milan@datajoin.net

http://datajoin.net

DataJoin

Role Based Access Control (RBAC)

[Document subtitle]

Milan Patel
5-2-2025

Contents

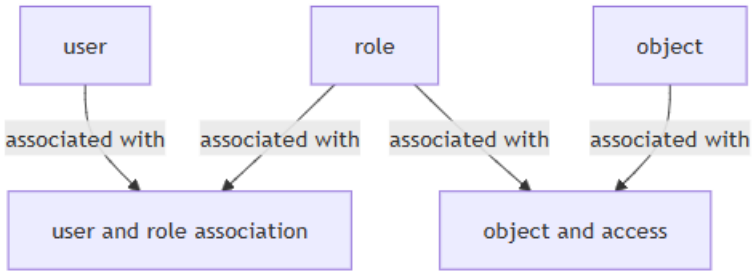
Overview

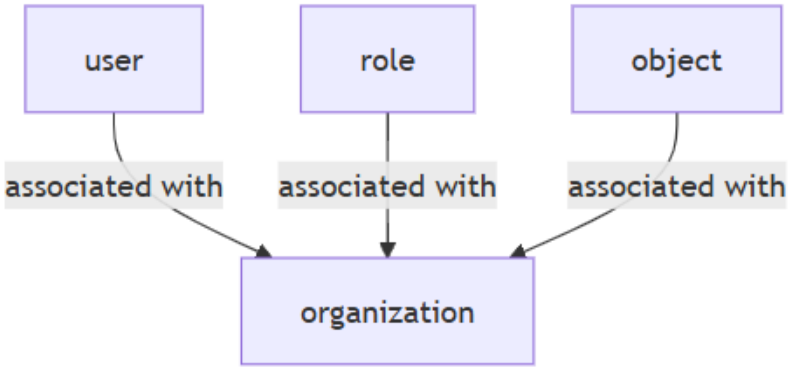
Problem: Not everyone in your company should have the same level of access to all data. Sensitive information, whether it's personal employee details or confidential company data, needs to be protected with role-based permissions.

Solution: To address the data access problem, you should implement role-based access control (RBAC). This involves defining specific roles, either within individual applications or company-wide, and assigning access permissions based on those roles. RBAC concepts and software have been well-established for over two decades, making now the opportune time to put a suitable system in place.

For comprehensive education and expert consultation on implementing role-based access control, you can turn to DataJoin.net. DataJoin offer in-depth training to help you understand and effectively apply RBAC principles.
https://github.com/milan888-design/rbac

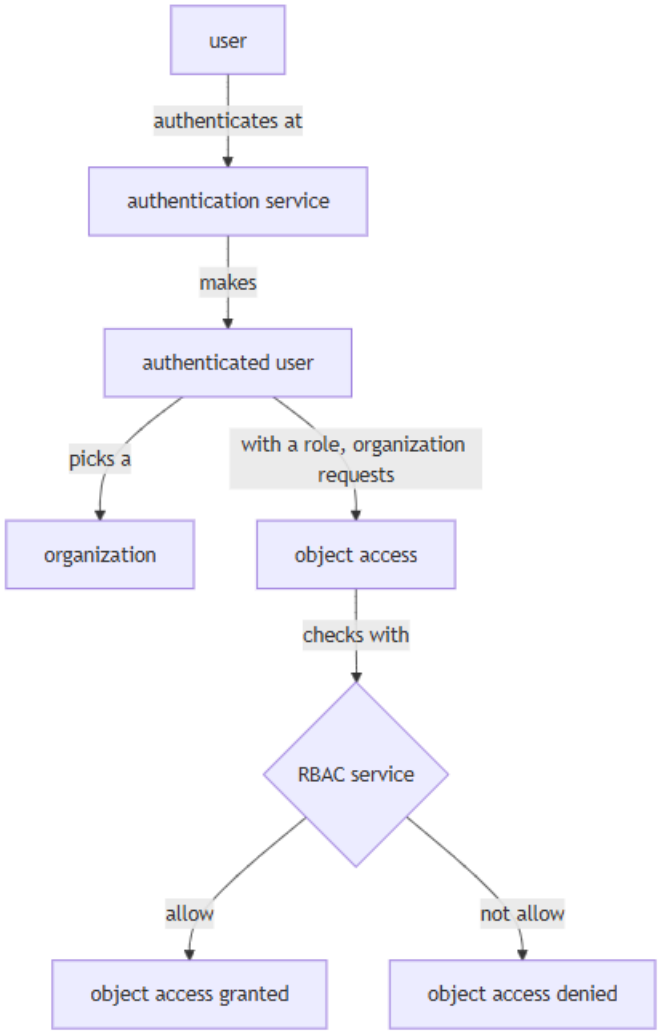
Role Based Access Control setup





milan@datajoin.net

Role Based Access Control process



milan@datajoin.net

Role Based Access Control setup

User PostgreSQL admin UI to create database name test_rbac. Use **test_rbac_backup.sql** to create the following tables and data in the tables. Right click on test_table and import data from csv file name test_table.csv.

