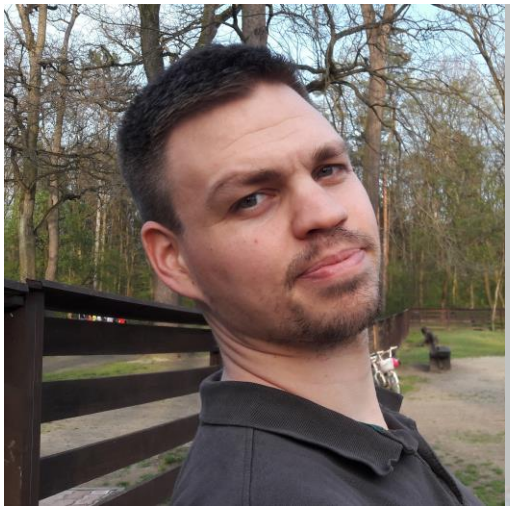


Dr. Hegedűs Péter, Dr. Ferenc Rudolf

Nagyméretű adatbázisok

Jelen tananyag a Szegedi Tudományegyetemen készült az Európai Unió támogatásával.

Projekt azonosító: EFOP-3.4.3-16-2016-00014

Szegedi Tudományegyetem
Cím: 6720 Szeged, Dugonics tér 13.
www.u-szeged.hu
www.szechenyi2020.hu

SQL Hadoop felett - Apache Hive

Összefoglalás

Ez az olvasólecke gyakorlati tudást ad a Hive SQL-alapú lekérdező réteg használatáról. Iránymutatást adunk arra nézve, hogyan lehet a Hive eszközt telepíteni, illetve virtualizált környezetben elindítani. Konkrét példákon keresztül bemutatjuk, hogyan lehet lokális fájlrendszerről, vagy a HDFS-en tárolt adatfájlokat betölteni, és hozzájuk SQL lekérdezéseket írni. Foglalkozunk a tabulált fájlok, valamint a struktúrát szövegek (pl. log bejegyzések és JSON) betöltésének módjaival a Hive kliens segítségével. E mellett betekintést kaphat az olvasó abba, hogyan lehet a Hive adattáblákat Java programkódból elérni a klasszikus JDBC API segítségével. A lecke fejezetei:

- 1. fejezet: **Apache Hive indítása Docker környezetben, tabulált szöveges állományok betöltése (olvasó)**
- 2. fejezet: **Struktúrált szöveg fájlok beolvasása (napló bejegyzések és JSON) beépített SerDe megvalósításokkal (olvasó)**
- 3. fejezet: **Apache Hive Java programból történő elérése JDBC API-n keresztül (olvasó)**

Téma típusa: **gyakorlati**
Olvasási idő: **50 perc**

📖 1. fejezet

Apache Hive indítása, tabulált szöveges adatok betöltése

Ahogy a kapcsolódó előadás olvasóleckéiben (`6e_BigData-sql-over-hadoop-SPOC`) már láttuk, az Apache Hive [1] egy SQL lekérdező és feldolgozó absztrakciós réteg a HDFS-en tárolt adatfájlok fölé. Ebben a fejezetben bemutatjuk hogyan lehet tabulált szövegfájlokat betölteni a Hive adattábláiba és azokon SQL lekérdezéseket végezni.

Telepítés/docker konténerek Hive-hoz

A Hive telepítése a megfelelő bináris csomag letöltéséből, kicsomagolásából, valamint a környezeti változók és konfigurációs állományok beállításából áll. Részletek a hivatalos dokumentációban [2] olvashatók. Mi a lokális gépre történő telepítés helyett egy előre előkészített docker container stack-et fogunk használni, mert a Hive használatához egy Hadoop klaszterre is szükség van. A következőkben bemutatott stack az összes szükséges docker image-t elindítja, ami szükséges a Hive azonnali használatához. A következő példák tetszőleges gépen futtathatók, ahol telepítve van a Dokcer környezet, valamint a Git verziókövető kliens.

