Big Data – Exercises

Spring 2020 – Week 1 – ETH Zurich

Prerequisites

In this exercise, you will brush-up the fundamental concepts of relational databases and SQL. If you havn't taken the Data Modelling and Databases course (or an equivalent bachelor course), we recommend you to read Garcia-Molina, Ullman, Widom: Database Systems: The Complete Book. Pearson, 2. Edition, 2008. (Chapters 1, 2, and 6)

Exercise 1: Query operations in SQL

1) Label each of the following SQL statements with its query type.

A. SELECT * FROM Posts WHERE Id = 123

B. SELECT Id, ParentId FROM Posts WHERE ParentId IS NOT NULL

C. SELECT u.Id, DisplayName
FROM Users AS u
JOIN Posts AS p ON u.id = p.OwnerUserId
GROUP BY u.Id, DisplayName

2) What makes SQL a declarative language and what advantages does that have?

3) What aspects of functional languages are present in SQL, and what advantages does that have?

Exercise 1: Solution

1)

A. SELECT Id, ParentId FROM Posts WHERE Id = 123

A selection is performed on the primary key.

B. SELECT Id, ParentId FROM Posts WHERE ParentId IS NOT NULL

A projection over two indexed columns.

C. SELECT u.Id, DisplayName
FROM Users AS u
LEFT JOIN Posts AS p ON u.id = p.OwnerUserId
GROUP BY u.Id, DisplayName

A join followed by projection (selecting columns from the newlly 'created' table) and a group-by.

2) We only describe *what* we want, not how this should be computed. We *declare* what our intent is. This shifts the implementation effort from the programmer to the database system. The hope is that the system has more information at hand, such as data size, data distribution, information about the hardware, in order to choose the best way to compute the result. This results into efficient computation with little effort from the programmer.

3) Results of a query can be used as input of another query, either in form of tables or in form of scalars. This makes SQL expressive.

Exercise 2: Explore the dataset

Here we will recall basic concepts from relational databases and try to illustrate them by example. First, some introductory questions:

1. What is a relational model?
2. In what logical shape is the data stored?

2. In what logical shape is the data stored?
3. What is a primary key and what is his purpose?
4. What does 'first normal form' refer to?

Now let us illustrate with few examples. For this we need to connect to the database we used in the first exercise. We repeat here the steps. We first set the credentials to connect.

In [ ]:

```
server='ethbigdata2019.database.windows.net'
user='student@ethbigdata2019'
password='BigData19'
database='beer.stackexchange.com'
```

Now, lets make sure that we can connect to the database for which we need the following scripts.

In [ ]:

```
!pip install pymssql==2.1.2
```

In [ ]:

```
!pip install ipython-sql
```

We can now load the extension and establish a connection to our database from above. Run the following cell and make sure the output says *"Connected: <connection string>"*.

In [ ]:

```
%load_ext sql
connection_string = 'mssql+pymssql://{user}:{password}@{server}:1433/{database}'.format(
      server=server, user=user, password=password, database=database)
print(connection_string)
%sql $connection_string
```

If all steps executed correctly, the cell bellow should produce result.

In [ ]:

```
%%sql
SELECT TOP 10 Id, DisplayName FROM Users
```

Now that you have established connection to the database, let us try to understand the it a bit better.

List of Tables

Run the following query which shows the content of a system table with the names of the tables. (This is specific to MS SQL Server.)

In [ ]:

```
%sql SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' %sql SELECT * FROM INFOR
MATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_CATALOG='beer.stackexchange.com'; TABLE_
CATALOG='$database';
```

List of attributes/columns

The following shows information about the attributes of the tables.

