CS2102: Database Systems (AY2223 – Sem 1)

Midterm Exam

Date: Monday, 3 October 2022, Time: 18:30–19:30

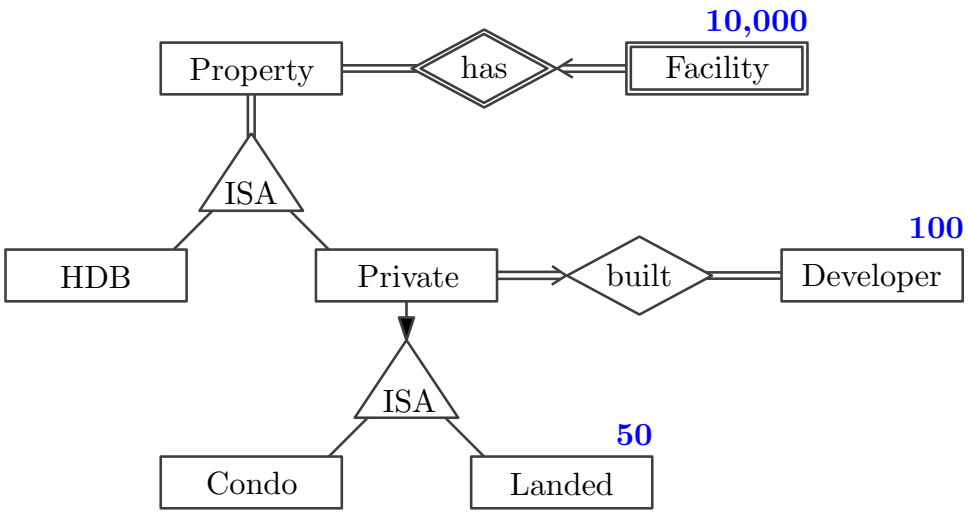
Submission Instructions

1. Please read **ALL** instructions carefully.
2. All the assessment is be done using Examplify; the assessment contains
   (a) Fill in the Blank Questions: 1.1
   (b) Multiple Response Questions: 1.2–1.3 (4 MRQs)
   (c) Essay Questions: 2.1–2.5 (5 SQL queries)
   (d) Your Internet connection will be blocked
   (e) This is an open-book exam
3. This assessment starts at 18:30 and ends at 19:30.
   - Submit your answers by 19:30
   - No additional time will be given to submit.
4. For the SQL questions, there are additional instructions below; in a nutshell:
   - Use an IDE or text editor to write your queries and test them using `psql`
   - Prepare your final answer before submitting it to Examplify according to the instructions in Section 2
   - Allocate sufficient time to prepare your final answers and to copy-&-paste them into Examplify.
5. Failure to follow each of the instructions above may result in deduction of your marks.

Good Luck!

1 ER model & Relational Algebra (10 Marks)

1.1 Cardinalities (2 Marks). The given Entity-Relationship Diagram below does not show any attributes as they are of no relevance for this task. What is relevant are the numbers on the top right of the entity sets with the colour **BLUE**. These numbers indicate the number of instances of the entity sets `Facility`, `Developer`, and `Landed`.



Complete the table below by identifying the **minimum** and **maximum** possible numbers of instances for the remaining entity sets.

	minimum	**maximum**
`Property`		
`HDB`		
`Private`		
`Condo`		

Solution:

	minimum	**maximum**
`Property`	100	10,000
`HDB`	0	10,000
`Private`	100	10,000
`Condo`	0	9,950

