**Business Standards**

1. There can be many type of members some of them can be CEO, COO, Product Manager, Project Manager, Team Manager, IT Admin, Senior Employee, Junior Employee, [et cetera](https://www.indeed.com/career-advice/starting-new-job/business-roles).
2. An Organization can have multiple Branches which can be divided based on the geographic location. Within a branch there can be multiple departments.
3. An Organization can have many number of Departments like HR, Production, R&D, [et cetera](https://www.business.com/articles/8-branches-of-business-management/).
4. A Department can have multiple Teams.
5. There can be different type of Teams like functional, self-managed, [et cetera](https://hygger.io/blog/5-types-of-teams-in-organizations/).

**Database Standards**

1. An Organization can have single, multiple or no­­­­# Branches
2. A Branch can have single, multiple or no# Departments
3. A Department can have single, multiple or no# Teams
4. A Team can have single, multiple or no# Managers.
5. A Team can have single, multiple or no\* Members.
6. A Team can have single, multiple or no\* Projects.
7. The Members within a Team can have same or different domains.
8. The Members may or may not have more hierarchy under them.
9. A Project can be assigned to single, multiple or no\* Teams.
10. A Project can have single, multiple or no\* Managers.
11. A Project can have single, multiple or no Tasks.
12. A Task can have single, multiple or no Sub-Tasks, and so on and so forth.
13. A Project/Task may or may not have deadlines, can have single, multiple or no milestones, may or may not have Priority, can have single, multiple or no Releases, can have single, multiple or no Budget, can have Heading, can have Description, can have single, multiple or no Releases,
15. Each member in an organization will have different roles and will have different access privileges.
16. Admin have the highest privileges. <<The privileges here means the privileges to make changes in the software>>
17. .
18. In some department there can be no projects like in HR department there might be no projects but there can be single, multiple or no Tasks.
19. .

# no Branches/Departments/Teams/Managers can be possible when software is used for personal use by a single user.

\* no Members/Projects is possible when a template is created but values are not assigned

**Other Standards**

1. The software has privileges assigned to several components. Based on the privileges the user’s software will have different features.
2. The software will have privileges for underlying components:
   1. Project:
      1. Creation
      2. Deletion (deleting projects will not erase the timestamp)
      3. Visibility
      4. Managers
      5. Members
      6. Deadlines
      7. Milestones
      8. Priority
      9. Budget
      10. Releases
      11. Heading
      12. Description
      13. Goals
      14. Files
      15. Flags
      16. Meetings
   2. Assigning Tasks inside a Project
   3. Tasks:
      1. Creation
      2. Deletion (deleting tasks will not erase the timestamp)
      3. Visibility
      4. Managers
      5. Members
      6. Deadlines
      7. Milestones
      8. Priority
      9. Budget
      10. Releases
      11. Heading
      12. Description
      13. Goals
      14. Files
      15. Flags
      16. Meetings
   4. Tickets:
      1. Creation
      2. Deletion
      3. Picking
   5. Access to Board
   6. Access to Team’s/Group’s Calendar:
      1. Schedule Event
      2. Update Event
      3. Delete Event
      4. Add Members to Event
      5. Assign Priority to Event
      6. Add Flags
      7. Heading
      8. Description
   7. Teams:
      1. Creation
      2. Members
      3. Team Name
      4. Team Goals
      5. Assign Roles
      6. Change Roles
   8. Chat Group (Unofficial)
      1. Members
      2. Managers
      3. Who can send Messages
      4. Visibility of Messages
   9. Team Chat Group (Official):
      1. Members
      2. Managers
      3. Who can send Messages
      4. Visibility of Messages
   10. Start Discussion (inside chats):
       1. Topic Name
       2. Members
       3. Managers
       4. Who can send Messages
       5. Visibility of Messages
   11. Flags:
       1. Creation
       2. Updating
       3. Deletion
       4. Who can use the flags
   12. Change access privileges

