Chapter 3 (cont)

Database security

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Outline

- Access control mechanisms:
  - DAC.
  - MAC.
  - RBAC.

3 Access Control Models

- DAC (Discretionary Access Control)
  - A type of security access control that grants or restricts the access of a user to an object.
  - A subject can grant/revoke privileges to/from another subjects conforming to a set of rules.
- MAC (Mandatory Access Control)
  - Define for a class of subjects a set of principles to directly or indirectly access classes of objects.
- RBAC (Role-based Access Control
  - A role is a set of privileges. Grant roles instead of individual privileges to a subject, then the subject will have all the privilege included in that role.

DAC

- Discretionary access control (DAC) is a type of security access control that grants or restricts object access via an access policy determined by an object's owner group and/or subjects.
- DAC mechanism controls are defined by user identification with supplied credentials during authentication, such as username and password.
- DACs are discretionary because the subject (owner) can transfer authenticated objects or information access to other users. In other words, the owner determines object access privileges.

DAC in commercial DBMS

- All commercial DBMS implement DAC.
- Current DAC models bases on System R model.
- System R: developed by Griffiths and Wade in 1976, is one of the first models introduced for Relational DBMS.
- System R: Base on authorizing object owner's admin privileges.

System R

- Managed objects: table and view.
- Privilege: select, insert, update, delete, drop, index (only for table), alter (only for table).
- Only support group, not support role.
- Use GRANT command to grant privilege, with GRANT OPTION.

System R

- The authorization is expressed via GRANT OPTION, meaning the granted user can re-grant the privilege to other users.
- A user can grant privileges on a relation to other users if he is the owner of the relation, or he is granted those privileges with GRANT OPTION.

GRANT command

GRANT *PrivilegeList*| ALL[PRIVILEGES]
ON *Relation* | *View*
TO *UserList* | PUBLIC
[WITH GRANT OPTION]

- Can grant privilege on table and view.
- Privilege applied to the entire table (or view).
- For update priviledge, indicating specific updatable columns is required.

GRANT – Ex:

A: GRANT select, insert ON NHANVIEN TO B
WITH GRANT OPTION;

A: GRANT select ON NHANVIEN TO C
WITH GRANT OPTION;

B: GRANT select, insert ON NHANVIEN TO C;

- C receives select privilege (from A and B) and insert privilege (from B).
- C can grant select privilege to other users, but C can not grant insert privilege.

GRANT command

- For each user, DBMS records:
  - A: Set of privileges this user has.
  - B: Set of privileges this user can grant to other users.
- When a user execute GRANT command, DBMS will
  - Identify the intersection of B and the privileges in Grant command.
  - If the intersection is empty, the command will not be run.

GRANT – Ex:

A: GRANT select, insert ON NHANVIEN TO C WITH GRANT OPTION;

A: GRANT select ON NHANVIEN TO B WITH GRANT OPTION;

A: GRANT insert ON NHANVIEN TO B;

C: GRANT update ON NHANVIEN TO D

WITH GRANT OPTION;

B: GRANT select, insert ON NHANVIEN TO D;

GRANT – Ex:

- In this example:
  - 1. Which command will be executed entirely?
  - 2. Which command will not be executed?
  - 3. Which command is partially executed?

TRẢ LỜI:

- 1. The first 3 commands (A is the table's owner).
- 2. The 4th command, C does not have Update privilege.
- 3. The 5th command, B has select, insert privilege, but B does not have grant option for insert privilege, so D only receive select privilege.

Revoke command

REVOKE *PrivilegeList*| ALL[PRIVILEGES]
ON *Relation* | *View*
FROM *UserList* | PUBLIC

- For revoking granted privileges.
- User can only revoke privileges granted by himself.
- User can not revoke grant option.
- A user will only loose a privilege if all users who granted the privilege to him have revoked it.

Revoke – Ex:

A: GRANT select ON NHANVIEN TO C WITH GRANT OPTION;

A: GRANT select ON NHANVIEN TO B WITH GRANT OPTION;

C: GRANT insert ON NHANVIEN TO D;

B: GRANT select ON NHANVIEN TO D;