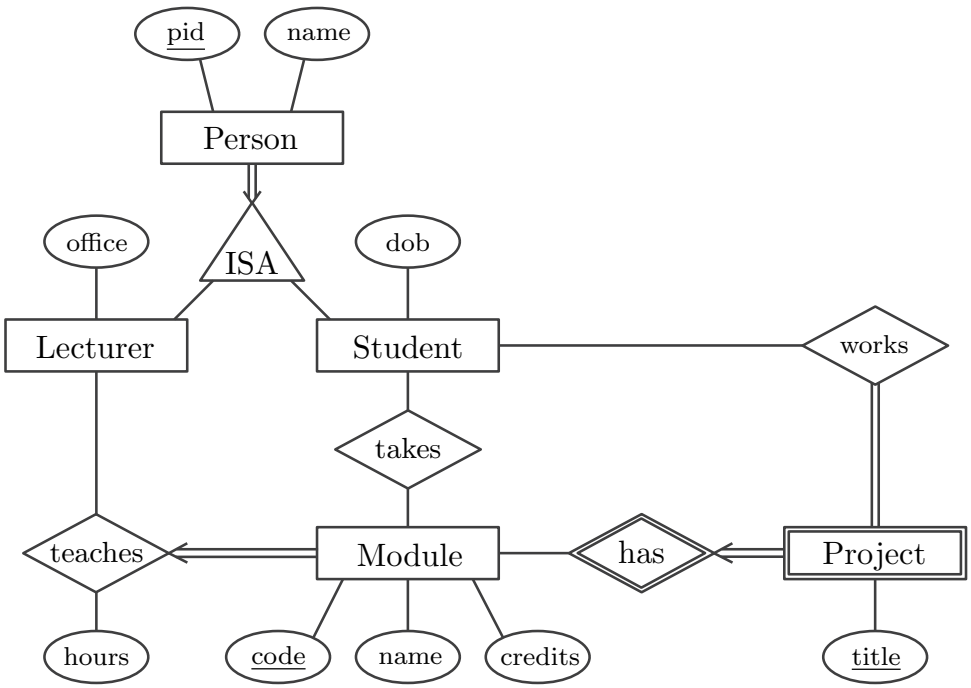
Explanation

- The 10,000 instance of `Facility` and the total participation constraint from `Property` specify the upper bound for `Property`: 10,000
- The 100 instances of `Developer` and the 2 constraints around `built` specify the lower bound for `Private`: 100
- The upper bound of `Property` specifies the upper bound for `Private`: 10,000 (since each property can be both HDB and private: e.g., executive condos are hybrid)
- The lower bound of `Private` specifies the lower bound of `Property`: 100
- The upper bound of `Private` minus the 50 instances of `Landed` specify the upper bound of `Condo`: 9,950
- The lower bound of `Condo` is 0
- The upper bound of `Property` specifies the upper bound of `HDB`: 10,000 (again, a property can be both an HDB and a private property)
- The lower bound of `HDB` is 0

NOTE

There is a note on Examplify specifying: "Your answer should be a number consisting of only digits *(e.g., if the answer is one thousand, you should write 1000 instead of 1,000)*." We attempted to sanitise the answer to remove any non-digit. If the resulting string is empty, then it is replaced by a `NULL` value. No other answers are accepted.

1.2 Constraints (4 Marks). The ER Diagram below models a very simple University Course Database.



Based on this ER diagram, the following database schema has been created:

- `Person(pid, name)`
- `Lecturer(pid, office)`
- `Student(pid, dob)`
- `Module(code, name, credits, lecturer_id, hours)`
- `Project(code, title)`
- `Takes(student_id, code)`
- `Works(code, title, student_id)`

The foreign key constraints are as follows:

- `Lecturer.pid` → `Person.pid`
- `Student.pid` → `Person.pid`
- `Module.lecturer_id` → `Lecturer.pid`
- `Project.code` → `Module.code`
- `Takes.student_id` → `Student.pid`
- `Takes.code` → `Module.code`
- `Works.student_id` → `Student.pid`
- `(Works.code, Works.title)` → `(Project.code, Project.title)`

Let's assume the application requirements include the following list of 8 constraints (there might be more, but focus on these 8):

- All Lecturers and Students are uniquely identified.
- Each project must be worked on by at least one student.
- The total number of teaching hours for a Lecturer cannot exceed 100h.
- Each Module is taught by exactly one Lecturer.
- A student can only work on a project that is part of a module the student is taking.
- The credits for a module must either be 2, 4, or 8.
- A Person must either be a Lecturer or a Student but not both.
- The name of a person cannot be an empty string.

Given these 8 constraints, answer the following 2 questions:

(a) Which of the 8 constraints above are **NOT CAPTURED** by the ER diagram? Select all that apply.

