CPS 216 Spring 2003
Homework #1
Assigned: Wednesday, January 22
Due: Monday, February 10

Note: This is a long homework. Start early! If you have already taken CPS 196.3 or an equivalent undergraduate course in databases, you may skip Problems 1-5 and complete Problems 6-8. Otherwise, you may complete Problems 1-5 and skip Problems 6-8.

Problem 1.

Consider a database containing information about military aircrafts used in the World War II. This database consists of the following four relations:

Model (*model*, *type*, *country*, *payload*, *range*, *speed*)
Plane (*id*, *model*, *year*)
Battle (*name*, *year*)
Outcome (*plane*, *battle*, *result*)

All aircrafts are categorized into models. The relation *Model* records the name of the model (e.g., "B-52," "F4F"), type (e.g., "fighter," "bomber," "reconnaissance plane"), the country that made the model, the payload of the model, its maximum range (in miles) and speed (in MPH). Relation *Plane* records the unique identification string of each plane, its model, and the year in which it entered service. Relation *Battle* gives the name and the year of each battle involving planes. Relation *Outcome* records the result ("downed," "damaged," "ok") for each plane in each battle. The key attributes are underlined. Please write ***relational algebra*** expressions to answer following queries.

(a) List model name and country for models with range over 1,500 miles.
(b) Find ids of planes with speed no less than 300 MPH.
(c) List ids and models of planes that were downed in the battle of "Pearl Harbor."
(d) For each plane engaged in the battle of "Midway," list its id, payload, speed, and the year in which it entered service.
(e) Find those countries that have both bombers and reconnaissance planes.
(f) Find those planes that "lived to fight another day," i.e., they were damaged in one battle but later fought in another.
(g) Find the model with the highest payload.
(h) Find ids of those planes that fought ***only*** in the battles that plane "007" fought in.
(i) Find ids of those planes that fought in ***every*** battle that plane "007" fought in.

Problem 2.

As discussed in class, the core operators in relational algebra are selection (σ_p), projection (π_L), cross product ($\times$), union ($\cup$), and difference ($-$).

(a) Show that the projection operator is necessary; that is, some queries that use the projection operator cannot be expressed using any combination of the other operators.

(b) Show that the union operator is necessary; that is, some queries that use the union operator cannot be expressed using any combination of the other operators.
(c) Show that the selection operator is necessary; that is, some queries that use the selection operator cannot be expressed using any combination of the other operators.

Problem 3.

Consider a relation R (A, B, C, D) with FD's $AB \rightarrow C$, $C \rightarrow D$, and $D \rightarrow B$.

(a) Show that $\{A, B\}$ is a key of R (remember a key has to be minimal).
(b) What are the other keys of R? (Hint: A must be in every key of R; why?)
(c) $D \rightarrow B$ is a BNCF violation. Using this violation, we decompose R into R_1 (B, D) and R_2 (A, C, D). What are the keys of R_1?
(d) What are the FD's that hold in R_1? Do not list them all; instead, give a set of FD's from which all other FD's in R_1 follow. This set of FD's is called a *basis*. When checking for BCNF violations, it suffices to check just the basis.
(e) Is R_1 in BCNF? Briefly explain why.
(f) What are the keys of R_2? (Hint: There is more than one.)
(g) What are the FD's that hold in R_2? Again, do not list them all; instead, give a basis.
(h) Is R_2 in BCNF? If yes, briefly explain why. Otherwise, decompose further until all decomposed relations are in BCNF, and then show your final results.

Problem 4.

Consider the following schema. The catalog lists prices charged for parts by suppliers.

Supplier (*sid*, *sname*, *address*)
Part (*pid*, *pname*, *color*)
Catalog (*sid*, *pid*, *price*)

Log into `rack40.cs.duke.edu` and run `~cps216/examples/db-supply/setup.sh` to setup a database with some sample data. For the SQL database schema, please refer to the file `create.sql` in the same directory. Write SQL statements to answer the following queries. Make sure that the result contains no duplicates. Use `DISTINCT` only when necessary.

Write all your queries in file named `hw1-4.sql`. When you are done, run "`db2 –tvf hw1-4.sql > hw1-4.out`" (you may need to run "`db2 connect to cps216`" before that and "`db2 disconnect all`" afterwards). Then, print out the file `hw1-4.out` and turn it in together with the rest of the homework.

You may find it useful to consult the online document titled "DB2 SQL Programming Notes," found under the "Programming Notes" section of the course Web page (`http://www.cs.duke.edu/courses/spring03/cps216/`).

(a) Find the names of parts for which there is some supplier.
(b) Find the names of suppliers who supply ***only*** red parts.
(c) Find the ids of suppliers who supply at least one red part ***and*** at least one green part.

(d) Find the names of parts supplied by "Lockheed Martin" and by no one else.
(e) Find the average price of red parts.
(f) Find the ids of suppliers who charge more for ***some*** part than the average price of that part.
(g) For each part, find the names of suppliers who charge most for that part.

Problem 5.

You are asked to design a relational database for a store. After brainstorming with the store managers, you come up with the following specification:

- The store has multiple departments, identified by their names.
- Each department may have many employees but only one of them is the department manager.
- Employees are identified by their names. We also need to record their salaries. Each employee may work in only one department. Managers are employees as well, and they manage the departments they work in.
- The store sells various items identified by item ids. Each item is carried by exactly one department, while each department may carry many items. For each item, we also need to keep a short description and its quantity in stock.
- The store deals with a number of suppliers identified by their names. We need to record their addresses. Each supplier supplies an item at a particular price. A supplier may supply any number of items, and the same items could be supplied by different suppliers (probably at different prices).
- The store receives orders identified by order ids. Each order has a date, a shipping address, and may include different quantities of multiple items.