A Hive stack indításához először töltsük le a docker leírókat és a teljes stack konfigurációt tartalmazó git repository-t a következő parancs segítségével:

```
$ git clone https://github.com/big-data-europe/docker-hive.git
```

A letöltött `docker-hive` mappa tartalmazza a `docker-compose.yml` stack konfigurációt, ebben láthatjuk milyen szolgáltatások (docker image-ek) indulnak el a Hive használatához:

- `namenode` - a Hadoop klaszter NameNode szervere
- `datanode` - egy darab Hadoop DataNode

- `hive-server` - a Hive szerever csomópontja
- `hive-metastore` - a Hive számára szükséges Metastore adatbázis, PostgreSQL alapú
- `presto-coordinator` - PrestoDB elosztott SQL lekérdező motor (ezt a gyakorlatokban nem fogjuk kihasználni, de Hive fölötti lekérdezések végrehajtására alkalmas)

A stack indításához lépjünk be a `docker-hive` mappába, és adjuk ki a következő docker parancsot:

```
$ docker-compose up -d
```

A stack sikeres indítása után a következő paranccsal ellenőrizhetjük, hogy minden container sikeresen elindult:

```
$ docker ps
```

Windows felhasználók figyelem!
Amennyiben Windows host-on indítjuk a docker stack-et, és a következő hibaüzenetet kapjuk "**ERROR: for namenode Cannot start service namenode: Ports are not available: listen tcp 0.0.0.0:50070: bind: An attempt was made to access a socket in a way forbidden by its access permissions.**", az azért van, mert a dokcer daemon bizonyos portokat lefoglal magának, amivel ütközik a docker konfigurációnk. A legegyszerűbb megoldás, ha módosítjuk a *docker-compose.yml* fájlt, és minden 50000-en felüli portszám esetén a mapping-et átírjuk, pl. - "50070:50070" helyett legyen - "30070:50070".

Tabulált szöveges adatok betöltése Hive-ba a lokális fájlrendszerről

Először lépjünk be a `hive-server` container-be és indítsunk egy terminált a következő parancs segítségével:

```
$ docker exec -it docker-hive_hive-server_1 bash
```

Először egy Hive-os példa szövegfájl, az `examples/files/employee_part.txt` tartalmát töltjük be egy Hive adattáblába. A szövegfájl tartalma a következő:

```
root@af7359462095:/opt/hive# cat examples/files/employee_part.txt
16|john|4000|USA
17|robert|2000|USA
18|andrew|4000|USA
19|katty|2000|USA
27|edward|4000|UK
29|alan|3000|UK
31|kerry|4000|UK
34|tom|3000|UK
35|zack|2000|UK
```

Ezt a fájl a HIve a lokális fájlrendszerből be tudja tölteni és elérhetővé tenni SQL lekérdezésekhez. Mielőtt ezt azonban meg tudná tenni, definiálnunk kell az adatok struktúráját, hasonlóan, ahogy egy relációs adatbázis esetén definiálja a tábla sémáját. Az adatmodell leíró nyelv (Data Definition Language - DDL) hasonló az SQL nyelvhez, bővebben a hivatalos dokumentáció DDL-t ismertető

részéből [4] tudunk tájékozódni. A fenti tabulált szövegfájl tartalomhoz hozzuk létre a következő adattáblát (`employee.hql`):

```
CREATE TABLE IF NOT EXISTS employee
(
    id INT,
    name STRING,
    salary INT,
    country STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
STORED AS TEXTFILE;
```