Solution:

- The total number of teaching hours for a Lecturer cannot exceed 100h.
- A student can only work on a project that is part of a module this student is taking.
- The credits for a module are either 2, 4, or 8.
- The name of a person cannot be an empty string.

How the Rest are Captured

- All Lecturers and Students are uniquely identified.
  - Having a primary key `pid` inherited from `Person`.
- Each project must be worked on by at least one student.
  - Total participation constraint on `Project` w.r.t. `works`.
- Each Module is taught by exactly one Lecturer.
  - Total participation constraint + cardinality constraint *(i.e., key constraint)* on `Module` w.r.t `teaches`.
- A Person must either be a Lecturer or a Student but not both.
  - Overlap constraint = false and covering constraint = true using the double-lined arrow on the ISA relationship.

(b) Which constraints **CANNOT BE CAPTURED** when converting the ER Diagram into the database schema without using `TRIGGER` and using only concepts covered in lectures 1 - 6? Select all that apply.

Solution:

- The total number of teaching hours for a Lecturer cannot exceed 100h.
- ~~A student can only work on a project that is part of a module this student is taking~~. *(This option can be enforced by adding more FK, so we ignore this choice from marking. See the explanation below.)*
- Each project must be worked on by at least one student.
- A Person must either be a Lecturer or a Student but not both.

How the Rest are Captured

- All Lecturers and Students are uniquely identified.
  - Having a primary key `pid` on both `Lecturer` and `Student`.
- The name of a person cannot be an empty string.
  - Using `CHECK (name <> '')`.
- Each Module is taught by exactly one Lecturer.
  - `Module` having `code` as the primary key implies that it can uniquely identify the `lecturer_id`. Hence, there is only one `lecturer_id` for each `code`.
- The credits for a module must either be 2, 4, or 8.
  - Using `CHECK (credit = 2 OR credit = 4 OR credit = 8)`.
- A student can only work on a project that is part of a module this student is taking
  - Adding the foreign key constraint
    `(Works.student_id, Works.code)`
    $\rightarrow$ `(Takes.student_id, Takes.code)`.
  - Need to also add the `NOT NULL` constraints to fully enforce the FK.
  - This translation does not totally follow the foreign key constraint specified in the question *(i.e., the translation does not follow from ER precisely but following the needed constraint)* BUT it can still be enforced by. Hence, may be captured *(if we believe the FK can be changed)* or may not be captured *(if we believe the FK cannot be changed)*.

Note on Covering but Non-Overlap Constraint

It may seem that covering constraint and non-overlap constraint on an ISA can be enforced by adding a field that is equal to 0 when the person is a lecturer and equal to 1 when the person is a student as follows:

```
CREATE TABLE Person (
  pid  INT  UNIQUE, /* behave like PK in the end */
  kind INT  NOT NULL CHECK (kind = 0 OR kind = 1),
  name TEXT NOT NULL,
  PRIMARY KEY (pid, kind) /* make sure can be used on FK */
);
CREATE TABLE Lecturer (
  pid    INT  UNIQUE,
  kind   INT  NOT NULL CHECK (kind = 0),
  office TEXT NOT NULL,
  PRIMARY KEY (pid, kind),
  FOREIGN KEY (pid, kind) REFERENCES Person (pid, kind)
);
CREATE TABLE Student (
  pid  INT  UNIQUE,
  kind INT  NOT NULL CHECK (kind = 1),
  dob  DATA NOT NULL,
  PRIMARY KEY (pid, kind),
  FOREIGN KEY (pid, kind) REFERENCES Person (pid, kind)
);
```

But this clearly does not follow the database schema created since the database schema uses `Person(pid, name)` instead of `Person(pid, kind, name)`. Similarly for `Lecturer` and `Student`.

1.3 Equivalence of Queries (4 Marks).

(a) Consider the following SQL query

```
SELECT l.pid, l.office
FROM   Lecturer l
WHERE  EXISTS (SELECT m.lecturer_id
               FROM   Module m, Project p
               WHERE  m.code = p.code
                 AND  m.lecturer_id = l.pid
                 AND  m.credits = 2);
```

Which of the following Relational Algebra expressions are equivalent to this SQL query? Select all that apply.