In [ ]:

```
%sql SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, DATA_TYPE \
    FROM INFORMATION_SCHEMA.COLUMNS \
    WHERE TABLE_CATALOG='beer.stackexchange.com' AND TABLE_SCHEMA <> 'sys'\
    ORDER BY TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, ORDINAL_POSITION;
```

For each table you can extract the primary key by running:

In [ ]:

```
%sql SELECT COLUMN_NAME \
    FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE \
    WHERE OBJECTPROPERTY(OBJECT_ID(CONSTRAINT_SCHEMA + '.' + QUOTENAME(CONSTRAINT_NAME)), 'IsPrimaryKey') = 1 \
       AND TABLE_NAME = 'Badges' AND TABLE_SCHEMA = 'dbo';
```

From the above returned results answer the following questions:

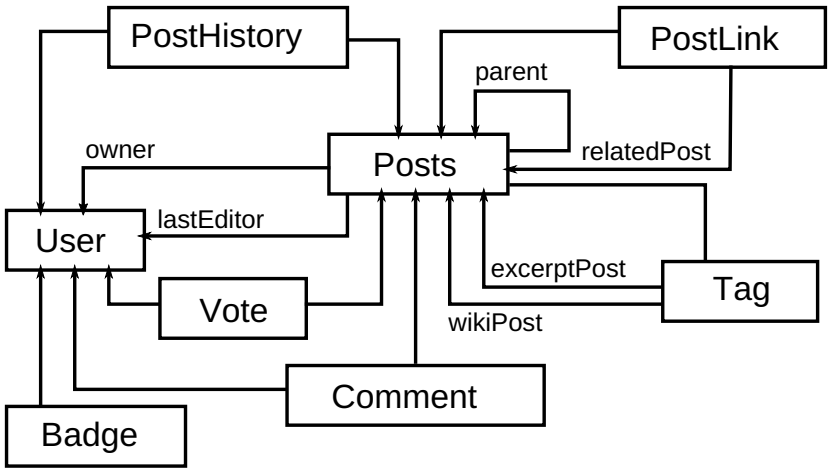
1. Which objects are modelled in the dataset and how do they relate (semantically) to each other?
2. Which are the primary keys for each table?

Where we got the data from (if interested)

- Info about the StackOverflow dataset
- Web interface to query it
- Link to the dataset (you don't need to do that!)

Exercise 2: Solution

1. The relational model is an approach to managing data where all data is represented in terms of tuples, grouped into relations.
2. In what logical shape is the data stored?
   - The data is stored as tables.
3. What is a primary key and what is his purpose?
   - The primary key is a column (attribute of data if you will) which uniquely identifies data records.
4. What does 'first normal form' refer to?
   - A table is in first normal form (1NF) if each column contains only atomic (undividable) values, and there are not repeating groups of columns. The latter refers to the fact that columns in the table are not allowed to relate to the same attribute.
5. Which objects are modelled in the dataset and how do they relate to each other?



6. Which are the primary keys for each table?
   - The primary keys in each of these tables has the name `Id`, and is of type `int` (Integer). The primary can have any name (i.e. does not have to be `Id` specifically), and can have any type, as long as it uniquely identifies each row in the table.

Exercise 3: Distribution of post scores

In this exercise, we want to find out how the scores of posts are distributed.

To start, write a query that selects the top 10 best-scored posts. (Note that `SELECT TOP 10` is MS SQL specific syntax. Other systems have different syntaxes to achieve the same thing.)

In [ ]:

```
%sql SELECT TOP 10 Id, Score, ViewCount, Body FROM Posts ORDER BY Score DESC;
```

We now know what the best posts look like. What about "more normal" posts? Write a query that counts (using the COUNT operation) the number of posts for each score.

In [ ]:

```
%sql SELECT Score, Count(*) FROM Posts GROUP BY Score ORDER BY Score DESC;
```

Did you use renaming in the query? If not try to rename the returned results from the count operation.

In [ ]:

```
%sql SELECT Score, Count(*) AS Count FROM Posts GROUP BY Score ORDER BY Score DESC;
```

The above query gives a very large result that is difficult to interpret. Let us write a query that rounds the scores of the posts to the nearest multiple of a constant that we define and counts the number of posts for each rounded score.

In [ ]:

```
%%sql
SELECT RoundedScore, Count(*) AS Count
FROM (
      SELECT ROUND((score+2.5)/5, 0) * 5 AS RoundedScore FROM Posts
   ) AS Rounded
GROUP BY RoundedScore
ORDER BY RoundedScore DESC;
```

Can you name the operation of calling a query from inside a query? What are the semantics of the GROUP By and ORDER By operations?

Using the right constant for the rounding, you can already get a better grasp of the distribution of scores. Here, we round each score to smallest integer multiple of 5 that is still strictly larger. (This is not the greatest way of rounding, but it will do for the purpose of this exercise.)

