DATA 514 Homework 3: Advanced SQL and Azure

Objectives: To practice advanced SQL. To get familiar with commercial database management systems (SQL Server) and using a database management system in the cloud (SQL Azure).

Assigned date: Jan. 22, 2019

Due date: Feb. 04, 2019. You have 2 weeks for this assignment.

What to turn in: `hw3-q1.sql`, `hw3-q2.sql`, etc (see below).

Assignment Details

This homework is a continuation of homework 2 but with three changes:

- The queries are more challenging
- You will get to use a commercial database system (i.e., no more SQLite 😃. SQLite simply cannot execute these queries in any reasonable amount of time; hence, we will use SQL Server, which has one of the most advanced query optimizers. SQL Server also has a very nice client application, SQL Server Management Studio, that you will get to use in this assignment.
- You will use the Microsoft Azure cloud.

Here is again the schema of the `Flights` database, for your reference:

```
FLIGHTS (fid int, year int, month_id int, day_of_month int,
         day_of_week_id int, carrier_id varchar(7), flight_num int,
         origin_city varchar(34), origin_state varchar(47),
         dest_city varchar(34), dest_state varchar(46),
         departure_delay double, taxi_out double, arrival_delay double,
         canceled int, actual_time double, distance double, capacity int,
         price double)
CARRIERS (cid varchar(7), name varchar(83))
MONTHS (mid int, month varchar(9))
WEEKDAYS (did int, day_of_week varchar(9))
```

We leave it up to you to decide how to declare these tables and translate their types in SQL Server. But make sure that your relations include all the attributes listed above. Note that SQL Server will complain if you try to ingest data and it needs to truncate it because VARCHAR fields are too small.

In addition, impose the following constraints:

- The primary key of the `FLIGHTS` table is `fid`.
- The primary keys for the other tables are `cid`, `mid`, and `did` respectively.
- `Flights.carrier_id` references `Carrier.cid`.
- `Flights.month_id` references `Months.mid`.
- `Flights.day_of_week_id` references `Weekdays.did`.

We provide the flights database as a set of plain-text data files in the linked `.tar.gz` archive. Each file in this archive contains all the rows for the named table, one row per line.

In this homework, you will do three things:

1. Create a database in the SQL Server database management system running as a service on Windows Azure.
2. Write and test the SQL queries below; keep in mind that the queries are quite challenging, both for you and for the database engine.
3. Reflect on using a database management system running in a public cloud.

A. Setting up an Azure SQL Database [0 points]

In this assignment, we want you to learn how to use an Azure SQL database from scratch. Your first step will thus be to setup a database in the Azure service and importing your data. This step may seem tedious but it is crucially important. We want you to be able to continue using Azure after the class ends. For this, you need to know how to use the system starting from nothing.

NOTE: These steps will take some time to complete, so start early!

Step 1: Create an Azure account and log in to Azure portal

You should have received an email from invites@microsoft.com inviting you to join an organization. Follow that link to apply the subscription to your account.

Afterwards, you will be forwarded to the Azure portal.

Step 2: Learn about Azure SQL Server

Spend some time clicking around, reading documentation, watching tutorials, and generally familiarizing yourself with Azure and SQL Server.