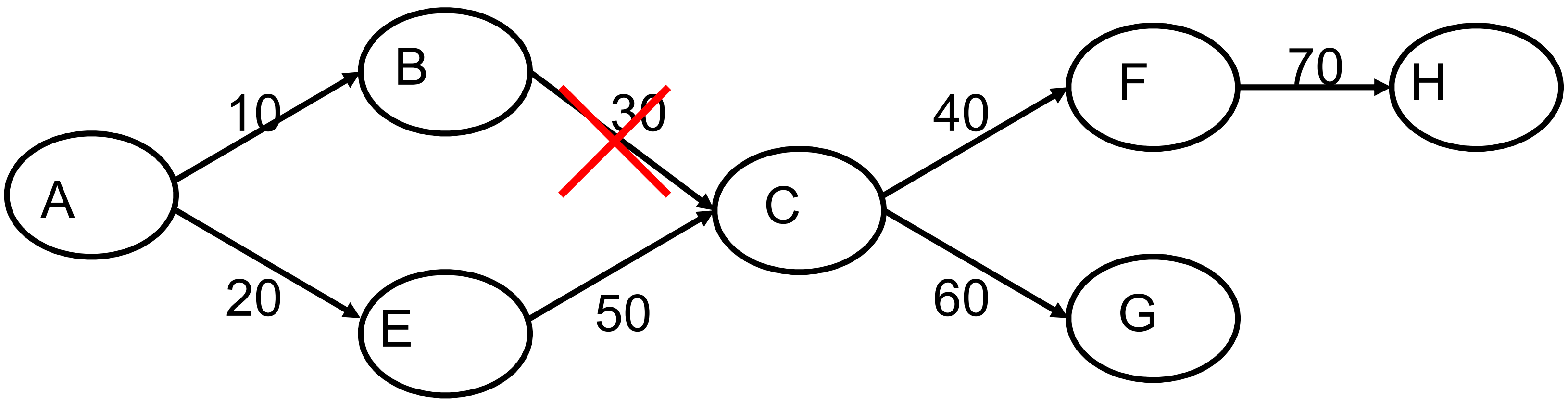
C: REVOKE select ON NHANVIEN FROM D;

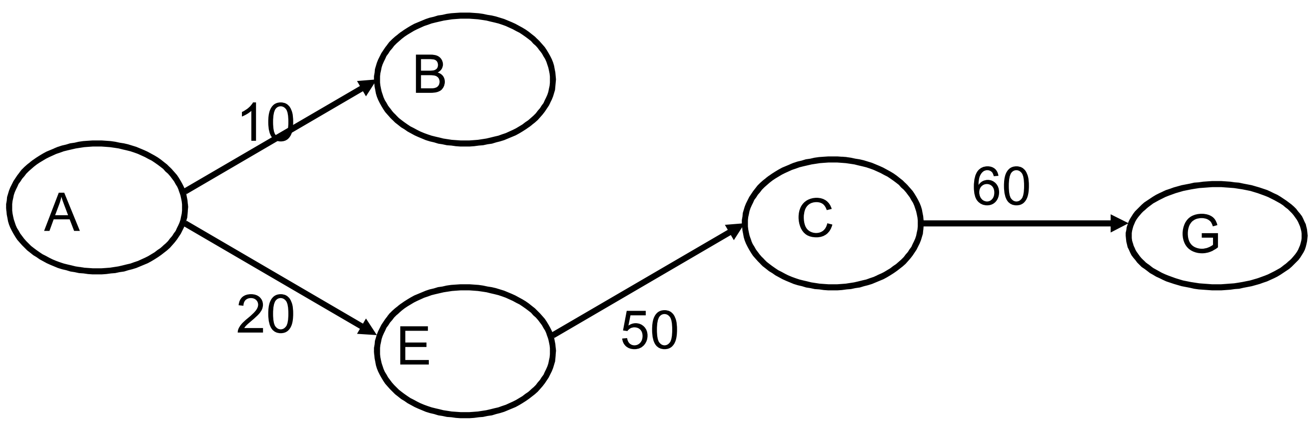
- After all these command, D can still select from NHANVIEN as B has not revoked this privilege from D.

Revoke

- Recursive revocation: When A revoke some privileges from B, DBMS will also revoke the privileges from all users who are granted by B.

