D. SIMULATOR

Isaac Yang, Yuxiang Lin

Outline

- Application Introduction
- Database Schema
- Queries Worth Noting
- Demo
- Comparison with Existing Application

Application Introduction

A detective simulator

- Set in a town (1000 inhabitants)
- User plays the role of a detective to capture the culprit of a series of murders

Motivation of the Concept

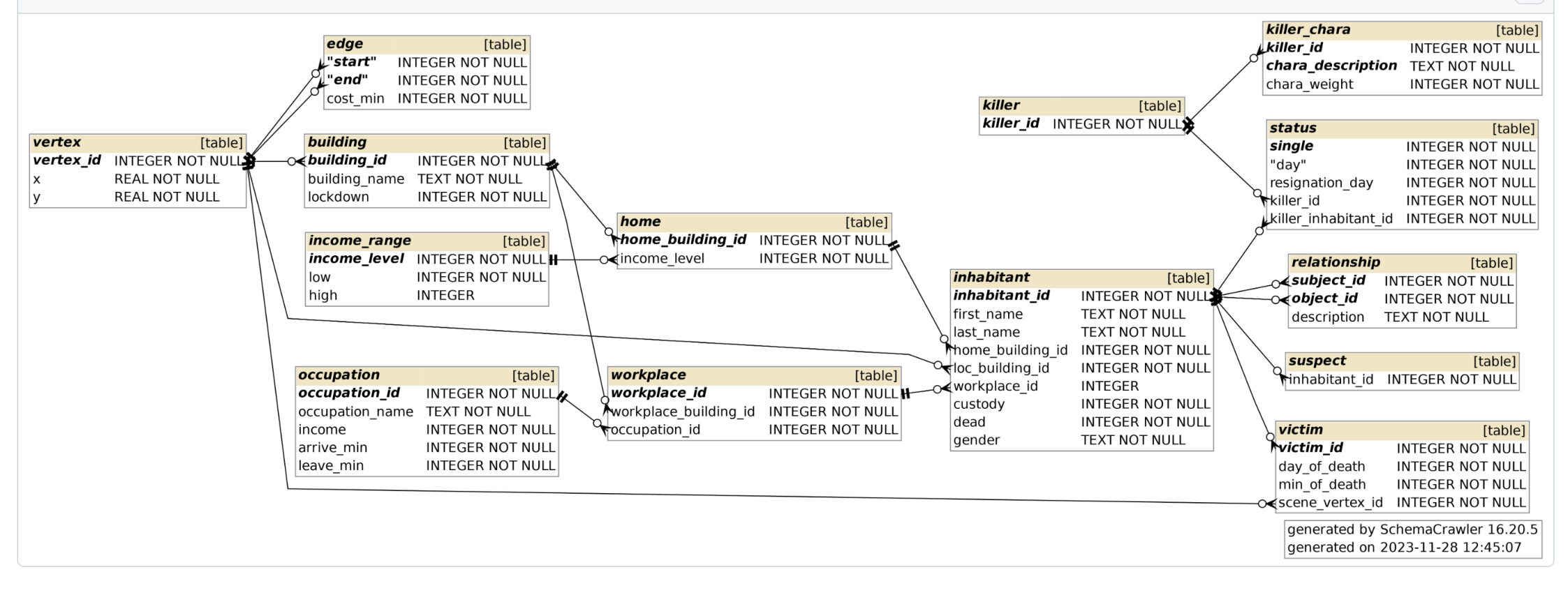
- Scarcity of murder cases for detective work training
- Provides a training ground for aspiring detectives

Simulation overview:

- Killer selected
- Kills every turn
- Simulation ends when murderer is accused or when time limit has been reached

DATABASE SCHEMA

Database Schema Overview



QUERIES WORTH NOTING

Random Generation of Loc-Time Pairs

```sql
CREATE TABLE src_dst(
  inhabitant_id INTEGER NOT NULL,
  src           INTEGER NOT NULL,
  dst           INTEGER NOT NULL,
  t_src         INTEGER NOT NULL,
  t_dst         INTEGER NOT NULL,
                PRIMARY KEY(inhabitant_id, src, dst)
);

CREATE TABLE loc_time(
  inhabitant_id INTEGER NOT NULL,
  vertex_id     INTEGER NOT NULL,
  arrive        INTEGER NOT NULL,
  leave         INTEGER,
  dst           INTEGER NOT NULL,
  t_dst         INTEGER NOT NULL,
                PRIMARY KEY(inhabitant_id, vertex_id, arrive)
);
```