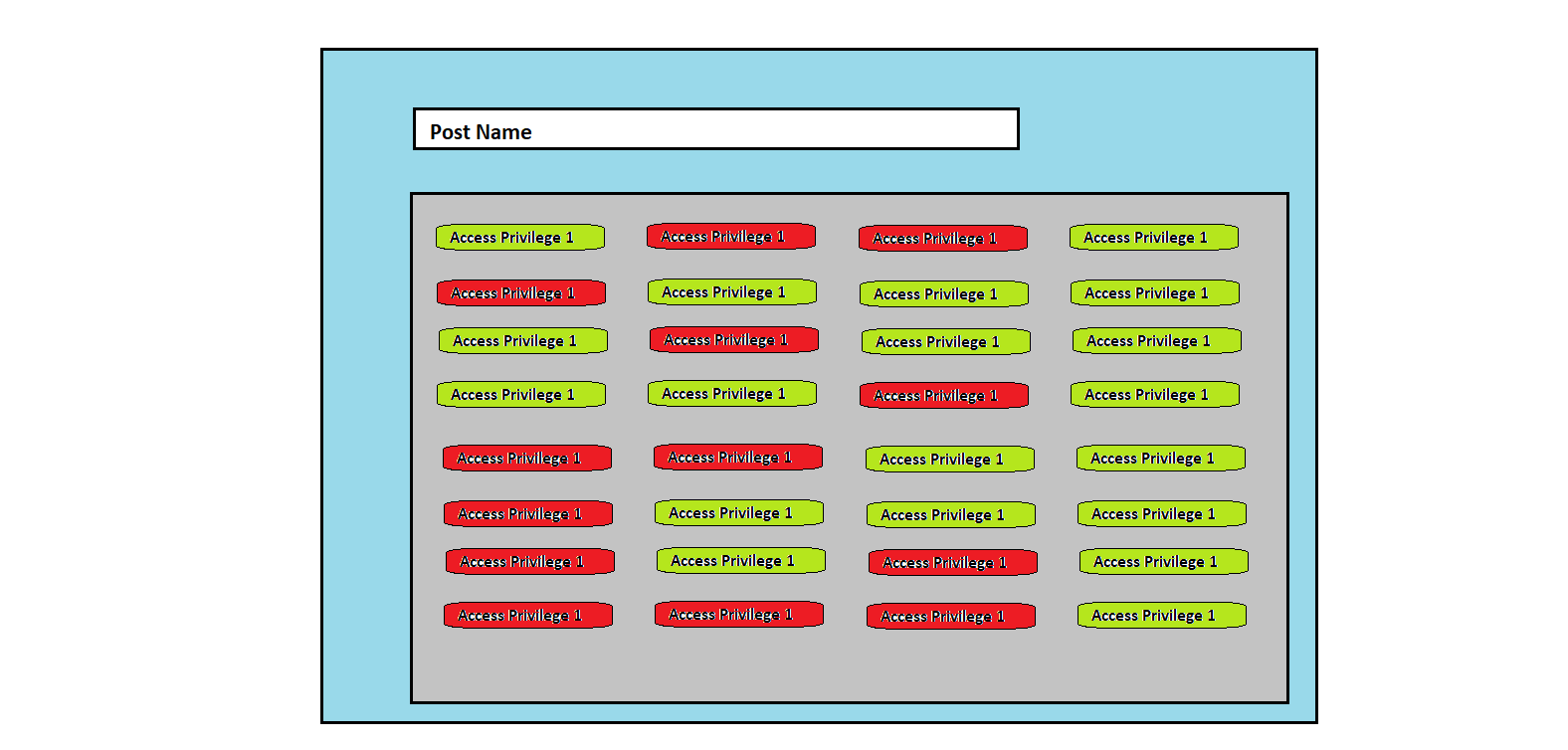
<More components will have privileges including workflow, analysis… It will be added later>

1. .

06/12/2020 2040

By default the application will have only one user in its database. This user will have highest access rights.

(Although the application can be used by a single person, we will address the application’s use only in case of an organization)

When an organization decides to use the application, the Admin will first download the application and log into the application using the root user and password provided. The admin will have the highest access privileges, admin will now register other users in the application’s database. The admin will have to first create some rules for the organization. These rules includes creating Posts and assigning access privileges to those posts. For implementing this feature the application must have an easy interface giving option to add name of post and select the available access rights.