Step 3: Create a database

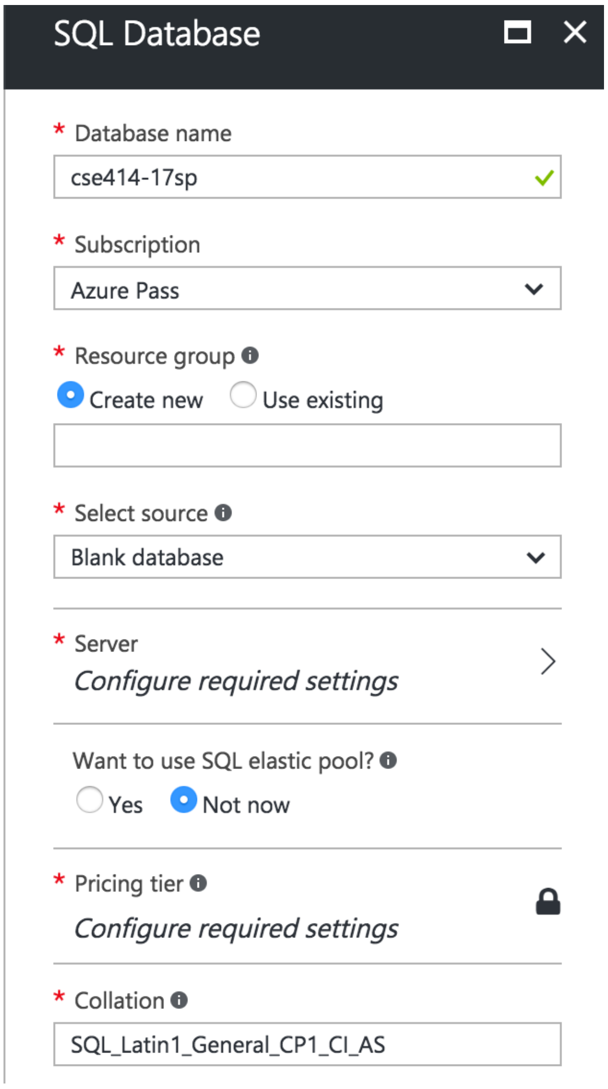
From the Azure portal, select "+ New" on the left, + New, then select "Databases", then select "SQL Database". This will bring up a panel with configuration options for a new DB instance.

Perform the following configuration:

- Choose a database name (e.g., "data514-19wi").
- Choose a name for the resource group that will be created (e.g., "myresourcegroup").
- Select a source: Blank database
- Create a new server by clicking on "Server" (it will say "Configure required settings"). A second panel will appear to the right; click on "Create a new server" and a third panel will appear to the right of this. Fill in the form as follows:
- Choose a name for the server (e.g., "fooBarSqlserver"). Unlike your database name, the server name must be unique across the universe of Azure SQL databases.

- Choose an admin login and password. (You will need this when access your database using tools other than the portal.)
- Set the location to "West US" or "West US 2".
- Make sure "Allow azure services to access server" is checked.
- SELECT.
- Change to a cheaper pricing tier. (The default is currently "Standard S2", which is too expensive.) To do this, click on "Pricing tier", which will open a pane with "Standard" selected. Find the slider for "DTUs", and turn it all the way down to 10. It should now say the monthly cost is only $15/month.
- APPLY
- Select "Pin to dashboard".

Your configuration should now look like this (again, replace the database name with data514-19wi):



SQL Database

* Database name: cse414-17sp

* Subscription: Azure Pass

* Resource group: [✓ Create new] Use existing

* Select source: Blank database

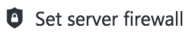
* Server: *Configure required settings*

Want to use SQL elastic pool? Yes [✓ Not now]

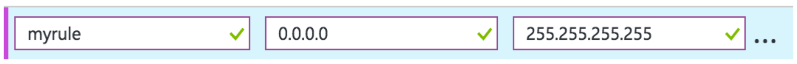
* Pricing tier: *Configure required settings*

* Collation: SQL_Latin1_General_CP1_CI_AS

Now click on the "Create" button to create this database. You will see a panel on the dashboard that says "Deploying SQL Database" while the database is being set up: this will take a while. Once that is done, it should open up a panel with details on your database. (If not, click on the panel for your SQL database to open it.)

Finally, click on the button near the top that says "Set server firewall" 

. You will need to change the settings before you can upload data. The easiest option is to add a rule that allows connections from any client, which you can do as follows:


myrule | 0.0.0.0 | 255.255.255.255

Be sure to click "Save" once you have added this rule.

Step 4: Try out the database

The simplest way to play with the database is using the built-in Query editor in the Azure portal. To launch this, click on the "Tools" button on top (to the left of the firewall settings button you clicked before).

Select "Query editor", and accept the warning that this is a preview feature (i.e., not necessarily what the final launched version will look like).

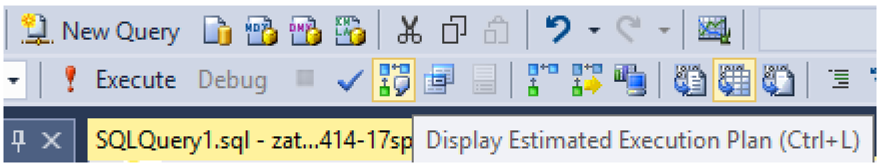
Click on the "Login" button and enter the username and password that you chose when you created your database in Step 3. Once you have done that, you can try entering SQL commands. Press the "Run" button to execute them.