We will not execute the same query but from within a Python script. This allows us to send the SQL query results to Matplotlib and plot them.

In [ ]:

```
%matplotlib inline
import matplotlib.pyplot as plt

# Store the result of the query in a Python object (add your query here!)
result = %sql SELECT RoundedScore, Count(*) AS Count \
    FROM ( \
          SELECT ROUND((score+2.5)/5, 0) * 5 AS RoundedScore FROM Posts \
      ) AS Rounded \
    GROUP BY RoundedScore \
    ORDER BY RoundedScore DESC;

# Convert the result to a Pandas data frame
df = result.DataFrame()

# Extract x and y values for a plot
x = df['RoundedScore'].tolist()
y = df['Count'].tolist()

# Print them just for debugging
print(x)
print(y)

# Plot the distribution of scores
fig, ax = plt.subplots()
ax.bar(range(len(df.index)), y, tick_label=[int(i) for i in x], align='center')
ax.set_xlabel('Score')
ax.set_ylabel('Number of Posts')
```

Exercise 4: Impact of Score Count on Scores

We now want to find out whether the number of posts of the owner of a post has an influence of the score of the post. To that goal, write queries that answer the following questions:

1. What are the 10 users with the highest number of posts?
2. What is the average number of posts per user?
3. Which are the users with a number of posts higher than 10?
4. How many such users exist?

In [ ]:

```
%%sql
SELECT TOP 10 u.Id, DisplayName, COUNT(p.Id) PostCount
FROM Users AS u
JOIN Posts AS p ON u.id = p.OwnerUserId
GROUP BY u.Id, DisplayName
ORDER BY PostCount DESC
```

In [ ]:

```
%%sql
SELECT AVG(CAST(PostCount AS FLOAT)) AS AveragePostCount
FROM (SELECT u.Id, DisplayName, COUNT(p.Id) PostCount
      FROM Users AS u
      LEFT JOIN Posts AS p ON u.id = p.OwnerUserId
      GROUP BY u.Id, DisplayName) AS PostCount
```

The average post count per user is below 1, so every user who has made a single post is considered "active". Let's change the definition of the question to make it more sensible and consider a user active only if he/she has made at least 10 posts.

In [ ]:

```
%%sql
SELECT * FROM
(
   SELECT u.Id, DisplayName, COUNT(p.Id) PostCount
   FROM Users AS u
   LEFT JOIN Posts AS p ON u.id = p.OwnerUserId
   GROUP BY u.Id, DisplayName
) AS NumPostsPerUser
WHERE PostCount > 10
ORDER BY PostCount DESC
```

In [ ]:

```
%%sql
SELECT COUNT(*) FROM
(
   SELECT u.Id, DisplayName, COUNT(p.Id) PostCount
   FROM Users AS u
   LEFT JOIN Posts AS p ON u.id = p.OwnerUserId
   GROUP BY u.Id, DisplayName
) AS NumPostsPerUser
WHERE PostCount > 10
```

Own exploration

We recommend that you try to interact with the database and construct your own queries of different semantics and difficulty. Knowledge of SQL is very valuable in disciplines which have to deal with big data volumes stored as a relational data model: the predominant approach for data storage currently is use.