Recursive revocation





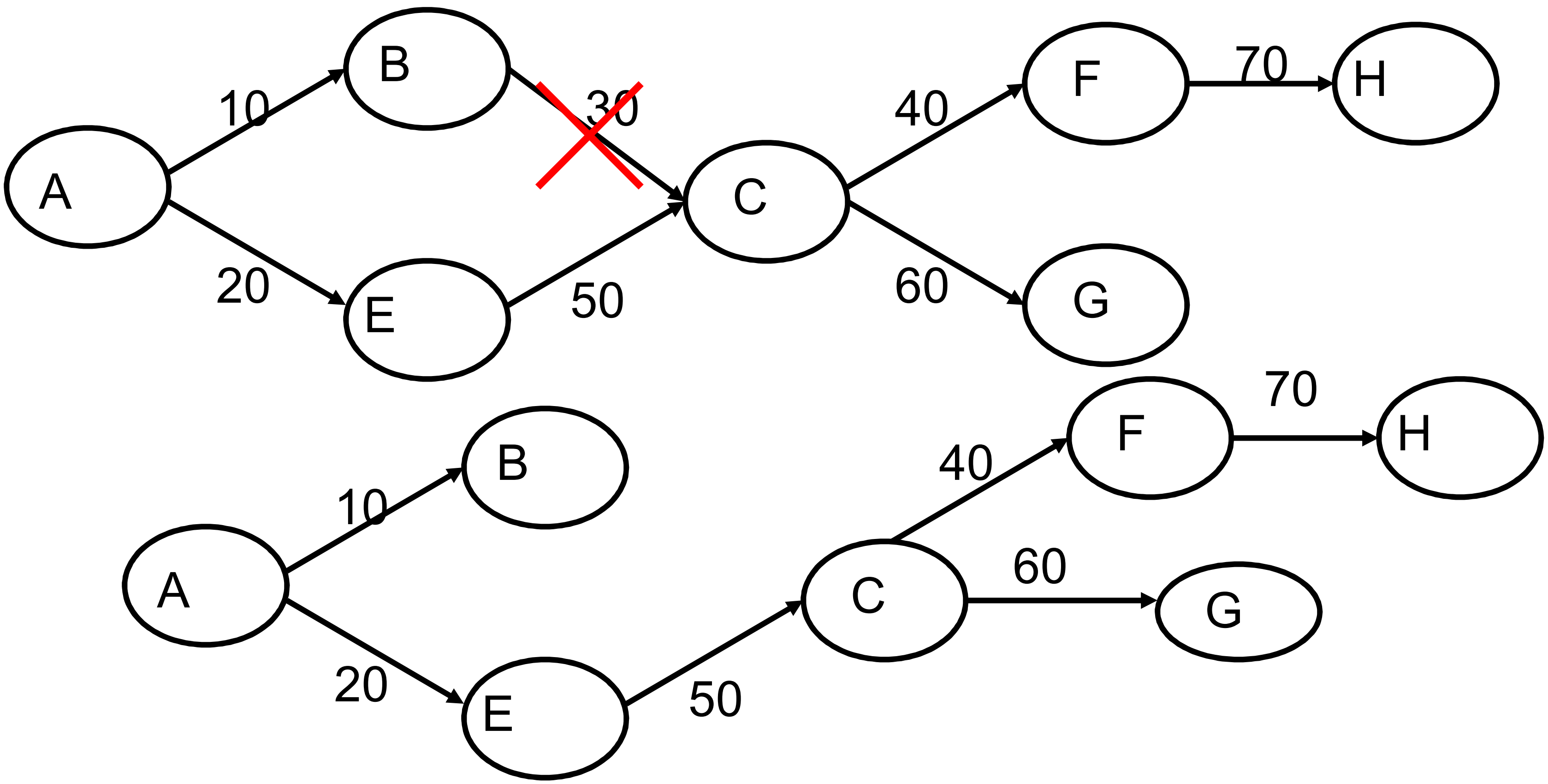
Recursive revocation

- In fact, when a user A leaves or changes his position, we only want to revoke his privileges. We do not want to revoke other users' privileges granted by A.

Recursive revocation

- Recursive revocation in System R bases on timestamps of every time privileges are granted.
- A variation of this approach is not based on timestamps, the purpose is to avoid Recursive revocation.
- Then, if C is revoked by B and C has the same privileges granted by another user (although later), the privileges that C granted to other users are still held.

A variation of Recursive revocation



Thu hồi quyền không dây chuyền (Noncascading revoke)

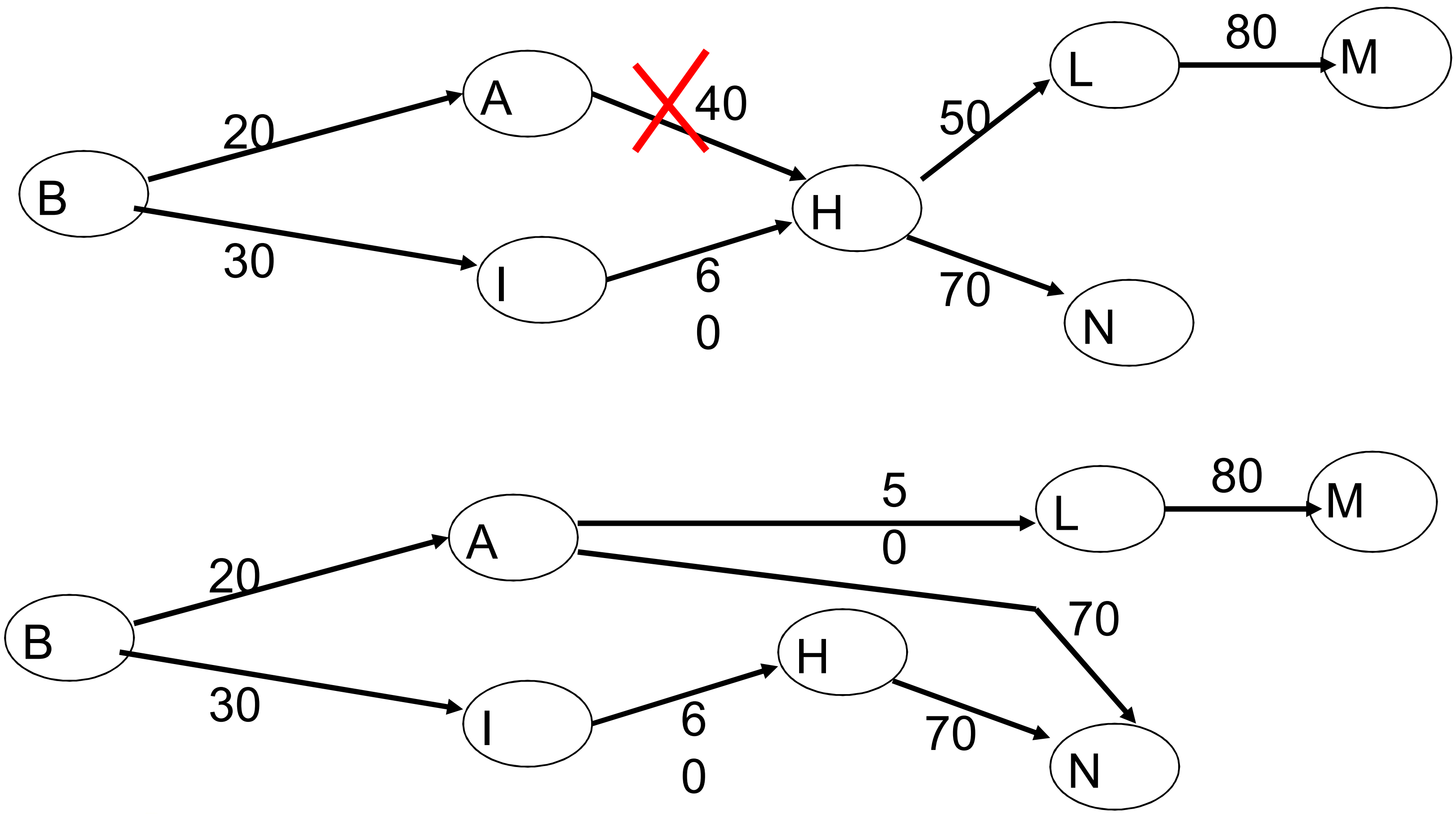
- Khi A thu hồi quyền truy xuất trên B thì tất cả quyền truy xuất mà B đã cấp cho chủ thể khác được thay bằng A đã cấp cho những chủ thể này.

