Database & Storage Security

Professor Dr. Mohammad Abu Yousuf
yousuf@juniv.edu

#Lecture – 3.1

Polyinstantiation Problem

Definition and need for polyinstantiation

- Polyinstantiation is a database technique that allows the database to contain multiple instances of the same data but with different classifications

- Polyinstantiation occurs because of mandatory policy

- In relational DBMS it is possible to have different tuples with the same key but with different classifications

Definition and need for polyinstantiation

- Polyinstantiation can affect relations, tuples and data elements

- Polyinstantiation arises because subjects with different classes are allowed to operate on the same relations

- Polyinstantiated relations are relations with different access classes

- Polyinstantiated tuples (also called entity polyinstantiation) are tuples with the same primary key but with different access classes associated to the primary keys

Definition and need for polyinstantiation

- Polyinstantiated elements (also called attribute polyinstantiation) are elements of an attribute which have different access classes but are associated with the same primary key and key class

- Polyinstantiation occurs as one of:
  - Visible polyinstantiation
  - Invisible polyinstantiation

ML relations – polyinstantiation

- *invisible polyinstantiation*
  - When a low user inserts data in a field which already contains data at higher or incomparable level
- *visible polyinstantiation*
  - When a high user inserts data in a field which already contains data at a lower level

ML relations – invisible polyinstantiation

Suppose a low user asks to insert a tuple with the same primary key as an existing tuple at a higher level; the DBMS has three choices:

1) Notify the user that a tuple with the same primary key exists at higher level and reject the insertion
2) Replace the existing tuple at higher level with the new tuple being inserted at low level
3) Insert the new tuple at low level without modifying the existing tuple at the higher level (i.e. polyinstantiate the entity)

Choice1 introduces a signaling channel

Choice 2 allows the low user to overwrite data not visible to him and thus compromising integrity

Choice 3 is a reasonable choice; as consequence it introduces a polyinstantiated entity

ML relations – invisible polyinstantiation Example

Name	C_{Name}	Dept#	$C_{Dept\#}$	Salary	C_{salary}	TC
Bob	Low	Dept1	Low	100K	Low	Low
Ann	High	Dept2	High	200K	High	High
Sam	Low	Dept1	Low	150K	High	High

Assume a low user issue the following insert operation

INSERT INTO Employee
VALUES (Ann, Dept1, 100k)

ML relations – invisible polyinstantiation Example

Name	C_{Name}	Dept#	$C_{Dept\#}$	Salary	$C_{Dept\#}$	TC
Bob	Low	Dept1	Low	100K	Low	Low
Ann	High	Dept2	High	200K	High	High
Sam	Low	Dept1	Low	150K	High	High
Ann	Low	Dept1	Low	100K	Low	Low

The tuples with primary key "Ann" are *polyinstantied*

ML relations – visible polyinstantiation

Suppose a high user asks to insert a tuple with the same primary key as an existing tuple at lower level; the DBMS has three choices:

1) Notify the user that a tuple with the same primary key exists and reject the insertion
2) Replace the existing tuple at lower level with the new tuple being inserted at the high level
3) Insert the new tuple at high level without modifying the existing tuple at the lower level (i.e. polyinstantiate the entity)

Choice 1 does not introduce a signaling channel; however, rejecting the insertion may result in a DoS problem

Choice 2 would result in removing a tuple at lower level and thus introduce a signaling channel

Choice 3 is a reasonable choice; as consequence it introduces a polyinstantiated entity

Example of polyinstantiated tuple

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	20K	S	S
Sam	S	Math	S	10K	S	S

A TS user tries to insert (Sam, CIS, 30K)

Example of polyinstantiated tuple

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	20K	S	S
Sam	TS	CIS	TS	30K	TS	TS
Sam	S	Math	S	10K	S	S

Example of polyinstantiated element

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	**20K**	**S**	S
Sam	TS	CIS	TS	30K	TS	TS

A TS user insert 30K as Salary of Ann

Figure 3

Example of polyinstantiated element

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	20K	S	S
Ann	S	CIS	S	30K	TS	TS
Sam	TS	CIS	TS	30K	TS	TS

After a TS user insert 30K as Salary of Ann

Example of polyinstantiated element

The view of the table for a subject with classification S based on the previous table for a polyinstantiated element

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	20K	S	S
Ann	S	CIS	S	30K	TS	TS
Sam	TS	CIS	TS	30K	TS	TS

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	20K	S	S

Polyinstantiation

- For read operations, subjects have read access to instances of multilevel relations accessing data at their level or below.
- For write (insert or update) operations, the effect depends on the access level of dominated by, incomparable with their level

Polyinstantiation

- Suppose an S-subject (i.e., a subject with classification S) wants to execute the operation

  INSERT INTO EMPLOYEE

  VALUES 'Sam', 'Math', '10K'

The operation is applied to Figure 1 and the result will be Figure 2

In this example, the subject clearance is dominated by the access class of data

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	30K	TS	TS
Sam	**TS**	**CIS**	**TS**	**30K**	**TS**	**TS**

Figure 1

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	30K	TS	TS
Sam	TS	CIS	TS	30K	TS	TS
Sam	S	Math	S	10K	S	S

Figure 2

Polyinstantiation

- Suppose an S-subject (i.e., a subject with classification S) wants to execute the operation

```
UPDATE EMPLOYEE
SET Salary = '20K'
WHERE Name = 'Ann'
```

The operation is applied to Figure 1 and the result will be Figure 3

In this example, the subject clearance is dominated by the access class of data

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	**30K**	**TS**	TS
Sam	TS	CIS	TS	30K	TS	TS

Figure 1

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	30K	TS	TS
Ann	S	CIS	S	20K	S	S
Sam	TS	CIS	TS	30K	TS	TS

Figure 3

Polyinstantiation

- Suppose a TS-subject (i.e., a subject with classification TS) wants to execute the operation

```
UPDATE EMPLOYEE
SET Dept = 'Math'
WHERE Name = 'Ann'
```

The operation is applied to Figure 3 and the result will be Figure 5 given next, where multiple rows are added

In this example, the subject clearance dominates the access class of data

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	30K	TS	TS
Ann	S	CIS	S	20K	S	S
Sam	TS	CIS	TS	30K	TS	TS

Figure 3

Polyinstantiation

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	30K	TS	TS
Ann	S	Math	TS	30K	TS	TS
Ann	S	CIS	S	20K	S	S
Ann	S	Math	TS	20K	S	TS
Sam	TS	CIS	TS	30K	TS	TS

Figure 5

ML relations – polyinstantiation

- The introduction of data classification in relational DBMS introduces polyinstantiation
- Several approaches have been developed to handle this problem
  - Approaches that allows polyinstantion
    - Sandhu&Jajodia, SeaView Model by Denning et al.
      - These approaches define the key of a multilevel relation to be a combination of the original key attributes and their classifications
    - Belief-based model by Smith and Winslett
  - Approaches that prevent polyinstantion
    - Require that all keys be classified at the lowest possible access class
    - Partition the domain of the primary key among the various access classes so that each value has a unique possible classification