Step 5: Use the database from SQL Server Management Studio

As convenient as the Azure portal Query editor is, you may want use SQL Server Management Studio (SSMS) in order to examine SQL Server's query execution plans. If you have Windows, SSMS may already be installed or you should be able to download and install it from Microsoft's web site. (For detailed instructions on installing SSMS, see this document.)

Another option is that you can launch a Windows VM directly in Azure. The process is similar to how we created a database instance above. Once your VM instance is running, open the panel for the instance and click on "Connect" to download a `.rdp` file. On a Mac, you can then connect to your Windows machine by using Microsoft Remote Desktop. Once connected, you can install SSMS using the link above.

When you launch SSMS, it will ask you to connect to a SQL server instance. Tell it the name of the server you created in Step 3 above. Then, select "SQL authentication" (rather than Windows authentication) and give it your username and password from Step 3. At that point, you should be able to see the tables of your database in the panel on the left side. Buttons to create a new query and execute it are in the middle of the menu bar, like what you see below:


New Query
Execute
Debug
SQLQuery1.sql - zat...414-17sp
Display Estimated Execution Plan (Ctrl+L)

Just to the right of the "Execute" button is an option to display the query execution plan. (It's highlighted yellow in the middle of the image above.) Once you do so, the query execution plan will appear in a panel on the bottom of SSMS, as shown here. We will discuss query plans in class and you may find it useful to examine them.

Now you are ready to move on to the next part of the assignment!

B. Ingesting Data (0 points)

Next, you will import all the data from HW2. Make sure that you execute your `CREATE TABLE` statements first so that the tables you will add tuples to already exist. Also, make sure that the types of the columns in the tables you created match the data.

To import data, you will need to use a utility called `bcp`, which should come with the command prompt on Windows. (Note that you can also use a Windows VM instance on Azure.) If you are using a unix system, you can use the freebcp utility, which is a part of freetds. On a Mac, you can install freebcp using the homebrew package manager:

```
brew install freetds
```

Make sure that homebrew installs freetds version at least 0.95.79. (You may need to run `brew update` and `brew upgrade` to get the most recent versions.)

The command to load `[file_name]` into the table `[table_name]` with `bcp` is:

```
bcp dbo.[table_name] in [file_name] -U [user_name]@[server_name] -P
[password] -S [server_name].database.windows.net -c -t [separator] -r
[newline] -d [database]
```

Here is an example with server `foosqlserver.database.windows.net`, user `bar`, and database name `data514-19wi`:

```
bcp dbo.Flights in flights-small.csv -U bar@foosqlserver -S
foosqlserver.database.windows.net -P "some-complicated-password" -c -t ","
-r "0x0a" -d data514-19wi
```

Usage of `freebcp` is similar to bcp except you need to use `-D` instead of `-d` when specifying the database and you need to use `\n` instead of `0x0a`. The same example from above becomes:

```
freebcp dbo.Flights in flights-small.csv -U bar@foosqlserver -S
foosqlserver.database.windows.net -P "some-complicated-password" -c -t ","
-r "\n" -D data514-19wi
```

Note: Mac users might have to change the double quotes to single quotes in the command above.

Note: When loading the `flights-small.csv` table, `bcp` may appear to hang toward the end; however, it probably did not hang. It is creating the index on the primary key, which will take what seems like an eternity but is approximately 20 min, and then it will complete.

As a sanity check, count the number of rows in `Flights` and make sure it matches the number of rows in the SQLite Flights table. If the count on SQL Server is too low, the data didn't all get imported. You will need to drop the table and try again.

If it continues to not work, you can try splitting the `flights-small.csv` file into multiple smaller files and importing them one at a time. On Mac, to split the file into 10 smaller files, run:

```
split -l 114868 flights-small.csv flights-small-part
```

On Windows, you can either use cygwin (to get the same command) or install something like HJ-Split.

C. SQL Queries (90 points):

Use SQL Server Management Studio

1. Click connect and login into your sql server
2. Choose your database and create new sql file, write your query, and execute it. See below for details.