Recommended own work: Set up an SQL database with the StackOverflow dataset

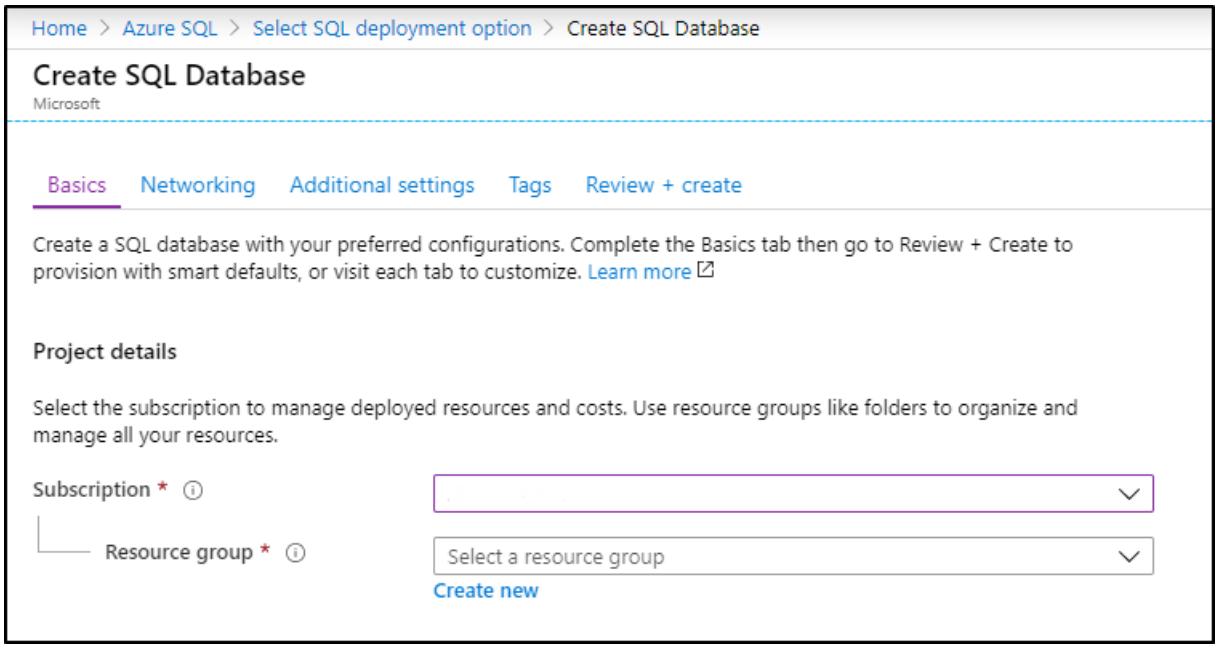
The loading will consist of the following steps:

1. Create your own SQL server.
2. Copy our StackOverflow export to your storage account.
3. Import the database dump into a new SQL database.
4. Test querying the server.

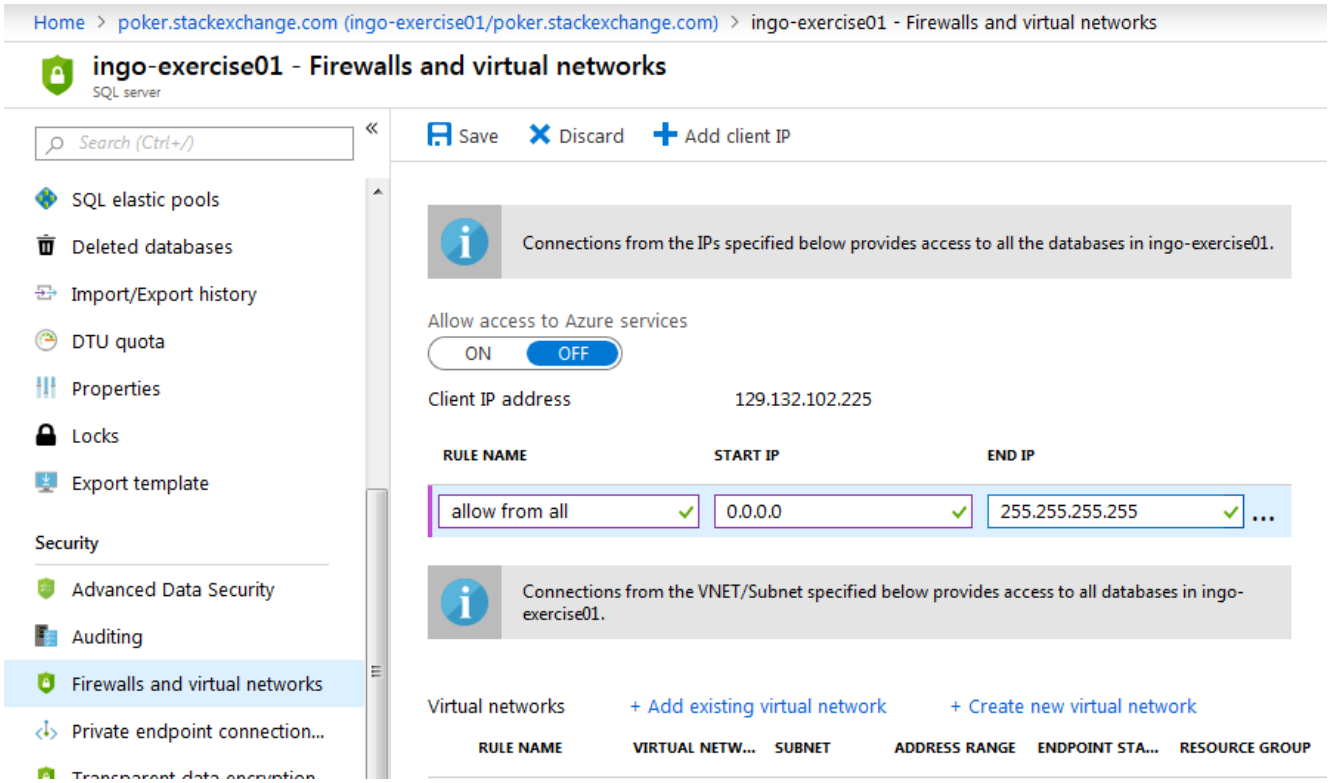
Step 1: Create your own SQL server.