Sea View Model

Sea View Model

- SEcure dAta VIEW was developed by Lunt, Denning, et al in 1987 in California
- Model has two layers:
  - MAC (Mandatory Access Control)
  - TCB (Trusted Computing Base)
- MAC enforces the security policy of the Bell-LaPadula and Biba models
- TCB defines:
  - Concept of multilevel relations
  - Supports DAC
  - Formalizes supporting policies

Sea View Model

- Sea View model uses subjects, objects, and access classes
- Access class consists of:
  - Secrecy class
  - Integrity class
- Secrecy class corresponds to the security level of Bell-LaPadula model
- Integrity class corresponds to the integrity level of Biba model

Sea View Model

- Objects of the MAC are the files to which access must be granted.
- Subjects of MAC are processes acting on behalf of users, and are assigned the classification of that user.
- Each user is assigned minimal secrecy (minsecrecy) and integrity (minintegrity) classes

Sea View Model

- The secrecy and integrity classes originally assigned to the user are denoted maxsecrecy and maxintegrity
- Variables of the subject uses (minsecrecy, maxintegrity)
- Variables of the subject uses (maxsecrecy, minintegrity)
- For each subject, the read class must dominate the write class

Sea View Model

- Subject is said to be trusted if the **readclass** strictly dominates the **writeclass**
- Trust is divided into two parts
  - Secrecy trust corresponds to strict inequality of secrecy classes
  - Integrity trust corresponds to strict inequality of integrity classes
- Subjects with secrecy trust could write data at a lower secrecy class than data read

Sea View Model

- Polyinstantiation integrity requires:
  - For every non-key attribute A_i with classification C_i there is a multivalued dependency $A_k, C_k \rightarrow A_i, C_i$
- Polyinstantiation integrity ensures that no two tuples will have the same primary key unless they represent polyinstantiated elements

Jajodia-Sandhu (J-S) Model

Jajodia-Sandhu (J-S) Model

- A write operation in Sea View model could potentially generate z^n new tuples where z is the number of elements in the access class and n is the number of non-key attributes.
- Jajodia and Sandhu point out how these could lead to false tuples getting added.
- This work led to changes in polyinstantiation integrity.
- J-S model introduces entity integrity and null integrity

Jajodia-Sandhu (J-S) Model

- Entity integrity requires:
  - no tuple can have a null value for an attribute that is part of the primary key
  - all key attributes must have the same classification (an important addition that makes all key values entirely visible or entirely invisible)
  - class of key attributes must dominate the class of non-key attributes

Jajodia-Sandhu (J-S) Model

- In multilevel relations, null values may have double meaning (either the value is null or the higher classification for that value makes the display null for low-classification subjects)
- Null integrity requires:
  - null values be classified at the same level of the key attributes of the tuple
  - null value be included by a non-null value independent of the classification of the non-null value

Jajodia-Sandhu (J-S) Model

- J-S model does not require the multivalued dependency of the Sea View model for polyinstantiation integrity.

- J-S model handles the read operations using the Bell-LaPadula model security of No Read-up principle.

- J-S model handles the write operation using the No Write-down principle (namely a user cannot affect instances of a relation with a lower or incomparable classification with his/her own clearance).

Example of J-S model Write

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	20K	TS	TS
Sam	TS	CIS	TS	30K	TS	TS

Figure 7

Assume that an S-subject performs

INSERT INTO EMPLOYEE VALUES 'John, CIS, 20K'

This produces the following S-instance

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	null	S	S
John	S	CIS	S	20K	S	S

Figure 8

Example of J-S model Write

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	20K	TS	TS
Sam	TS	CIS	TS	30K	TS	TS

Figure 9 shows the TS-instance view of Figure 7 after the INSERT operation performed by the S-subject

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	20K	TS	TS
Sam	TS	CIS	TS	30K	TS	TS
John	S	CIS	S	20K	S	S

Figure 9

Example of J-S model Write

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	20K	TS	TS
Sam	TS	CIS	TS	30K	TS	TS
John	S	CIS	S	20K	S	S

Assume that a TS-subject performs

INSERT INTO EMPLOYEE VALUES 'Bob, CIS, 20K'

on table in Figure 8

This produces the following **S-instance**:

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	null	S	S
John	S	CIS	S	20K	S	S

Figure 10

Example of J-S model Write

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	20K	TS	TS
Sam	TS	CIS	TS	30K	TS	TS
John	S	CIS	S	20K	S	S

Figure 11 shows the TS-instance view after the INSERT operation on table in Figure 8 performed by the TS-subject

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Bob	TS	CIS	TS	20K	TS	TS
Ann	S	CIS	S	20K	TS	TS
Sam	TS	CIS	TS	30K	TS	TS
John	S	CIS	S	20K	S	S

Figure 11

Example of J-S model Write

Assume now that an S-subject who is unaware of the existence of Sam as TS wants to perform the following operation on table in Figure 11:

INSERT INTO EMPLOYEE VALUES "Sam, CIS, 20K"

To avoid down-channel signaling, a new tuple must be entered with S-classification with primary key Sam. This is called necessary polyinstantiation

The following is an S-instance of Figure 11 after the above INSERT operation

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Ann	S	CIS	S	null	S	S
Sam	S	CIS	S	20K	S	S
John	S	CIS	S	20K	S	S

Figure 12

Example of J-S model Write

Figure 13 shows the TS-instance view after the INSERT operation on table in Figure 11 performed by the S-subject

User	C_{user}	Dept	C_{dept}	Salary	C_{salary}	TC
Bob	S	Math	S	10K	S	S
Bob	TS	CIS	TS	20K	TS	TS
Ann	S	CIS	S	20K	TS	TS
Sam	TS	CIS	TS	30K	TS	TS
Sam	S	CIS	S	20K	S	S
John	S	CIS	S	20K	S	S

Figure 13

Access Control -- RBAC

Role Based Access Control (RBAC)

- With Role-Based Access Control (RBAC), access decisions are based on the roles that individual users have as a part of an organization.
- Roles are closely related to the concept of user groups in access controls.
- Role brings together a set of users on one side and a set of permissions on the other whereas user groups are typically defined as a set of users.