Négy mezőt definiálunk a szövegfájl oszlopainak megfelelően, majd a `ROW FORMAT DELIMITED` utasítással jelezzük, hogy tabulált szöveg az adatforrás. A `FIELDS TERMINATED BY '|'` utasítással megadjuk, hogy a mező elválasztó karakter a `|`. Az utolsó utasítás azt jelzi, hogy az adatok szöveges fájlként léteznek. Ezután készen állunk a Hive parancssor elindítására (a két lehetőség közül az újabb és preferált `beeline` klienst használjuk, aminek egyetlen JDBC URL paramétere van) és a tábla leíró betöltésére. Ehhez a docker container-en belül adjuk ki a következő parancsot:

```
root@af7359462095:/opt/hive# beeline -u jdbc:hive2://localhost:10000 -i
employee.hql
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hive/lib/log4j-slf4j-impl-
2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop-
2.7.4/share/hadoop/common/lib/slf4j-log4j12-
1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Connecting to jdbc:hive2://localhost:10000
Connected to: Apache Hive (version 2.3.2)
Driver: Hive JDBC (version 2.3.2)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Running init script employee.hql
0: jdbc:hive2://localhost:10000> CREATE TABLE IF NOT EXISTS employee
. . . . . .> (
. . . . . .>     id INT,
. . . . . .>     name STRING,
. . . . . .>     salary INT,
. . . . . .>     country STRING
. . . . . .> )
. . . . . .> ROW FORMAT DELIMITED
. . . . . .> FIELDS TERMINATED BY '|'
. . . . . .> STORED AS TEXTFILE;
No rows affected (1.957 seconds)
```

A Hive Metastore-ban eltárolódott az új táblánk leírója, most már kiadhatjuk az adat lokális fájlrendszerről való betöltését kezdeményező parancsot a beeline terminálból:

```
0: jdbc:hive2://localhost:10000> LOAD DATA LOCAL INPATH
'/opt/hive/examples/files/employee_part.txt' OVERWRITE INTO TABLE employee;
No rows affected (1.219 seconds)
```

Ha a parancs hibaüzenet nélkül lefutott, most már készen állunk arra, hogy a tabulált szövegként létező adaton SQL lekérdezéseket hajtsunk végre. Amennyiben a tábla már tartalmazott adatokat, azok az `OVERWRITE` kulcsszó miatt felülíródnak. Próbáljuk ki a következő lekérdezéseket:

```
0: jdbc:hive2://localhost:10000> SELECT * FROM employee;
+--------------+----------------+------------------+-------------------+
| employee.id  | employee.name  | employee.salary  | employee.country  |
+--------------+----------------+------------------+-------------------+
| 16           | john           | 4000             | USA               |
| 17           | robert         | 2000             | USA               |
| 18           | andrew         | 4000             | USA               |
| 19           | katty          | 2000             | USA               |
| 27           | edward         | 4000             | UK                |
| 29           | alan           | 3000             | UK                |
| 31           | kerry          | 4000             | UK                |
| 34           | tom            | 3000             | UK                |
| 35           | zack           | 2000             | UK                |
+--------------+----------------+------------------+-------------------+
9 rows selected (0.257 seconds)

0: jdbc:hive2://localhost:10000> INSERT INTO employee VALUES(99, "peter", 10000,
"USA");
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the
future versions. Consider using a different execution engine (i.e. spark, tez) or
using Hive 1.X releases.
No rows affected (1.859 seconds)
0: jdbc:hive2://localhost:10000> SELECT * FROM employee;
+--------------+----------------+------------------+-------------------+
| employee.id  | employee.name  | employee.salary  | employee.country  |
+--------------+----------------+------------------+-------------------+
| 99           | peter          | 10000            | USA               |
| 16           | john           | 4000             | USA               |
| 17           | robert         | 2000             | USA               |
| 18           | andrew         | 4000             | USA               |
| 19           | katty          | 2000             | USA               |
| 27           | edward         | 4000             | UK                |
| 29           | alan           | 3000             | UK                |
| 31           | kerry          | 4000             | UK                |
| 34           | tom            | 3000             | UK                |
| 35           | zack           | 2000             | UK                |
+--------------+----------------+------------------+-------------------+
10 rows selected (0.165 seconds)

0: jdbc:hive2://localhost:10000> SELECT country, AVG(salary) as avg_salary FROM
employee GROUP BY country;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the
future versions. Consider using a different execution engine (i.e. spark, tez) or
using Hive 1.X releases.
+----------+-------------+
| country  | avg_salary  |
+----------+-------------+
| UK       | 3200.0      |
```

```
| USA       | 4400.0       |
+-----------+--------------+
2 rows selected (1.341 seconds)
```

