CS2102: Database Systems (AY2022/2023 – Sem 1)

Assignment 1

Deadline: Saturday, 1 October 2022, Time: 11.59 pm

Instructions

- The assignment consists of 10 half-mark SQL questions; thus a total of 5 marks.
- The deadline for submission is October 1 (Saturday) at 11.59 pm.
- **Late submission penalty:** One mark will be deducted for each late day up to two late days; submissions after the second late day will receive zero marks and will not be graded.
- The assignment is to be submitted using Canvas.
- As the assignment will be auto-graded, it is very important that your submitted answers conform to the requirements in the Instructions (Section 1).

Good Luck!

1 Instructions

In this assignment, you will formulate 10 SQL queries and copy your solutions into the provided template file `cs2102-assignment1-views.sql`. As the assignment will be auto-graded, it is very important that your submitted answers conform to the following requirements:

- For each question, you are to write a **single `CREATE OR REPLACE VIEW` SQL statement** to answer the question. You **MUST** use the view schema provided for each question without changing any part of the view schema (i.e., view name, column names, or the order of the columns). For example, the provided view schema for Question 1 is "`v1 (area, num_stations)`", and if your answer to this question is the query "`SELECT 'dummy' AS area, 10 AS num_stations`", then you must enter your answer in the answer box as follows:

```
CREATE OR REPLACE VIEW v1 (area, num_stations) AS
SELECT 'dummy' AS area, 10 AS num_stations
;
```

- Each `CREATE OR REPLACE VIEW` statement must terminate with a semicolon.
- Each answer must be a syntactically valid SQL query without any typographical errors: a SQL answer that is syntactically invalid or contains very minor typographical error will receive 0 marks even if the answer is semantically correct.
- For each question, your answer view must not contain any duplicate records.
- Each question must be answered independently of other questions, i.e., the answer for a question must not refer to any other view that is created for another question.
- You are allowed to create the answer view using a single CTE statement (defining possibly multiple temporary tables). If your answer uses a CTE statement to define temporary table(s), you must not use any of the other 10 answer views `v1`, `v2`, ..., `v10`.
- If your answer uses the CTE statement, you must define at most 2 temporary tables.
- Your answers Questions 1-9 must not use SQL constructs that are not covered in class. Question 10 aims to encourage you to explore a bit the functionality of PostgreSQL, so anything is allowed here; we also give some hints and pointers.
- Your answers must be executable on PostgreSQL.

- To help you test and debug your solutions in PostgreSQL, we will provide a Web interface where you can check the correctness of your query. More details can be found in Section 4.

Before you Submit. Before you submit your completed template file to Canvas, check (a) if you updated the `student` view to contain your Student ID and NUSNET ID, and (b) if your template file is valid. The easiest way to test this is to import it into the database, e.g., using

From the command line:
`psql -d cs2102_singaporedb -f cs2102-assignment1-views.sql`

Within `psql` and connected to `cs2102_singaporedb`:
`\i cs2102-assignment1-views.sql`

You might need to adjust the database name and path of the .sql file to match your settings.

The exact command will depend on the database name and the path of your template file. If your template file is valid and does not throw any errors, you should see the following output:

```
DROP VIEW
CREATE VIEW
CREATE VIEW
...
CREATE VIEW
```

containing 11 `CREATE VIEW` messages, 1 for the student view and 10 for the query views. Note that any given `CREATE OR REPLACE VIEW` statement that you did not complete will cause an error. You need to remove those empty statements to successfully import that file. **Only submit the template file if you can import it without errors!**

Lastly, please rename the template file to contain your student number and NUSNET id (e.g., `cs2102-assignment1-views-A00000000-e0000000.sql`)! This is to be doubly sure that we can associate all submissions correctly with all students.

2 Database

We already made the database we use for this assignment available on Canvas. However, for completeness, we provide you here with the ER diagram and the `CREATE TABLE` statements. Below are some additional information that will help answering the questions:

- Table `mrt_connections` contains all direct train connections between 2 stops, including stops from different lines, for both directions. For example, the table includes the tuples `('dt14', 'ew11')` and `('ew11', 'dt14')` since you can get from Bugis to Lavender with just one stop.
- We provide you with the function `geodistance(lat1, lng1, lat2, lng2)` to compute the distance between two latitude/longitude-pairs in kilometers. For example, the query below computes the distance between the School of Computing (1.29478, 103.7738) and the Bukit Timah campus (1.3186, 103.81739):

```
SELECT geodistance(1.29478, 103.7738, 1.3186, 103.81739);
```

  This will give you a distance of about 5.52 km.
- All string values are lowercase!

