Project Reports

Project report #1

1. Describe the problem concisely. Why is a database (as opposed to, say, a simple set of files) a good idea for this task?

2. Describe the intended classes of users of your database system.

3. Identify 5 main "things" about which you will need to keep information, and the information you will need to keep.

4. Describe realistic situations where using your database system will require handling any two of the operations (see Tasks and Operations in the **narrative**); in each situation, you may consider one or more operations. (The assignment is to describe a *total* of two operations, using a *total* of either one or two situations.)

5. Sketch the APIs (application program interfaces; for this project, an API of an operation is just inputs and outputs for the operation) required for each of the operations listed in Tasks and Operations in the **narrative**. (In some cases the output will be just a confirmation.) Submit the APIs.

6. Give short descriptions (no more than 50 words each) of the views of the data that correspond to the intended classes of users, one view per user class. Each view should reflect all and only the data (rather than operations) in the database that is relevant to all operations for the given class of users. For example, the billing-staff view may reflect all basic information about the working staff, supplier, and customers, and may represent the entire database as just one table for them, including the amounts owed to suppliers, paychecks for staff, and cashback bonuses for customers.

7. Construct local E/R diagrams, one for each view in item 6. In each diagram, you need to reflect all database information that is relevant to all operations for the given class of users. (See item 6 above and the **narrative**).

8. Document the local E/R diagrams. Highlight any design decisions - specifically, describe why you have the entities and relationships that you do.

9. Derive a local relational schema from each of the above local E/R diagrams (see item 7).

10. Document the local relational schemas. Highlight any design decisions - specifically, explain why you decided to make the relations you did and how each entity and relationship in the E/R diagram is captured in the relational schema.

Project report #2

1. Derive a global relational database schema from the schemas you obtained at step 9 in project report 1. Normalize to at least 3NF. Document your database schema and the explanations why your relation schemas are in at least 3NF. Note: to conclude that your relation schemas satisfy (at least) 3NF (and to get full credit for this item), you need to consider and discuss *all* nontrivial functional dependencies that the instructor can reasonably assume might hold on your schemas, rather than only those dependencies that are obvious and do hold on your schemas.

2. Describe any design decisions for the global schema. Identify and explain all integrity constraints of the following types: NOT NULL, key, and referential integrity. Describe which attributes are allowed to be NULL, why, and what a NULL value means for each attribute on which it is allowed.
3. Using the terminal for MariaDB for all your relation schemas create base relations with the right attribute domains and with *all* the integrity constraints you listed in item 2 of this report.
   Populate the base relations with 4-8 rows each.

4. Write interactive SQL queries for each operation in the **narrative**, and test the queries on the terminal for MariaDB:

   4.1 Make assumptions to justify the constants in your queries: for example, for the operation "show all staff members who do billing" assume that the database has information about three staff members who do billing. Use these fictitious constants to ask specific SQL queries and to make specific updates on your stored relations. For the queries, choose the constants so that a non-empty answer set is returned; you may need to add additional rows to do so. For each operation in Tasks and Operations in the **narrative**, submit the query or update (whichever is appropriate) for the operation and the printout of the response to the query/update on the terminal for MariaDB. Note: A lot of points will be taken off if you miss some of the operations in the **narrative**.

4.2 Use the EXPLAIN directive in MariaDB to study execution plans for your queries (from item 4.1 above). Find two queries for which EXPLAIN shows full table scans; print the EXPLAIN outputs for these queries. For these two queries, create appropriate indexes; print the EXPLAIN output that shows the use of the indexes.
For each query, submit the printouts of:

(1) the SQL query,
(2) the execution plan (EXPLAIN) for the query which shows at least one full table scan,
(3) your index-creation statement in SQL, and
(4) the execution plan (EXPLAIN) for the query where the full table scan has been replaced with a use of your index.

4.3 For any two of your SQL queries ("select" statements only, rather than "insert/delete/update") with joins, explain why the queries are correct - one explanation per query. If you come up with erroneous solutions prior to the correct one, also include those as part of the explanation. For each query, submit the query itself and your explanation. Your explanations will consist of two parts:

- the corresponding relational algebra expression, and
- a *correctness proof* that shows that the query answer always corresponds to the query specification;
  for example, a correctness proof of the query

  “SELECT e.Lname, d.DeptName
  FROM Employee e, Department d
  WHERE e.DeptNo = d.DeptNo,"

  with specification
  "return the last names of all employees together with the names of their departments,"

  can be as follows:
  "Suppose *e* is any tuple in the Employee relation, and *d* is any tuple in the Department relation, such that the value *e.DeptNo* is the same as the value *d.DeptNo*. Each such combination of tuples *(e, d)*gives personal information about one employee, together with all information on the

department for that employee. For each such combination *(e, d)*, the query returns the value of Lname and the value of DeptName. These values are the last name of the employee and the name of the department for the employee. But this is exactly what our query should return; see the specification."

Project report #3

1. (Write all the required applications and test them appropriately. (A lot of points will be taken off if your code does not close DBMS connections.) Submit all source code to the submit board. Your application-code language must be different from SQL and must include functionalities for sending SQL commands to a database-management system and for receiving responses from the DBMS. Java with its JDBC functionalities is one example of permitted application-code language.

2. In at least two of the applications (see item 2 above), use transactions to ensure that the applications work correctly even if they encounter unexpected events. (Example: a credit-card authorization fails because a staff member has entered an invalid credit-card number.) Document the program logic of the transactions in the applications. Submit, as parts of your project report 3 paper, (1) the parts of the code that contain the transactions (points will be taken off by the grader if these parts of the code are not easy to locate), and (2) your documentation. Make sure that your transactions use the COMMIT and ROLLBACK statements.

3. Document your programs. The documentation should be part of the code (submitted via the submit board), but will be graded in addition. In a separate submission that is a part of your project 3 report paper, highlight *high-level* design decisions, which are any choices/decisions that you had to make when designing your database and applications. As part of the documentation submitted as part of your project 3 report paper, explain who in your team played what functional role (e.g., software engineer, database designer/administrator, etc, see above under Organization) in each part (1 through 3) of the project.