For each question below, write a single SQL query to answer that question (you can use subqueries this time), and save your submission in individual files `hw3-q1.sql`, `hw3-q2.sql`, etc. For each query, add a comment to each query the number of rows your query returns, how long the query took, and the first 20 rows of the result (if the result has fewer than 20 rows, output all of them). You can find the query time on the right side of the yellow bar at the bottom of the window. You can simply copy and paste the first rows into the comment.

Remember to follow the SQL programming style rules from HW2, repeated here for your convenience:

- Give explicit names to all tables referenced in the `FROM` clause. For instance, instead of writing:

  ```
  select * from flights, carriers where carrier_id = cid
  ```

  write

  ```
   select * from flights as F, carriers as C where F.carrier_id = C.cid
  ```

  So that it is clear which table you are referring to.

- Similarly, reference to all attributes must be qualified by the table name. Instead of writing:

  ```
  select * from flights where fid = 1
  ```

  write

  ```
  select * from flights as F where F.fid = 1
  ```

This will be useful when you write queries involving self joins.

- The same applies to aggregates and projections as well. Instead of writing:

  ```
  select fid, avg(price) from flights group by fid
  ```

  write

  ```
  select F.fid, avg(F.price) from flights as F group by F.fid
  ```

We reserve the rights to deduct points for queries that don't conform to these style guidelines, even if they are correct.

Note that SQL Server interprets NULL values differently than sqlite! Try using it in a `WHERE` predicate and you will see the difference.

Now answer the following questions:

1. For each origin city, find the destination city (or cities) with the longest direct flight. By direct flight, we mean a flight with no intermediate stops. Judge the longest flight in time, not distance. (15 points)

   Name the output columns `origin_city`, `dest_city`, and `time` representing the the flight time between them. Do not include duplicates of the same origin/destination city pair. Order the result by `origin_city` and then `dest_city`.

   [Output relation cardinality: 334 rows]

2. Find all origin cities that only serve flights shorter than 3 hours. You can assume that flights with `NULL` actual_time are not 3 hours or more. (15 points)

   Name the output column `city` and sort them. List each city only once in the result.

   [Output relation cardinality: 109]

3. For each origin city, find the percentage of departing flights shorter than 3 hours. For this question, treat flights with `NULL actual_time` values as longer than 3 hours. (15 points)

   Name the output columns `origin_city` and `percentage` Order by percentage value. Be careful to handle cities without any flights shorter than 3 hours. We will accept both `0` and `NULL` as the result for those cities.

   [Output relation cardinality: 327]

4. List all cities that cannot be reached from Seattle though a direct flight but can be reached with one stop (i.e., with any two flights that go through an intermediate city). Do not include Seattle as one of these destinations (even though you could get back with two flights). (15 points)

   Name the output column `city`.

[Output relation cardinality: 256]

5. List all cities that cannot be reached from Seattle through a direct flight nor with one stop (i.e., with any two flights that go through an intermediate city). Warning: this query might take a while to execute. We will learn about how to speed this up in lecture. (15 points)

   Name the output column `city`.

   ~~(Hint: Do not forget to consider all cities that appear in a flight as an `origin_city`)~~ (You can assume all cities to be the collection of all `origin_city` or all `dest_city`)

   (Note: Do not worry if this query takes a while to execute. We are mostly concerned with the results)

   [Output relation cardinality: 3 or 4, depending on what you consider to be the set of all cities]

6. List the names of carriers that operate flights from Seattle to San Francisco, CA. Return each carrier's name only once. Use a nested query to answer this question. (7 points)

   Name the output column `carrier`.

   [Output relation cardinality: 4]

7. Express the same query as above, but do so without using a nested query. Again, name the output column `carrier`. (8 points)

D. Using a Cloud Service (10 points)

The DBMS that we use in this assignment is running somewhere in one of Microsoft's data centers. Comment on your experience using this DBMS cloud service. What do you think about the idea of offering a DBMS as a service in a public cloud?

Save your answer in a file called `hw3-d.txt` in the `submission` directory.

Submission Instructions

Answer each of the queries above and put your SQL query in a separate file. Call them `hw3-q1.sql`, `hw3-q2.sql`, etc (and `hw3-d.txt` for the last question). Make sure you name the files exactly as is. Put your files inside `hw3/submission`.