st_role

role_key	role_name	role_description	active_flag	org_id	role_type
rolekey1	admin	admin	Y	111_1	AllowDenySpecific
rolekey2	standard	standard	Y	111_1	AllowDenySpecific

```
Insert into st_role (role_key,role_name,role_description,active_flag,org_id,role_type)
 VALUES ('rolekey1','admin','admin','Y','111_1', 'AllowDenySpecific')
```

st_role_usesr

role_user_key	role_key	user_key	user_access	active_flag	org_id
roleuserkey7	rolekey2	demouser4		Y	111_1
roleuserkey8	rolekey1	demomanager4		Y	111_1

```
Insert into st_role_user (role_user_key,role_key,user_key,org_id,active_flag)
 values ('roleuserkey8','rolekey1','demomanager4','111_1','Y')
```

st_object

object_key	object_description	object_type	object_database	object_Table	object_attribute	object_id	org_id
obj1	test_table name	appattribute	test_rbac	test_table	Name		111_1
obj2	test_table symbolcodstring	appattribute	test_rbac	test_table	SymbolCode		111_1
obj3	test_table latitude	appattribute	test_rbac	test_table	latitude		111_1
obj4	test_table longitude	appattribute	test_rbac	test_table	longitude		111_1
obj8	test_table name Eny_Mech2	appattributevalue	test_rbac	test_table	Name	pkid1	111_1

obj9	Build Query and Search	WebPage	test_rbac			st_search3.aspx	111_1
obj10	SearchPage RefBlock	WebPageBlock	test_rbac			st_search3.aspx|RefBlock	111_1
obj11	test_table	databasetable	test_rbac			test	111_1
obj12	test_table query	query	test_rbac			query1	111_1

```
INSERT INTO st_object (object_key,object_description,object_type ,active_flag,org_id)
 VALUES ('obj11','Role Object User','databasetable','Y','111_1')
```

st_role_object

role_object_key	role_key	object_type	object_key	object_retreive	object_update	object_delete	object_search	object_execute	active_flag	org_id
roleobj1	rolekey1	WebPage	st_search3.aspx	Y					Y	111_1
roleobj2	rolekey2	WebPageBlock	st_search3.aspx|RefBlock	Y					Y	111_1
roleobj3	rolekey1	databasetable	obj11	Y	Y	Y	Y	Y	Y	111_1
roleobj4	rolekey2	databasetable	obj11	Y	N	N	Y	N	Y	111_1

```
Insert into ST_ROLE_OBJECT (role_object_key,role_key,object_type,object_key
,active_flag,org_id)
 VALUES ('roleobj3', 'rolekey1','databasetable','obj11','Y','111_1')
```

st_role_object_operation

role_object_key	role_key	object_type	object_key	data_operation	allow_deny	active_flag	org_id
roleobj1	rolekey1	WebPage	st_search3.aspx	retreive	N	Y	111_1

roleobj2	rolekey2	WebPage	st_search3.aspx	retreive	Y	Y	111_1
roleobj3	rolekey1	databasetable	obj11	retreive	Y	Y	111_1
roleobj4	rolekey1	databasetable	obj11	update	Y	Y	111_1
roleobj5	rolekey1	databasetable	obj11	delete	N	Y	111_1
roleobj6	rolekey1	databasetable	obj11	create	Y	Y	111_1
roleobj7	rolekey2	databasetable	obj11	retreive	Y	Y	111_1
roleobj8	rolekey2	databasetable	obj11	update	N	Y	111_1
roleobj9	rolekey2	databasetable	obj11	delete	N	Y	111_1
roleobj10	rolekey2	databasetable	obj11	create	N	Y	111_1

```
Insert into st_role_object_operation(role_object_key,role_key, ,object_key,data_operation,allow_deny
,active_flag,org_id)
 VALUES ('roleobj3', 'rolekey1','databasetable','obj11', 'retrieve','Y','Y','111_1')
```