2.1 ER Diagram

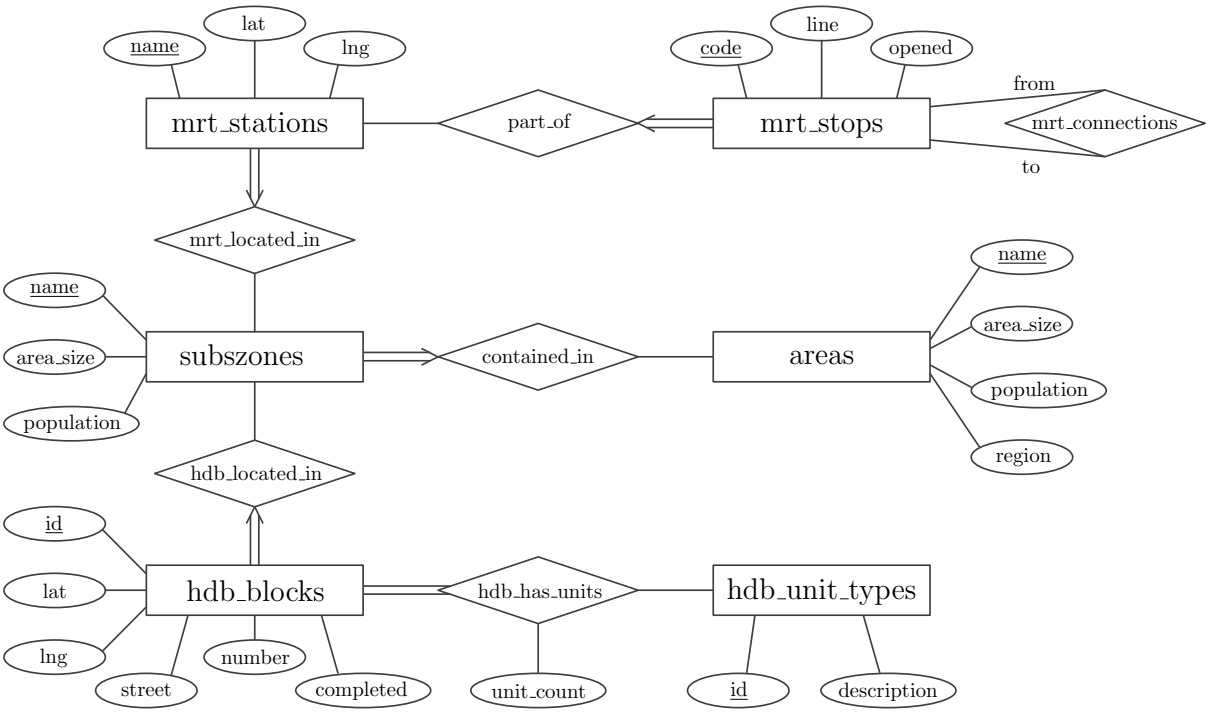

Figure 1: ER Diagram of Database

2.2 Database Schema

```
CREATE TABLE areas (
 name TEXT PRIMARY KEY,
 area_size NUMERIC CHECK (area_size >= 0),
 population INTEGER CHECK (population >= 0),
 region TEXT NOT NULL
);
```

```
CREATE TABLE subzones (
 name TEXT PRIMARY KEY,
 area_size NUMERIC CHECK (area_size >= 0),
 population INTEGER CHECK (population >= 0),
 area TEXT NOT NULL,
 FOREIGN KEY (area) REFERENCES areas (name)
);
```

```
CREATE TABLE hdb_blocks (
 id INTEGER PRIMARY KEY,
 number TEXT NOT NULL,
 street TEXT NOT NULL,
 postal_code INTEGER NOT NULL CHECK (postal_code > 0),
 completed INTEGER NOT NULL,
 max_floor INTEGER NOT NULL CHECK (max_floor >= 0),
 lat NUMERIC NOT NULL,
 lng NUMERIC NOT NULL,
 subzone TEXT NOT NULL,
 FOREIGN KEY (subzone) REFERENCES subzones (name)
);
```

```
CREATE TABLE hdb_unit_types (
 id CHAR(10) PRIMARY KEY,
 description TEXT NOT NULL
);
```

```
CREATE TABLE hdb_has_units (
 block_id INTEGER NOT NULL,
 unit_type TEXT NOT NULL,
 unit_count INTEGER NOT NULL CHECK (unit_count >= 0),
 PRIMARY KEY (block_id, unit_type),
 FOREIGN KEY (block_id) REFERENCES hdb_blocks (id)
```

```
);
```

```
CREATE TABLE mrt_stations (
 name TEXT PRIMARY KEY,
 lat NUMERIC NOT NULL,
 lng NUMERIC NOT NULL,
 subzone TEXT NOT NULL,
 FOREIGN KEY (subzone) REFERENCES subzones (name)
);
```

```
CREATE TABLE mrt_stops (
 code CHAR(4) PRIMARY KEY,
 line CHAR(2) NOT NULL,
 opened INTEGER NOT NULL CHECK (opened >= 0),
 station TEXT NOT NULL,
 FOREIGN KEY (station) REFERENCES mrt_stations (name)
);
```

```
CREATE TABLE mrt_connections (
 from_code CHAR(4),
 to_code CHAR(4),
 PRIMARY KEY (from_code, to_code),
 FOREIGN KEY (from_code) REFERENCES mrt_stops (code),
 FOREIGN KEY (to_code) REFERENCES mrt_stops (code)
);
```

3 Questions

Q1: As one would expect, the access to MRT stations is not uniformly distributed across Singapore. Find all areas that contain at least 5 MRT stations. Return the name of the areas together with the number of MRT stations in each area.

```
CREATE OR REPLACE VIEW v1 (area, num_stations) AS

;
```

