to compile the two .java classes and to generate synthetic .sql files for the database we will use in this assignment (alternatively, you can also do this from Eclipse, using a setup similar to the setup in the previous SQL assignment – don't forget to specify the argument 100 for DatabaseGenerator):

```
javac DatabaseGenerator.java
javac TestQueries.java
java DatabaseGenerator 100
```

The following files will be created:

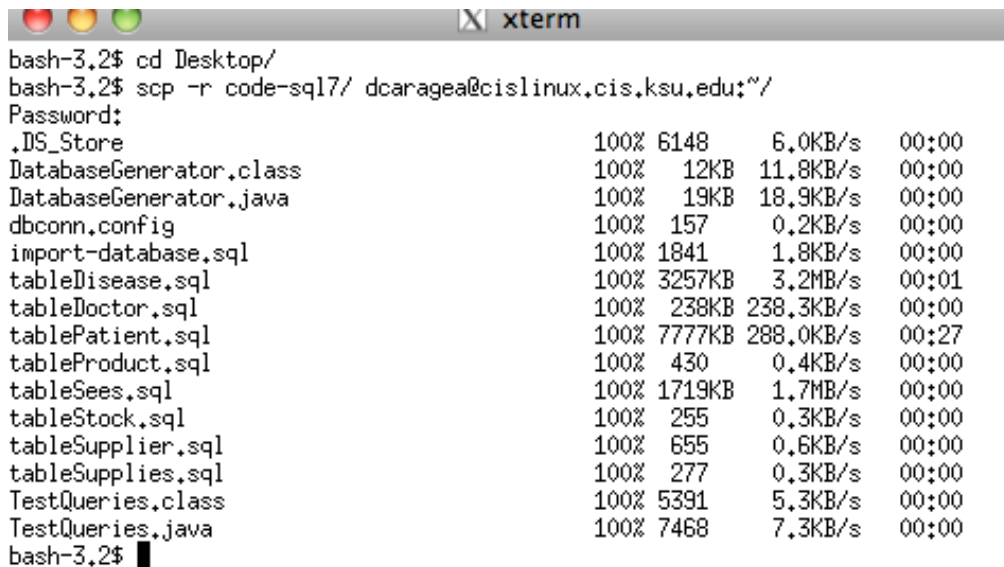
```
tablePatient.sql
tableDisease.sql
tableDoctor.sql
tableSees.sql
tableProduct.sql
tableStock.sql
tableSupplier.sql
tableSupplies.sql
```

```
bash-3.2$ cd Desktop/code-sql7
bash-3.2$ ls
DatabaseGenerator.java  dbconn.config
TestQueries.java        import-database.sql
bash-3.2$ javac DatabaseGenerator.java
bash-3.2$ javac TestQueries.java
bash-3.2$ ls
DatabaseGenerator.class  TestQueries.class      dbconn.config
DatabaseGenerator.java   TestQueries.java       import-database.sql
bash-3.2$ java DatabaseGenerator 100
bash-3.2$ ls
DatabaseGenerator.class  import-database.sql    tableSees.sql
DatabaseGenerator.java   tableDisease.sql       tableStock.sql
TestQueries.class        tableDoctor.sql        tableSupplier.sql
TestQueries.java         tablePatient.sql       tableSupplies.sql
dbconn.config           tableProduct.sql
bash-3.2$
```

3. Import the file `import-database.sql` into your MySQL DB – this will create eight new tables in your database (specifically, `Patient`, `Disease`, `Doctor`, `Sees`, `Product`, `Stock`, `Supplier`, `Supplies`).
4. Populate the tables using the `.sql` data files above. Note that you might not be able to import all the files through the `phpmyadmin` interface, as some of these files are larger than our usual files. You will need to import them (at least the largest ones) from command line – this is actually the method preferred by regular SQL users. To do that, you will first need to ssh to a CIS machine:

```
bash-3.2$ ssh dcaragea@cislinux.cis.ksu.edu
Password: enter-your-CIS-password
```