A

$$Q_1 = \text{Lecturer} \bowtie \rho_{\text{pid} \leftarrow \text{lecturer\_id}}(\text{Module}) \bowtie \text{Project}$$
$$Q_{ans} = \pi_{\text{pid,office}}(\sigma_{\text{credits}=2}(\pi_{\text{pid,office,credits}}(Q_1)))$$

B

$$Q_1 = \sigma_{\text{credits}=2}(\text{Module}) \bowtie \pi_{\text{code}}(\text{Project})$$
$$Q_{ans} = \pi_{\text{pid,office}}(\text{Lecturer} ⟕_{\text{pid}=\text{lecturer\_id}} Q_1)$$

C

$$Q_1 = \text{Lecturer} \bowtie_{\text{pid}=\text{lecturer\_id}} \text{Module}$$
$$Q_{ans} = \pi_{\text{pid,office}}(\sigma_{\text{credits}=2}(\pi_{\text{pid,office,credits}}(Q_1) \bowtie \text{Project}))$$

D

$$Q_1 = \text{Lecturer} ⟖_{\text{pid}=\text{lecturer\_id}} \text{Module}$$
$$Q_{ans} = \pi_{\text{pid,office}}(\sigma_{\text{credits}=2}(Q_1) \bowtie \pi_{\text{code}}(\text{Project}))$$

E None of the others.

Solution: This query returns the `pid` and `office` values of all lecturers that teach at least one 2-MC module that has at least one project.

- Correct solutions: A, D
- In case of D: The Right Outer Join is over a FK, so it degenerates to an inner join.

The following are not equivalent:

- B: The Left Outer Join will give us all the lecturers.
- C: Before the Natural Join, the shared attribute `code` gets removed, so we get the Cross Product.

(b) Consider the following Relational Algebra expression

$$Q_1 = \pi_{\text{code,pid,hours}}(\rho_{\text{pid}\leftarrow\text{lecturer\_id}}(\text{Module}))$$

$$Q_{ans} = \pi_{\text{name,office}}(\sigma_{\text{hours}>40}(\text{Person} \bowtie \text{Lecturer} \bowtie Q_1))$$

Which of the following SQL queries are equivalent to this Relational Algebra expression? Select all that apply.

A:

```
SELECT p.name, l.office
FROM   Person p, Lecturer l
WHERE  p.pid = l.pid
  AND  l.pid = ANY (SELECT lecturer_id
                    FROM   Module m
                    WHERE  m.hours > 40);
```

B:

```
SELECT p.name, l.office
FROM   Person p, Lecturer l
WHERE  p.pid = l.pid
  AND  l.pid IN (SELECT m.lecturer_id
                 FROM   Module m
                 WHERE  m.hours > 40);
```

C:

```
SELECT p.name, l.office
FROM   Person p, Lecturer l
WHERE  p.pid = l.pid
  AND  NOT EXISTS (SELECT 1
                   FROM   Module m
                   WHERE  m.lecturer_id = l.pid
                     AND  m.hours <= 40);
```

D:

```
SELECT p.name, p.office
FROM   Person p, Lecturer l, Module m
WHERE  p.pid = l.pid
  AND  l.pid = m.lecturer_id
  AND  m.hours > 40;
```

E: None of the others

Solution: The query finds the names and office of all lecturers that teach at least one module with more than 40 hours.

- A, B

The following are not equivalent

- C: We cannot flip the condition and make a NOT EXISTS. This would give us all lecturers that teach only modules with more than 40 hours; only then the inner query is empty. But a lecturer can teach 2 modules, one with 30 and one with 50 hours.
- D: The query will contain duplicates if the same lecturer teaches more than 1 module with more than 40h.

Additional Exercise
If the two queries are not equivalent, it means there is a set of data such that running the two queries will produce different results. Can you find such set of data?

2 Formulating SQL Queries (10 Marks)

In this part of the exam your task is to formulate queries in SQL. We use the same database that we introduced for Assignment 1. This means you can directly run and test your queries using `psql`. If needed, we provide the ER diagram and the `CREATE TABLE` statements for the database schema in the Appendix. But you should be very familiar with the database by now.