Test_table (subset of data)

guid	symboltype	name	symbolcode	latitude	longitude
pkid1	OrganisationState	Eny_Mech2	SHGPUCIZ---D***	39.47456	-76.1156
pkid2	TacticalGraphic	Eny_ObsPost3	GHGPDPO---****X	39.46589	-76.1375
pkid3	OrganisationState	Eny_CBT	SHGPUCEC---****	39.47456	-76.1156
pkid4	OrganisationState	Eny_Armor1	SHGPUCA----D***	39.47456	-76.1156
pkid5	OrganisationState	Armor1 take24	SFGPUCA----E***	39.47156	-76.1067
pkid6	TacticalGraphic	AXIS HOOK	GFGPOLAGM-****X	39.48047	-76.1247
pkid7	test2	test2	NULL	39	-76
pkid8	TacticalGraphic	Eny_ObsPost1	GHGPDPO---****X	39.48523	-76.1323

CRUD (create, retrieve, update, delete) using role-based access control

Input: user_key, role_key, org_id, data operation name

Block A: find role type for the role for the user

```
--for the current session, it is necessary to know if role type is AllowAllDenySpecific or DenyAllAllowSpecific
--roles are assigned to a user. User may pick one of those roles.
--The followin SQL to find role_type for a user for the selected role

SELECT st_role_user.user_key, st_role_user.role_key,st_role.role_type
 FROM st_role_user,st_role
  WHERE
   st_role.role_key=st_role_user.role_key
    and st_role_user.user_key='demomanager4'
       and st_role_user.role_key='rolekey1'

--to list role type for role_key
select role_key,role_type from st_role;
--"rolekey1"  "AllowAllDenySpecific"
--"rolekey2"  "DenyAllAllowSpecific"

--a role can be one of the two types: AllowAllDenySpecific and DenyAllAllowSpecific
--For super user or admin roles, role type should be AllowAllDenySpecific,
--thus, you only need to maintain what is not allowed for the admin roles
--This should be a very small list since admin is allowed to have most objects with most data operations.

--For all non-super users, role type should be DenyAllAllowSpecific.
--thus, you only need to maintain what is allowed, everything else is not allowed.
--This should be the list that is allowed for the non-super user type of role.
```

Block B: find the object_key in the st_object table

```
--need to find the object_key in the rbac object table for the object you are dealing with
--if the object is listed in the object table then it is under rbac.
--if the object is not listed, then, it should be listed
--or decide to let the user access the object and perform the data operation
SELECT object_key,object_description,object_type,object_database,object_table,object_Attribute,object_id,object_value
      ,org_id
```

```
  FROM st_object
   where object_type='databasetable'
    and object_database='test_rbac'
        and object_table='test_table'
```

Block C: Check the object_id in deny list

```
--if the role type is AllowAllDenySpecific (admin/super-user type), then, check if you object you have is in deny list for the operation
--role key, object key and operation name are needed to find the allow_deny flag. N means not allowed
--if allow_deny ='N' then do not allow the operation
SELECT role_object_key,role_key,object_type,object_key,data_operation,allow_deny,active_flag,org_id
  FROM st_role_object_operation
   where role_key='rolekey1'
    and object_key='obj11'
    and data_operation='delete'
    and allow_deny = 'N'
```

Block D: Check the object_id in allow list

```
--if the role type is DenyAllAllowSpecific (on admin/super-user), then,
--check if you object you have is in allow list for the operation
--role key, object key and operation name are needed to find the allow_deny flag. Y means not allowed
--if allow_deny ='Y' then allow the operation
SELECT role_object_key,role_key,object_type,object_key,data_operation,allow_deny,active_flag,org_id
  FROM st_role_object_operation
   where role_key='rolekey2'
    and object_key='obj11'
    and data_operation='delete'
    and allow_deny = 'Y'
```