Non-cascading revoke

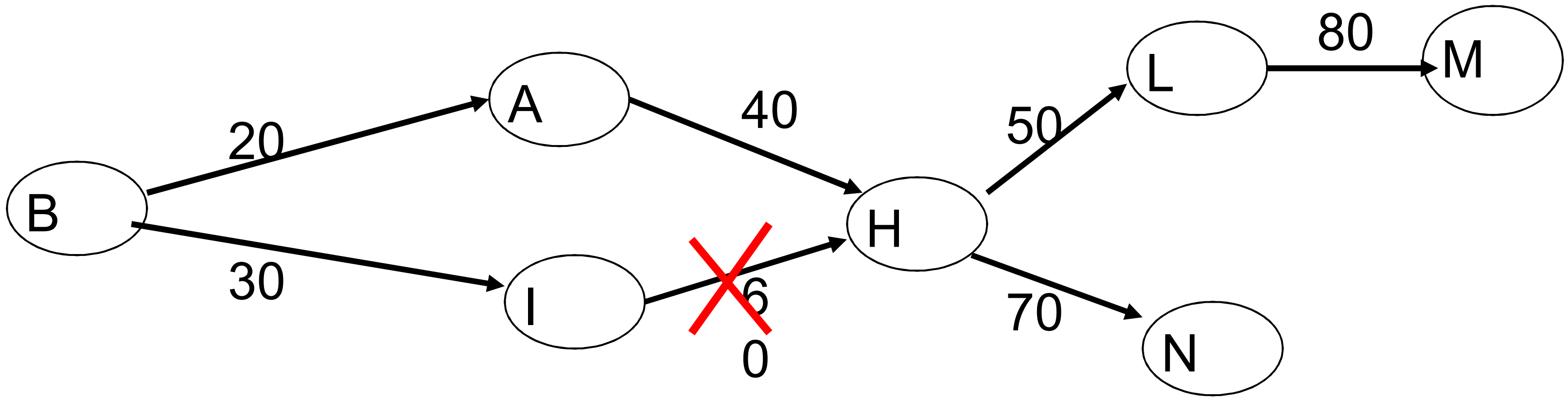
Notice:

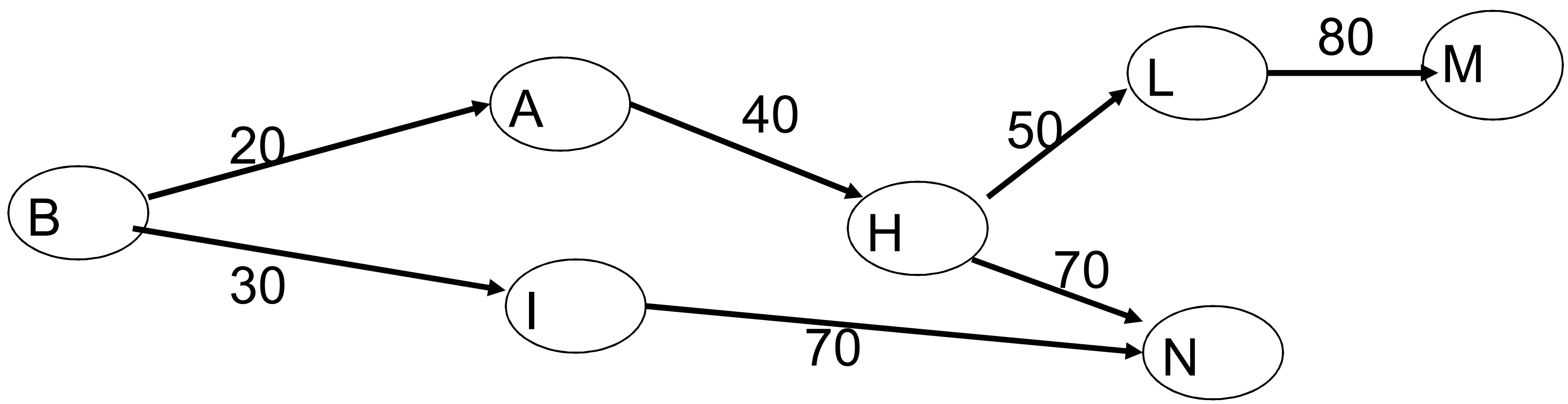
- Because B is granted privileges from multiple users (other than A), not all the privileges granted by B are actually granted by A. And A is only considered to be a substitute for B when B grants the privileges (A has granted B) to other users.
- A will be the user who granted the privileges B granted after receiving that authorization from A with WITH GRANT OPTION. With the privileges granted to B by $C \neq A$, then B in turn grants to others, B is still the user who grants these privileges.

Non-cascading revoke



Non-cascading revoke





Non-cascading revoke

- Note that with the privilege H granted to L, after revocation of the privilege, it is not allowed to replace I as the granter because this privilege was granted before I granted to H.

View and the content-based authorization

- In most RDBMSs, view is a commonly used mechanism to support content-based access control
- Use predicates to limit the content of data that needs authorization.
- Only those records satisfying the predicate are considered the objects of authorization.

View and the content-based authorization

- Content-based access control in RDBMS is done as follows:
  - Define a view V that uses predicates to select the data rows that you want to grant to subject S.
  - Grant S with privileges on view V.

View and the content-based authorization

- For example, suppose we want to grant user B permission to access only employees who have salary less than 20,000 :
  - CREATE VIEW V_NHANVIEN AS
    SELECT * FROM NHANVIEN
    WHERE LUONG < 20000;

  - GRANT Select ON V_NHANVIEN TO B;

Query processing steps

- Parsing
- Catalog lookup
- Authorization checking
- View Composition
  - The query on the view will be converted to the query on the base tables through this step.
  - The result will be based on the predicate of the query and the predicate that defines the view.

B:
```
SELECT * FROM V_NHANVIEN
        WHERE CONGVIEC = 'Lap trinh vien';
```

Query after view composition:
```
SELECT * FROM NHANVIEN
        WHERE LUONG < 20000 AND
         CONGVIEC = 'Lap trinh vien';
```

- Query optimization

Comment

- Because authorization checking is done before the view composition step, the privileges checked are based on the view, not on the base tables used to define the view.
- View is useful when granting permissions on columns - just create the view of the columns to which we want to grant access.
- View is also useful in granting access to statistical data (data generated from AVG, SUM, etc.)
- Privileges on Views can be granted or revoked like privileges on tables.
- Users who want to create a View must have Select privilege on the base tables.

View and the content-based authorization

- View definer: User who defines the view.
- The privileges view definer has on view depend on :
  - The semantics of views or the base tables used to create views.
  - The privileges that view definer has on base tables.
- The alter and index privileges do not apply to view, so the view definer never has these privileges on view.

```
A: CREATE VIEW V1 (MANV, TONGTIEN)
   AS SELECT MANV, LUONG+THUONG
   FROM NHANVIEN WHERE CONGVIEC = ‘Lap trinh vien’
```

The update operation is not defined on the TONGTIEN field of the view, so A will not be able to update this field.

Authorization on view

- To determine the privileges that view definer has on his view, BDMS must:
  - Identify the intersection of privileges that the view definer has on base tables and the privileges for operations that can be performed on the view.

Authorization on view – Ex:

- Consider the table NHANVIEN and suppose A is the person who created table NHANVIEN

  A: GRANT Select, Insert, Update ON NHANVIEN to D;

  D: CREATE VIEW V1 AS SELECT MANV, LUONG FROM NHANVIEN;

  D: CREATE VIEW V2 (MANV, LUONG_NAM) AS SELECT MANV, LUONG*12 FROM NHANVIEN;

- D can perform all the operations on V1 as D can do on table NHANVIEN, namely Select, Insert, Update.
- However, D can only perform on V2 the Select and Update command on column MANV.

Authorization on view – Ex:

- Can definitely grant privileges on view:
  - The privileges that a user can grant are the privileges that he has with a grant option on base tables.
  - For example, user D cannot grant any privileges on view V1 and view V2 that D has defined because D was not granted with a grant option before.