Copy your `.sql` data files to this machine, if they aren't there already. You can do this using `scp`.



```
bash-3.2$ cd Desktop/
bash-3.2$ scp -r code-sql7/ dcaragea@cislinux.cis.ksu.edu:~/
Password:
.DS_Store                                100% 6148      6.0KB/s   00:00
DatabaseGenerator.class                  100% 12KB      11.8KB/s  00:00
DatabaseGenerator.java                   100% 19KB      18.9KB/s  00:00
dbconn.config                           100% 157       0.2KB/s   00:00
import-database.sql                      100% 1841      1.8KB/s   00:00
tableDisease.sql                         100% 3257KB    3.2MB/s   00:01
tableDoctor.sql                          100% 238KB     238.3KB/s 00:00
tablePatient.sql                         100% 7777KB    288.0KB/s 00:27
tableProduct.sql                         100% 430       0.4KB/s   00:00
tableSees.sql                           100% 1719KB    1.7MB/s   00:00
tableStock.sql                           100% 255       0.3KB/s   00:00
tableSupplier.sql                        100% 655       0.6KB/s   00:00
tableSupplies.sql                        100% 277       0.3KB/s   00:00
TestQueries.class                        100% 5391      5.3KB/s   00:00
TestQueries.java                         100% 7468      7.3KB/s   00:00
bash-3.2$
```

To connect to the database, type the following command

```
dcaragea@cougar:~/code-sql7$ mysql -p -h mysql.cis.ksu.edu
Enter password: enter-your-MYSQL-password
```

Next, you need to specify the database that you want to use (this will be your user name):

```
mysql> use dcaragea
...
Database changed
```

Now, you can finally import your .sql files using the source command, as in the following example:

```
mysql> source tablePatient.sql
```

See screenshot below.

```
xterm

Password:
Last login: Tue Nov  1 21:03:08 2011 from ip70-179-155-252.fv.ks.cox.net
=====
      cougar, Linux 2.6.32-33-generic
      Kansas State University, Computing and Information Sciences
      Need help? Visit http://support.cis.ksu.edu/
=====
dcaragea@cougar:~$ cd code-sql7/
dcaragea@cougar:~/code-sql7$ ls
DatabaseGenerator.class  import-database.sql  tableSees.sql
DatabaseGenerator.java   tableDisease.sql     tableStock.sql
TestQueries.class        tableDoctor.sql      tableSupplier.sql
TestQueries.java         tablePatient.sql     tableSupplies.sql
dbconn.config            tableProduct.sql
dcaragea@cougar:~/code-sql7$ mysql -p -h mysql.cis.ksu.edu
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 399475
Server version: 5.0.60-log Gentoo Linux mysql-5.0.60-r1

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use dcaragea
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> source tablePatient.sql
Query OK, 100000 rows affected (6.81 sec)
Records: 100000  Duplicates: 0  Warnings: 0

mysql> source tableDisease.sql
Query OK, 132763 rows affected (5.47 sec)
Records: 132763  Duplicates: 0  Warnings: 0

mysql> source tableDoctor.sql
Query OK, 5000 rows affected (0.17 sec)
Records: 5000  Duplicates: 0  Warnings: 0

mysql> source tableSees.sql
Query OK, 119988 rows affected (3.71 sec)
Records: 119988  Duplicates: 0  Warnings: 0

mysql> source tableProduct.sql
Query OK, 19 rows affected (0.32 sec)
Records: 19  Duplicates: 0  Warnings: 0

mysql> source tableStock.sql
Query OK, 19 rows affected (0.01 sec)
Records: 19  Duplicates: 0  Warnings: 0

mysql> source tableSupplier.sql
Query OK, 9 rows affected (0.01 sec)
Records: 9  Duplicates: 0  Warnings: 0

mysql> source tableSupplies.sql
Query OK, 27 rows affected (0.35 sec)
Records: 27  Duplicates: 0  Warnings: 0

mysql> █
```