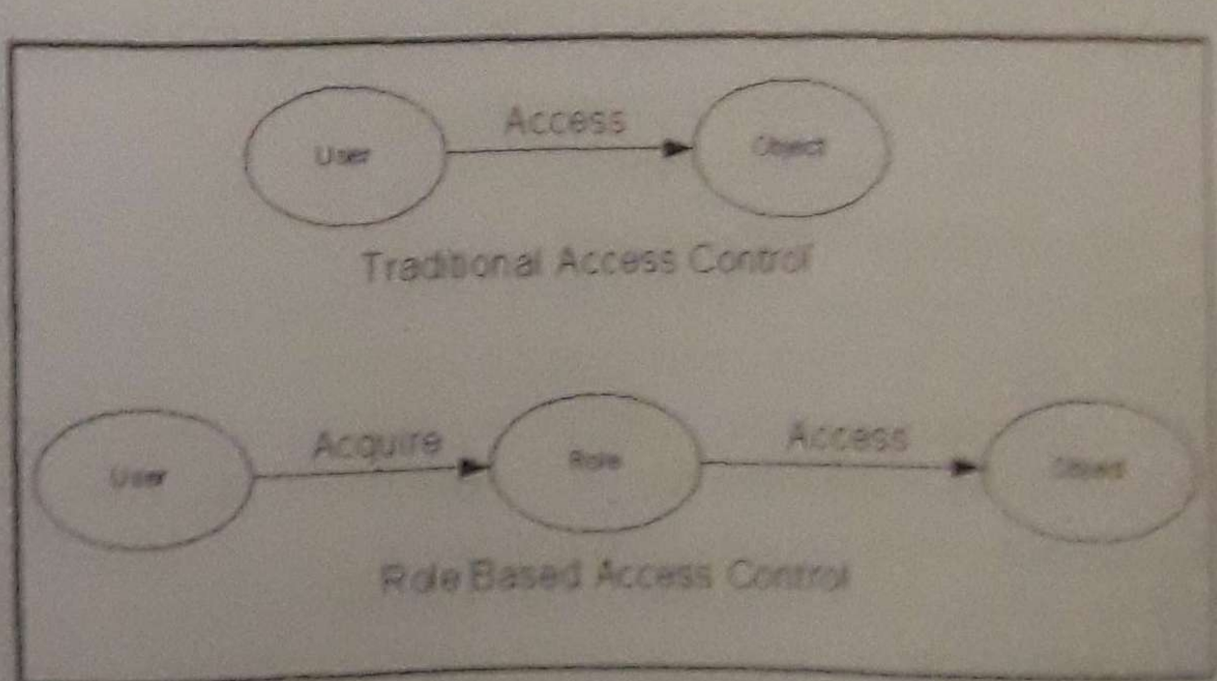
Role Based Access Control (RBAC)

- Many organizations base access control decisions on "the **roles** that individual users take on as part of the organization".
- They prefer to centrally control and maintain access rights that reflect the organization's protection guidelines.
- With RBAC, role-permission relationships can be predefined which makes it simple to assign users to the predefined roles.
- The combination of users and permissions tend to change over time, the permissions associated with a role are more stable.

Role Based Access Control (RBAC)

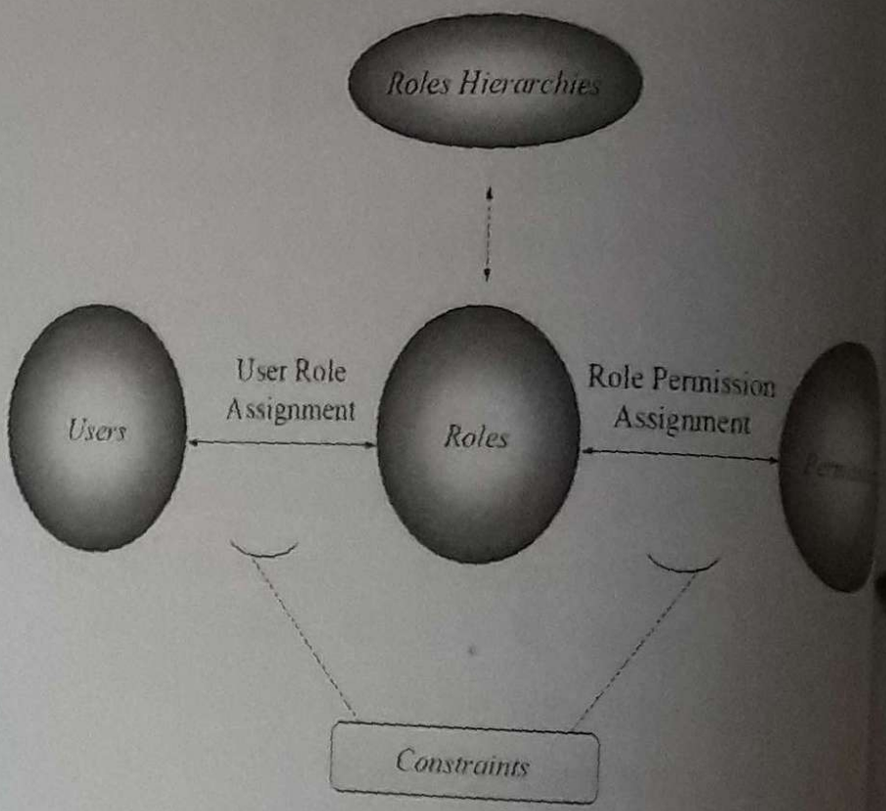
- The concept of role-based access control began with multi-user and multi-application on-line systems pioneered in 1970's.
- Introduces the concept of a **role** and a **permission**
  - A permission is an association between a transformation procedure and an object
    - A permission can be thought as an object-method pair or a class-method pair in an object-oriented environment
    - A permission can be thought as a table-query pair or a view-query pair in a database application
    - Permission to perform an operation on an object is assigned to roles, not to users
- Users are assigned to roles
- Users acquire their permissions based on the roles they are assigned

Role Based Access Control (RBAC)



Role Based Access Control (RBAC)

- Access control in organizations is based on "roles that individual users take on as part of the organization"
- A role "is a collection of permissions"

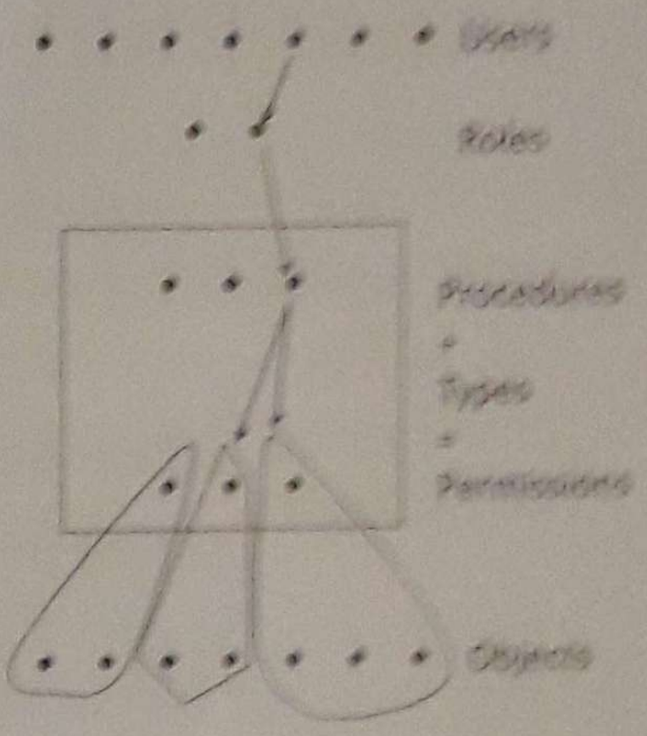


Role Based Access Control (RBAC)