Authorization on view – Ex:

- Consider the following statements :
  - A: GRANT Select ON NHANVIEN TO D WITH GRANT OPTION;
  - A: GRANT Update, Insert ON NHANVIEN TO D;
  - D: CREATE VIEW V4 AS SELECT MANV, LUONG FROM NHANVIEN;

D's privileges on V4:

- Select with Grant Option;
- Update, Insert without Grant Option;

DAC – Positive permissions / Negative permissions

- System R and most DBMS use close policies.
  - With closed policy, lack of privileges means that there is no access.
- When user accesses a data object, DBMS looks up in the list of privileges that he has, if no suitable privileges found, access is denied.
  - Drawback: The lack of access privileges does not prevent the user from receiving privileges from another user.
  - For example, x does not have privileges on object o, but in the case of a system that uses a policy of sharing administrative rights, the owner who has the privileges to grant access on o may accidentally grants privileges on o to x.
- Negative permission is introduced to solve this problem.

DAC – Positive permissions / Negative permissions

- Positive permissions : List of can-use privileges.
- Negative permissions: List of can-NOT-use privileges.
- However, this can cause some conflict.

Eg: A can WRITE to table NHANVIEN.

A can not READ from table PHONGBAN.

A can not WRITE to column LUONG of table NHANVIEN.

- Often Negative permissions get the priority.

DAC – Positive permissions / Negative permissions

- Negative permissions is enforced as privileges blocking.
- When a user is assigned Negative permissions on an object, his Positive permissions on the object are blocked, until the Negative permissions are revoked.

  Advantages:

  - If accidentally assigned negative permissions to users, they can be revoked.
  - It is possible to block a person's access for a period of time by assigning negative permissions and then revoking them.

DENY command

- DENY {ALL [PRIVILEGES] | *permission*[,...*n*]} { [(*column*[,...*n*])] ON { *table* | *view*} |

  ON {*table* | *view*} [(*column*[,...*n*])] | {*procedure* | *extended_procedure*} }

  TO *security_account*

Ex:

DENY SELECT, INSERT, UPDATE

ON NHANVIEN

TO A, B

Revoking Granted and Denied Permissions

- **Use REVOKE:**
  - **We can revoke Positive permissions (granted by GRANT command)**
  - **We can revoke Negative permissions(blocked by DENY command)**
- REVOKE and DENY are alike in that they prohibit some operations.
- REVOKE and DENY differ in that REVOKE delete a privilege granted in the past, while DENY blocks a privilege to be used in future.

DAC – Context constraint

- In fact, users are only allowed to access data for a certain period of time.
- There should be a mechanism to support access within a given period.
  - Ex: the mechanism which only allowing part-time workers to access data only between 9am and 1pm from 1/1/98.
- In most DBMS, this is often implemented in application programs.
  - Disadvantage: When confirming and changing access control policies, it is not guaranteed that this policy is enforced.
- The Time-based access control model is proposed to address this problem.

DAC – Context constraint

- Effective time :
  - Each access privileges has a Effective time
  - After the expiry of the Effective time, the access privileges are automatically revoked without the need of the administrator revoking them.
- Usage cycle of access privileges :
  - Cyclic access rights can be positive or negative permissions. If in the same period of time the user has both a positive or negative permissions on the same object and the same access method, then the priority is negative.
- Inference mechanisms based on inference rules
  - Inference rules denote the constraint of privileges over time.
  - These rules allow inference of new access privileges based on the existence or non-existence of other access privileges for a specified period of time.
  - For example: If two users work on the same project, they must have the same access privileges on the same objects.

DAC – Context constraint

- Access privileges are defined as a set of 5 attributes auth = (s, o, m, pn, g).

Where as:

- **s** (subject), **g** (granter) U (user list).
- **m** $\in$ M (access method).
- **o** $\in$ O(object).
- **pn** $\in$ {+ , -}(pos/neg permission).

- *Ex:*

(B,o1, read, +, C) : C grant to B privilege to read o1.

(B, o1, write, -, C) : C block B from writing to o1.

DAC – Context constraint

- Time-based access control is the triple ([begin,end], P, auth).

Where :

- **begin** is start time.
- **end** is end time.
- **P** is the cycle expression.
- **auth** is the access privilege.

- The privileges will take effect in cycle P with the access day >= t_b (begin day) and <=t_e (end day).

Ex 2: A1= ([1/1/94,], Mondays, (A, o1, read, +, B)) granted by B, denote that A can read from o1 on Monday from 1/1/94.

DAC – Context constraint

- Using negative permissions can lead to some conflict.
- Ex:
  - Assume that we have A2 = ([1/1/95,], Working-days, (A, o1, read, -, B)) along with A1= ([1/1/94,], Mondays, (A, o1, read, +, B)).
  - From 1/1/95, A has Neg and Pos permissions on o1 at the same time for the same read operation.
  - Solve: The negative permissions take priority.

