Relational Databases and SQL - Course Project Instructions

Uri Goldstein, IDC Herzliya

Version 1.2, 24/02/2021 – Clarified start and end dates for role assignments in 2.3.6
Version 1.1, 01/02/2021 – Fixed numbering and mix-up in 2.3.2 and 2.3.3
Version 1.0, 25/01/2021 - Published

1. Goal and Background

As this year's students you are tasked with submitting a database implementation project as the major part of your course grade. The goal of the project is to demonstrate your understanding and practical experience of the material of the course. Use this project to show how the skills you learned throughout the semester can be used to build a real, usable database.

The central task in this year's project is to implement the database for an ***RBAC mechanism***.

RBAC stands for ***Role-Based Access Control***. An RBAC mechanism a security feature of a software system that controls what data users can access and which actions they can perform in the system.

For example, consider the Moodle software that the IDC uses for e-learning. You, as users of type "student" in Moodle, can access only specific course websites that you registered for. Other students might not have access to the same courses.

Users in other roles, for example "course instructors", can do different things to courses, like change their contents or delete them entirely. All of this is managed through Moodle's internal RBAC mechanism.

In this project you will build a database that can provide an RBAC mechanism with (almost) all the data that it needs to get its job done.

2. Requirements

Below are the specific requirements to be implemented in your submission for the project. We start by reviewing the data entities, followed by their relationships, their data fields and finish with a small set of queries that you are to implement and include in your submission.

2.1. Entities

All the business entities of the RBAC mechanism are listed below. Your database must represent all these entities according to relational principles that we have learned in the course.

2.1.1. Subjects

A ***subject*** is a user of the system. The main function of the RBAC mechanism is to control what subjects can do in a software system.

2.1.2. Resources

A ***resource*** in the RBAC mechanism represents a part of the software system to which access needs to be controlled.

For example, consider the grade for one of your courses in Moodle. Access to the grade needs to be limited so that only you and specific faculty or administration members can see or change it. Similar examples can be email messages in Gmail or even a specific database table.

2.1.3. Operations

An ***operation*** represents a type of access to a resource. When you view your grade in Moodle, "view" is the operation (the grade being the resource). When a teacher "edits" your grade that is a different operation on the same resource.

Typical operations can be read, create, edit, delete, login, grant, deny, approve, enable, disable etc. etc.

Note that not all operations can be (or should be) applied to all resources. Only specific operations are paired to specific resources (typically 1-4, sometimes more).

2.1.4. Permissions

A ***permission*** is the approval for doing something in the software system. A permission is a pairing of an *operation* and a *resource*.

Examples can be:

- The permission to login (operation) to the system (resource).
- To read (op.) articles (res.) in an online scientific journal.
- To create (op.) a new document (res.) in an online office system.
- Even to grant (op.) RBAC access (res.) to other users.

2.1.5. Roles

A ***role*** typically represents a job function, or a group of subjects and the permissions assigned to them. For example, "student", "teaching instructor", "teaching assistant" and "system administrator" are all different roles in Moodle.

Roles are the main feature and innovation of RBAC. Using roles makes it a lot easier to map the many users of a software system to its many different types of permissions.

Subjects in the system are assigned zero or more roles. Each role grants its subjects a set of permissions.

For example, consider Homer Simpson.

Homer works at the Springfield Nuclear Power Plant as a safety inspector. As part of his job Homer can enter the plant's control room and inspect how the reactor is working.

In this example:

- The power plant is the ***system*** (it is not a software system but still makes a good example).
- Homer is the ***subject***.
- "Safety inspector" is the ***role***.
- Entering the plant's control room is a ***permission*** ("enter" is the ***operation*** and "control room" is the ***resource***).
- "Shutting down the reactor" is another example of a ***permission*** but it is **not** assigned to Homer's only ***role*** as "safety inspector" which means Homer is **not** allowed by RBAC to do it.

2.2. Relationships

Below is a list of all the entity relationships in our RBAC mechanism. **Note** the important distinction between "can" and "must".

