Rubric

- If your code does not run on the lab machines, you will get 0.
- The marking weight is as follows:
  - 10%: Creation and populating the tables.
  - 15%: Experiments for Q1.
  - 20%: Experiments for Q2.
  - 25%: Experiments for Q3.
  - 30%: Experiments for Q4.

Appeals

If you don't agree with the marking and/or have questions about it **please contact your TA (not the instructor)**, the instructor will be contacted by the TAs if there's a need for a "second opinion."

Preamble

In this assignment, you will be provided with a database from which you will derive three databases of different sizes in order to benchmark the performance of different queries with and without indices and different database settings. As done in previous assignments you will need to strictly follow the submission instructions provided below.

Tasks

The following tables are to be derived from this open dataset publicly available at Kaggle. The semantics of their attributes should be clear from their names and the comments indicate the .csv file to be used as the source and the columns to be used (there are more files and attributes that are not needed for the purposes of this assignment, you should only use those that are required).

Data source: https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce?select=olist_order_items_dataset.csv

```
                                                    -- olist_customers_dataset.csv
CREATE TABLE "Customers" (
      "customer_id"      TEXT,                        -- customer_id
      "customer_postal_code"   INTEGER,               -- customer_zip_code_prefix
      PRIMARY KEY("customer_id")
);
```

```
                                                              --olist_sellers_dataset.csv
CREATE TABLE "Sellers" (
      "seller_id"  TEXT,                                       -- seller_id
      "seller_postal_code"     INTEGER,                        -- seller_zip_code_prefix
      PRIMARY KEY("seller_id")
);
                                                              --olist_orders_dataset.csv
CREATE TABLE "Orders" (
      "order_id"   TEXT,                                       -- order_id
      "customer_id"      TEXT,                                 -- customer_id
      PRIMARY KEY("order_id"),
      FOREIGN KEY("customer_id") REFERENCES "Customers"("customer_id")
);
                                                             --olist_order_items_dataset.csv
CREATE TABLE "Order_items" (
      "order_id"   TEXT,                                      -- order_id
      "order_item_id"    INTEGER,                             -- order_item_id
      "product_id"       TEXT,                                -- product_id
      "seller_id"  TEXT,                                      -- seller_id
      PRIMARY KEY("order_id","order_item_id","product_id","seller_id"),
      FOREIGN KEY("seller_id") REFERENCES "Sellers"("seller_id")
      FOREIGN KEY("order_id") REFERENCES "Orders"("order_id")
);
```

You will populate the four tables above using the respective `.csv` files (highlighted in orange). The table and attribute names (highlighted in yellow) must be as specified above. Then you will create three different databases, which must be called `A3Small.db`, `A3Medium.db` and `A3Large.db`, respectively. The cardinalities of the four tables above for each of these three databases should be approximately as follows:

Cardinality of the tables for each database

		A3Small.db	A3Medium.db	A3Large.db
Tables	Customers	~10k	~20k	~33k
	Sellers	~500	~750	~1k

`Order`	~10k	~20k	~33k
`Order_items`	~2k	~4k	~10k

Each group is to create their own databases by sampling tuples **randomly** from the given `.csv` files. Note that the three databases must enforce the primary and foreign keys constraints as described in the table definitions. There are several ways to do that, the approximate cardinalities in the table above should give you some freedom in that respect.

Using each of the databases you created you will experiment with four scenarios:

1. **Uninformed:** You will undefine the primary and foreign keys of all tables and will disable the creation of SQLites's auto-indexing. (Refer to SQLite's setting of PRAGMA automatic_index). **You must not create any indices.**
2. **Self-optimized:** You will (re)define the primary and foreign keys of all tables and will enable the creation of SQLites's auto-indexing. You must still not create any indices.
3. **User-optimized:** This is the same scenario as the "Self-optimized" one but you are expected to create indices (freely) that would optimize the performance of the query load.

The idea of using these scenarios is to enable one to investigate how varied the performance of a DBMS can be depending on how one sets it up.

Then, for each of the scenarios, you will execute the following four queries:

Q1: Given a random customer_postal_code from Customers, find how many orders containing more than 1 item have been placed by customers who have that customer_postal_code.

Q2: Create a VIEW called OrderSize which has two columns, oid and size, where oid is an order_id and size is the total number of items in that order. Using the view OrderSize, extend Q1 with the orders that have items more than the average number of items in the orders.

Q3: Rewrite query Q2 but not using any VIEW.

Q4: Choose a random customer with more than one order and for that customer's orders, find in how many (unique) postal codes the sellers provided those orders.

As you will note all queries above have a "random" query attribute. Those must be sampled randomly from the existing tables in each version of the database, and each query must be executed 50 times, and the average running time collected. The reason for that is to minimize the effect of caching and cold-start.

Finally, you will produce charts showing the performance (query execution time) for each query in each scenario for each database size.

