COSC3380: Checking Referential Integrity and Normalization in a Relational Database

Instructor: Carlos Ordonez

1 Introduction

You will develop a program that generates SQL statements (SELECT + temp tables) to check referential integrity and normalization (simplified).

The input is a database consisting of a set of SQL tables (relations) and a primary key (PK) for each table, as well as foreign keys. We assume all keys are simple keys (one column). The schema of the tables will be specified as a text file, and the actual tables will be already stored in the database. While your program must generate the output as text files, all the processing will be done with SQL queries.

The output is a single text file showing if each table has referential integrity and if it is normalized or not. More details below.

2 Program requirements

Your program will generate SQL code based on input tables whose basic schema and referential integrity constraints are defined in a simple text file.

1. Referential integrity:

   For each table the program must state if referential integrity is correct: Y/N. Since the key is simple and we assume PKs are clean you can check only 3NF.

2. Normalization:

   For each table the program must state if it is normalized Y/N. Since the key is simple and we assume PKs are clean you can check only 3NF with simple keys.

3. Languages and OS.

   The OS for our DBMS and for testing is Linux (any distro is fine). However, it is feasible to develop programs in Windows and MacOs and later test them in Linux. The programming language is: Python (Java or Javascript require instructor permission). The DBMS is: PostgreSQL.

4. You cannot export or read tables row by row, doing processing outside the DBMS. The bulk of processing must be done with SQL queries and temporary tables.

5. At the end, you must display a summary. Your program must state Y/N if the database fulfills referential integrity. If all tables are in 3NF (and therefore BCNF) your program must state Y.

2.1 Programming details

1. The input is a text file with the table schemas, primary and foreign key and referential integrity. The user must provide the input text file with database name, table names, column names and keys. If a table has no PK specified or FK has an invalid column name, your program should display an error message and skip such table. Notice important aspects: (1) the database may contain more tables than those given in the input file; (2) the text file may contain less columns that the actual columns in the table; (3) a FK specification is optional: there may be tables without FKs.

2. The output is a text file displaying Y/N for each table, for referential integrity and normalization, respectively. The file format will be specified by the TAs.

3. Your program should generate a plain SQL script file for all the queries used in your program (correctly formatted as shown in the textbook and seen in class, following SQL format/indentation standards).

4. The input text file specifies the tables to be analyzed, and their schema. You can assume each table has one primary key and up to 3 foreign keys.

5. The program should drop any derived table in advance to avoid exceptions.

6. SQL code generation: Your program will get just one key without specifying other (secondary) candidate keys. You can use any combination of SELECT statements, temporary tables and views. You can name your tables/columns any way you want, and assume unique column names in the database. You are not allowed to modify the input tables in any way.

7. Python: you can use any Py library or Python statements to parse the input text file. You can store table and column names on any data structure including lists, arrays or trees. You do not have to worry about Python speed or memory usage since the ”heavy lifting” is done by the DBMS.

8. SQL:

   Write to a text file ”checkdb.sql” all the SQL statements generated during the checking of normalization. This SQL should be well formatted to be understood by a person, not as a long line.

   SQL is not case sensitive. Therefore, convert all your input SQL, tables names, and so on to either lower or uppercase. It is preferred SQL keywords are in uppercase and table/column names in lower case.

3 Program call and output specification

Here is a specification of input/output. There is only one input parameter: the input file with table schemas.

- syntax for Python call:
  “python checkdb.py database=dbxyz.txt”.

- If the connection to the DBMS fails, the program must display an error message. If the connection to the DBMS is successful, you should generate the “checkdb.sql”script file regardless the selected option, and dump clean SQL statements generated by your program.

- Input database/tables checking:

  Your program will be tested with several database schema text files that are ”clean”. But you must make elementary error checking for non-existent tables and invalid column names.

The input tables have integer numbers or strings (no dates, no floating point).

- The SQL of your generated script file should be properly formatted and indented (one SELECT term per line, JOIN/WHERE/GROUP in different line).
- Output:

  A text file, with one checked table per line. The program must work in two phases: (1) referential integrity; (2) normalization. Output sorted by table name.

4 Examples of input and output

Recall that all tables are given in the input database file. You are not expected to figure out table and column names from the database connection, but you can do it.

Input specification example:

```
dbxyz.txt
--------------------------------------
T1(K(pk),K2(fk:T2.K2),A,B)
T2(K2(pk),C)
T3(K3(pk),D,E)
T4(K4(pk),K5(pk),K3(fk:T3.K3),F)
--------------------------------------
```

Sample output file:

```
refintnorm.txt
----------------------------------------
     referential integrity     normalized
T1                Y                  Y
T2                N                  Y
T3                N                  Y
T4                Y                  Y

DB referential integrity: N
DB normalized: Y
----------------------------------------
```

5 Grading

The TAs will provide the input file and the expected output for a sample test database. There will be several tables in such database. These test cases will give the students the opportunity of evaluate their own projects, and make corrections before the official submission. The TAs will use a different database for grading, with different table and column names. The TA will use scripts for automatic grading and will use a different database as input, with different table and column names. You can expect tables to have no more than 200 rows, to make SQL development faster and easier to check.

Score: A program not uploaded to the UH linux server will get 0 (zero, the TAs have no obligation to run testing outside this server). A program with syntax or data type errors will get 10. A working program,

producing correct for some cases, will get a score 30 or better. A program passing checks with most tables can get 60 or better.

Code originality: The Python code and SQL statements will be checked for plagiarism with source code analysis tools (we cannot disclose them): any significant similarity with some other student code will be reported to the instructor. Notice reformatting code, changing variable names, changing order of functions, changing for loops for while loops, repackaging into different files, adding spurious variables; all of them can be easily detected since the ”logic” solving the problem is the same. Warning: copying/pasting code from the Internet (stack overflow, github, tutorials) may trigger a red flag: the instructor will decide if there is cheating or not.