- Access Control policy is embodied in various components of RBAC such as
  - Role-Permission relationships
  - User-Role relationships
  - Role-Role relationships
- These components collectively determine whether a particular user will be allowed to access a particular piece of data in the system.
- RBAC components may be configured directly by the system owner or indirectly by appropriate roles as delegated by the system owner
- The policy enforced in a particular system is the net result of the precise configuration of various RBAC components as directed by the system owner
- The ability to modify policy to meet the changing needs of an organization is an important benefit of RBAC

Role-Based Access Control

- The complexity of administration is reduced through
  - Assigning users to roles
  - Assigning permissions to roles
  - Organising roles into a hierarchy



Role-Based Access Control

RBAC supports three well-known security principles:

- Least Privilege
- Separation of duties
- Data Abstraction

- Least Privilege is supported because RBAC can be configured so only those permissions required for tasks conducted by members of the role are assigned to role.
- Separation of duties is achieved by ensuring that mutually exclusive roles must be invoked to complete a sensitive task.
- Data abstraction is supported by means of abstract permissions such as credit and debit for an account.
- The degree to which data abstraction is supported will be determined by the implementation details

RBAC

- Access depends on role/function, not identity
  - Example: Rahim is **bookkeeper** for Math Dept. He has access to financial records. If he leaves and Karim is hired as the new **bookkeeper**, Karim now has access to those records. The role of "bookkeeper" dictates access, not the identity of the individual.

Challenges in RBAC

- Policy must be clearly defined or RBAC breaks down completely
- Roles must be created that reflect business needs
- Permissions for roles to access objects must be determined
- Membership is each role must be determined

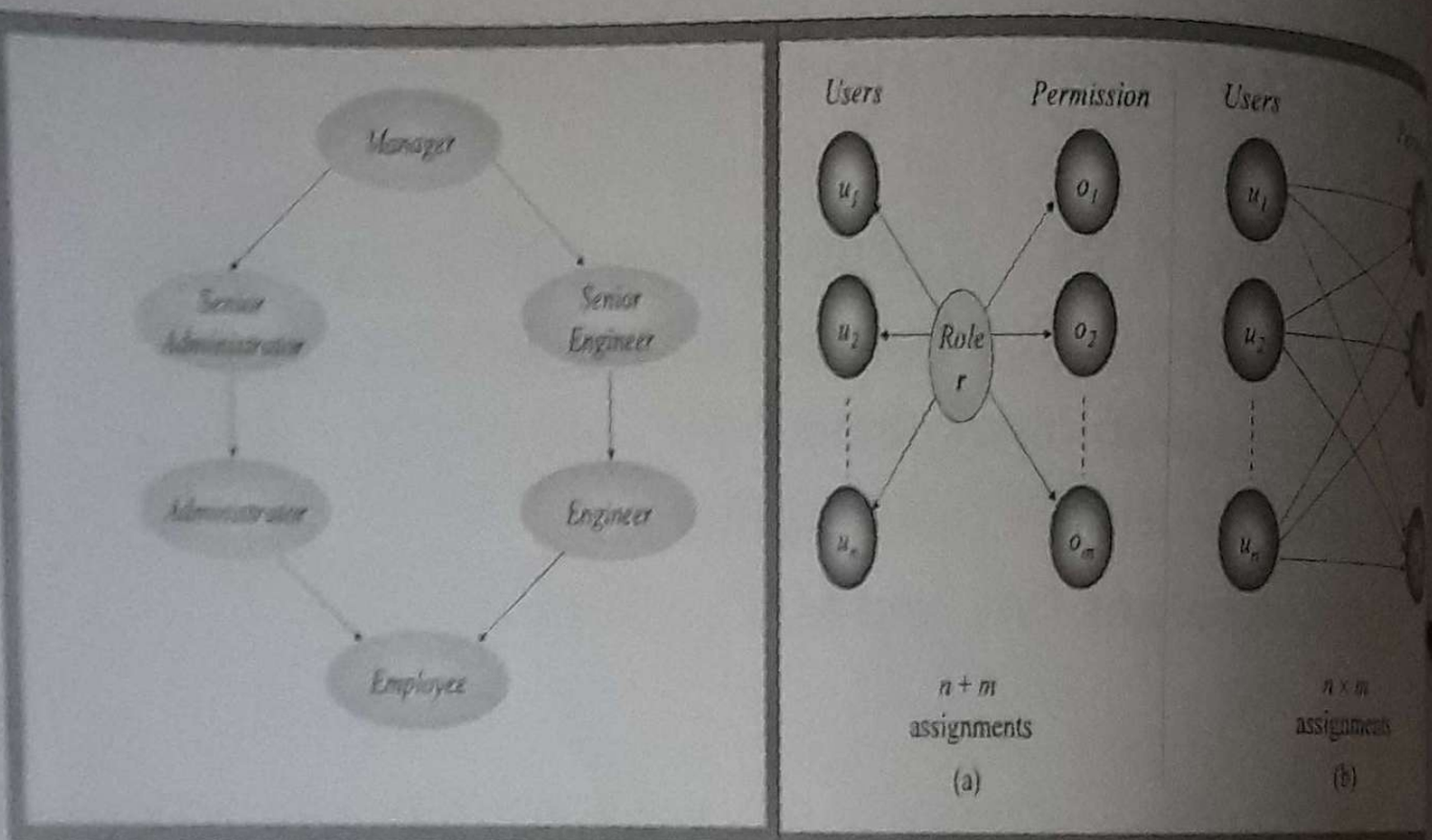
Advantages of RBAC

- Allows Efficient Security Management
  - Administrative roles, Role hierarchy
- Principle of least privilege allows minimizing damage
- Separation of Duties constraints to prevent fraud
- Allows grouping of objects
- Policy-neutral - Provides generality
- Encompasses DAC and MAC policies

Disadvantages

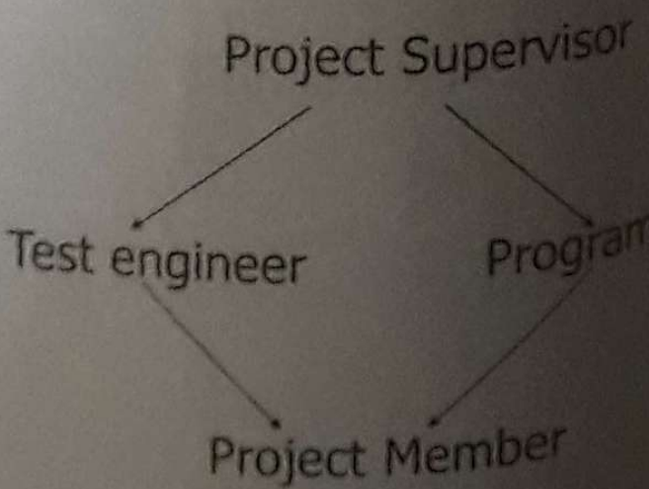
- This is good model for a static, closed, centralized organization where you can form a Role Hierarchy. But, this model is not efficient for distributed or dynamic system.

