DTS Exercise 1

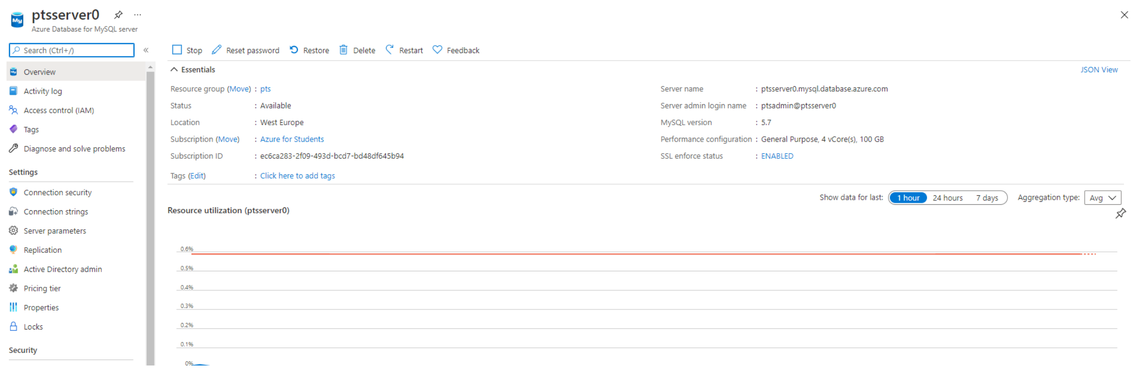
For this exercise, we are going to use the following technologies and tools:

- Microsoft Azure + Azure Database for MySQL
- MySQL Workbench 8.0.20 CE
- Docker tools (`https://docs.docker.com/get-docker/`)
- Hadoop 3.2.1
- Sqoop 1.4.7

1. Cloud databases

Cloud databases bring many opportunities and benefits for users, such as: no need for manual server administration, high availability, scalability, etc. MS Azure Cloud is a platform, which offers implementations of various DBMSs (e.g. SQL Server, MySQL, PostgreSQL) as cloud services (the so called "Database-as-a-service" model). For the exercise, we are going to use the student licence for a partially free access to Azure services. First, you need to activate your licence at `https://azure.microsoft.com/en-us/free/students/`. Login with your "student.um.si" email address.

To create and deploy a cloud MySQL server in the MS Azure platform you need to register/login into the Azure portal available at `https://portal.azure.com/`. After successful re-login, create a new resource. In the category "Databases" select "Azure Database for MySQL". Enter the necessary data and wait for a couple of minutes for the resource to deploy.



Resource group (Move)	: pts
Status	: Available
Location	: West Europe
Subscription (Move)	: Azure for Students
Subscription ID	: ec6ca283-2f09-493d-bcd7-bd48df645b94
Tags (Edit)	: Click here to add tags
Server name	: ptsserver0.mysql.database.azure.com
Server admin login name	: ptsadmin@ptsserver0
MySQL version	: 5.7
Performance configuration	: General Purpose, 4 vCore(s), 100 GB
SSL enforce status	: ENABLED

Slika 1: Successfully deployed MySQL server in Azure.

When the resource is ready, you need to configure remote connections to the server because by default the access to the server is protected with a firewall and it is not publicly available. On the left-side menu, select the "Connection security" option and, as a new firewall rule, add your current IP address. Save the changes.

Now you can test the connection to the deployed cloud MySQL server within MySQL Workbench. In the Azure portal for your resource, on the left-side menu, select the "Connection strings" option. You get a list of different connection strings that include data necessary for connecting to the server and/or given database (*hostname, port, username*. For instance, for a general JDBC connection, the connection string has the following form:

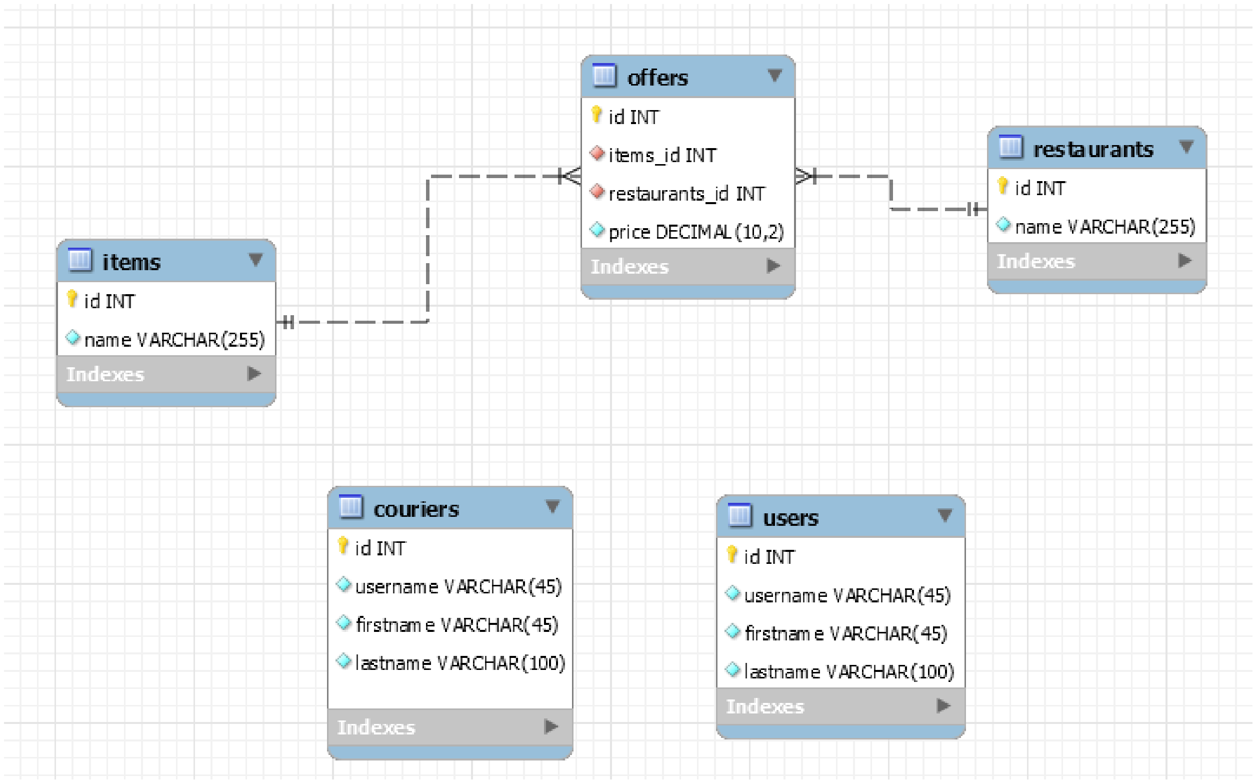
```
String url ="jdbc:mysql://ptsserver0.mysql.database.azure.com:3306/{your_database}?
useSSL=true&requireSSL=false"; myDbConn = DriverManager.getConnection(url,
"ptsadmin@ptsserver0", {your_password});
```

The required data is entered in Workbench when creating a new connection (hostname="ptsserver0.mysql.database.azure.com", username="ptsadmin@ptsserver0", password=your_password). When you successfully connect to the server, you will see only the "sys" schema (i.e. database) in the schema list.

Importing CSV data into MySQL database

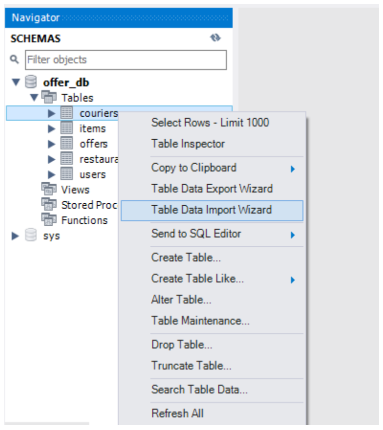
In this step, we are going to create a new MySQL database *offer_ db*, where we are going to store data about restaurant offers, i.e. data about restaurants, items and their prices in given restaurants as well as users and couriers.

Create a new database (schema) called *offer_ db* based on the ER model shown in Figure 2.



Slika 2: ER model of the *offer_ db* database.

After creating the tables, we are going to import data. To successfully import data from the local file system into the database, we need to prepare CSV documents with the same structure as the target database tables (e.g. the CSV file with data about items must have columns *id* and *name*). Download the prepared CSV documents from eŠtudij: *items.csv, offers.csv, users.csv, couriers.csv* and *restaurants.csv*. In Workbench, right click on each table and select option *Table Data Import Wizard*. Select the appropriate CSV document and finish data import into all tables.

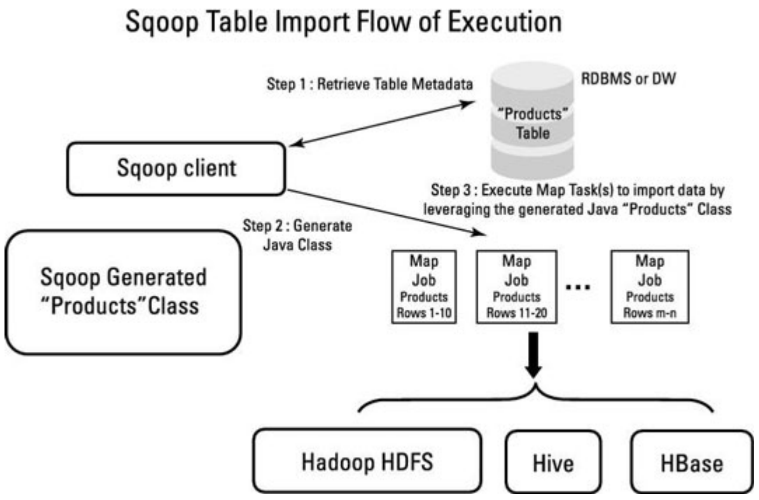


Slika 3: Importing data from a CSV document into MySQL by using the *Table Data Import* wizard.