(This is an adaptation of this tutorial.)

1. In the portal in the left menu, click on "Create a resource", search for "SQL server", then select "SQL server (logical server)" and finally "create".
2. Fill in the form below with values of your choice. Your form also asks for a resource group. Create a new resource group, which you may call "exercise01". Select "Pin to dashboard".



1. To check whether the database server has been created, go to "Resource groups" in the menu on the left, then open your new resource group ("exercise01") from the list. You should see the SQL server in the list.
2. Now to the settings of your database server, then open the firewall settings. Open the firewall for everyone by adding a rule named "open for all" with start IP "0.0.0.0" and end IP "255.255.255.255" in the following form. Click "save" to finish.

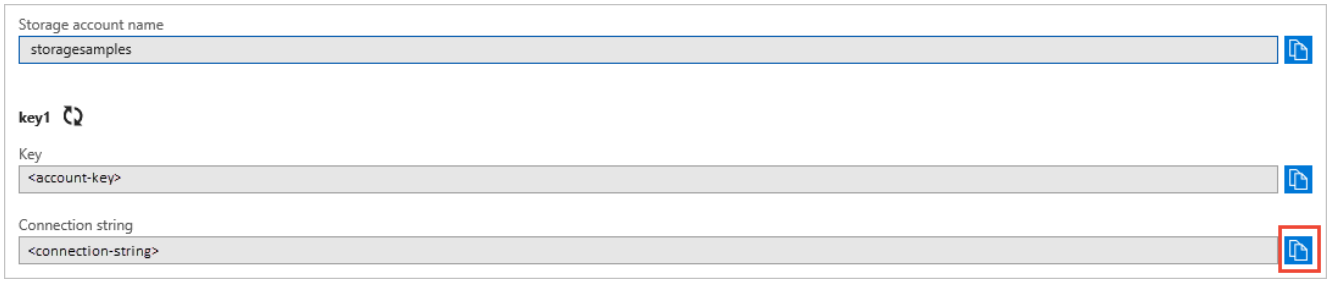


Transparent data encryption

No vnet rules for this server

Step 2: Copy our StackOverflow export to your storage account

1. First you need to create a storage account. (More about that next week!) To do that, go to "Storage accounts" in the left menu. Click on "Add" at the top of the blade and fill out the following form. Choose the "exercise01" resource group, select "Locally redundant storage (LRS)" as replication mode, and let all other values at their defaults.
2. Open the new storage account (you may need to wait a while before it has been created), go to "Access keys" (under "Settings") and copy one of its keys to the clipboard.



1. Paste the key and the account name here. The third variable holds the name of the container (a container is essentially a folder) that we will create a bit later. Run the cell.

