COSC3380: Checking Referential Integrity and Normalization in a Relational Database

Instructor: Carlos Ordonez

1 Introduction

You will develop a program that generates SQL statements (CREATE TABLE, SELECT, DELETE FROM) to check referential integrity and normalization (simplified).

The input is one database, consisting of a set of SQL tables (relations) with one primary key (PK) for each table, as well as a list of foreign keys (FKs). The input database may change in each run. The schema of the tables will be given as a text file, with one table per line. The actual tables with data will be already stored in the input database. Your program must generate a short output text file, but 90% of processing will be done with SQL queries. You may create temp tables or views, to solve the problem in steps.

IMPORTANT: To simplify, all keys are simple keys (one column) and we will not do normalization with respect to every candidate key (CK) (general, more complicated textbook definition). That is, do not worry about composite keys (2 or more attributes) and do not worry about CKs in the input file.

The input is a text file with a list of tables already stored inside the database, where each table has a PK and a potential list of FKs (zero or more). The output is a single text file showing if each table fulfills referential integrity and if it is normalized or not (on the normal form (2NF, 3NF, BCNF) indicated below). More details below.

2 Theory to solve HW

You can easily translate these equations into SQL to solve this homework. You should use the JOIN keyword to join tables instead of the WHERE clause. Set size can be computed with a simple SELECT count(*) query on SELECT.. GROUP BY or SELECT DISTINCT... There are many ways to write the queries, with/without DISTINCT or GROUP BY, with/without temp tables.

Example: Consider two tables $T_1(\underline{J}, K, B, C, D), T_2(\underline{K}, E, F)$, where T_1 has FK K, referencing T_2.

- Referential integrity. Referential integrity is correct if $|T_1| = |T_1 \bowtie_K T_2|$. It is acceptable $|T_2| \neq |T_1 \bowtie_K T_2|$.
- Functional dependencies: FDs $J \rightarrow K$, $J \rightarrow B$ come from PK J. FD $B \rightarrow C$ if $\pi_B(T_1) = \pi_{BC}(T_1)$. FD $D \rightarrow K$ if $\pi_D(T_1) = \pi_{DK}(T_1)$.
- T_1 is in 3NF/BCNF if every FD has J on the left side. In other words, there are no FDs with K, B, C, D on the left side, unless they are CKs.
- T_2 is in 3NF/BCNF if every FD has K on the left side. In other words, there are no FDs with E, F on the left side, unless they are CKs.

3 Program requirements

Notice the tables may change for each run. Therefore, your program must generate (dynamically create) SQL statements based on input tables whose basic schema and referential integrity constraints (foreign keys) are defined in a simple text file.

1. Referential integrity:

   For each table the program must state if referential integrity is correct: Y/N.

   Simplification: Since the PK and FKs are simple and we assume PKs are clean you can directly check 3NF/BCNF, jumping 1NF (trivial in SQL) and 2NF (only a conmcern for composite PKs). However, you are encouraged to review 1NF, 2NF definitions in the textbook. A table without a FK is trivially correct.

2. Normalization:

   For each table the program must state if it is normalized Y/N.

   Simplification: Since the PK is simple and we assume PKs are clean you can check only 3NF/BCNF with simple keys. 3NF is violated if an FD does not involve the PK and there are duplicate rows in the projection.

3. Languages and OS.

   The OS for our DBMS and for testing is Linux (Amazon cloud server provided). However, it is recommended 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 logic and 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.

3.1 Programming details

1. INPUT

   The input is a text file with the list of table schemas, primary and foreign key (referential integrity). The python program call must provide the input text file, which contains 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. That is, your program should continue processing.

   important aspects about input file: (1) the database may contain more tables than those given in the input file (they should be ignored); (2) the table description may contain less columns that the actual columns in the table; (3) a FK specification is optional: there may be tables without FKs, but they are probably referenced by other tables.

   Notice this text file is a substitute of a connection to the DBMS to extract column names and data types, from the internal catalog, which would make code development significantly more complicated (interactive). In short, this input text file is a simple mechanism to show database information. You can expect the input file to be clean, but there may be minor unintentional errors. That is, you are not expected to run extensive syntax checks on this file.

2. OUTPUT

   The output is a text file displaying Y/N for each table, for referential integrity and normalization, respectively. The specific file format and test cases will be specified by the TAs (do not include any extra output or messages, do not create one file per table).

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. Notice the table may have candidate keys, not indicated in tghe text file. In general, the first column will the PK, but FKs can appear anywhere (2nd, 3rd column, and so on).

5. Your program should not crash and stop if there is some error or referential integrity violation. The program should drop any derived table in advance to avoid exceptions. Test your program well with invalid input.

6. SQL code generation: There is only 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, hash tables 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.

4 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:
  "python3 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. That is, you should not waste time catching every potential error in the text file like a missing parenthesis, pkk instead of pk, repeated column name, inconsistent column names, and so on.

  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 should be sorted by table name.

5 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. Notice that you cannot state if a table is OK, by just reading the text file (other than tables without FKs)

Input specification example:

```
dbxyz.txt
-------------------------------------
T1(id(pk),i(fk:T2.i),A,B)
T2(i(pk),C,D)
T3(j(pk),E,F)
T4(deptid(pk),deptname,j(fk:T3.j),numemployees)
-------------------------------------
```

Output file example (output depends on input table contents):

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

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

6 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.