- A ***role*** can have 0 or more ***subjects***.
- A ***subject*** can have 0 or more ***roles***.
- A ***role*** can have 0 or more ***permissions***.
- A ***permission*** can be part of 0 or more ***roles***.
- A ***permission*** is a pairing of a single ***operation*** to a single ***resource***.

2.2.1. Role Hierarchy

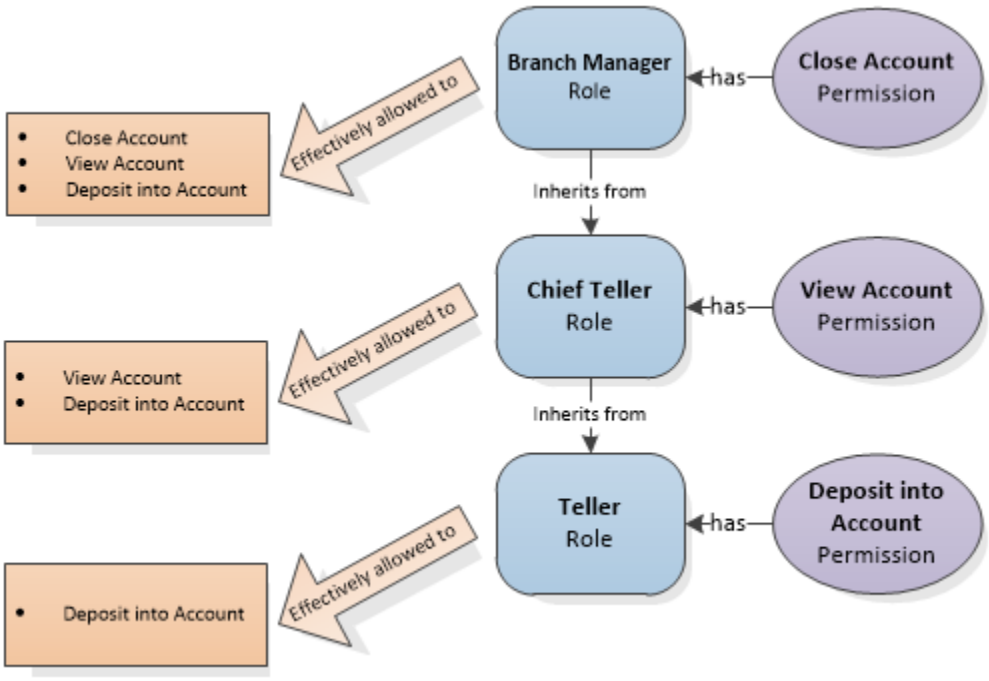
In our RBAC implementations Roles are arranged in a hierarchy where one role can inherit the permissions (but not the subjects!) of another role. To understand this, consider the following example of a bank.

The lowest level role in our example bank would be that of a "teller". Tellers can deposit money into a customer's account, but they are now allowed to see the account details.

A higher-level role is that of the "chief teller". The chief teller inherits from the "teller" role so they can deposit money for customers. The chief teller role also has an extra permission directly assigned to: "view customer account".

The highest-level role in the bank is that of a "branch manager". It inherits from the "chief teller" role and has an extra, direct permission to close a customer's account.

The result of this hierarchy example is shown in the diagram below.



Note: The role hierarchy only effects ***permissions***. A role does not inherit its parent role's ***subjects***.

Note: In our RBAC mechanism a role can inherit from 0 or 1 other roles. We do not support multiple inheritance.

2.3. Data properties

Below are the data properties for the entities described in 2.1.

Note: Names can be assumed to be no longer than 64 characters long. Names can be in any language and have non-Latin characters. You should set the appropriate character set and collation at the database level.

2.3.1. Subjects

A subject has these properties:

- An *email address*. This uniquely identifies the subject. Assume addresses do not exceed 256 characters.
- A *first name*.
- A *last name*.
- An optional *phone number*.

2.3.2. Resources

- A resource must have a unique *name*. Resource names are typically nouns (ex: "Loan", "Page", "Game").

2.3.3. Operations

- An operation must have a unique *name*. Operation names are typically verbs (ex: "Approve", "Create", "Read").