RBAC

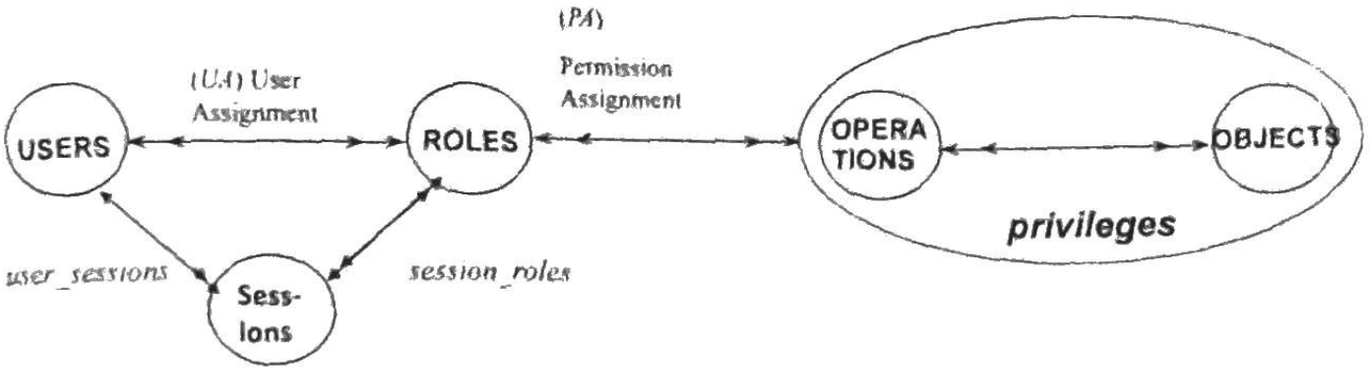


RBAC (cont'd)

- Is RBAC a discretionary or mandatory access control?
  - RBAC is policy neutral; however individual RBAC configurations can support a mandatory policy, while others can support a discretionary policy.
- Role Hierarcies
- Role Administration



RBAC (NIST Standard)



- Many-to-many relationship among individual users and privileges
- Session is a mapping between a user and an activated subset of assigned roles
- User/role relations can be defined independent of role/privilege relations
- Privileges are system/application dependent
- Accommodates traditional but robust group-based access control

An important difference from classical models is that Subject in other models corresponds to a Session in RBAC

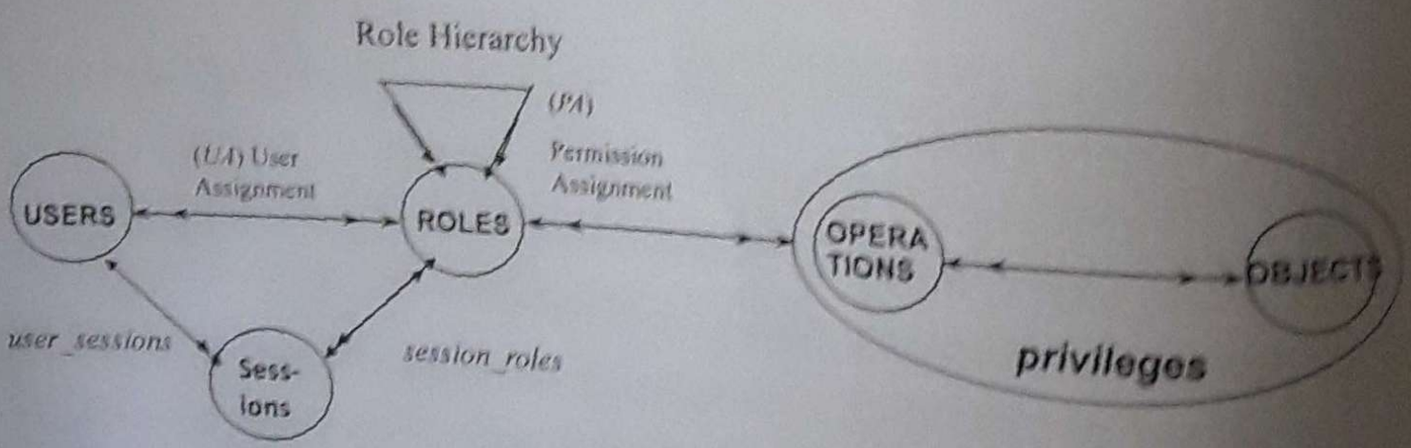
Core RBAC (relations)

- Permissions = $2^{\text{Operations x Objects}}$
- UA $\subseteq$ Users x Roles
- PA $\subseteq$ Permissions x Roles
- *assigned_users*: Roles $\rightarrow 2^{\text{Users}}$
- *assigned_permissions*: Roles $\rightarrow 2^{\text{Permissions}}$
- *Op*(p): set of operations associated with permission p
- *Ob*(p): set of objects associated with permission p
- *user_sessions*: Users $\rightarrow 2^{\text{Sessions}}$
- *session_user*: Sessions $\rightarrow$ Users
- *session_roles*: Sessions $\rightarrow 2^{\text{Roles}}$
  - *session_roles*(s) = {r | (session_user(s), r) $\in$ UA)}
- *avail_session_perms*: Sessions $\rightarrow 2^{\text{Permissions}}$

Hierarchical RBAC

- It adds requirements for supporting role hierarchies. A hierarchy is mathematically a partial order defining a seniority relation between roles, whereby the seniors roles acquire the permission of their juniors, and junior roles acquire the user membership of their seniors. This standard recognizes two types of role hierarchies.
  - General Hierarchical RBAC: In this case, there is support for an arbitrary partial order to serve as role hierarchy, to include the concept of multiple inheritance of permissions and user membership among roles.
  - Limited Hierarchical RBAC: Some systems may impose restrictions on the role hierarchy. Most commonly, hierarchies are limited to simple structures such as trees and inverted trees