The green one are the one that are associated with post and red ones are not. These access rights are not real life access rights but the software level access rights like access to project/team creation. These post and their values will be stored in the database and can be changed in future.

For the time being there won’t be default post but after sufficient development of software we will provide some of them.

The admin can now register new users in the database and put up their information which will include some mandatory filed and some optional ones. Mandatory filed will include essential components like username, user-id, user-password (some random default one), post, special privileges, et cetera. Optional ones can be address, DOB, email-id, et cetera. Again the mandatory and optional fields are not related to real world but based on software level, the mandatory ones are those which will make sure the proper functioning of application.

For registering the new users admin will have an interface for entering certain values. Admin can select the post of the person. Also admin can create special privileges for the particular individual. These special privileges will have more priority over those provided via Post. Note that these special privileges does not mean that we give more rights but rather it can be used to restrict the person too as these special privileges have higher priority so in this list admin can remove some access rights which are provided in the post.

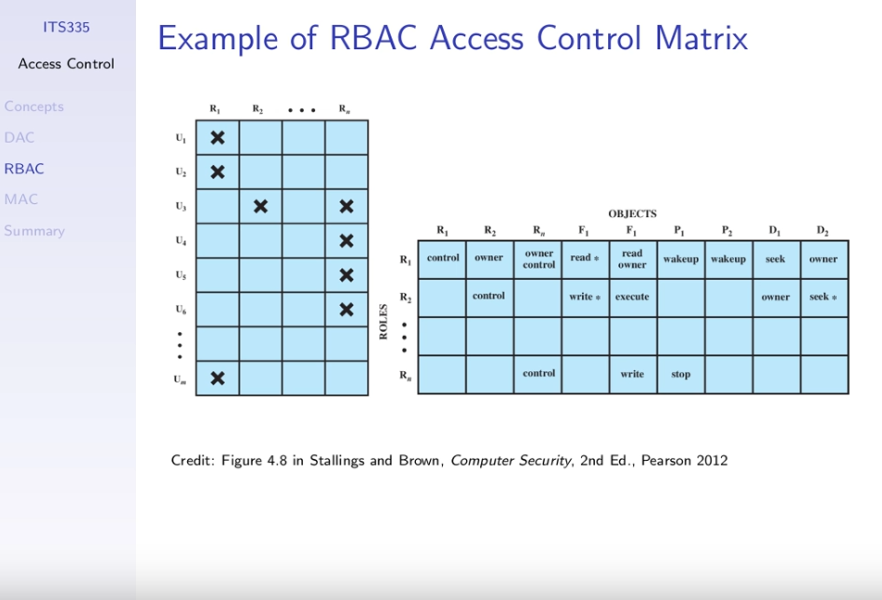
Within the database the users’ data will be stored in several different tables. Most preferably it will be separated on basis of personal and professional information. Personal information table will contain data like address, DOB, DOJ, et cetera. The professional table will contain data like user-id, email-id, post, special privileges, et cetera.

To remove redundancy a table will be created which will contain data related to post and its associated access rights.

06/11/2020 2344

It is possible that a user can have multiple posts or a scenario where the post of person changes is very common. A user can have multiple roles in real life and to achieve this from the software perspective it must be possible to dynamically update these posts, add new post to existing one, assign a temporary post, remove a post, and other similar actions.

Some way must be found to implement this in the database, the option to assign multiple post/roles, dynamically change these roles/posts.

[](https://www.youtube.com/watch?v=NwHx467ZT9I)

As shown in above figure a table can be used to maintain the list of users. The table consists of the some set of roles as columns and the users’ entry as row. For every user the value of the role is set to true or false based on the business requirements.

Another table must be used to that maps the roles to the access rights. It will contain the access rights as columns and the row entry for the role/post in the table will indicate whether that role/post has that right.

Here scalability can be an issue. If in some future scenario we want to add new roles/post or delete some existing ones then the users data must be updated. Also if we want to CUD access rights then the AccessRights table must be updated.

For example, if future if we add new access right to the software then a new column must be added in the AccessRights table. Now the admin has to change the value to this new right for every roles if some default is not set. Also if admin creates new role/post then the values for all the access rights must be specified which might be an overhead and might also lead to redundancy.