Instructions. To ensure successful submission and grading of your SQL queries adhere to the following instructions:

- 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 the question is `"q2 (area)"`, and if your answer to this question is the query `"SELECT name FROM areas;"`, then you must enter your answer in the answer box as follows:

```
CREATE OR REPLACE VIEW q2 (area) AS
SELECT name
FROM areas
;
```

- Each `CREATE OR REPLACE VIEW` statement must terminate with a semicolon.
- Each answer must be a syntactically valid SQL query. Test with `psql`!
- 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.
- Your answers must not use SQL constructs that are not covered in Lectures 1-6.
- Your answers must be executable on PostgreSQL. Test with `psql`!
- You must not enter any extraneous text for your answer (e.g., "My answer is: ...), or enter multiple answers (i.e., submit multiple SQL statements). Your answer should not contain any comments.

Recommendations. We recommend the following steps to answer each question:

- Write the answer query using your IDE or text editor of choice, and test it on your machine with `psql`.
- Once you are happy with your query, add the `CREATE OR REPLACE VIEW` statement for the corresponding question in front of the query. You can test the complete statement again with `psql` and create the view.
- Copy the complete `CREATE OR REPLACE VIEW` statement into the answer field in Examplify (don't forget to the semicolon at the end!)

2.1 Query 1 (1 Mark): Find the number of subzones in the `'bukit merah'` area with a population greater than 15,000!

```
CREATE OR REPLACE VIEW q1 (num_subzones) AS

;
```

Solution:

```
CREATE OR REPLACE VIEW q1 (num_subzones) AS
SELECT COUNT(*) AS num_subzones
FROM subzones
WHERE area = 'bukit merah'
AND population > 15000
;
```

2.2 Query 2 (2 Marks): Find the names of all areas in the `'central'` region that do not contain any MRT station!

```
CREATE OR REPLACE VIEW q2 (area) AS

;
```

Solution:

```
CREATE OR REPLACE VIEW q2 (area) AS
SELECT a.name
FROM areas a
WHERE a.region = 'central'
AND NOT EXISTS (SELECT 1
                FROM mrt_stations m, subzones s1
                WHERE m.subzone = s1.name
                AND s1.area = a.name)
;
```

2.3 Query 3 (2 Marks): Find the name of the top-3 areas with the most MRT stops (not stations)! Include the number of stops in your result set! Additional note: You can ignore potential ties – that is, two or more areas having the same number of stops – the result for the top-3 areas is unambiguous.

```
CREATE OR REPLACE VIEW q3 (area, num_stops) AS

;
```

Solution:

```
CREATE OR REPLACE VIEW q3 (area, num_stops) AS
SELECT z.area, COUNT(*) AS num_stops
FROM subzones z, mrt_stations m, mrt_stops s
WHERE z.name = m.subzone
AND m.name = s.station
GROUP BY z.area
ORDER BY num_stops DESC
LIMIT 3
;
```

2.4 Query 4 (2 Marks): Find the names of all MRT stations along the East-West Line (`'ew'`) from which you can reach any other line in just 1 stop. For example, from `'aljunied'` (`'ew9'`) you can reach `'paya lebar'` (`'cc9'`), so `'aljunied'` should be in your result set. In contrast, `'kallang'` (`'ew10'`) only connects to stations of the `'ew'` line, so it should not be in your result set.

```
CREATE OR REPLACE VIEW q4 (mrt_station) AS

;
```

Solution:

```
CREATE OR REPLACE VIEW q4 (mrt_station) AS
SELECT s1.station
FROM mrt_stops s1, mrt_connections c, mrt_stops s2
WHERE s1.code = c.from_code
AND s2.code = c.to_code
AND s1.line = 'ew'
AND s2.line <> 'ew'
GROUP BY s1.station
;
```

2.5 Query 5 (3 Marks): Find all areas that are served by 3 or more different MRT lines – that is, all areas that contain any number of MRT stations of at least 3 different lines. So if an area contains 5 MRT stations but from only 1 or 2 lines, this area should not be part of your result. Include the area names and the number of lines that serve each area in your result and sort by the number of lines in descending order.

```
CREATE OR REPLACE VIEW q5 (area, num_lines) AS

;
```