Q2: You need to move and are looking for a `‘1room‘` HDB unit. As a good CS2102 student, you always want to come to the lecture in I3-AUD, so you do not want to live far away. Find the number and street of the 10 HDB blocks that have `’1room’` units and are closest to i3-AUD! Include the distance rounded by 2 decimals; the in-built method `ROUND()` makes this easy. The location of I3-AUD is at lat/lng-coordinate `(1.29271, 103.7754)`. Although it should not affect the result, do the rounding before the sorting. Hint: You will need the method `geodistance()` to answer this query.

```
CREATE OR REPLACE VIEW v2 (number, street, distance) AS

;
```

Q3: You like to live in lively areas, so you favor areas with many HDB blocks. For all areas, find the number of HDB blocks located in each area. Sort the areas with respect to the number of blocks in descending order. Hint: Don’t forget the areas in which no HDB blocks are located at all.

```
CREATE OR REPLACE VIEW v3 (area, num_blocks) AS

;
```

Q4: The naming scheme in Singapore is generally intuitive, but there are some exceptions. Find all areas that have an MRT station named after them, but where these MRT stations are not located in the respective areas. For example, `'pioneer'` is the name of both an area and an MRT station. However, the MRT station `'pioneer'` is located in the area `'jurong west'`. Return the name of all those areas like `'pioneer'`.

```
CREATE OR REPLACE VIEW v4 (area) AS

;
```

Q5: You want to run an ad for your startup. However, the budget is tight, and you only have money to advertise at 5 MRT stations. Of course, you want to make sure that your ad is seen by a lot of people. Find the 5 MRT stations along the East-West line (`'ew'`) that have the most HDB blocks in a radius of 300 m! Return the names of the MRT stations together with the number of blocks within their vicinity of 300 m. Hint: You will need the method `geodistance` to answer this query.

```
CREATE OR REPLACE VIEW v5 (mrt_station, num_blocks) AS

;
```

Q6: You were not really happy with the results you got from Q2 and therefore want to explore more options. However, you are still looking for a `'1room'` unit. So as a good first step: Find all the subzones in which you could not get a '1room' HDB unit. Exclude all the subzones that do not contain any HDB blocks! Return the name of each subzone.

```
CREATE OR REPLACE VIEW v6 (subzone) AS

;
```