A második lekérdezés warning üzenete beszédes. Kiderül belőle, hogy a Hive MapReduce job-bá konvertálja az SQL-ben megfogalmazott lekérdezéseket, és az előálló eredményt adja vissza. Az is kiderül az üzenetből, hogy ez a módszer már túlhaladott (és nem a leghatékonyabb), a legújabb Hive használatakor érdemes megfontolni az Apache Spark [5] vagy Apache Tez [6] végrehajtó motor használatát a klasszikus MapReduce helyett.

Tabulált szöveges adatok betöltése Hive-ba HDFS-ről

Nem csak a Hive lokális fájlrendszeréből tudunk adatokat betölteni és lekérdezni, hanem természetesen a HDFS-en már létező fájlokra is ún. külső (external) táblákat tudunk illeszteni. Ehhez példaként használjuk fel a `sales_enrties.csv` állományt a `2g_BigData-data-transform-SPOC` gyakorlatról. Először másoljuk be a csv fájlt a lokális gépünkről a NameNode-ot futtató docker container-be, majd onnan a HDFS-re az alábbi parancsok segítségével:

```
$ docker cp sales_entries.csv docker-hive_namenode_1:/
$ docker exec -it docker-hive_namenode_1 bash
root@fcac992a9a07:/# hadoop fs -mkdir /csv_data
root@fcac992a9a07:/# hadoop fs -put sales_entries.csv /csv_data
```

Hozzuk létre a `sales_entries.csv`-nek megfelelő táblát Hive-ban:

```
root@af7359462095:/opt/hive# beeline -u jdbc:hive2://localhost:10000
0: jdbc:hive2://localhost:10000> CREATE EXTERNAL TABLE sales_entries (
. . . . . .> pid STRING,
. . . . . .> offer_date STRING,
. . . . . .> offer_text STRING
. . . . . .> )
. . . . . .> ROW FORMAT DELIMITED
. . . . . .> FIELDS TERMINATED BY ';'
. . . . . .> STORED AS TEXTFILE
. . . . . .> LOCATION 'hdfs://namenode:8020/csv_data'
. . . . . .> TBLPROPERTIES ("skip.header.line.count"="1");
No rows affected (0.129 seconds)
```

Az utasítás majdnem ugyanúgy néz ki mint az imént. A különbség (a más elválasztó karakteren kívül), hogy az `EXTERNAL` kulcsszót használtuk az elején, és a végén egy `LOCATION` paranccsal megadtuk az előbb feltöltött csv fájlt tartalmazó mappa elérési útvonalát HDFS-en (a `TBLPROPERTIES` segítségével beállítottuk, hogy a fejléc sort hagyja ki a csv feldolgozásból). Ez azt jelenti a Hive számára, hogy az adatok már léteznek, azt nem kell neki létrehozni HDFS-en az alapértelmezett helyre. Nincs is ezután szükség a `LOAD` utasításra, a csv fájl azonnal lekérdezhető SQL-lel:

```
0: jdbc:hive2://localhost:10000> SELECT * FROM sales_entries;
+--------------------+---------------------------+--------------------------+
| sales_entries.pid  | sales_entries.offer_date  | sales_entries.offer_text |
+--------------------+---------------------------+--------------------------+
| 3344556677         | 2020-01-05                | "terrific offer"         |
| 1122334455         | 2017-03-02                | ""                       |
| 2233445566         | 2019-10-12                |                          |
+--------------------+---------------------------+--------------------------+
3 rows selected (0.122 seconds)
```