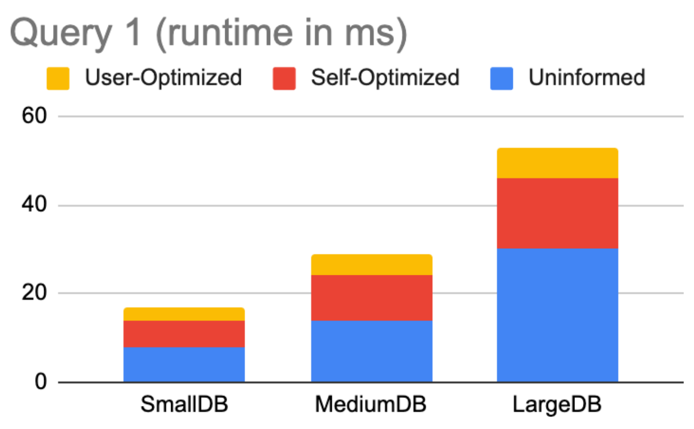
How will all these experiments be done?

You will write four Python applications, one for each of the queries above –they must be named `Q1A3.py`, `Q2A3.py`, `Q3A3.py` and `Q4A3.py` respectively. Each application will execute the respective query 50 times (drawing random input from the database tables as needed), under each of the three scenarios above for each database size. Query performance (running time) is to be collected and plotted at the closing of each application.

For instance, for Q1, your application `Q1A3.py` will:

1. Connect to `A3Small.db`
    1. Set scenario "Uninformed"
    2. Execute Q1 50 times (collecting query execution time).
    3. Disconnect `A3Small.db` and reconnect (this is to minimize caching effects by SQLite)
    4. Set scenario "Self-optimized"
    5. Execute Q1 50 times (collecting query execution time)
    6. Disconnect `A3Small.db` and reconnect (this is to minimize caching effects by SQLite)
    7. Set scenario "User-optimized"
    8. Execute Q1 50 times (collecting query execution time)
    9. Disconnect `A3Small.db`
2. Connect to `A3Medium.db`
    1. Repeat steps 2-9
    2. Disconnect `A3Medium.db`
3. Connect to A3Large.db
    1. Repeat steps 2-9
    2. Disconnect `A3Large.db`
4. Plot query performance results.

The chart with the query results to be produced at the end can look like the following one, but you are free to choose other charts/formats, as long as they serve to reflect the same information. Note that (1) you can find some sample code to produce stacked bar charts here and (2) the numbers in the chart below are merely illustrative (do not take them as a reference for your own results).



The same four main steps above would be done for queries Q2, Q3, and Q4 as well. Finally, the four charts (one for each query/application) must be named `Q1A3chart.png`, `Q2A3chart.png`, `Q3A3chart.png` and `Q4A3chart.png`, respectively.

Submission

It is VERY important that all tables, attributes and file names are EXACTLY as specified above. Failure to do so may yield a mark of zero grade to the whole assignment.

There should be only ONE submission per group. In order for all group members to receive credit the correct group number must be used as detailed below.

Your submission must include:

1. The files for the three databases you created: `A3Small.db`, `A3Medium.db` and `A3Large.db`.
2. The four Python applications you developed: `Q1A3.py`, `Q2A3.py`, `Q3A3.py` and `Q4A3.py`.
3. The charts produced by each of the four applications: `Q1A3chart.png`, `Q2A3chart.png`, `Q3A3chart.png` and `Q4A3chart.png`.
4. A `README.txt` (in plain text format) file
    1. It must contain at the top: the group number, the ccids and names of all group members and the list of resources used (other than regular course material) and/or people you collaborated with (as much as it is allowed as per our course's policy) or the single line "We declare that we did not collaborate with anyone outside our own group in this assignment".
    2. Your readme file should also include a short report explaining for each of the four queries what indices you created for the "User Optimized" scenarios and the

reasoning behind those choices . Submissions without a `README.txt` file will be penalized with a 30% deduction of the final mark.

Your submission is one `.tgz` file (i.e., it must be a gnu zipped tar file) containing two folders:

1. The first folder is to be named `GroupXXA3_DBs` (where `XX` is your group number) and must contain ONLY the three database files, i.e., `A3Small.db`, `A3Medium.db` and `A3Large.db`.
2. The second folder is to be named `GroupXXA3` (where XX is your group number) and it must contain all other files (except the .db files) as described above, i.e., `Q1A3.py`, `Q2A3.py`, `Q3A3.py`, `Q4A3.py` , `Q1A3chart.png`, `Q2A3chart.png`, `Q3A3chart.png`, `Q4A3chart.png` and `README.txt`.

Note that eClass does not support versioning of submissions, and each new submission replaces your previous one. This makes last-minute submissions somewhat risky. **Avoid last-minute submissions, and check your submissions after an upload (ideally on the lab machines) to make sure the right content is uploaded.** And again, follow the instructions above exactly, any deviations may render your submission non-gradable and therefore worth 0 (zero) marks.