Q7: Starting your train journey at the first stop of a line gives you typically a higher chance to find a seat. Find the names of any MRT station that mark the end station of any MRT line. For example, Jurong East is an end station of the North-South line (`'ns'`), just not an end station of the East-West line (`'ew'`).

```
CREATE OR REPLACE VIEW v7 (mrt_station) AS

;
```

Q8: Say you have to come to the campus each weekday via the Kent Ridge MRT station (code: `'cc24'`). You're looking for a new accommodation but you want to find a place near an MRT station that brings you to the Kent Ridge station in no more than 10 stops. Find the names of all MRT stations of any line that qualify for that! For example, the Redhill MRT station is only 5 stops away and should therefore be in the result set. Return the name of the MRT station together with the number of required stops; sort by the number of stops in ascending order!

```
CREATE OR REPLACE VIEW v8 (mrt_station, num_stops) AS

;
```

Q9: How many HDB blocks along the Downtown Line (code: `'dt'`) have been completed after a nearby station has been opened. Here, "nearby" is defined by an HDB block and a MRT station being in the same subzone. For example, subzone `'tampines west'` contains the Downtown Line station `'tampines'` which opened 2017. There are a total of 12 HDB blocks in subzone `'tampines west'` that have been completed 2017 or later. This means your result should contain the tuple `('tampines west', 12)`. Get these numbers for all 29 subzones containing a MRT station of the Downtown line; return the name of the subzone and the number of blocks!

```
CREATE OR REPLACE VIEW v9 (subzone, num_blocks) AS

;
```

Q10: Have you ever noticed that some MRT stops seem to be "missing"? For example, there is no stop with the code `'dt4'` between stops `'dt3'` and `'dt5'` along the Downtown Line; not yet, at least. Find the codes of all missing MRT stops! This query can be a bit tricky, and there are probably very different ways to solve it. Here is a hint: Identify all stops codes that one would expect (like `'dt4'`):

- You can use the highest stop code for each MRT line (e.g., `'dt35'`) to derive this set of all possible station codes
- Have a look at the in-built functions `CAST`, `SUBSTRING`, and `GENERATE_SERIES` provided by PostgreSQL. But note that your solution is not required to use these functions.

```
CREATE OR REPLACE VIEW v10 (stop_code) AS

;
```

4 Check Your Solution

To help you test and debug your answers in PostgreSQL, we will provide a Web interface where you can check the correctness of your queries. But please keep the following things in minds:

- Please use this interface only to test the correctness of a query **after** you have written and run the query using your local PostgreSQL installation of the database!
- If you think that your solution is correct, but the check (partially) fails, please send us an email with your solution so we can clarify if maybe the question can be misinterpreted or our reference solution is flawed.

4.1 Web Interface

You can find the Web interface here: http://172.26.191.108/modules/cs2102/sql/. Note the machine is only accessible within the NUS network (or via VPN). To test the correctness of a query, you need to do the following steps:

- Select the query you want to check using the dropdown box (e.g., Query 1)
- Copy your solution for the query **without the** `CREATE OR REPLACE VIEW` **part** into the textbox.
- Click "Check Query" and wait for the report (depending on the complexity of your query this might take a bit)

The screenshot in Figure 2 shows an example for a report. The current answer for the test query `'Query 0'` is:

```
SELECT number, street, completed
FROM hdb_blocks
WHERE id = 123;
```

Feel free to use and modify the query to see how the changes affect the results of the report. For example, in the screenshot, the correct attribute `completed` has been renamed to `silly_name` to illustrate the warning the returned attribute names do not match the expected attribute names (but again, this just causes a warning as the `CREATE OR REPLACE VIEW` statements handle the final renaming of the attributes).