📖 2. fejezet

Egyéb strukturált szövegfájlok betöltése Hive-ba

Szövegfájlok betöltése reguláris kifejezéssel

A fentiekben bemutattuk a tabulált szövegfájlok betöltésének módjait. Sokszor előfordul azonban, hogy a lekérdezni kívánt adatfájl nem tabulált, de valamilyen egységes formában tartalmaz információt. Tipikus példa a log fájlok, ahol minden sor azonos felépítésű, de az adatok nem tagolódnak mezőkre. A Hive lehetőséget ad az ilyen fájlok közvetlen betöltésére és SQL lekérdezésére is reguláris kifejezések segítségével (ezen kívül több fajta szerializáló/deszerializáló metódust is tartalmaz beépítve [7]). A fájl típusának meghatározásakor a beépített `RegexSerDe` módot kell választani, és egy reguláris kifejezéssel megadni, hogy a szövegfájl egyes részei milyen mezők értékeire képződjenek le. Tekintsük az alábbi példa log fájlt (`sample.log`):

```
64.242.88.10 - - [07/Mar/2004:17:09:01 -0800] "GET
/twiki/bin/search/Main/SearchResult?scope=text&search=Joris%20*Benschop[^A-Za-z]
HTTP/1.1" 200 4284
164.42.82.14 - - [08/Mar/2004:13:11:21 -0800] "GET
/twiki/bin/search/Main/SearchResult?scope=text&search=Joris%20*Benschop[^A-Za-z]
HTTP/1.1" 200 4284
64.242.88.10 - - [17/Mar/2004:07:55:11 -0800] "GET
/twiki/bin/search/Main/SearchResult?scope=text&search=Joris%20*Benschop[^A-Za-z]
HTTP/1.1" 200 4284
```

Ha ebből a logból minden bejegyzés esetén szeretnénk kinyerni az IP címet, a log bejegyzés napját és az elért URL címét, az alábbi Hive tábla definíciót használhatjuk (a `sample.log` fájlt másoljuk fel a Hive container-be és lokális adatként töltsük be a táblába):

```
CREATE TABLE logs
(
    ip STRING,
    day STRING,
    url STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES ("input.regex" = "(\\S+).*?\\[(.*?)\\s.*?(/.*?)\\?.*");

LOAD DATA LOCAL INPATH '/opt/hive/sample.log' OVERWRITE INTO TABLE logs;
```

A reguláris kifejezés három illeszkedő minta része alkotja majd az egyes sorok mezőinek értékeit. A fenti adatok betöltése után a táblánk tartalma a következő:

```
0: jdbc:hive2://localhost:10000> SELECT * FROM logs;
+---------------+----------------------+-------------------------------------+
|    logs.ip    |       logs.day       |               logs.url              |
+---------------+----------------------+-------------------------------------+
| 64.242.88.10  | 07/Mar/2004:17:09:01 | /twiki/bin/search/Main/SearchResult |
| 164.42.82.14  | 08/Mar/2004:13:11:21 | /twiki/bin/search/Main/SearchResult |
| 64.242.88.10  | 17/Mar/2004:07:55:11 | /twiki/bin/search/Main/SearchResult |
+---------------+----------------------+-------------------------------------+
3 rows selected (0.124 seconds)
```

Adatok betöltése JSON fálból

Létezik beépített JSON `SerDe` megoldás is, amivel közvetlenül JSON fájlból tudunk adattáblát feltölteni. Hozzunk létre egy táblát a `personal_entries.json` állományban tárolt adatokhoz (lásd `2g_BigData-data-transform-SPOC` gyakorlati anyag):

```
CREATE TABLE personal_entries
(
    pid STRING,
    name STRING,
    gender STRING,
    last_contacted STRING,
    birth_year INT,
    useless_info ARRAY<INT>
)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe';

LOAD DATA LOCAL INPATH '/opt/hive/personal_entries.json' OVERWRITE INTO TABLE
personal_entries;
```