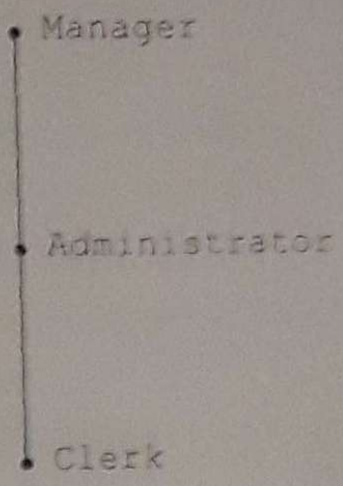
Hierarchical RBAC



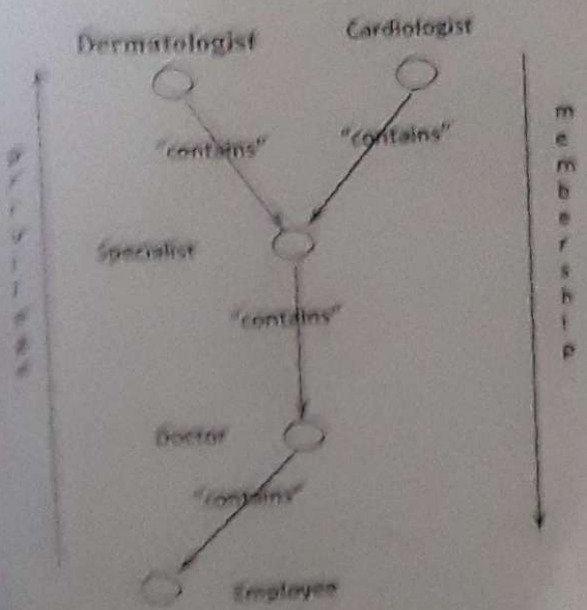
- Role/role relation defining user membership and privilege inheritance
- Reflects organizational structures and functional delineations
- Two types of hierarchies:
  - - Limited hierarchies
  - - General hierarchies

The Role Hierarchy

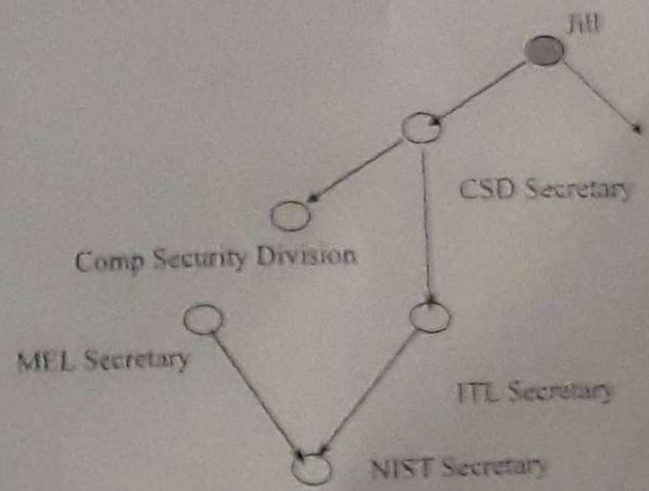
- The set of roles is partially ordered
- Models hierarchical structure of enterprise
- Aggregates permissions and implicitly assigns users to roles
  - Further simplifies administration
  - The `Manager` role inherits the permissions of the `Administrator` and `Clerk` roles
  - A user assigned to the `Manager` role can activate the `Administrator` or `Clerk` role
- Separation of duty can be defined on roles
  - No user can be assigned to both the purchase order clerk and financial clerk roles



The Role Hierarchy



a-Limited Hierarchies



Added Advantages:

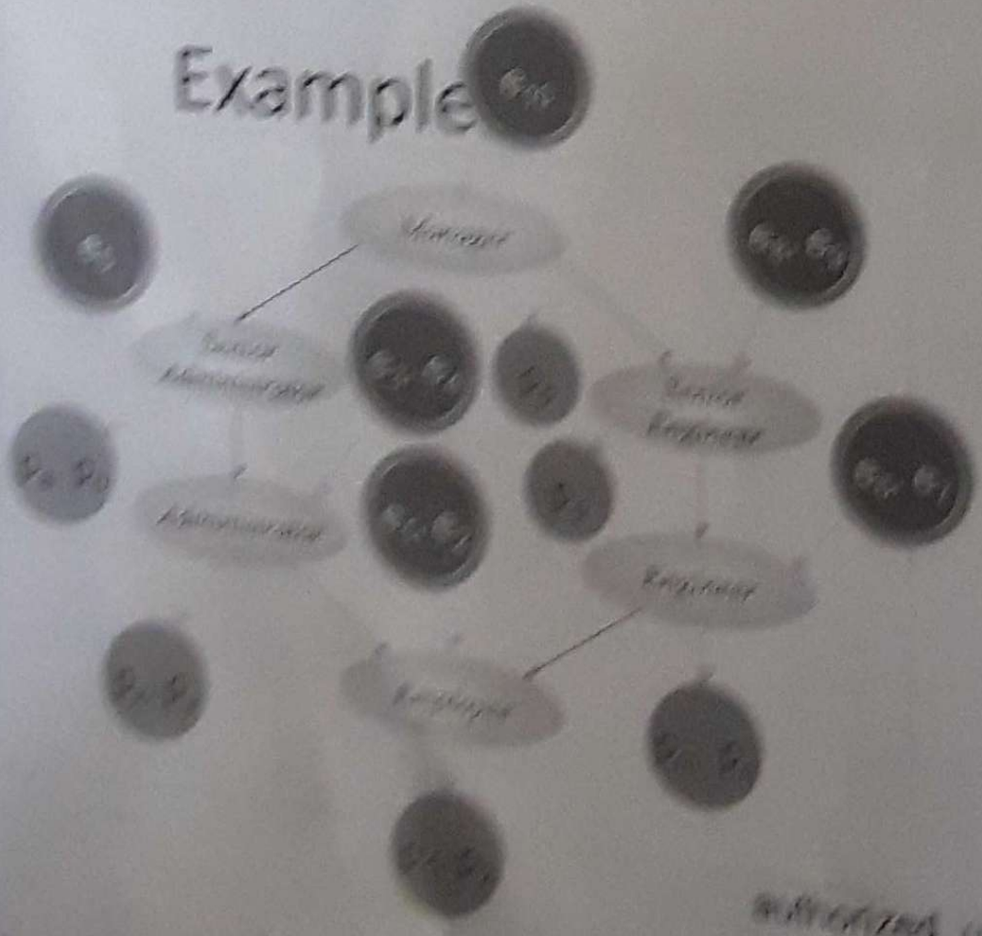
- User's can be included on edges of graph
- Role's can be defined from the privileges of two or more subordinate roles

b-General Hierarchies

RBAC with General Role Hierarchy

- authorized_users: Roles → 2^{Users}
  authorized_users(r) = {u | r' ≥ r & (r', u) ∈ UA}
- authorized_permissions: Roles → $2^{permissions}$
  authorized_users(r) = {p | r' ≥ r & (p, r') ∈ PA}
- RH ⊆ Roles x Roles is a partial order
  - called the inheritance relation
  - written as ≥
  $(r_2 \geq r_1) \rightarrow$ authorized_users(r_2) ⊆ authorized_users(r_1) &
  authorized_permissions(r_2) ⊆ authorized_permissions(r_1)