In the Roles table for a user the value of the particular role must be specified whether it is true or false. This appears to be an overhead because we have to specify the role’s value even if the user does not have it. And in some very big organization where there can be large number of roles then for creating or updating a user’s entry the value of each and every role must be specified which has good chances of human errors. <A counter argument can be that we can provide an easy GUI for doing this task. Rather than setting the true/false value for every role the admin can select the particular role from the list of roles and only the selected ones will have their values set to true and other will be set to false>

A better scenario will be to have only the list of assigned roles. This is not possible in a sql database, however in a nosql database we can just maintain a list of assigned roles, or rather we can directly store the available access right. This will also remove condition of conflicting rights as mentioned below. Although update in the situation can be troublesome task as it require changing the list which can be easily done in sql database as it requires to only set a flag to true/false. Also if we store the list of provided access rights for each user than it can produce considerable performance issues when the database size grows as the precious memory is used to store the list which could have been done using some flags. Even if we store the roles and not the individual rights then too the size of the data entry can create issues especially when the size of data becomes very big.

https://www.geeksforgeeks.org/sql-vs-nosql-which-one-is-better-to-use/

NOTE: An important point to note here is the priority of roles assigned. Two different roles can provide conflicting access rights and to resolve it one must have higher precedence. Here if we try to add Special Rights feature then it might also create conflicts with the existing access rights.

As we have put up topic of conflicting rights it might be a good time to address situation of hierarchical roles. It might be possible in an organization that a person with higher post can act like a person with lower post to checkout some work. On the other hand this condition might not be available in other organization, a higher post cannot write down the line because of some business logic like abstraction/maintaining security like in mandatory access control a personal with higher access privileges cannot write on something with lower privileges because this might lead to leaking of data which should not be available to lower level.

Hard-coding roles in application’s backend.

NoSQL can be used to store the logs as these are not very commonly used. They are useful when some analysis is to be done or in our application’s perspective when some anomaly occurs; one can get the data from these logs to trace back the root cause. As these nosql database store data in json like format so all the relevant data will be stored in a single entry and will also be easy to work with. However the performance might reduce but that can be dealt with in this situation as use of these logs data is quite less.

