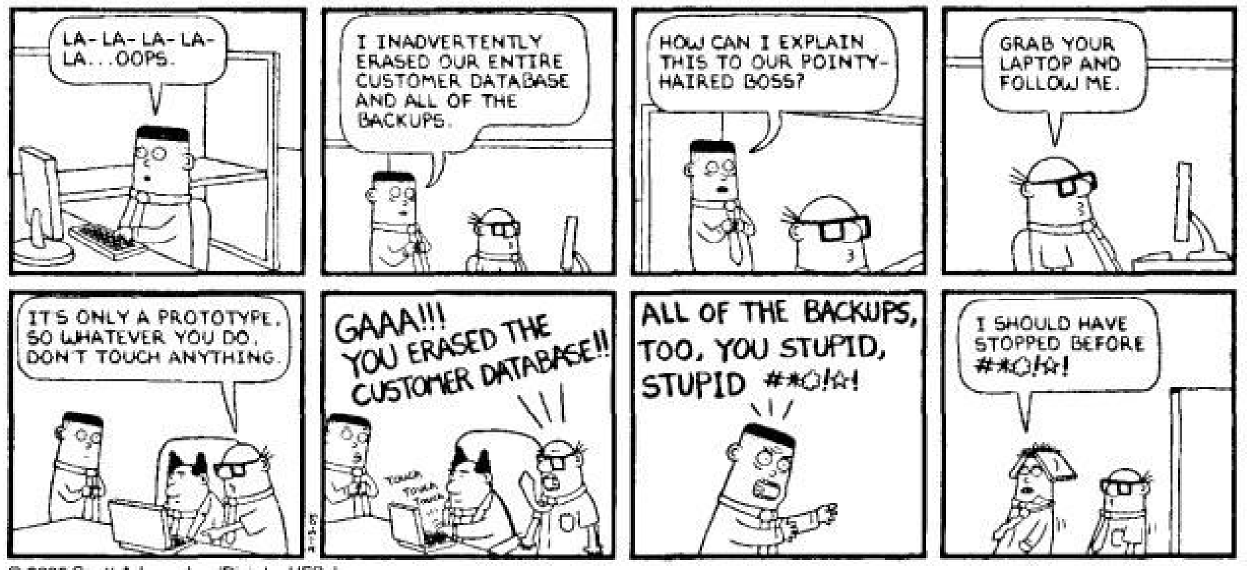
CS 251 Inlab 7 : Relational Databases

Refer to the general instructions and submission guidelines at the end of this document before submitting.


LA- LA- LA- LA- LA... OOPS.
I INADVERTENTLY ERASED OUR ENTIRE CUSTOMER DATABASE AND ALL OF THE BACKUPS.
HOW CAN I EXPLAIN THIS TO OUR POINTY-HAIRED BOSS?
GRAB YOUR LAPTOP AND FOLLOW ME.
IT'S ONLY A PROTOTYPE. SO WHATEVER YOU DO, DON'T TOUCH ANYTHING.
GAAA!!! YOU ERASED THE CUSTOMER DATABASE!!
TOUCH TOUCH TOUCH
ALL OF THE BACKUPS, TOO, YOU STUPID, STUPID ##O!☆!
I SHOULD HAVE STOPPED BEFORE ##O!☆!

In this lab, you are going to play with relational databases. A **Database** is a software **system** that is responsible for **data management**. Now, we know that data can be stored and managed in a multitude of ways. Relational databases approach it using a basic construct called **Table**. A Table is just what it sounds like, it has several columns, a header row and member rows. The concept itself is very intuitive.

In Relational Databases data is stored in the form of tables. Some basic operations for any table are

1. Create table
2. Insert Row(s)
3. Update Row(s)
4. Delete Row(s)
5. Select Row(s) etc..

To formalize the operations on tables a language called **SQL** (pronounced as SEQUEL) was introduced. Any operation that you want to perform on tables/databases can be written in SQL. That SQL statement is written called a **Query**. A good reference for writing SQL queries can be found here.

A simple SQL query can have the form

```
SELECT column1, column2, ...
FROM table_name;
```

One important thing to note is that unlike programming languages like C/C++, Python, Java, etc... there is no single standard for SQL. In that, the syntax of the same SQL query might change with the actual relational database implementation you use. Some of the RDB implementations are MySQL, Postgres, SQLite. Today we will be using SQLite, so please stick to the SQL syntax that works for **SQLite**.

Once a query is written, it is passed to the database for execution. There are a couple of ways to do that. One might directly type it into an interactive db session or use a third party language to talk to the database and pass the query. Here we are going to use **Python3** as a mediator between us and SQLite.

Gotta Catch 'Em All [50 points]

You have been provided with files **`pokemon.csv`**, **`abilities.csv`** and **`pokemon_abilities.csv`**. **`pokemon.csv`** which represents a table containing pokemon id, name and other columns. **`abilities.csv`** contains powers of some pokemon. **`pokemon_abilities.csv`** contains a mapping between the above two tables.

For each of the following tasks, you will need to write one or two SQL statements.