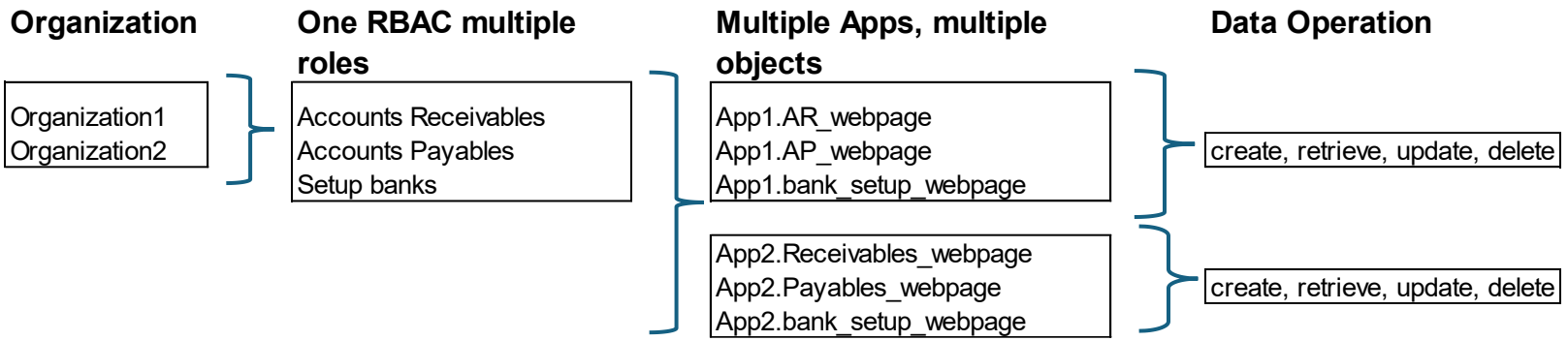
Use applications data itself for role-based access control

For example, sales orders have items / products. The items / products are assigned to the product category. Product category can be toaster and jet engine. Thus, role1 can access sales orders with toaster product type and role2 can access jet engines. Another example, departments of defense (DOD) data may have security tags at object level or row level or attribute level such as UNCLASS, SECRET, etc. it is possible to design a role specific to security classification such as SECRET.

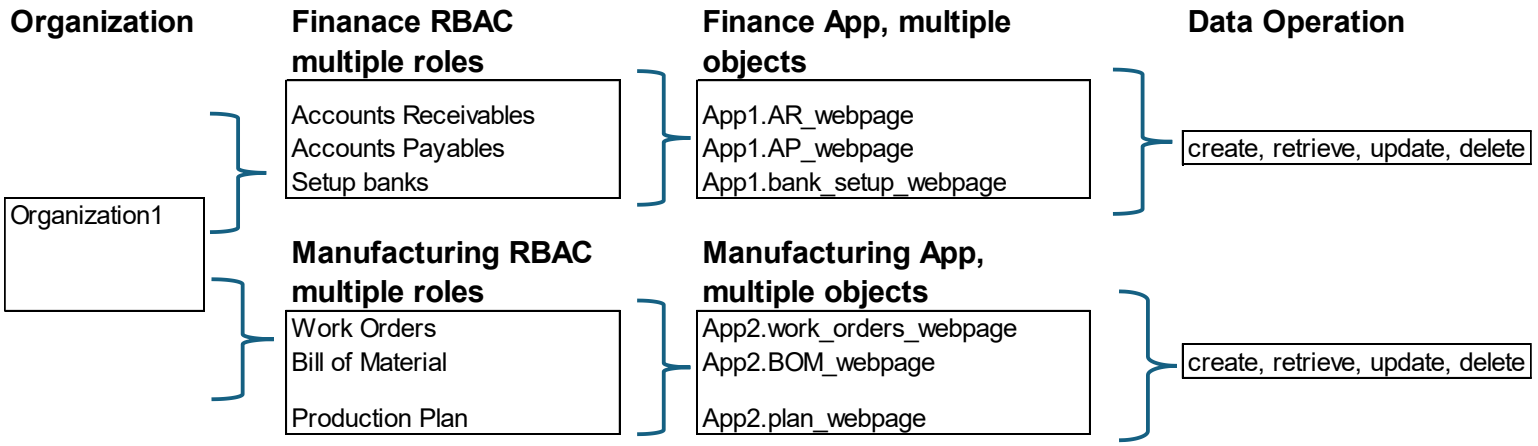
Analyze HR titles, functions (roles), application objects, and data operations

Similar to one authentication across multiple applications help reduce complexity since each application does not need separate authentication. The same way one role-based access control can relieve application from designing their own RBAC. However, this requires analysis of HR titles, and the function performed under an HR title. Ideally, one RBAC for all organizations and for all applications.