Random Generation of Loc-Time Pairs Cont'd

```sql
CREATE TRIGGER insert_loc_time AFTER INSERT ON loc_time
WHEN
  NEW.vertex_id <> NEW.dst
BEGIN
  INSERT INTO loc_time
      SELECT NEW.inhabitant_id, end, NEW.leave + cost_min,
             NEW.leave + cost_min + ABS(RANDOM()) % (NEW.t_dst - (NEW.leave + cost_min) + 1
             - (SELECT MIN(d) FROM dist WHERE dist.src = end AND dist.dst = NEW.dst)),
          NEW.dst, NEW.t_dst
        FROM edge
       WHERE start = NEW.vertex_id
             AND NEW.leave + cost_min +
             (SELECT MIN(d) FROM dist WHERE dist.src = end AND dist.dst = NEW.dst)
             <= NEW.t_dst
    ORDER BY RANDOM()
       LIMIT 1;
END;
```

Kill Sequence

```
DROP TEMP TABLE IF EXISTS pot_victim;
CREATE TEMP TABLE pot_victim(
    inhabitant_id INTEGER,
    vertex_id INTEGER,
    start_min INTEGER,
    end_min INTEGER
)
```

- Marking of potential victim
  - Intersection of time and space
- MAX and Min for noting intersecting interval

```
INSERT INTO pot_victim
SELECT DISTINCT B.inhabitant_id, B.vertex_id, MAX(A.arrive, B.arrive), MIN(A.leave, B.leave)
FROM status, loc_time AS A, loc_time AS B
WHERE A.inhabitant_id = status.killer_inhabitant_id AND
        A.vertex_id = B.vertex_id AND
        A.arrive <= B.leave AND
        A.leave >= B.arrive
```

Kill Sequence Cont'd

```
DROP TEMP TABLE IF EXISTS weighed_pot_victim;
CREATE TEMP table weighed_pot_victim (
  inhabitant_id INTEGER,
  description TEXT,
  chara_weight INTEGER,
  vertex_id INTEGER,
);
```

- Weighing potential victims based on Killer_chara
- Cartesian product on killer_chara
  - Matching inhabitants with characteristic by case

```
INSERT INTO weighed_pot_victim
WITH killer_info AS (
    SELECT inhabitant.* FROM status, inhabitant
    WHERE status.killer_inhabitant_id = inhabitant.inhabitant_id
)
SELECT inhabitant_id, description, chara_weight, vertex_id
FROM pot_victim NATURAL JOIN inhabitant AS i NATURAL JOIN home AS h NATURAL JOIN workplace,
    killer_chara AS k, status
WHERE status.killer_id = k.killer_id AND
    CASE
        WHEN k.description = "low income" THEN income_level = "low income"
        WHEN k.description = "high income" THEN income_level = "high income"
        WHEN k.description = "neighbor" THEN EXISTS(
            SELECT * FROM killer_info
            WHERE h.home_building_id = killer_info.home_building_id
        )
        WHEN k.description = "rapist" THEN EXiSTS(
            SELECT * FROM killer_info
            WHERE killer_info.gender <> i.gender
        )
        WHEN k.description = "colleague" THEN EXISTS(
            SELECT * FROM killer_info WHERE killer_info.workplace_id = i.workplace_id
        )
        ELSE EXISTS(
            SELECT * FROM relationship
            WHERE subject_id = killer_inhabitant_id AND
                object_id = inhabitant_id AND
                relationship.description = "Relative"
        );
```

DEMO

Comparison with Existing Application: *SQL Murder Mystery*

A crime has taken place and the detective needs your help. The detective gave you the crime scene report, but you somehow lost it. You vaguely remember that the crime was a **murder** that occurred sometime on **Jan.15, 2018** and that it took place in **SQL City**. Start by retrieving the corresponding crime scene report from the police department's database.

Exploring the Database Structure

Experienced SQL users can often use database queries to infer the structure of a database. But each database system has different ways of managing this information. The SQL Murder Mystery is built using SQLite. Use this SQL command to find the tables in the Murder Mystery database.

Run this query to find the names of the tables in this database.

