Database reverse engineering and assessment for Android applications

Project goal
In this project, you will play the role of expert consultants, whose mission is to analyze, understand and assess a real-life data-centered system. Each group (3 students) will study a different small/medium-sized open-source Android application. Each application is (partly) written in Java and uses one (or several) relational database(s).

The general goal of the group is to reverse engineer the database of the studied application, and assess both the quality of the database schema and of the database manipulation programs.

Recommended steps
The project includes 4 mandatory steps + 1 bonus step. The bonus step allows the group to get bonus points, and therefore to reach an excellent grade. Each step aims at applying the approaches, techniques, and tools presented during the lectures, to the Android project under consideration.

STEP 1

Reverse engineer the explicit physical database schema (PS) into a (possibly enriched) logical schema (LS) and then abstract LS as a conceptual schema (CS).

N.B. If the Android application relies on several databases, you may choose to reverse engineer only one of those databases, provided that you clearly justify your choice.

The schema enrichment can use as inputs:
- the explicit database schema (primary keys, column/table names and columns types);
- the applications programs (see STEP 2).

If the database schema does not include explicit foreign keys (FKs), the group will check if implicit FKs can be discovered from the schema or from the queries (see lectures).

If the database schema already includes explicit FKs, the group will check:
(1) to what extent explicit FKs can be inferred from the explicit physical schema or from the queries (compute precision and recall values - see lectures);
(2) if additional (implicit) FKs can be discovered.

The physical database schema will be enriched with the implicit foreign keys discovered, if any.

Output of step 1: A brief description of PS. Schema analysis and query analysis results (implicit FK discovery).
All database schemas (PS, LS and CS).

STEP 2
Locate and analyze the queries in the program's source code (SQLInspect may help you here). Derive the logical subschema (LSS), i.e., the logical subschema that is actually used by the programs. In other words, LSS can be obtained by removing all schema elements (tables/columns) from LS that are not used at all by the programs.

Output of step 2: Detailed, commented statistics about the database queries, their type (select | delete | insert | update | create | . . .), their complexity, their distribution over the code base, etc. The logical subschema (LSS).

STEP 3
Apply a *what-if analysis* to your system. Assume you must evolve the physical subschema (PSS) according to at least 10 different evolution scenarios: e.g., rename | update | remove certain tables/columns, split a given table into two tables, merge two tables into one table, add new columns, change the type of certain columns, remove a column from a table, etc.

For each schema evolution scenario considered, assess the impact on the programs, in terms of what has to be changed to preserve global consistency (whenever possible).

Output of step 3: Full description of each chosen evolution scenario and of its impact in terms of program adaptation, i.e., which source code fragments (classes | methods | lines of code) would be affected in this evolution scenario.

STEP 4
Comment on the global quality of the database schema and of the database manipulation code. Make some recommendations for improvement, if relevant.

Output of step 4: Documented list of merits and drawbacks of (1) the database schema and (2) the database manipulation source code. For each (type of) drawback(s), a list of concrete recommendations to improve the quality of the system.

BONUS STEP
Consider *several versions* of the studied application. Select at least 3 different versions: 1 version from the initial status of the development, 1 version from the middle of its evolution history, and 1 more recent version (e.g., the current version). Analyse the evolution of the DB schema and/or queries, and comment on the evolution of the global system quality.

Output of step BONUS: Illustrated description of your cross-version analysis results.

Final outcome
Each group will deliver:

(1) a project report in PDF format, summarizing the results of each step;
(2) a link to a public, documented GitLab/GitHub repository including the full results of steps 1 to 4.

The PDF project report will systematically refer to the contents of the GitLab/GitHub repository.

Dates and deadlines

October 19, 2023
Project kick-off and presentation of SQLInspect.

The teaching team post the project instructions and a documented list of candidate Android projects on WebCampus.

October 26, 2023
The groups of students are composed by the students and communicated to jean.albrecq@unamur.be

Each group chooses a spokesperson, a different Android project among the provided list.

The groups work on their project. There will be no lecture that week.

November 23, 2023
Project sanity check: each group will quickly report on progress status, completed steps, issues, etc.

Date of the oral exam (TBD), January 2024
The spokesperson sends the final project report (in PDF) to jean.albrecq@unamur.be

with email subject « INFOM218 project report - group # ».

On that date, the GitLab/GitHub repository will then be considered in its final state.

Each group of students presents its project results, with slides (15 min, including Q&A). Then each member of the group presents the individual oral exam (one student every 15 min).

Reminder:
Course grade = 40% project grade + 60% exam grade.