Task 1 [5 points]

Create a python script **`create_tables.py`**, which when executed

1. creates a SQLite DB file named **`pokedex.db`** and
2. creates 3 tables named
   a. **POKEMON** with table structure (i.e. column names and types) as in **`pokemon.csv`**. You can assume that all numbers are integers
   b. **ABILITIES** for **`abilities.csv`**
   c. **POKEMON_ABILITIES** for **`pokemon_abilities.csv`**

We will use the **`pokedex.db`** created in this task for all further tasks in this lab.

Grading will be done for this task by inspecting the state of `pokedex.db` after running `create_tables.py`

Task 2 [5 points]

Create a python script **`insert.py`**, which when executed populates the tables i.e.

1. Inserts all rows of **`pokemon.csv`** in **POKEMON** table
2. Similarly, inserts all rows of **`abilities.csv`** in **ABILITIES** table
3. And, inserts all rows of **`pokemon_abilities.csv`** in **POKEMON_ABILITIES** table

You may want to confirm that the inserts are indeed successful by reading and printing the data in tables.

Grading will be done for this task by inspecting the state of `pokedex.db` after running `create_tables.py` & `insert.py`

Task 3 [7 points]

Apparently some rogue pokemon crept into the database. So before proceeding further it is essential that we clean the data. Fortunately for us, all the rogue pokemon's identifier starts with the string "rogue".

Write a python script **`clean.py`** which when executed **deletes** all rows in POKEMON table with identifier starting with string "rogue". Note: if rogue occurs in the middle of the identifier (like in **"**tyrogue**")** then it is **NOT** a rogue pokemon.

You may want to verify that the rogue pokemon and only the rogue pokemon are deleted at this point. If you find that you messed up in this step, then you need to delete **`pokedex.db`** and re-run scripts in Tasks 1 & 2 to reset the database [Although better ways exist].

Grading will be done for this task by inspecting the state of `pokedex.db` after running `create_tables.py` & `insert.py` & `clean.py`

Task 4 [6 points]

Surely Prof. Samuel is getting sloppier by the day. It seems that he has made a horrible mistake while collecting the data.

In **ABILITIES** table for all the rows where **`is_main_series`** is **`0`,** the **`generation_id`** should have been **`8`**. It seems he has entered **`5`** instead.

Write a python script **`update.py`** which when executed corrects the above-mentioned error.

You may want to verify that the update operation is successful. If you find that you messed up in this step, then you need to delete `pokedex.db` and re-run scripts in Tasks 1 & 2 & 3 to reset the database [Although better ways exist].

Grading will be done for this task by inspecting the state of `pokedex.db` after running `create_tables.py` & `insert.py` & `clean.py` & `update.py`

Task 5 [7 points]

Now that we are all set and done with our pokedex database, we can try to see some interesting patterns in the data

Write a python script **`rank.py`** which **prints** (only the) **identifier** of **top 3** pokemon after ordering them by **base_experience** in **descending** order line by line.

For ex.

```
$ python3 rank.py
bulbasaur
ivysaur
venusaur
```

Grading will be done for this task by matching the output of `rank.py` after running `create_tables.py` & `insert.py` & `clean.py` & `update.py`

Task 6 [8 points]

Next, we try to find the pokemon with the most experience in every species.

Write a python script **`group.py`** which when executed, groups entries in **POKEMON** table w.r.t. **`species_id`** and prints the max of **`base_experience`** in every group line by line **ordered decreasingly** w.r.t max of **`base_experience`**. Limit the number of prints to 3.

For ex.

```
$ python3 group.py
236
142
64
```

Grading will be done for this task by matching the output of `group.py` after running `create_tables.py` & `insert.py` & `clean.py` & `update.py`

Task 7 [12 points]

Now that we know a few things about our pokemon, we want to map every pokemon to its abilities.

Write a python script `join.py` which when executed, performs an (inner) join on tables **POKEMON**, **POKEMON_ABILITIES** and **ABILITIES** on `pokemon_id` and `ability_id` and creates a selection consisting of columns named `pokemon_id`, `ability_id`, `pokemon_identifier`, `ability_identifier`.

This selection will have rows, each containing a pokemon identifier and its ability identifier. Use it to print each pokemon name and the list of its powers in the selection in the following format, abilities ordered in lexicographic order and pokemon ordered according to their name

```
bulbasaur=[chlorophyll,overgrow]
kakuna=[shed-skin]
...
```

Observe that some pokemon can have multiple abilities and some pokemon can have no abilities at all.

Grading will be done for this task by matching the output of `join.py` after running `create_tables.py` & `insert.py` & `clean.py` & `update.py`

Credits

1. veekun/pokedox for the data

General Instructions

- Make sure you know what you write, you might be asked to explain your code at a later point in time
- The submission will be graded automatically, so stick to the naming conventions strictly
- Do not submit your `pokedex.db`
- The deadline for this lab is **Thursday, 3rd October, 17:15.**

After creating your directory, package it into a tarball `<roll_number>-inlab7.tar.gz`
We will untar your submissions using

```
$ tar xvf <roll_number>-inlab7.tar.gz
```

Make sure that when the above is executed, the resulting <roll_number>-inlab7/ directory has the correct directory structure.

The directory structure should be as follows (nothing more nothing less)

```
<roll_number>-inlab7
     ├── clean.py
     ├── create_tables.py
     ├── group.py
     ├── insert.py
     ├── join.py
     ├── rank.py
     └── update.py
```