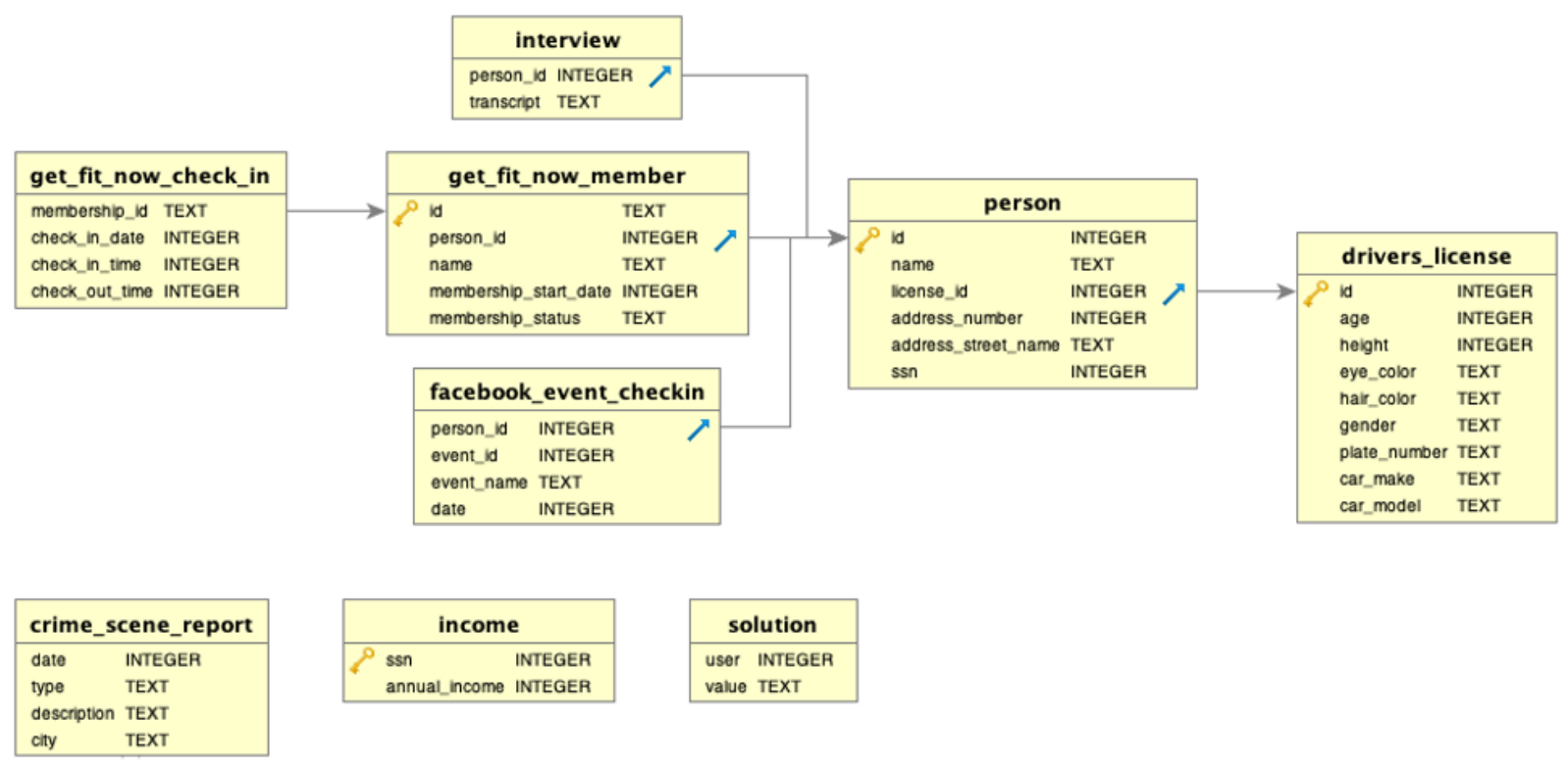
This command is specific to SQLite. For other databases, you'll have to learn their specific syntax.

```
1 SELECT name
2   FROM sqlite_master
3  where type = 'table'
```

RUN ⇩ RESET

https://github.com/NUKnightLab/sql-mysteries

Comparison with Existing Application: *SQL Murder Mystery*



Q&A

Contributions

- Project link: https://github.com/chikubidestroyer/D.-Simulator

- Database schema version 1:
  - Collective effort amongst Isaac, Yuxiang, and Shanruo
  - Annotations written by Shanruo
- Database schema version 2:
  - ER modeling designed by Isaac
  - Changes to final database schema reflected by Yuxiang
- Database data generator:
  - building, income range, home, occupation and workplace written by Yuxiang
  - vertex, edge, inhabitant, killer, killer chara, status, relationship, victim written by Isaac

Contributions Cont'd

- Query implementation:
  - Shortest path, inhabitant location-time pair generation, witness count, and plausible via point implemented by Yuxiang
  - Victim selection, victim commonality, inhabitant query implemented by Isaac
  - Isaac also experimented with implementing the inhabitant location-time pair generation in procedures, which is not used in the final game
- Gameplay implementation:
  - Simple queries that provide basic information to the users are implemented by both Yuxiang and Isaac
  - Load-save functionality is written by Yuxiang
  - Checking the end-game condition and a view for edges that are not blocked *is written by Isaac*
- Front-End UI:
  - Individual effort by Yuxiang