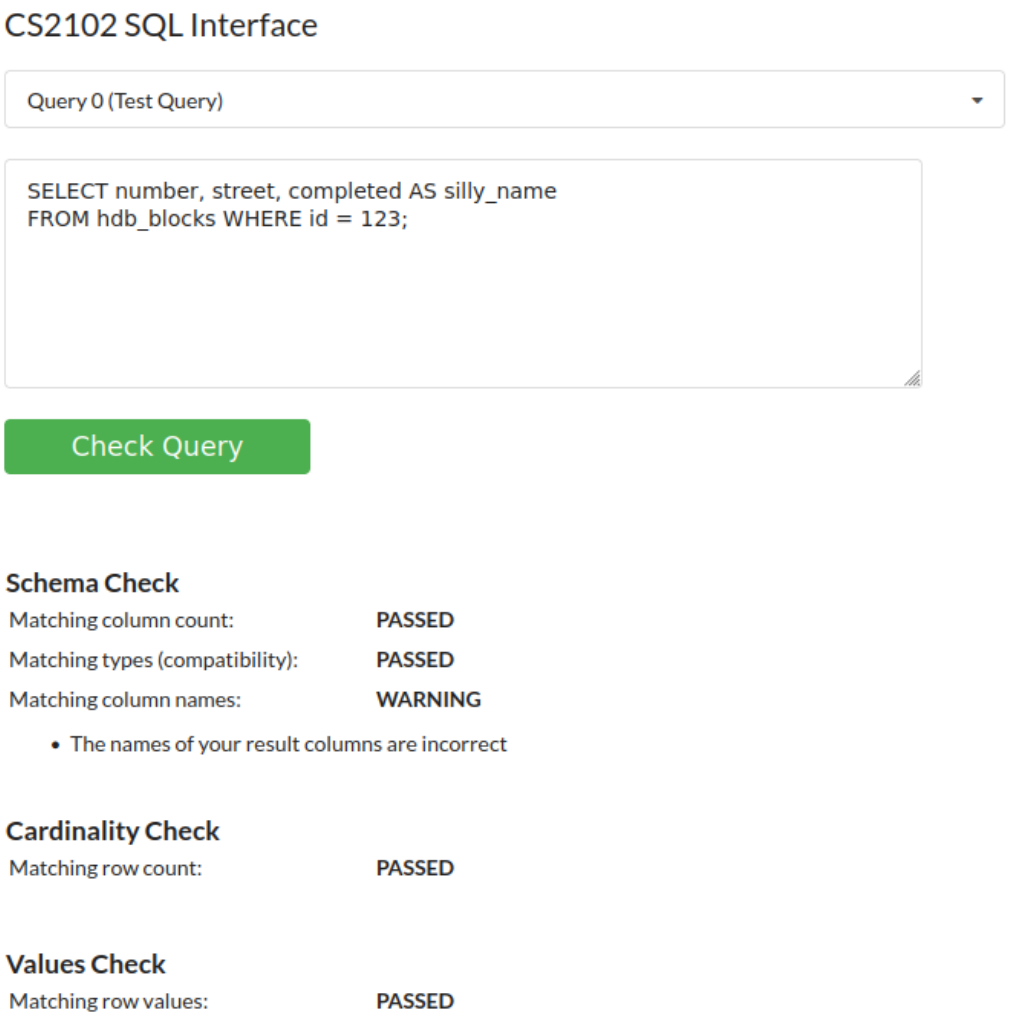

Figure 2: Screenshot of SQL Web Interface to check you queries.

4.2 Description of Report

The report performs 3 types of checks:

- **Schema Check:** The Schema Check will check if your solution and the reference solutions have "comparable" schemas. This means:
  - Both schemas have the same number of columns.
  - Both schemas are union-compatible (i.e., the respective columns have comparable data types).
  - Both schemas should have matching column names (note this will throw only a warning as the final column names will be enforced when creating the view)
- **Cardinality Check:** The Cardinality Check will check if your solution and the reference solutions have the same number of output rows/tuples. If the cardinalities

do not match, the report will indicate how many rows/tuples are missing.

- **Values Check:** The Values Check will check if your solution and the reference solutions return the same result.

Important:

- Note that the checks are not independent. For example, if the Schema Check fails because the number of columns do not match, the Values Check will naturally also fail. So try to address any failed checks in a meaningful order.
- A query that passes all checks is not 100% guaranteed to be correct; although the likelihood should be high. For the evaluation of your submission, we will use a modified version of the database. This is to avoid that some students might might submiy "hard-coded" queries. For example, if a valid solution for a query would be "`SELECT COUNT(*) AS num_blocks FROM hdb_blocks;`" the query "`SELECT 12389 AS num_blocks;`" would yield the same result. To catch such sneaky solution, we use a modified version of the database which will not affect correction solutions.