Introduction to Database Systems
Homework 3: Design Improvements and Tuning

Eleni Tzirita Zacharatou and Omar Shahbaz Khan

Fall 2023

1 SQL DDL [30 points]

Write SQL DDL commands to create the WASP database from Homework 2. Your script must implement the official solution ER diagram posted on LearnIT, must run in PostgreSQL as a whole, and must use the methodology given in the textbook and lectures. The relations must include all relevant primary key, candidate key, foreign key, and NOT NULL constraints. If certain restrictions cannot be enforced by these standard constraints, you can omit them (but if you are having fun, feel free to implement queries to detect violations, or triggers to prevent them). Please keep in mind that your SQL DDL script should address any additional requirements from Homework 2 that couldn't be represented in the ER diagram but can be implemented in the DDL (e.g., additional keys). Select reasonable types for attributes.

2 Practical Normalization [30 points]

In the homework quiz on LearnIT, you will find a script to create and populate five independent relations, each of which has seven columns and a primary key with a few thousand rows. For this project, *use only the Rentals relation!*

Each relation models a potential real-life database design situation, with some design flaws that must be addressed. In short, each of the relations has embedded a set of functional dependencies, which are not stated explicitly. You must a) find these dependencies and b) use them to guide the decomposition of the relations to 3NF/BCNF normal forms, using the methodology covered in class.

Note: In the homework quiz on LearnIT, you can find some additional material (slides, code, and video) to help you get started with this exercise.

2.1 Steps

More specifically, for the Rental relation, take the following steps:

(a) Find all the important FDs in the relations, given the constraints and simplifying assumptions that are listed in detail below.

Note: We strongly recommend to use your favourite programming language to generate an SQL script with all the FD detections queries, using the SQL query template covered in slide 44 of the provided slides for this quiz. The program could take as input a list of column names, and output all the required SQL queries into a text file. You can then run the text file using `psql`*. Figure 1 shows example Python code to get you started.*

(b) Decompose the relation until each sub-relation is in 3NF/BCNF, while preserving all non-redundant FDs. Write down the resulting schema description in a simple Relation(columns) format.

(c) Write the detailed SQL commands to create the resulting tables (with primary keys and foreign keys) and populate them, by extracting the relevant data from the original relations.

Note: In this homework, use the "homework" version of commands to create the new relations (see slide 9 of the provided slides for this quiz), so that you can compare the result of the decomposition with the original relation. Specifically, you should not run any ALTER TABLE commands.

(d) Select the correct normal form for the decomposed schema.

You can find a video from last year explaining the process on LearnIT.

```
-------------------------------------------------------------------------
def PrintSQL(Att1,  Att2)
    print ("Here make the SQL query to check %s --> %s" % (Att1, Att2));
    print ("Use the query in slide 44 of the homework slides on LearnIT");

R = ['A', 'B', 'C']
for i in range(len(R)):
    for j in range(len(R)):
        if (i != j):
            PrintSQL(R[i], R[j])
-------------------------------------------------------------------------
```

Figur 1: Python skeleton for code to check all FDs for a relation $R(A, B, C)$ with three attributes in step (a).

2.2 Simplifying Assumptions = Reality

In this project, assume that the following simplifying assumptions hold for each of the relations (this is the "reality" against which you check them):

- The relations must each be considered in isolation. The columns have short names that hint at their meaning, but you should not count on implicit FDs derived from the expected meaning of the columns. In short, the column names may be misleading.

- Assume that all functional dependencies in each relation (aside from primary key dependencies and dependencies derived from that) can be found by checking only FDs with one column on each side, such as $A \rightarrow B$.

  Note: As discussed in lectures, you can use SQL queries to detect potential FDs. Using the assumptions above, it is indeed relatively simple to create a program to output queries for all possible checks for FDs, which you can then run with `psql`*, thus automating the detection process.*

- If you find a functional dependency that holds for the given instance, you can assume that it will hold for all instances of the relation.

  Exception: When an ID column and a corresponding string column both determine each other, consider that only the ID column determines the string column, not vice versa. For example, if both $CID \rightarrow CN$ and $CN \rightarrow CID$ are found to potentially hold, then consider that only $CID \rightarrow CN$ is valid.

- The only dependencies you need to consider for decomposition are a) the dependencies that can be extracted from the data based on the assumptions above, and b) the given key constraints. As covered in the lecture, you can ignore trivial, unavoidable and redundant functional dependencies.

3 Index Selection [20 points]

Consider the following relation with information on luxury properties:

Property (id, type, price, square_meters, ...)

This table stores luxury properties that are very expensive but not very big (typically between 100-200 square meters). The average price per square meter is 300,000. You are given a set of existing *unclustered* B+-tree indexes for the Property relation. For each query, select the index (or indexes) that a good query optimizer is most likely to use to process the query, or select "no index" if a full table scan would yield better performance than any of the available indexes.

3.1 Available Indexes

The available indexes are:

(a) Property(id)

(b) Property(square_meters)

(c) Property(price)

(d) Property(square_meters, price)

(e) Property(square_meters, price, id)

(f) Property(square_meters, price, type)

(g) No index

3.2 Queries

The queries are:

Query 1:
```
select id, type, price
from Property
where square_meters > 200;
```

Query 2:
```
select square_meters
from Property
where price = 25,000,000;
```

Query 3:
```
select id
from Property
where square_meters > (select AVG(price)/250,000 from Property);
```

Query 4:
```
select id, type, price
from Property;
```

4 Passing Grade

The maximum grade is 80 points. You can see the detailed breakdown of points on the LearnIT quiz. To pass the homework, you need to have *at least 20/80* points.