Note: When you are not using your MySQL server in the Azure platform, stop the server to save on the resources usage!

2. Apache Sqoop

Apache Sqoop ("SQL-to-Hadoop & Hadoop-to-SQL") is a tool within the Hadoop ecosystem for migrating data between relational stores and Hadoop data storage solutions (HDFS, HBase). Sqoop can also be used for transferring data in batches between Hadoop and external data storage systems. For importing data into Hadoop we can use the "Sqoop import" tool.


Sqoop Table Import Flow of Execution
Step 1 : Retrieve Table Metadata
RDBMS or DW
"Products" Table
Sqoop client
Step 3 : Execute Map Task(s) to import data by leveraging the generated Java "Products" Class
Step 2 : Generate Java Class
Sqoop Generated "Products" Class
Map Job Products Rows 1-10
Map Job Products Rows 11-20
Map Job Products Rows m-n
Hadoop HDFS
Hive
HBase

Slika 4: Sqoop process of data import into Hadoop.

The prerequisites for Sqoop installation are installed Java 8 and Hadoop. For this exercises, we are going to use the Sqoop v.1.4.7 compatible with the selected Hadoop 3.2.1 version.

The necessary environment is prepared as a Docker image publicly available in the Docker Hub repository under the name "sestakmartina/hadoop-sqoop:0.3". If the Docker tools are installed on your computer, you can download the prepared *docker-compose* file from eŠtudij, which contains the definition of a Docker container as a service that includes the necessary Hadoop and Sqoop configurations.

First, we pull the Docker image from DockerHub with the following statement:

```
    docker pull sestakmartina/hadoop-sqoop:0.3
```

When you are positioned in the directory where that file is in the system terminal, you can start the Docker container with:

```
    docker-compose up
```

This statement is going to run a container with a virtual operating system in your system, which includes the Hadoop ecosystem tools. In the next step, we need the ID of this container, which we obtain by running "docker ps" in the terminal. Then we execute the following statement to enter into the terminal of the virtual system:

```
    docker exec -it <container name> bash
```

To start using HDFS commands in the command line, please execute the following command in the command prompt (**important:** this line needs to be executed upon every start of the Docker container!):

```
    export PATH=$PATH:$HADOOP_HOME/bin
```

Before starting the Hadoop cluster for the first time, you need to reformat the HDFS namenode by running:

```
$HADOOP_HOME/bin/hdfs namenode -format
```

This will delete all previous data from HDFS, prepare the necessary folder structure and load all JAR dependencies.

Now we can also run the components of the Hadoop cluster i.e. HDFS in YARN by executing:

```
    $HADOOP_HOME/sbin/start-all.sh
```

Importing data from a relational database into HDFS via Sqoop

If it is not there, we add the ID primary key column to the table with the *auto_ increment* property. To simplify the data import process into HDFS from the MySQL database, where data is separated across three different tables, we are first going to create a view in the relational database, where we are going to join all data from all tables:

```
create view v_offers as
    select offers.id as offer_id, restaurants.id as rest_id, restaurants.name
    as rest_name, items.id as item_id, items.name as item_name, price from offers
    join restaurants on offers.restaurants_id=restaurants.id
    join items on offers.items_id=items.id;
```

The result of this query can be directly saved into CSV file, which will be the source dataset for processing purposes later.

We check if the Hadoop cluster processes are running with the *jps* statement. In the process list, you should see 6 running processes if everything was started properly. Then, we execute the following query to import data from the cloud MySQL database into the local HDFS:

```
sqoop import --connect "jdbc:mysql://<hostname>:3306/<db_name>?
useSSL=true&requireSSL=false&serverTimezone=Europe/Amsterdam" --username <username>
--password "<password>" --table <view name> --target-dir="/pts" -m 1
```

In this statement you can see that the connection string from the Azure portal has been supplemented with the information about the timezone in order to avoid different system times. In addition to the username and password, Sqoop import also requires information about the source MySQL table and target directory in HDFS.

Important: In case you are using a local installation of MySQL Server to host your database, the *hostname* parameter in the Sqoop command must be changed to *host.docker.internal*!

If the import runs successfully, in the HDFS web UI, you will see the *pts* directory with a document in the form of *part-m-00000*. In the terminal, we can print the contents of this file with:

```
hadoop fs -cat /pts/part-m-00000
```

The result of this statement prints all data from the MySQL database. Repeat the steps for the *users* and *couriers* tables to finish importing all data into HDFS.

Note: When you are done with your work, stop Hadoop and SSH processes with the following statement:

```
$HADOOP_HOME/sbin/stop-all.sh
```