Example



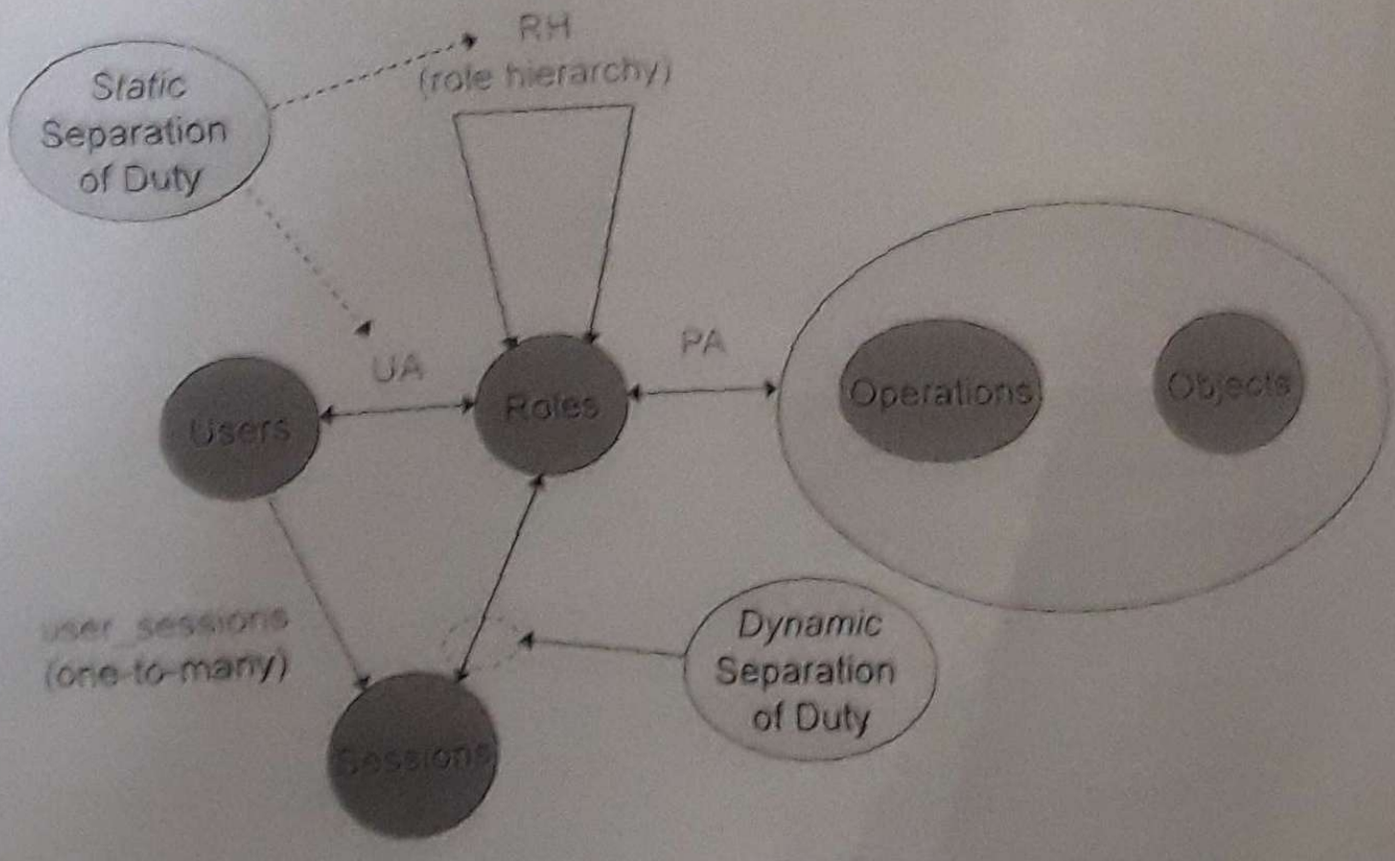
authorized_users(Employee)?
authorized_users(Administrator)?

authorized_permissions(Employee)?
authorized_

Constrained RBAC

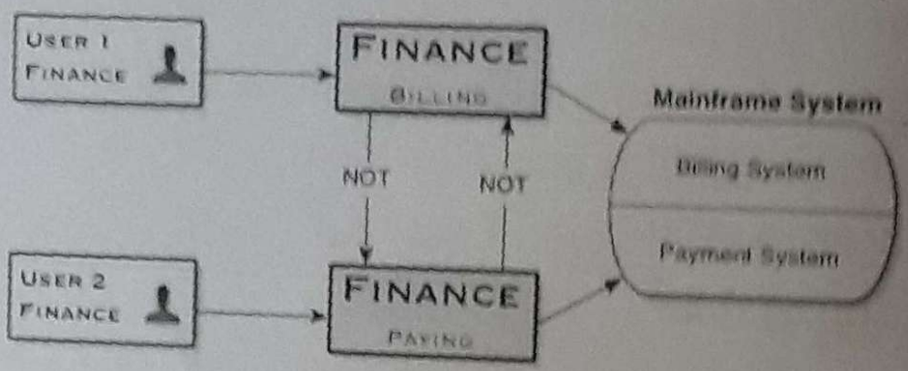
- It adds separation of duty relations to the RBAC model.
- Its purpose is to ensure that failures of omission or commission within an organization can be caused only as a result of collusion among individuals.
- To minimize the likelihood of collusion, individuals of different skills or divergent interests are assigned to separate tasks required in the performance of a business function.
- The motivation is to ensure that fraud and major errors cannot occur without deliberate collusion of multiple users.
- This RBAC standard allows for both static and dynamic separation of duty