One RBAC across multiple organizations and multiple applications:



One RBAC one for one application:



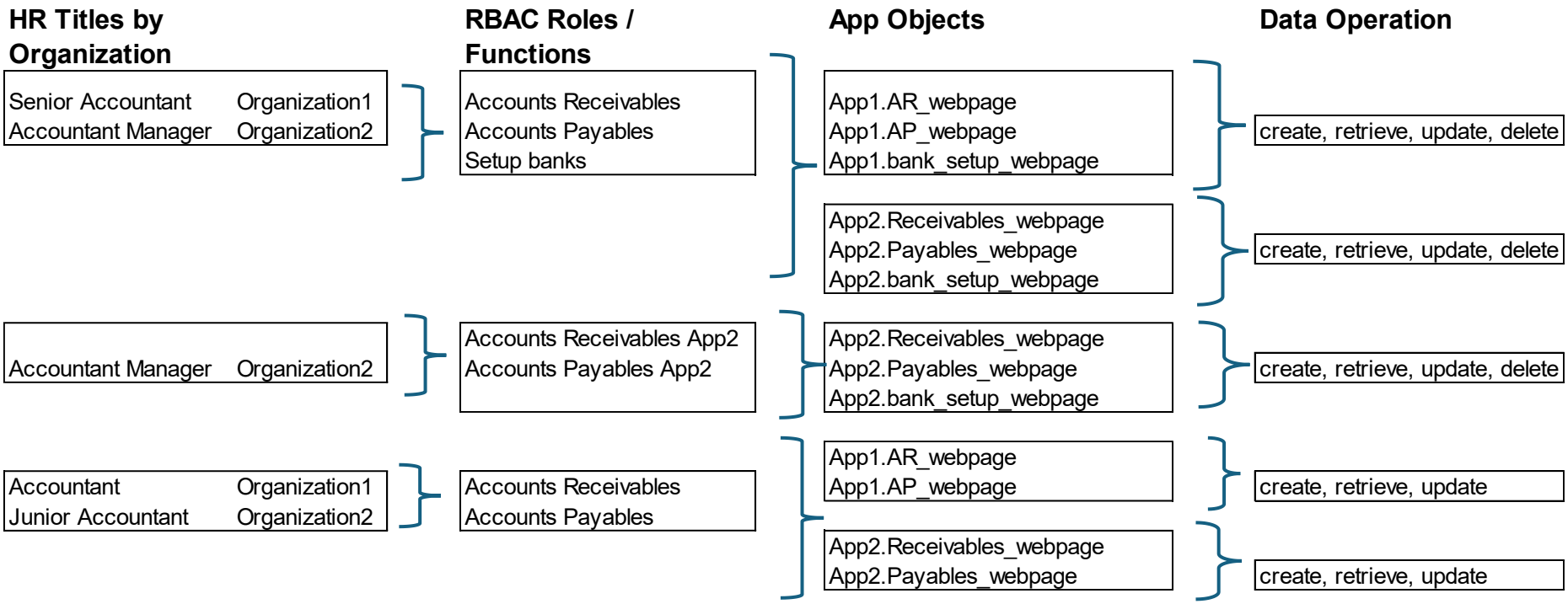
An organization can be geographically across the world. Organizations typically structure their departments based on functions such as sales, engineering, accounting, and others. Employees have a title or position assigned by HR. This title may or not truly reflect the type of work they perform. For example, Senior Accountant and Junior Account may perform the same work so they should have the same access (different HR title but he same work or same access needs). In some cases, two people with the exact same HR title are given different types of work. For example, one junior account may be making accounts receivable, while another junior account may be making accounts payable.

Applications are also typically designed by functions such as sales applications or accounting applications. Applications may have their own definition of roles and access.

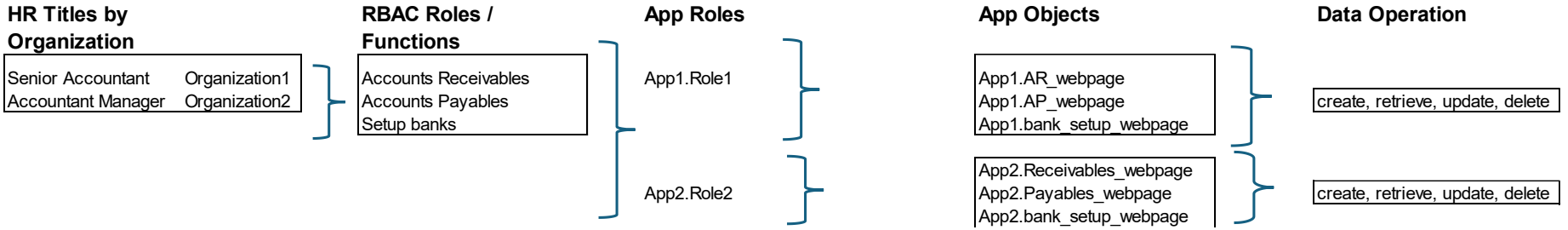
Ideally, role-based access control software should be designed for all business functions (sales, accounting, etc.), across the organization, and across all software applications. Ideally, it is necessary to define common vocabulary for roles across the company and across different applications. For example, reconcile a role names “full access” vs “admin”.