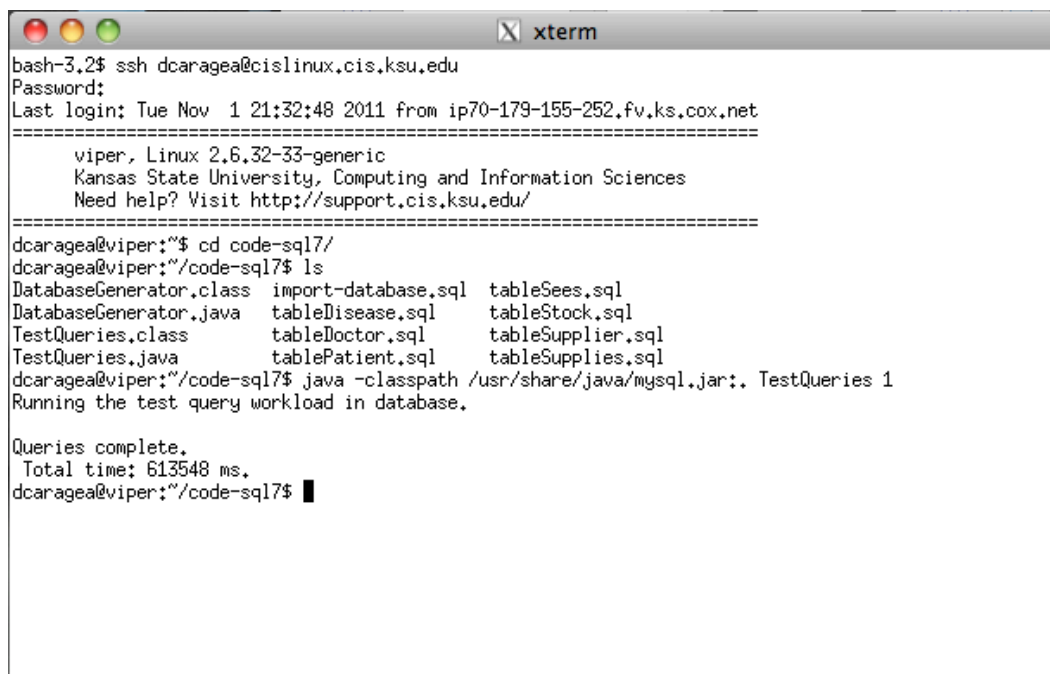
5. Run the test queries with the following command:

```
java TestQueries 1
```

Be aware that running the queries will take a few minutes.

The numbers 100 and 1 represent the scale factors; normally you wouldn't change them, but if you want you may increase them to test your database tuning solution more extensively.

You can run this through command line from your CIS account as shown in the following screenshot. **Don't forget to specify the classpath to the JDBC driver (see example in the screenshot below).**



```
bash-3.2$ ssh dcaragea@cislinux.cis.ksu.edu
Password:
Last login: Tue Nov  1 21:32:48 2011 from ip70-179-155-252.fv.ks.cox.net
=====
viper, Linux 2.6.32-33-generic
Kansas State University, Computing and Information Sciences
Need help? Visit http://support.cis.ksu.edu/
=====
dcaragea@viper:~$ cd code-sql7/
dcaragea@viper:~/code-sql7$ ls
DatabaseGenerator.class  import-database.sql  tableSees.sql
DatabaseGenerator.java  tableDisease.sql     tableStock.sql
TestQueries.class       tableDoctor.sql      tableSupplier.sql
TestQueries.java        tablePatient.sql     tableSupplies.sql
dcaragea@viper:~/code-sql7$ java -classpath /usr/share/java/mysql.jar:. TestQueries 1
Running the test query workload in database.

Queries complete.
Total time: 613548 ms.
dcaragea@viper:~/code-sql7$
```

These steps should go smoothly. If you get any error up to here, contact the instructor.

PROJECT DESCRIPTION

You are the database administrator for the Mercy Regional Health Center (MRHC). In this project, the MRHC database is generated by the Java program DatabaseGenerator.java, and imported by the script import-database.sql and the associated .sql synthetic data files.

The database has the following schema:

```
Patient(pid INTEGER, fname VARCHAR(20), lname VARCHAR(20), age
INT, street VARCHAR(20), city VARCHAR(10), zipcode VARCHAR(5))
Disease(pid INTEGER, disease VARCHAR(20))
```

```

Doctor(did INTEGER, fname VARCHAR(20), lname VARCHAR(20),
specialty VARCHAR(20))
Sees(pid INTEGER,did INTEGER)
Product(eid INTEGER, description VARCHAR(20))
Stock(eid INTEGER, quantity INTEGER)
Supplier(sid INTEGER, name VARCHAR(20), street VARCHAR(20), city
VARCHAR(10), zipcode VARCHAR(5))
Supplies(eid INTEGER,sid INTEGER)

```

The Patient, and Doctor tables contain information about the patients and doctors, respectively. The Disease table stores which patient has which disease(s). The Sees table says which patient is seeing which doctor(s). [Note: For the Patient table, it is usually better to use date-of-birth instead of age since the date-of-birth does not change every year. However, for the purposes of this problem, it is easier to use age.]