Ezután le is kérdezhetjük a JSON-ben lévő adatokat:

```
0: jdbc:hive2://localhost:10000> SELECT pid, name, gender, last_contacted,
birth_year, useless_info FROM personal_entries;
+-------------+-------------+---------+----------------------+-------------+----
-----------+
|     pid     |    name     | gender  |    last_contacted    | birth_year  |
useless_info  |
+-------------+-------------+---------+----------------------+-------------+----
-----------+
| 1122334455  | John Doe    | M       | 2017-03-02T11:43:00  | 1987        |
[1,2,3]       |
| 2233445566  | Jane Doe    | F       | 2019-10-12T15:22:34  | 1965        |
[2,3,4]       |
| 3344556677  | John Smith  |         | 2020-01-05T08:27:12  | 1999        |
[1,2,3,4,5]   |
+-------------+-------------+---------+----------------------+-------------+----
-----------+
3 rows selected (0.115 seconds)
```

Sőt, `JOIN` művelettel akár össze is kapcsolhatjuk a `sales_entries.csv` tartalmával, hiszen azt is betöltöttük egy Hive táblába:

```
0: jdbc:hive2://localhost:10000> SELECT p.pid, p.name, p.gender,
p.last_contacted, p.birth_year, s.offer_date, s.offer_text FROM personal_entries
AS p, sales_entries AS s WHERE p.pid=s.pid;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the
future versions. Consider using a different execution engine (i.e. spark, tez) or
using Hive 1.X releases.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hive/lib/log4j-slf4j-impl-
2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop-
2.7.4/share/hadoop/common/lib/slf4j-log4j12-
1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Execution log at: /tmp/root/root_20200823120243_de13b96b-1df6-4943-8017-
3a816cdc3c05.log
2020-08-23 12:02:47     Starting to launch local task to process map join;
maximum memory = 477626368
2020-08-23 12:02:48     Dump the side-table for tag: 1 with group count: 3 into
file: file:/tmp/root/c52b52c9-7b77-4daa-9a8b-07650839de85/hive_2020-08-23_12-02-
43_026_891102812468550297-10/-local-10004/HashTable-Stage-3/MapJoin-mapfile11-
-.hashtable
2020-08-23 12:02:48     Uploaded 1 File to: file:/tmp/root/c52b52c9-7b77-4daa-
9a8b-07650839de85/hive_2020-08-23_12-02-43_026_891102812468550297-10/-local-
10004/HashTable-Stage-3/MapJoin-mapfile11--.hashtable (401 bytes)
2020-08-23 12:02:48     End of local task; Time Taken: 1.14 sec.
+-------------+-------------+-----------+----------------------+---------------
+---------------+--------------------+
|    p.pid    |   p.name    | p.gender  |   p.last_contacted   | p.birth_year  |
s.offer_date  |   s.offer_text     |
+-------------+-------------+-----------+----------------------+---------------
+---------------+--------------------+
| 1122334455  | John Doe    | M         | 2017-03-02T11:43:00  | 1987          |
2017-03-02    | ""                 |
| 2233445566  | Jane Doe    | F         | 2019-10-12T15:22:34  | 1965          |
2019-10-12    |                    |
| 3344556677  | John Smith  |           | 2020-01-05T08:27:12  | 1999          |
2020-01-05    | "terrific offer"   |
+-------------+-------------+-----------+----------------------+---------------
+---------------+--------------------+
3 rows selected (6.779 seconds)
```