In [ ]:

```
YOUR_ACCOUNT_KEY    = '...'
YOUR_ACCOUNT_NAME   = '...'
YOUR_CONTAINER_NAME = 'exercise01'
```

4. Install a management library for Azure storage.

In [ ]:

```
!pip install azure-storage==0.33.0
```

Now we can get a list of files in the storage container we created for you (again, next week, we will understand a bit better what is going on):

In [ ]:

```
from azure.storage.blob import BlockBlobService
from azure.storage.blob import PageBlobService
from azure.storage.blob import AppendBlobService
from azure.storage.blob import PublicAccess
from azure.storage.models import LocationMode
from azure.storage.blob import ContentSettings

# Name of storage account and container of the course
COURSE_ACCOUNT_NAME   = 'bigdataforeng2020'
COURSE_CONTAINER_NAME = 'exercise01'

# Connect to it
block_blob_service = BlockBlobService(account_name=COURSE_ACCOUNT_NAME)

# List all blobs in course's container
try:
    blobs = block_blob_service.list_blobs(COURSE_CONTAINER_NAME)
    for blob in blobs:
        print('Name: {} \t\t Type: {}'.format(blob.name,blob.properties.blob_type))
except:
    print("You don't have an access to %s "%(COURSE_CONTAINER_NAME))
```

5. Finally, we can copy the files from the course's container to a container we create on your account.

In [ ]:

```
# Connect to your account
```

```
your_service = BlockBlobService(account_name=YOUR_ACCOUNT_NAME, account_key=YOUR_ACCOUNT_KEY)

# Create a container where all files will be uploaded
try:
    status = your_service.create_container(YOUR_CONTAINER_NAME)
    if status == True:
        print("New container has been created")
    else:
        print("Container already exists")
except:
    print("Something went wrong.")

# Upload files to your storage from course's storage
try:
    blobs = block_blob_service.list_blobs(COURSE_CONTAINER_NAME)
    for blob in blobs:
        source = block_blob_service.make_blob_url(COURSE_CONTAINER_NAME,blob.name)
        your_service.copy_blob(YOUR_CONTAINER_NAME, blob.name, source)
    print("The files have been copied successfully")
except:
    print("Something went wrong.")

# List all files in your container
try:
    blobs = your_service.list_blobs(YOUR_CONTAINER_NAME)
    for blob in blobs:
        print('Name: {} \t\t Type: {}'.format(blob.name,blob.properties.blob_type))
except:
    print("Something went wrong.")
```

Step 3: Import the database dump into a new SQL database

Follow this guide with the SQL server and the `beer.stackexchange.com.bacpac` file you uploaded to your account **using the cheapest pricing tier available** ("B Basic"). In the form, leave the "Database name" field as it is; by default it will take on the name of the `.bacpac file`, and this is what we need.

Importing the database may take a while. You can check the progress in the alert bubble at the top right of the portal. You can also go the page of your SQL server and open "Import/Export history".

Step 4: Test querying the server

First, let's make sure that the connection library (PyMSSQL) and the SQL extension for Jupyter (ipython-sql) are still installed by running the next cells. If things don't work or you need more explanation what is going on, look at the notebook of last week.

In [ ]:

```
!pip install pymssql==2.1.2
```

In [ ]:

```
!git clone https://github.com/catherinedevlin/ipython-sql.git
!cd ipython-sql && LC_CTYPE="C.UTF-8" python setup.py -q install
```

Restart the kernel now if the extension was installed anew, then continue. Then fill in and run the following cell.

In [ ]:

```
# The name of your server is the one you chose in step 1
server='<your-db-name>.database.windows.net'

# The user is of form <your-admin-login>@<your-db-server-name>. You chose both in step 1.
# <your-db-server-name> is only the part *before* '.database.windows.net'
user='<your-admin-login>@<your-db-server-name>'

# The password is the one you chose in step 1
password='...'

# This is the name of the database.
# By default, it will coincide with the name of the .bacpac file that you used above.
# Warning: if this name contains dashes (-) in it, the subsequent code will not work
```

```
database='beer.stackexchange.com'
```

In [ ]:

```
%load_ext sql
connection_string = 'mssql+pymssql://{user}:{password}@{server}:1433/{database}'.format(
        server=server, user=user, password=password, database=database)
%sql $connection_string
```

Finally, the following cell should produce a result. (You may need to wait until the import process is complete.)

In [ ]:

```
%%sql
SELECT TOP 10 Id, DisplayName FROM Users;
```

If everything fails, use the credentials of our server from last week. If that fails as well, use the webinterface of StackOverflow to query live data.