Laboratory #2.1

DBxic: our toy database system

What to do:

1. Download the sources of DBxic from Campus Virtual.
2. Follow the attached tutorial to set it up and start using it
3. Study the system and consider these questions:
    a. We learnt that query processing pipelines can be lazy or eager. Which type does DBxic implements?
    b. Do you observe any query optimization? Where in the source code should you look for them?
    c. Which type of replacement strategy does the buffer manager of DBxic use?
    d. Imagine that we want DBxic to be able to use value constraints (e.g., a numeric column which can only take values between 18 and 99); which changes in the source code would be required?

    You'll need to show me something that supports your answer.

What:

- Next week, you'll need to answer these questions to me personally
- Prepare any diagram/screenshot/text that you may need for the explanation

How:

- In pairs

Delivery during our Lab slot: October 14th (19:00)

Tutorial on the use of DBxic

DBxic is a simple implementation in Java of a DBMS based on a similar software called *Attica* developed at the University of Edinburgh years ago.

Its objective is to visualize the different basic subcomponents of a DBMS. To keep it simple, the system has a few limitations that we will discuss in this tutorial

Setting DBxic up

Download the sourcecode from CampusVirtual. Unzip it and open the project with your favorite IDE. I personally use Intellij IDEA.

Before running the DBMS, we need to modify the config file. It is a file called `dbxic.props` that you'll find in the root directory of the project. Open it with a text processor and add your local path to the project in the two indicated lines:

```
# DBxic's database directory
dbxic.base.dir=/<PATH_IN_YOUR_LOCAL_MACHINE>/dbxic/db
# DBxic's temporary files directory
dbxic.temp.dir=/<PATH_IN_YOUR_LOCAL_MACHINE >/dbxic/db/tmp
# The number of blocks in the buffer
dbxic.buffersize=50
```

Once you have modified it, you can run the project from:

```
org.adbs.dbxic.engine.core.DBMS
```

If everything is correct, you should now have DBxic running!

Have a look to the directory `/<PATH_IN_YOUR_LOCAL_MACHINE>/dbxic/db`. A file called `catalog` has appeared there. It is the one which saves the description of our DB. You'll find more files here as you create new tables in the DB.

Using DBxic

There are a few things that you should know about how DBxic works:

- Always use a semicolon (;) to indicate the end of any statement. SQL statements can be written in multiple lines as long as the semicolon is properly used.

- Attributes **always** need to be referenced as `<table_name>.<attribute_name>`
- Select all (`select * from ...`) is not supported yet.
- Find the complete list of allowed commands and basic description with: `commandlist;`

Here you have a complete example of the use of DBxic:

```
create table student (id integer, name string, birth_year
integer);
create table department (id integer, name string, place string);
create table teacher (id integer, name string, phd_year integer,
department_id integer);
catalog;
describe table student;
insert into department values (13, 'Physics', 'Barcelona');
insert into department values (20, 'Maths', 'Bellaterra');
insert into department values (15, 'CS', 'Pals');
insert into teacher values (1, 'Bill Gates', 1979, 15);
insert into teacher values (2, 'Steve Jobs', 1981, 15);
insert into teacher values (3, 'Einstein', 1927, 13);
select teacher.name, department.name from teacher, department
where teacher.department_id = department.id and teacher.phd_year >
1975;
select department.id, department.name from department where
department.id < 19;
drop table student;
```

Populating the DB with large relations

To test the algorithms that we are going to develop during this laboratory, we might need large relations. To save time, I have implemented a simple script that does the work. You'll find it at:

```
test.org.adbs.dbxic.engine.InflateRelation
```

In the `main` function of the class, you only need to specify the name of the table, the number of tuples that you want to generate, and the list of attributes of the relation. Each attribute is

indicated as a pair of attribute's name and type. Type may be `integer`, `string`, or `limit_X_Y`, (meaning an integer attribute which can take values from `X` to `Y-1`).

This is an example of the execution of the script for a table teacher:

```
create table teacher (id integer, name string, phd_year integer,
                      department_id integer);
insert into teacher values (0, 'ahrwxyd', 9682, 6);
insert into teacher values (1, 'peirupb', 1727, 0);
...
insert into teacher values (49, 'bvozush', 7313, 7);
exit;
```

Then, you only need to copy the SQL statements returned by this script and paste them into the feed of DBxic. Note that it always finishes with an `exit;`. With this, you'll need to restart DBxic, but we make sure that the changes persist.

Modifying DBxic SQL parser

DBxic includes a parser for the (limited) SQL language at:

```
org.adbs.dbxic.engine.parser
```

You only need to rewrite the grammar specification at `dbxic_sql_parser.jj` so that it does whatever you need, and compile it with the parser generator `javacc` [1]:

```
javacc dbxic_sql_parser.jj
```

The rest of files in that subdirectory, including the java sourcecodes of the parser `XicQLParser.java`, will be automatically generated by the parser generator.

[1] You might need to install it: https://javacc.github.io/javacc/