You are encouraged (but not required) to use the E/R model to help you design the schema. Make sure that you apply the relational database design theory to check for redundancies in your design, and remove them if necessary.

Keep all SQL statements you write for this problem in a file named `hw1-5.sql`. When you are done, run "`db2 –tvf hw1-5.sql > hw1-5.out`". Then, print out the file `hw1-5.out` and turn it in together with the rest of this homework.

(a) Create the final relational schema using SQL `CREATE TABLE` statements on the DB2 server. Choose appropriate data types for your columns, and remember to declare any keys, foreign keys, `NOT NULL`, and `CHECK` constraints when appropriate.
(b) Make up some sample data and write SQL `INSERT` statements to populate the database. Each table should contain at least five rows.
(c) For each table you have created and populated, run the following query: "`SELECT * FROM` *table_name* `FETCH FIRST 10 ROWS ONLY;`".
(d) Write a query to find, for each department, the name of its manager, the number of employees in this department, the number of orders for items carried by this department, and the number of suppliers that supply items carried by this department. (Each order should be counted at most once even if it contains multiple

items carried by this department. Similarly, each supplier should be counted at most once even if it supplies multiple items carried by this department.)

Problem 6.

The following questions are based on the paper "A Relational Model of Data for Large Shared Data Banks," by Codd.

(a) Suppose that an instance of R (A, B) is "joinable" with an instance of S (B, C). What functional dependencies must hold in order for the "join" of R with S to be unique (as discussed by Codd in Section 2.1.3)?
(b) Why didn't Codd include the selection operator (σ_p) in the list of relational operators he discussed in Section 2.1?

Problem 7.

The following questions are based on the paper "A History and Evaluation of System R," by Chamberlin et al.

(a) In Section 3 under the subsection titled "The Optimizer," authors described two join methods that were believed to be optimal or nearly optimal under most circumstances. One notably efficient join method is missing, however. What is this missing join method?
(b) In Section 4 under the subsection titled "The SQL Language," authors introduced the notion of "outer-joins." Subsequently, syntax for outer-join was added to the SQL standard. Strictly speaking, however, the new syntax is not necessary (except perhaps from a user-friendly point of view). Show that you can write the outer-join between tables R (A, B) and S (B, C) in SQL without using the outer-join syntax.

Problem 8.

For this problem, you may work either independently or in groups of two. Your task is to develop a relational ***algebra*** query interface for DB2. Your interface should accept relational algebra query strings as input, translate them into SQL, send the SQL queries to DB2, and print out the results received from DB2. In developing this interface, you may need to consider the following issues:

- *Relational algebra syntax and parsing.* First, you will need to define the syntax of relational algebra queries and write a parser to convert arbitrarily complex query strings into an internal representation (perhaps in the form of an expression tree). Since parsing is not our focus, you may choose a syntax that is easy to parse. For example, you can use Reverse Polish Notation to avoid the use of parentheses to define the nesting of operators. A query

  $$\pi_{R.A,\, S.D} (\sigma_{B = \text{'Bart'} \vee C = 5} R \bowtie_{C \leq D} S)$$

  might be encoded as "`R \select{B = 'Bart' or C = 5} S \join{C <= D} \project{R.A, S.D}`", with the understanding that join is a binary operator while select and project are unary operators.

- *Using schema information.* For some relational algebra queries, knowledge about the schema is required for correct translation. For example, to translate the natural join query, $R \bowtie S$, we need to know the names of columns in R and S in order to generate the correct `SELECT` and `WHERE` clauses. (SQL actually supports a `NATURAL INNER JOIN` construct officially, but unfortunately it is not implemented in most commercial systems.) For another example, to translate the full table/column rename operator, e.g., $\rho_{S(A, B, C)} R$, we need to know the names of columns in R in order to rename them in the `SELECT` clause. Thus, to do a complete job in translation, your interface needs to query DB2 for schema information.
  Another use of schema information is to detect semantic errors, e.g., references to nonexistent columns, type mismatches, incompatible schemas, etc.
  To make your life easier for this problem, you may choose not to support natural join and column rename, but you must support regular join and table rename (e.g., $\rho_S R$). You may also delegate detection of semantic errors to DB2, but you must pass on any error messages returned by DB2 to the users of your interface.
- *JDBC/ODBC application or `db2` wrapper?* One approach is to implement the interface as a JDBC/ODBC database application. This approach allows the interface to query DB2 for schema information and use it in translation and error checking. The downside of this approach is that the interface must also handle the presentation of query results and error messages. An alternative approach is to implement the interface on top of the `db2` command-line processor. The interface simply needs to translate the query into SQL, execute SQL using `db2`, and pass on whatever output back to the user. This approach is simpler to implement and easier to debug, but accessing schema information is more difficult.
- *Difference between set and bag semantics.* Remember that relational algebra is duplicate-free, while SQL permits duplicates. You may need to use `DISTINCT` when necessary.
- *Additional "bells and whistles."* This problem is open-ended. If you have time and enjoy programming databases, you might consider adding the following features:
  - A graphical user interface to help users explore the database schema and contents and construct relational algebra queries (perhaps as expression trees).
  - More informative error messages.
  - Support for natural join, column rename, outerjoin, semijoin, etc.
  - The ability to assign names to relational algebra expressions and use them in subsequent queries (just like SQL views).

For this problem, turn in a pointer to the directory containing your documentation, source code, and executable. Please document the relational algebra syntax you support, known bugs/limitations, nifty features you have implemented, and instructions to compile and run your program. The best interface will be used in teaching future database courses at Duke.