07/12/2020 1216

https://www.youtube.com/watch?v=Tcvsefz5DmA

https://www.youtube.com/watch?v=Nw1ymxcLIDI

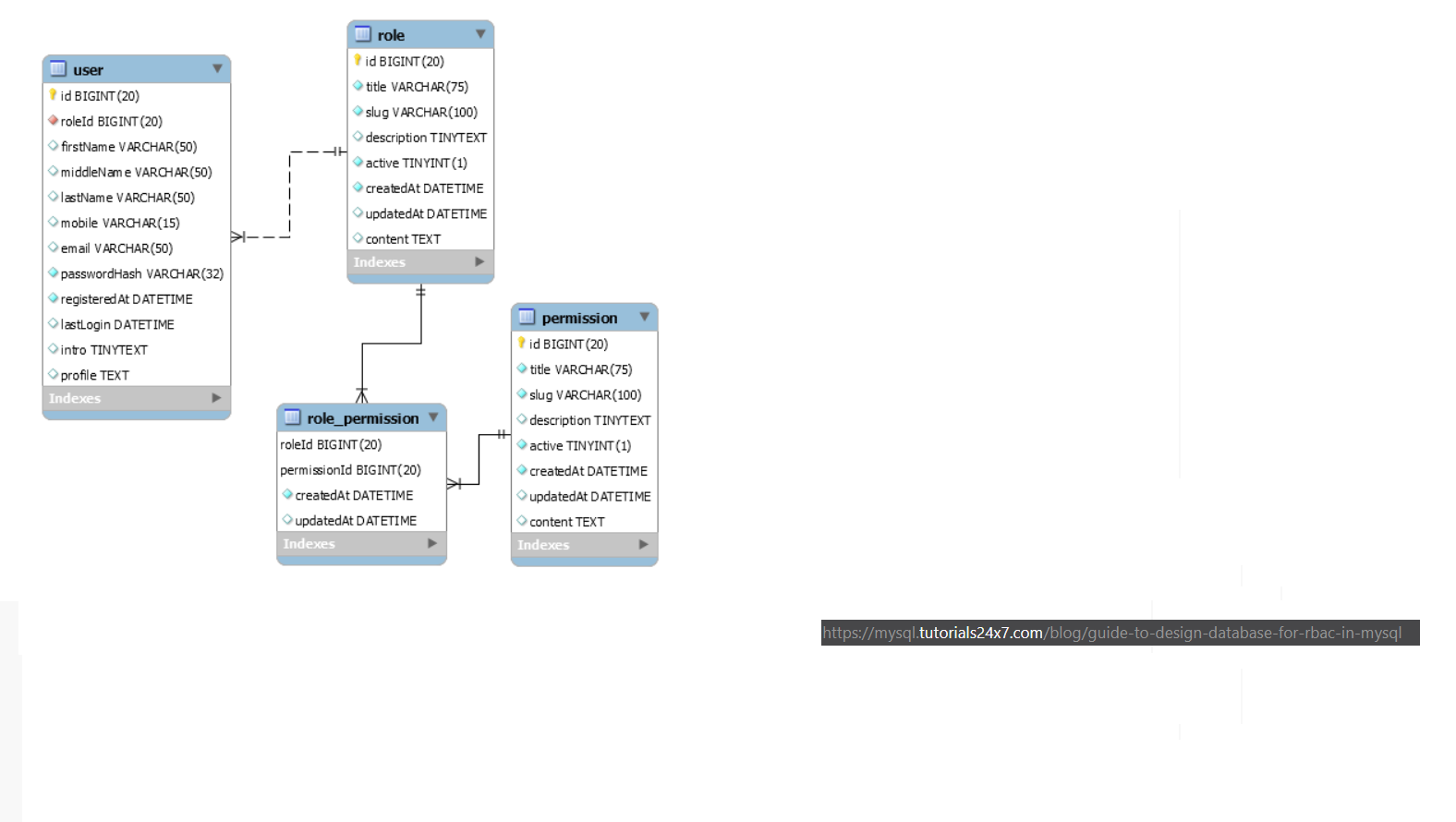
<https://www.youtube.com/watch?v=Lng6Xz1gBGs>

https://www.imperva.com/learn/data-security/role-based-access-control-rbac/

IMPORTANT: For storing passwords we can use hash algorithms.