DAC – Context constraint

- The inference rule is defined as the triple ([begin, end], P, A <OP> $\mathscr{A}$) where:
  - **begin** is the start day.
  - **end** is the end day .
  - **P** is the cycle expression,
  - **A** is the privilege.
  - *$\mathscr{A}$ is the Bool expression of the access privilege*.
  - **OP** is one of the operators: WHENEVER, ASLONGAS, UPON.
- The semantics of each operator in the inference rules :

([begin, end], P, A WHENEVER $\mathscr{A}$) : privilege A takes effect at time t $\in$ cycle P and t $\in$ [t_b, t_e] when $\mathscr{A}$ takes effect.

- Ex :

  A1 = ([1/1/95,1/1/96], Working-days, (M, o1, read, B))

  R1= ([1/1/95,], Summer-time, (S, o1, read, +, Bob) WHENEVER (M, o1, read, +, B)).

→S can only read object o1 in summer time, from1/1/95 when M can read o1.

DAC - Comment

- Advantage:
  - DAC is flexible in policy, so it is applied by most DBMS
- Disadvantage:
  - Lack of information flow control to protect DB against Trojan Horse attack.

- RBAC (Role based Access Control)
  - Most DNMS support RBAC.
  - RBAC can be used in conjunction with DAC or MAC or used independently.
  - Most DBMS only support flat RBAC.

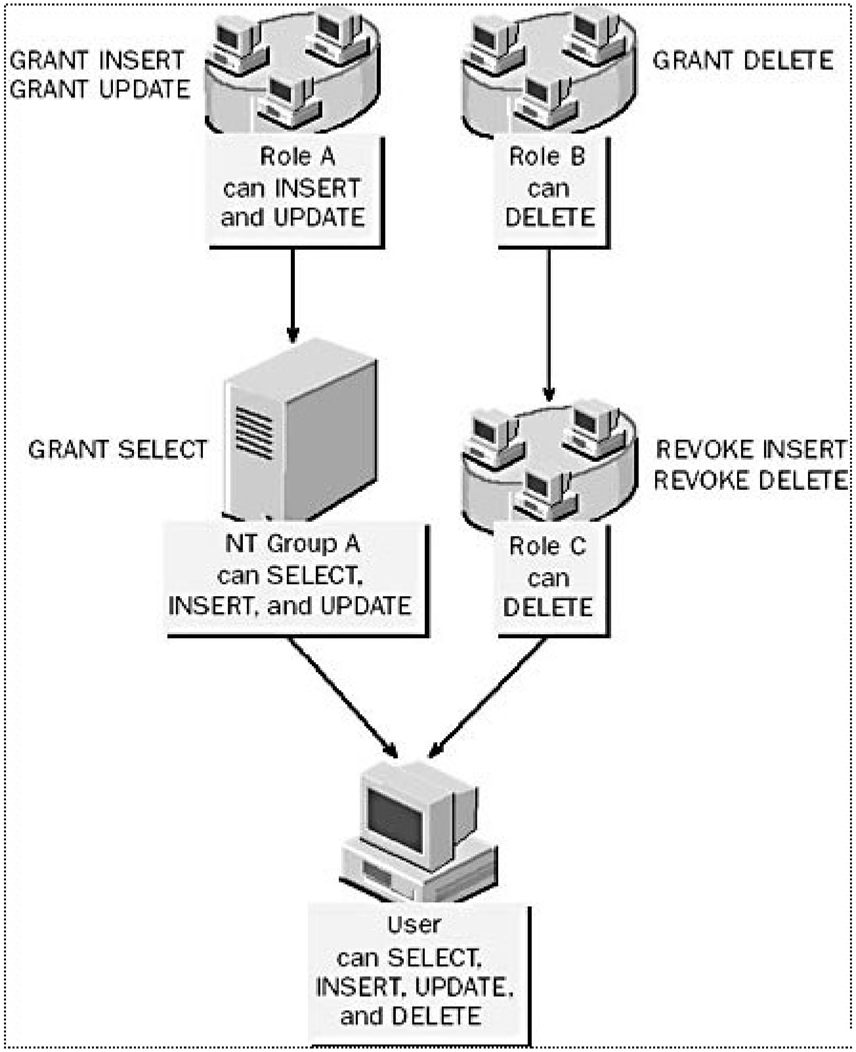
Role and Group

- At a basic level, roles can be considered equivalent to groups.
  - A privilege can be assigned to one or more groups or one or more roles, and a group or role is associated with one or more privileges.
  - Assigning a user to a group or role allows the user to use the privileges of that group or role.
  - The main difference between group and role is that group is a set of users (not a set of permissions). A role is a collection of users as well as a collection of permissions. A role is the intermediary object to bring these two sets together.