Solution:

```
CREATE OR REPLACE VIEW q5 (area, num_lines) AS
SELECT tab.area, COUNT(*) AS num_lines
FROM (SELECT s.area, h.line
      FROM mrt_stops h, mrt_stations m, subzones s
      WHERE h.station = m.name
      AND m.subzone = s.name
      GROUP BY s.area, h.line) tab
GROUP BY tab.area
HAVING COUNT(*) >= 3
ORDER BY num_lines DESC
;
```

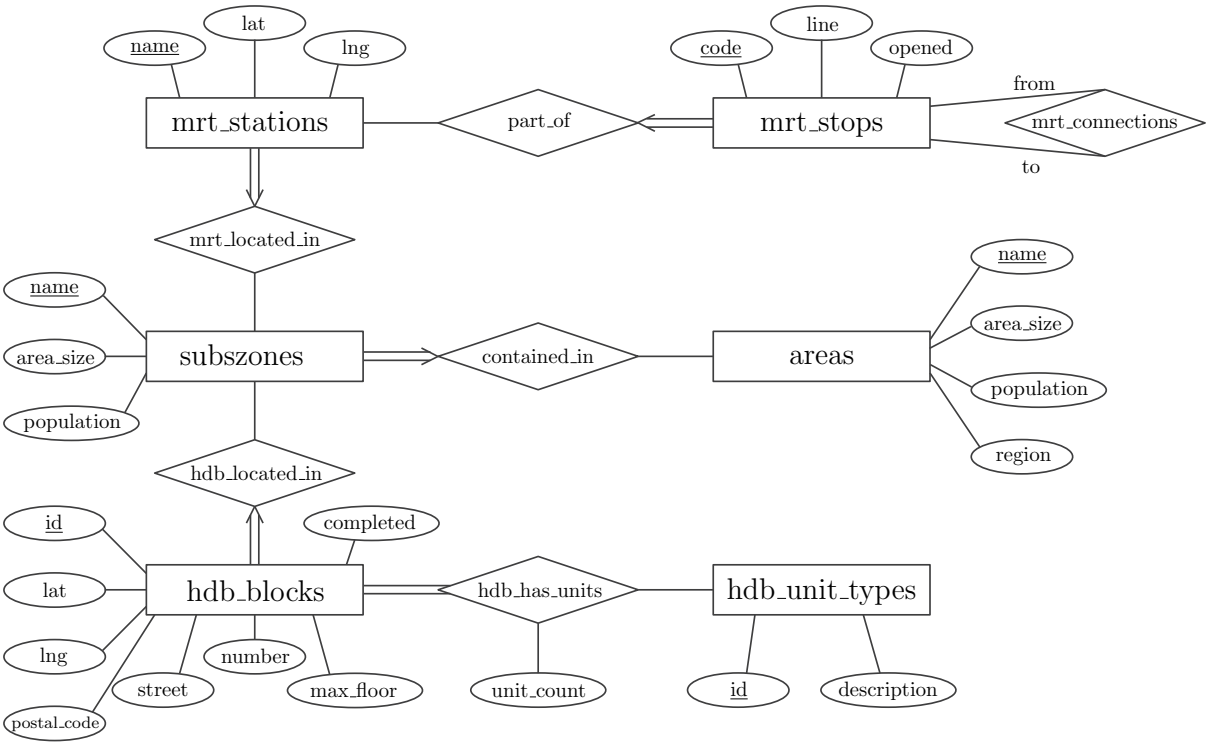
NOTE

This is done mainly by semi-automated marking followed by manual marking *(to ensure no hard-coding of answers and/or give partial marks to answers that do not pass the test cases)*. We attempted to add the necessary `CREATE OR REPLACE VIEW` to the answer whenever possible.

However, the automatic addition may not fully capture all cases and some other errors may still occur. Although this may prevent an immediate 0, it does not guarantee no penalty will be given for missing the `CREATE OR REPLACE VIEW`.

A Appendix

A.1 ER Diagram of Database for Task 2



A.2 CREATE TABLE statements of Database for Task 2

```
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_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)
);

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