The Product table contains information about the medical supplies and equipment that the MRHC has. The Stock table maintains the remaining quantity of each product. The Supplier table contains information about the supplier companies, whereas the Supplies table says which products are supplied by which supplier companies. For each relation, the attributes that form the primary key are underlined. Additionally,

- Disease.pid is a foreign key referring to Patient.pid
- Sees.pid is a foreign key referring to Patient.pid
- Sees.did is a foreign key referring to Doctor.did
- Stock.eid is a foreign key referring to Product.eid
- Supplies.eid is a foreign key referring to Product.eid
- Supplies.sid is a foreign key referring to Supplier.sid

Database Tuning

The staff at MRHC access the database extensively by running TestQueries.java. In TestQueries.java, you'll see that there are six kinds of queries that they run very often (e.g., finding the list of doctors a specific patient has seen, finding the number of patients with a specific disease). You are in charge of administrating the database and optimizing it for performance. As you saw, TestQueries.java takes up to 10 minutes to run: you need to improve that. **Report the actual time (in ms) that it took for you.**

Create a file tune.sql where you put some star dust to sprinkle over the database that makes TestQueries run faster. After your star dust has settled, TestQueries will run much faster and your customers will love you. The MRHC is paying you a lot of money, so they would like the TestQueries to run a lot faster. Although they currently do not have any INSERT statements in their workload, the database should be able to support them without a significant cost. Therefore, **you are allowed to use no more than 6-8 indexes** (in addition to the default ones created by MySQL on primary keys). Also, **for each index, explain in 1-2 sentences (as a comment in tune.sql) why you use that particular index (as opposed to another one - if applicable). Report the time it took to run all the test queries, after indexing.**

In this assignment, you will only tune the performance by adding indexes. You should create indexes on one or more attributes. You may create several indexes per table. Be aware that, by default, MySQL builds a primary clustered index on the primary key of each table. You will not be able to create clustered indexes on attributes of your choice. **However, you are required to mention in your tune.sql file, as a comment, what indexes – if any - would give even better performance if they were clustered. In case you think that no indexes would improve the performance further if clustered, mention that explicitly in the tune.sql file, also as a comment.**

You should focus ONLY on optimizing the workload `TestQueries.java`. Some other queries may become slower as a result of your optimization, but that's OK as long as `TestQueries` runs faster.

Note that in practice, physical database tuning is more involved and includes table partitioning, materialized views, and others.

To create an index, use the following command:

```
CREATE INDEX index_name ON table_name(column_name1, column_name2, ..., column_namek);
```

Hint 1: For the type of DB engine that we are using (InnoDB) only BTREE indexes are allowed - and BTREE indexes are generally better than HASH indexes (similar performance even for point queries).

<http://dev.mysql.com/doc/refman/5.0/en/create-index.html>

So, you will have to create BTREE indexes only and they will be unclustered. You will not be able to create clustered indexes, as MySQL creates clustered indexes on the primary key by default and there can be only one clustered index per table.

You might notice that if you use something like:

```
Create INDEX ind_dname USING HASH ON Doctor(fname, lname);
```

you will not get an error. However, a BTREE index is created instead of a HASH index. This is a known bug reported at:

<http://bugs.mysql.com/bug.php?id=36869>

Hint 2: If you want to get an idea about what you should be aiming for (what numbers you should expect to get in this assignment), last year, some students managed to reduce the time from roughly 10 minutes to approximately 2500-3000 ms.

Hint 3: You may want to check what plan MySQL picks for a given query. For that you

may use EXPLAIN, e.g.

```
explain select count(*) from Supplier where city='Topeka';
```

For more information on indexes in MySQL, see:

<http://dev.mysql.com/doc/refman/5.0/en/mysql-indexes.html>

<http://dev.mysql.com/doc/refman/5.0/en/create-index.html>

<http://dev.mysql.com/doc/refman/5.0/en/innodb-index-types.html>

TESTING AND RUNNING YOUR SOLUTION For this question, you need to create a file tune.sql that you will turn-in. After you create the file, to test your solution, run the test queries again; compare the time between the run before you create indexes and the run after you create indexes (and report those times as comments in the tune.sql file).

If you need to start from scratch (i.e. remove your indexes and get back MySQL's default indexes), the drop command will be helpful.

Acknowledgments: The assignment has been adapted from a similar assignment at University of Washington.