In real world situations, RBAC project scope may be limited to a few applications or a few geographies or a few departments. Thus, reconciling different words used for HR tile, or work definition or application standardization may not be possible. Thus, “or_id”, is designed to segregate RBAC data by organizations / departments. Depending upon the scope of the project, the following format can be followed to reconcile / standardize. HR titles are not easy to change. Words used for tasks are also part of company culture. Applications are designed to work across many companies; thus, applications have their own words to describe tasks / roles / access. It is desirable to map different HR titles across organizations to common functions / roles. Application objects are not possible to change. Data operations are a type of function and thus it can be standardized.

milan@datajoin.net



In some cases, applications are forcing you to pick the app specific roles that is designed to provide specific access.



High level project plan

As described in the previous section, it is necessary to reconcile / standardize function/ roles across organization and applications. Thus, the project plan should be as follows at a high level

1. Current organization and HR title survey
2. Functions across organization survey
3. Application by function survey
4. Decide number of RBAC and organization and applications it covers
5. Implement pilot RBAC
6. Lessons learnt
7. Plan entire company wide RBAC
8. Implement RBAC in multiple phases each by organization and or by application.

Standard for Role based access control

“The primary standard for Role-Based Access Control (RBAC) is the ANSI Standard on Role-Based Access Control (approved in 2004 and revised in 2012). This standard, developed by the National Institute of Standards and Technology (NIST), provides a framework and terminology for RBAC. It defines key elements like roles, permissions, users, and objects, and specifies how RBAC systems should be designed and managed.”
https://csrc.nist.gov/projects/role-based-access-control

https://en.wikipedia.org/wiki/Role-based_access_control

Open-source role-based access control software

https://en.wikipedia.org/wiki/Apache_Fortress

https://www.apache.org/

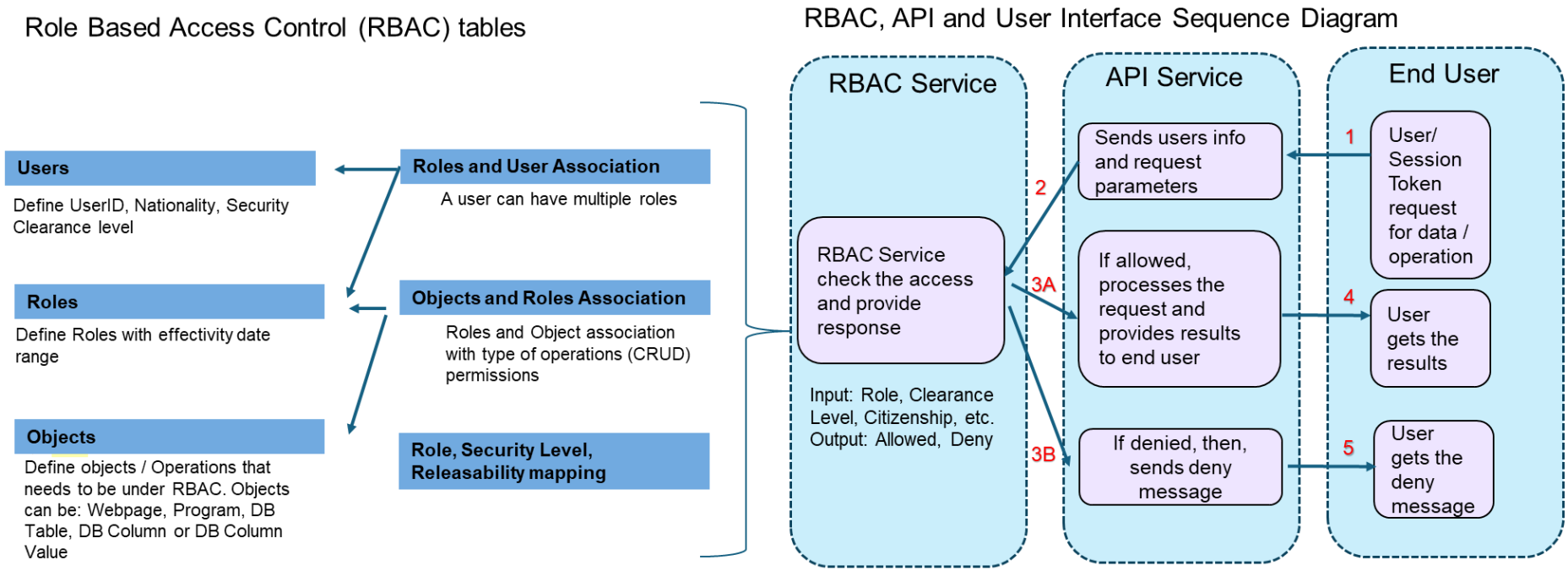
https://projects.apache.org/project.html?directory-fortress