2.3.4. Permissions

- A permission must have a unique *name*. The name is typically a combination of the operation and the resource. (ex: "Approve a loan").

2.3.5. Roles

- A role must have a unique *name*. Role names typically describe job or system functions. (ex: "Employee", "System Administrator").

2.3.6. Assignments of Roles to Subjects

When a role is assigned to a subject, the assignment can **optionally** have a *start date* (and time) and/or an *end date* (and time).

- If a role assignment to a subject has a start date, that means the role is assigned to the subject only if the start date has arrived already. That is, the date and time right now are after the start date and time.

- Likewise, if an assignment has an end date, that means the role is enabled only until the end date has passed.

- An assignment where both *start date* and *end date* have not been defined is always enabled (never ignored).

For example: Student A is assigned the "Final Project Submitter" role with the start date being January 25, 2021 at 20:00 and the end date being March 17, 2021 at 23:59. Given that the current date is February 24, 2021 at 22:00 the assignment is enabled.

2.4 Queries

As part of your submission, you should implement the three queries specified below.

All queries should be sorted by subject name (ascending) and output these columns:

- Subject Id.
- Subject Name.
- Permission Id.
- Permission Name.

2.4.1 Subjects and direct permissions

Write a query that returns all the subjects in the system along with all their **direct** permissions. "Direct" here means that the permissions are assigned to their roles directly and not inherited from parent roles.

In this query it does not matter if the role is active or not (i.e., you can ignore the start date and end date properties of role assignments).

2.4.2 Subjects and effective permissions

Augment the previous query so that **effective** permissions are shown. I.e. permissions from a subject should come from the subject's roles, their parent roles and their parent roles' parent roles (up to 3 levels of inheritance max).

You can ignore start and end dates of role assignments in this query too.

2.4.3 Subjects and effective, enabled permissions.

This query is the essence of the RBAC mechanism.

Write a query that returns all the subjects in the system along with all their **effective** and **enabled** permissions. Roles which have their assignments to subjects disabled (because of start and/or end dates) should not be included in the output (See *2.3.6 Assignments of Roles to Subjects* for details of when a role assignment is enabled or disabled).

3. Submissions

3.1 Pairing

You can submit the project by yourself or with one other student. If you need help in finding someone to pair with, please email me.

3.2 Date and Location

The last date to submit your work is **Wednesday, March 17, 2021 at 23:59**.

Submission must be through the dedicated task set up in Moodle. Please do not email submissions as they cannot be graded if you do.

3.3 Contents

Your submission should include the following files. Please upload them individually to the Moodle task rather than zip them together.

3.3.1 `rbac-schema.sql`

A SQL file that, when run, creates the new database, its tables and related SQL objects (keys, constraints etc.)

3.3.2 `rbac-data.sql`

A SQL file that, when run, populates the database with some sample data. The queries that you need to implement should also go in this file.

Note: I recommend that you create sample data that demonstrates the correctness of your database structure and your 3 queries. For example, I recommend that you include roles with, and without parents and "grandparents."

3.3.3 `rbac.mwb`

This file is **optional**. You can include a MySQL Workbench Model (`.mwb`) file showing all tables in the database along with their relationships with each other.

3.3.4 `rbac.pdf`

This file is **optional**. A short PDF document detailing your design decisions or any other information relevant to your submission.

4. Grading

The grade for this project accounts for 70% of the total course grade. The other 30% coming from your home exercises. A minimum grade of 60 in the project is required for passing the course.

Submissions will be graded on the following principles:

4.1 Completeness and Correctness

Submissions are expected to fulfill all business requirements detailed above correctly.

4.2 Relational design principles

Submissions are expected to follow relational design principles taught in the course. For example, referential integrity, avoiding the 3 anomalies etc.

4.3 Readability

Keep your code organized and easy to read. Using consistent naming and casing (upper/lower case) is recommended (but, as promised, **not** required).

5. Questions are most welcome!

If you have any questions or concerns over the project, its requirements, or submissions, please don't hesitate to email me at uri.goldstein@post.idc.ac.il. It might take me a short while to get back to you, but I will be more than happy to help!

Good luck!

Uri