Constrained RBAC



Separation of Duties

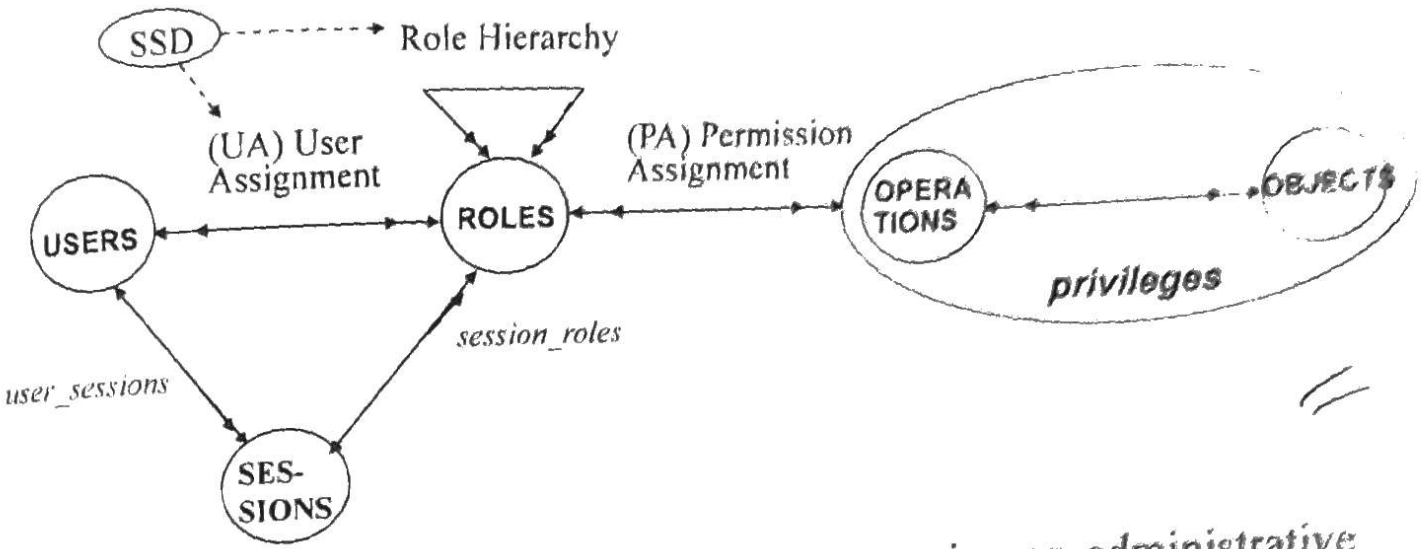
- No user should be given enough privileges to misuse the system on their own.
- Statically: defining the conflicting roles
- Dynamically: Enforcing the control at access time



Static Separation of Duty Relations

- Separation of duty relations are used to enforce conflict of interest policies. Conflict of interest in a role-based system may arise as a result of a user gaining authorization for permissions associated with conflicting roles.
- One means of preventing this form of conflict of interest is though *static separation of duty* (SSD), that is, to enforce constraints on the assignment of users to roles.
  - An example of such a static constraint is the requirement that two roles be mutually exclusive; for example, if one role requests expenditures and another approves them, the organization may prohibit the same user from being assigned to both roles.
- The SSD policy can be centrally specified and then uniformly imposed on specific roles. Because of the potential for inconsistencies with respect to static separation of duty relations and inheritance relations of a role hierarchy, we define SSD requirements both in the presence and absence of role hierarchies.

Static Separation of Duty Relations



SoD policies deter fraud by placing constrains on administrative actions and there by restricting combinations of privileges that are available to users

E.g., no user can be a member of both Cashier and AR Clerk roles in Accounts Receivable Department

Static Separation of Duty Relations

- ***Static Separation of Duty.*** SSD relations place constraints on the assignments of users to roles. Membership in one role may prevent the user from being a member of one or more other roles, depending on the SSD rules enforced.

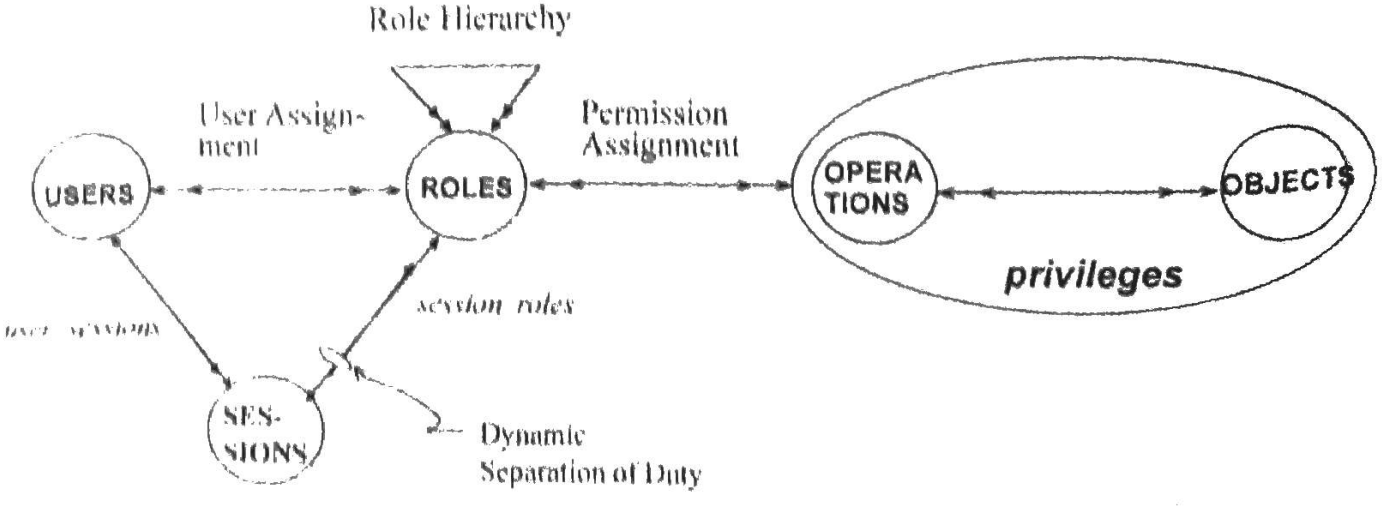
- ***Static Separation of Duty in the Presence of a Hierarchy***. This type of SSD relation works in the same way as basic SSD except that both inherited roles as well as directly assigned roles are considered when enforcing the constraints.

Dynamic Separation of Duty Relations

- Dynamic separation of duty (DSD) relations, like SSD relations, limit the permissions that are available to a user. However DSD relations differ from SSD relations by the context in which these limitations are imposed.

- DSD requirements limit the availability of the permissions by placing constraints on the roles that can be activated within or across a user's sessions.

Dynamic Separation of Duty Relations



DSoD policies deter fraud by placing constrains on the roles that can be activated in any given session there by restricting combinations of privileges that are available to users

RBAC's Benefits

	ESTIMATED TIME (IN MINUTES)		
	RBAC	NON-RBAC	DIFFERENCE
Assign existing privileges to new users	6.14	11.39	5.25
Change existing users' privileges	9.29	10.24	0.95
Establish new privileges for existing users	8.86	9.26	0.40
Termination of privileges	0.81	1.32	0.51

Cost Benefits

- Saves about 7.01 minutes per employee, per year in administrative functions
  - Average IT admin salary - $59.27 per hour
  - The annual cost saving is:
    - $6,924/1000; $692,471/100,000
- Reduced Employee downtime
  - if new transitioning employees receive their system privileges faster, their productivity is increased
  - 26.4 hours for non-RBAC; 14.7 hours for RBAC
  - For average employee wage of $39.29/hour, the annual productivity cost savings yielded by an RBAC system:
    - $75000/1000; $7.4M/100,000