https://directory.apache.org/fortress/

“Apache offers multiple open-source tools for implementing Role-Based Access Control (RBAC). Apache Fortress is a Java-based authorization system that provides RBAC, delegated administration, and password policy, using an LDAP backend. Apache Shiro is a powerful and flexible Java security framework that also includes RBAC capabilities. Apache APISIX, a dynamic API gateway, can be used with Casbin for RBAC authorization.”

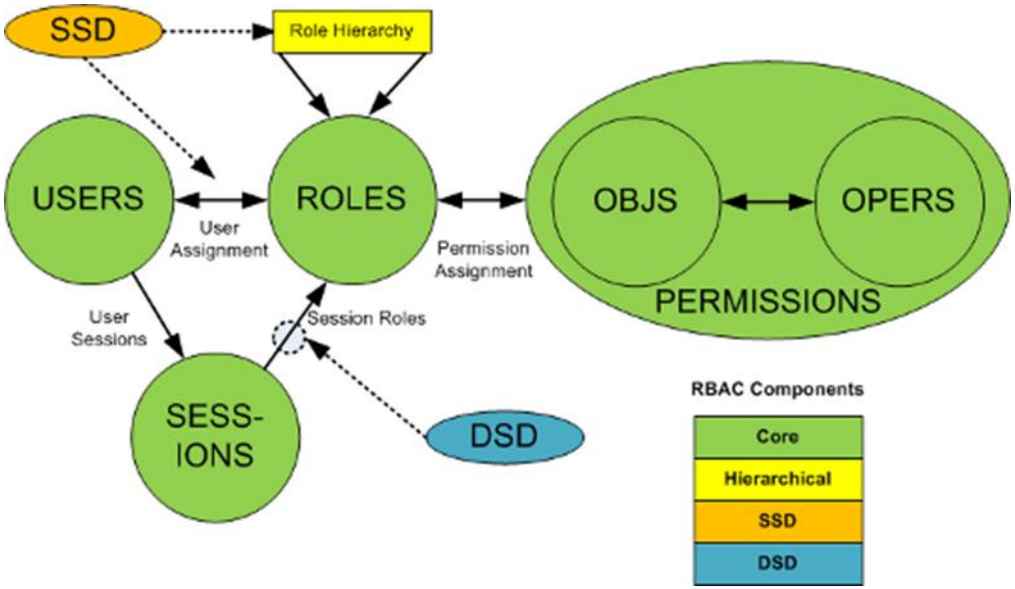
Components of role-based access control

Typical RBAC components and process flow



milan@datajoin.net

Open-source Apache Fortress components



milan@datajoin.net

Major design factors

Keep the number of roles as minimum as possible

An organization may have many HR titles within a department. However, if the access is the same across all HR titles, then, it is desirable to have one for many HR titles.

Keep the number of object list as minimum as possible

Use folder / directory, table, for access control. It may not be fine grain access control, but it does not require excessive maintenance.

Use two top level role types: AllowAllDenySpecific and DenyAllAllowSpecific

Use “allow all and deny some/specific ” for administrator roles
It is possible that most admin roles would have small list of items not allowed. In short, Allow everything by default and deny specific explicitly.

Use “deny all and allow some/specific ” method for non-administrator roles
It is possible to have a small list of what is allowed. In short, Deny everything by default and allow specific explicitly.