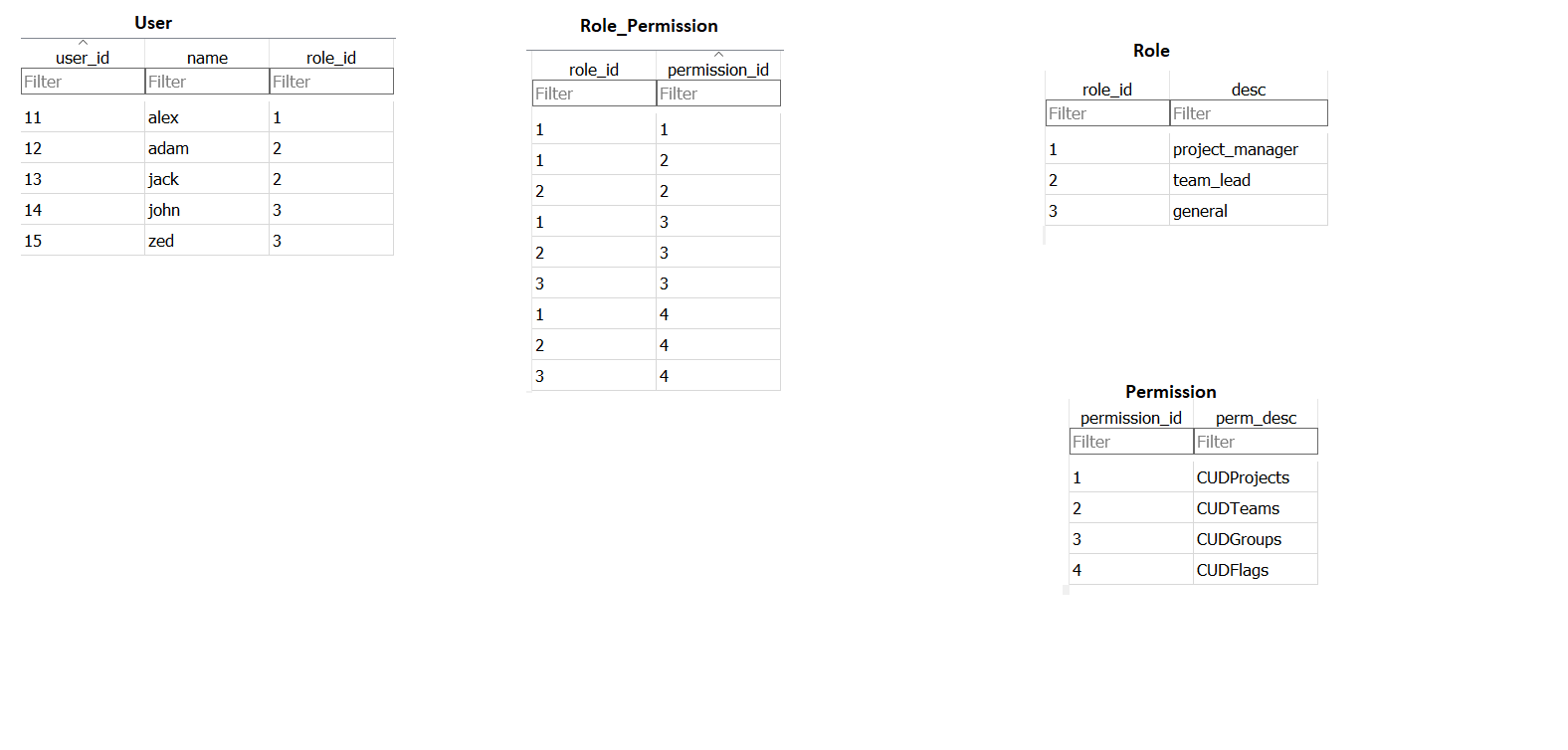
07/12/2020 2200

NEW DATABASE STRUCTURE. IGNORE THE ONE ABOVE.

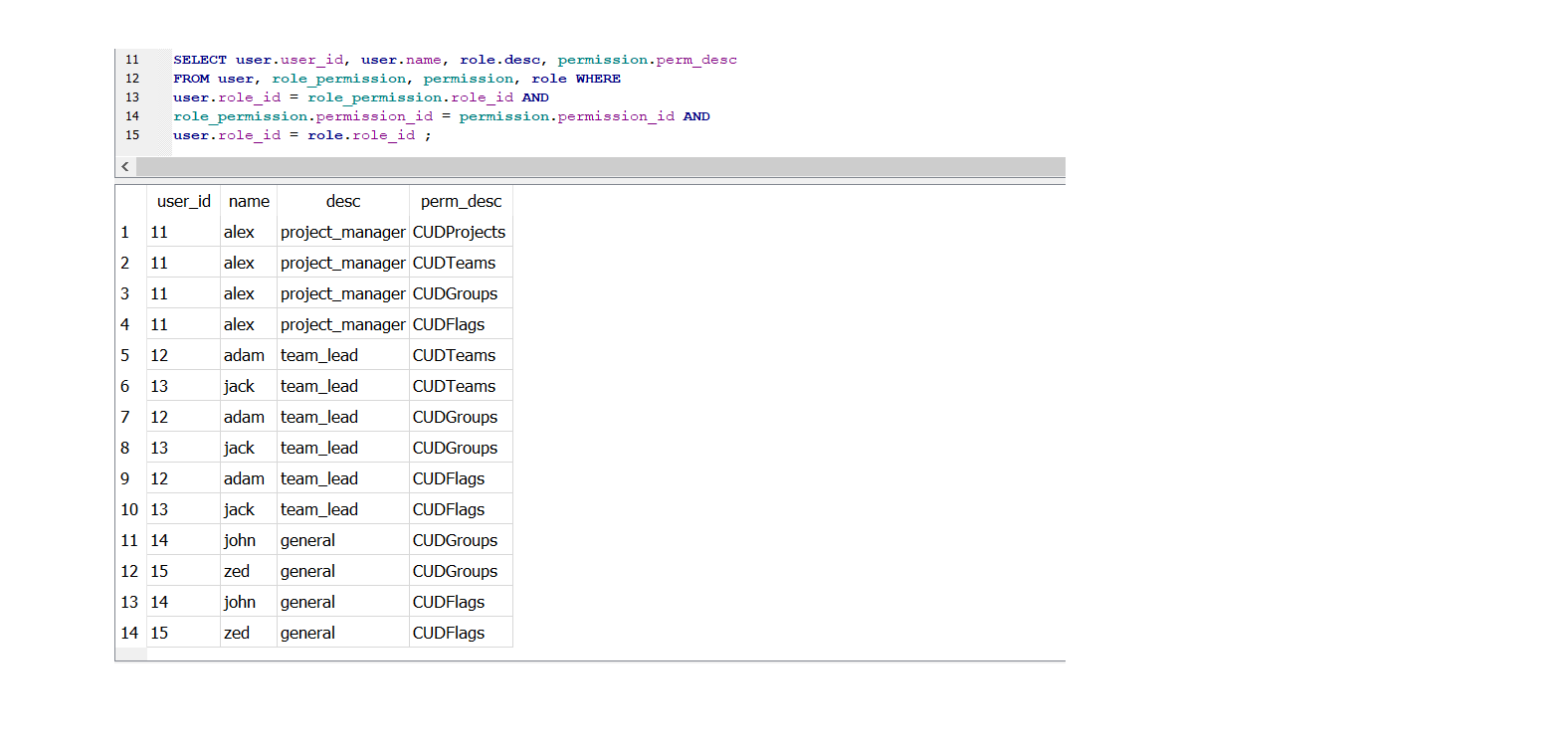
[](https://mysql.tutorials24x7.com/blog/guide-to-design-database-for-rbac-in-mysql)