📖 3. fejezet

Hive programozása Java-ban JDBC API-n keresztül

Mivel a Hive egy JDBC driver-t is nyújt az adatok eléréséhez (amit egyébként a `beeline` kliens is használ az adatok eléréséhez), a Hive könnyen használható Java kódból is a JDBC API- keresztül. Gyakorlatilag úgy programozhatjuk a HDFS-en lévő adatfájlok Hive rétegen keresztüli SQL lekérdezéseit, mintha egy klasszikus relációs adatbázishoz csatlakoznánk. Példaként tekintsük az alábbi Java programot (`org/example/HiveJDBCClient.java`), ami kilistázza a `personal_entries.json` fájl tartalmát a `personal_entries` Hive adattábla lekérdezése által:

```
package org.example;

import java.sql.*;
```

```
/**
 * Sample Hive JDBC client
 *
 */
public class HiveJDBCClient
{
    private static String driverName = "org.apache.hive.jdbc.HiveDriver";
    private static String LIST_ALL_ENTRIES = "SELECT * FROM personal_entries";

    public static void main( String[] args ) {
        try {
            Class.forName(driverName);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            System.exit(1);
        }

        try(Connection conn =
DriverManager.getConnection("jdbc:hive2://localhost:10000/default", "hive", "");
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(LIST_ALL_ENTRIES);
        ) {
            System.out.println(String.format("%s\t\t%s\t\t%s\t%s\t\t%s", "pid",
                                                                         "name",

"gender",

"last_contacted",

"birth_year"));
            while (rs.next()) {
                System.out.println(String.format("%s\t%s\t%s\t%s\t%d",
rs.getString(1),

rs.getString(2),

rs.getString(3),

rs.getString(4),

rs.getInt(5)));
            }
        } catch (SQLException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

A program fordítása és futtatása a következő parancsokkal végezhető el:

```
$ javac org/example/HiveJDBCClient.java

$ java -cp .:lib/* org.example.HiveJDBCClient
pid             name            gender  last_contacted          birth_year
1122334455      John Doe        M       2017-03-02T11:43:00     1987
2233445566      Jane Doe        F       2019-10-12T15:22:34     1965
3344556677      John Smith              2020-01-05T08:27:12     1999
```

Ahhoz, hogy a csatlakozás sikeres legyen a fenti Hive docker stack-nek futnia kell (a Hive szerver 10000-es portja forwardolva van localhost-ra is, ezért működik a lokális csatlakozás). Fontos, hogy a futtatásnak van pár függősége, amiket a `lib` könyvtárból töltünk be, ez első sorban a Hive JDBC driver, valamint a HTTP kommunikációhoz szükséges könyvtárak.

✅ További feladatok

1. Töltsük be a `billing_entries.json` fájlt is egy Hive táblába (teljes cím adatokkal), és írjunk egy olyan SQL lekérdezést, ami mindhárom adatforrást (`personal_entries.json`, `billing_entries.json`, `sales_entries.csv`) összekapcsolja a PID alapján! ⭐⭐
2. Írjunk egy olyan Java programot, ami a Hive JDBC driver segítségével lekérdezi azokat a személyeket a `personal_entries.json` és `sales_entries.csv` fájlokból, akik 1970 után születtek, és kaptak ajánlatot (`offer_text` mező nem üres)! ⭐
3. Kérdezzük le Apache Drill [8] segítségével a `personal_entries.json` tartalmát HDFS-ről! ⭐
4. Kapcsoljuk össze Apache Drill [8] segítségével a `personal_entries.json`, `billing_entries.json`, valamint a `sales_entries.csv` tartalmát egy lekérdezéssel! ⭐ ⭐
5. A gyakorlatban bemutatott lekérdezéseket hajtsuk végre Apache Imapala (docker stack [9] használható) segítségével is (használjuk az `imapla-shell`-t és a fenti adat definíciókat)! ⭐ ⭐⭐

Referenciák

[1] https://hive.apache.org/

[2] https://cwiki.apache.org/confluence/display/Hive/GettingStarted#GettingStarted-InstallationandConfiguration

[3] https://bigdataprogrammers.com/load-csv-file-in-hive/

[4] https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL

[5] https://spark.apache.org/

[6] https://tez.apache.org/

[7] https://cwiki.apache.org/confluence/display/Hive/SerDe

[8] https://drill.apache.org/

[9] https://github.com/parrot-stream/docker-impala