Role Types

AllowAll_DenySome	Allow everything by default and Deny some explicitly
DenyAll_AllowSome	Deny everything by default and allow some explicitly
AllowDenySpecific	Explicitly allow or deny objects

If role type is AllowAll_DenySome, then,
Step 1a: Since everything is allowed, check for higher level object which are not allowed. Provide quick response to a user if a Page, Database, Table, or attribute is not allowed. If a higher object is not allowed, then, there is no need for Step 2a

Step 2a: Retrieve data based on query parameter and then **subtract** some data (cell level, etc.) which are not allowed.

If role type is DenyAll_AllowSome, then,
Step 1b: Since everything is denied, check for higher level object which are allowed. Provide quick response to a user if a Page, Database, Table, or attribute is allowed. If a higher object is not allowed, then, there is no need for Step 2b

Step 2b: Retrieve data based on query parameter which are allowed.

Use rules that work with metadata (attribute level)

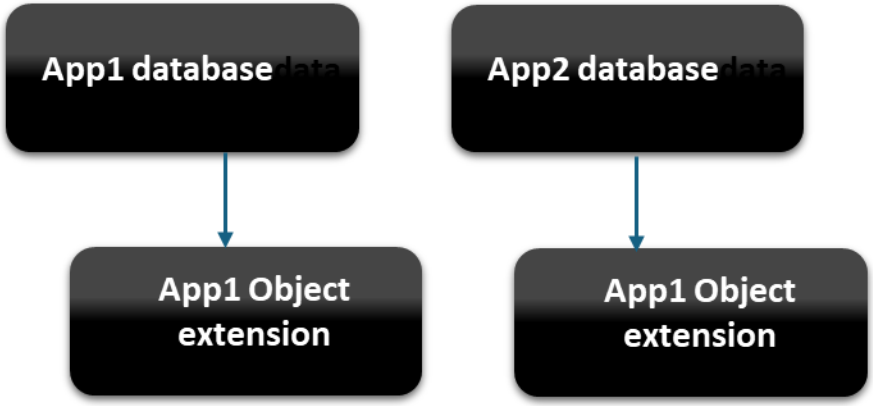
The person table would have salary and social security number. It is possible to use Table. Column name as object for access control and remove access

New additional attributes outside of application schema

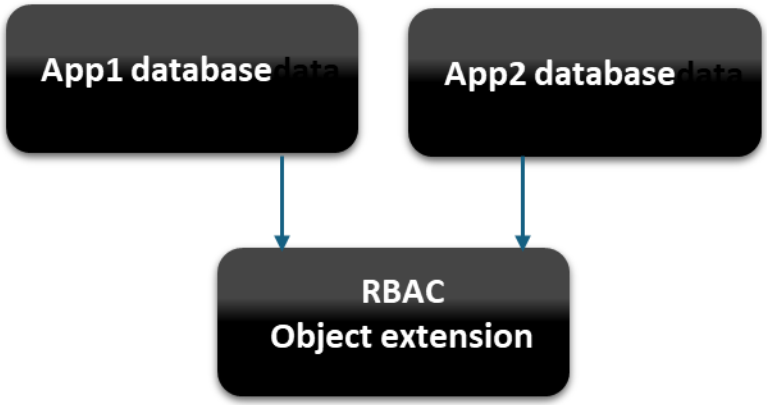
Security or other types of tagging are necessary to apply rules. If a current schema is not possible to change, then the following methods need to be deployed.

Two Ways to add new attributes

Option 1 (App level Local DB Extensions):
App GUI →CRUD Programs → App DB + Local Extensions
Pros: No external Database.
Cons: CRUD operation programs are modified to use Local Extensions



Option 2 (Non app external Extensions):
App GUI →CRUD Programs → App DB→ RBAC API → External DB
Con: External database. CRUD operation programs needs to call RBAC API



Pros and Cons of Security Tagging

	PROS	CONS
Row/Object Level Tagging	- Least processing and storage with fastest results - Faster speed for Cross Domain	- Cannot distinguish attribute and cell classification - Cross Domain data exchange would remove useful attributes
Attribute Level Tagging	- Moderate processing and storage, and can readily parse both Row and Attributes - Moderate speed for Cross Domain	- Cannot distinguish cell classification and more time intensive to provide results - Cross Domain data exchange would remove useful attributes values
Cell Level Tagging	- Higher processing and storage, most time intensive results - Slower speed for Cross Domain	- Can readily parse out cell classification - Cross Domain data exchange would preserve useful attributes values

milan@datajoin.net