As depicted in the above diagram the database structure will use multiple tables with similar names like User, Role, Permission, Role\_Permission. The user table will store some essential fields like role\_id, user\_id (unlike above other details can be placed in some other table which can be linked using user\_id). The Role table will contain role\_id, role\_name. The Permission table will contain attributes permission\_id, permission\_name. The Role\_Permission table will be a bridge table mapping the roles with the particular rights.

(Other attributes might be added later if we need them)



Sample data that will be stored in the database.



Querying on the data will give a list of available permission rights. These permission rights can be then stored in the backend of the application and used to provide appropriate features of the software.

Using this schema Admin can create and update roles and assign rights to those roles. Also it will be easy to create or delete access rights which was a problem in the schema decided prior (access matrix one). The major problem though is that the querying must be done correctly otherwise it can raise security issues. Also the database will give a list of rights so the backend will have to enumerate over it and store it locally in some dictionary or array.

IMPORTANT: To deal with issue of conflicting access rights we can add another field “priority” in the Role table which will can be used to deal with the issue. Using this we can implement the Special Privileges feature by assigning that one highest priority.

08/12/2020 1746

Steps:

Start main.py file. This file starts the main file of the view that opens up the login window.

On entering values and clicking on Login the data is send to a module in controller. This module now connects to the model to authenticate. In this authentication module it fetches the data from database and validates it.

If validation is not successful it sends a response to controller that the credentials are wrong. The controller module then instructs the view to show appropriate message.

If successfully validated this module itself again request the database for access rights of the particular user. Based on the received data it then gives the data to controller

Reference:

<https://www.youtube.com/watch?v=C4NP8Eon3cA>

<https://mysql.tutorials24x7.com/blog/guide-to-design-database-for-rbac-in-mysql>

<https://www.youtube.com/watch?v=Tcvsefz5DmA>

<https://www.giacomodebidda.com/mvc-pattern-in-python-introduction-and-basicmodel/>

<https://realpython.com/the-model-view-controller-mvc-paradigm-summarized-with-legos/>

<https://www.vitoshacademy.com/hashing-passwords-in-python/>

https://sqreen.github.io/DevelopersSecurityBestPractices/safe-password-storage/python