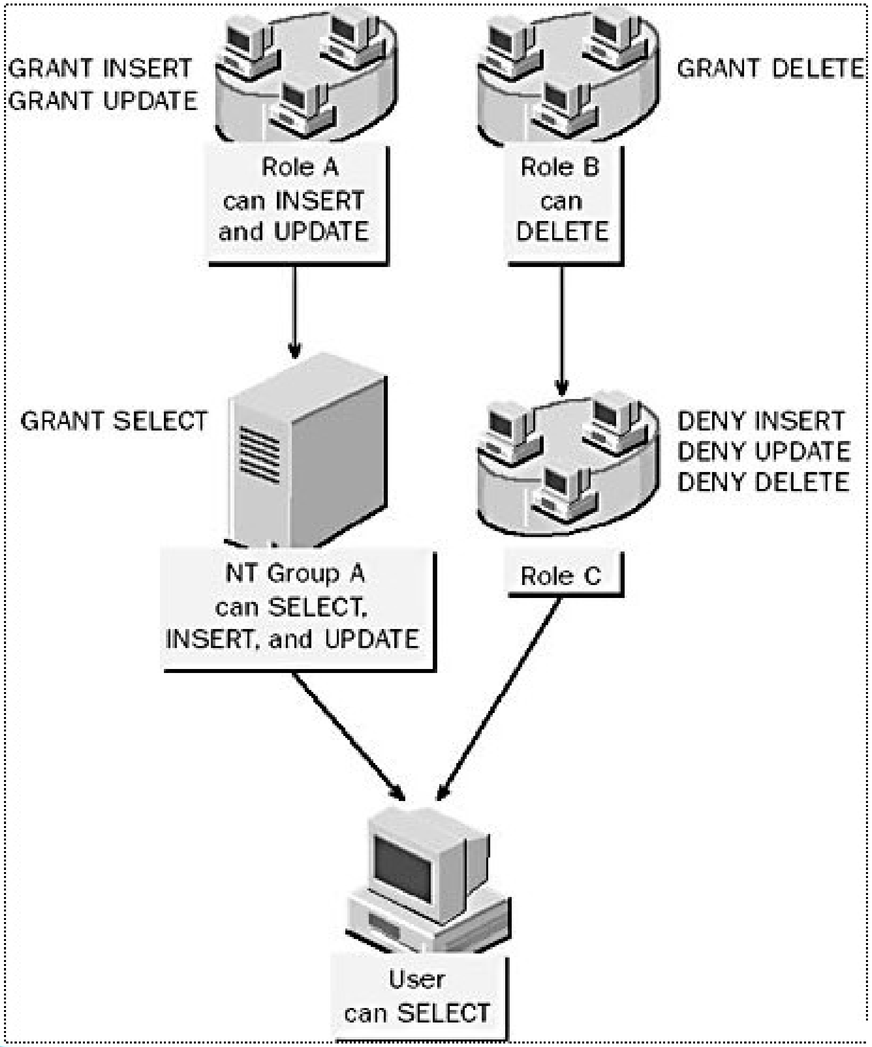
RBAC

- Applied in the early 1970s.
- The main concept of RBAC is that the privileges are associated with roles.
- When the number of subjects and objects is large → the number of privileges can become extremely large.
- If users are in high demand, granting and revoking will happen regularly.
- With RBAC, it is possible to pre-limit role- privilege relationships, which makes assigning users to predefined roles easier.
- Without RBAC it would be difficult to determine which privilege is to be granted to which user.
- Users are assigned the appropriate roles. This makes it simple to manage permissions.
- In an organization, different job position are categorized into roles and users are assigned roles based on their responsibilities and capabilities.

Ex:


GRANT INSERT
GRANT UPDATE
GRANT DELETE
Role A
can INSERT
and UPDATE
Role B
can
DELETE
GRANT SELECT
REVOKE INSERT
REVOKE DELETE
NT Group A
can SELECT,
INSERT, and UPDATE
Role C
can
DELETE
User
can SELECT,
INSERT, UPDATE,
and DELETE

Ex:



GRANT INSERT
GRANT UPDATE
Role A
can INSERT
and UPDATE
GRANT DELETE
Role B
can
DELETE
GRANT SELECT
NT Group A
can SELECT,
INSERT, and UPDATE
DENY INSERT
DENY UPDATE
DENY DELETE
Role C
User
can SELECT

Ex:

Account	Permission assigned	Result
Role A	GRANT SELECT	Members of role A have SELECT permission
Role B, member of role A	GRANT INSERT	Members of role B have SELECT permissions (because role B is a member of role A) and INSERT permission
User A, member of role B	DENY INSERT	User A has SELECT permission because it is a member of role A. User A does not have INSERT permission because INSERT has been denied to this user
Role A	DENY SELECT	Members of role A do not have SELECT permission

Account	Permission assigned	Result
Role B, member of role A	GRANT SELECT	Members of role B do not have SELECT permission because role B is a member of role A, which denies the SELECT permission
User A, member of role B	GRANT INSERT	User A has INSERT permission only
Role A	GRANT SELECT	Members of role A have SELECT permission
Role B, member of role A	REVOKE SELECT	Members of role B have SELECT permission because they still get it from role A
User A, member of role B	GRANT INSERT	User A has SELECT permissions (because the user is a member of role B) and INSERT permissions

MAC

- MAC (Mandatory Access Control)

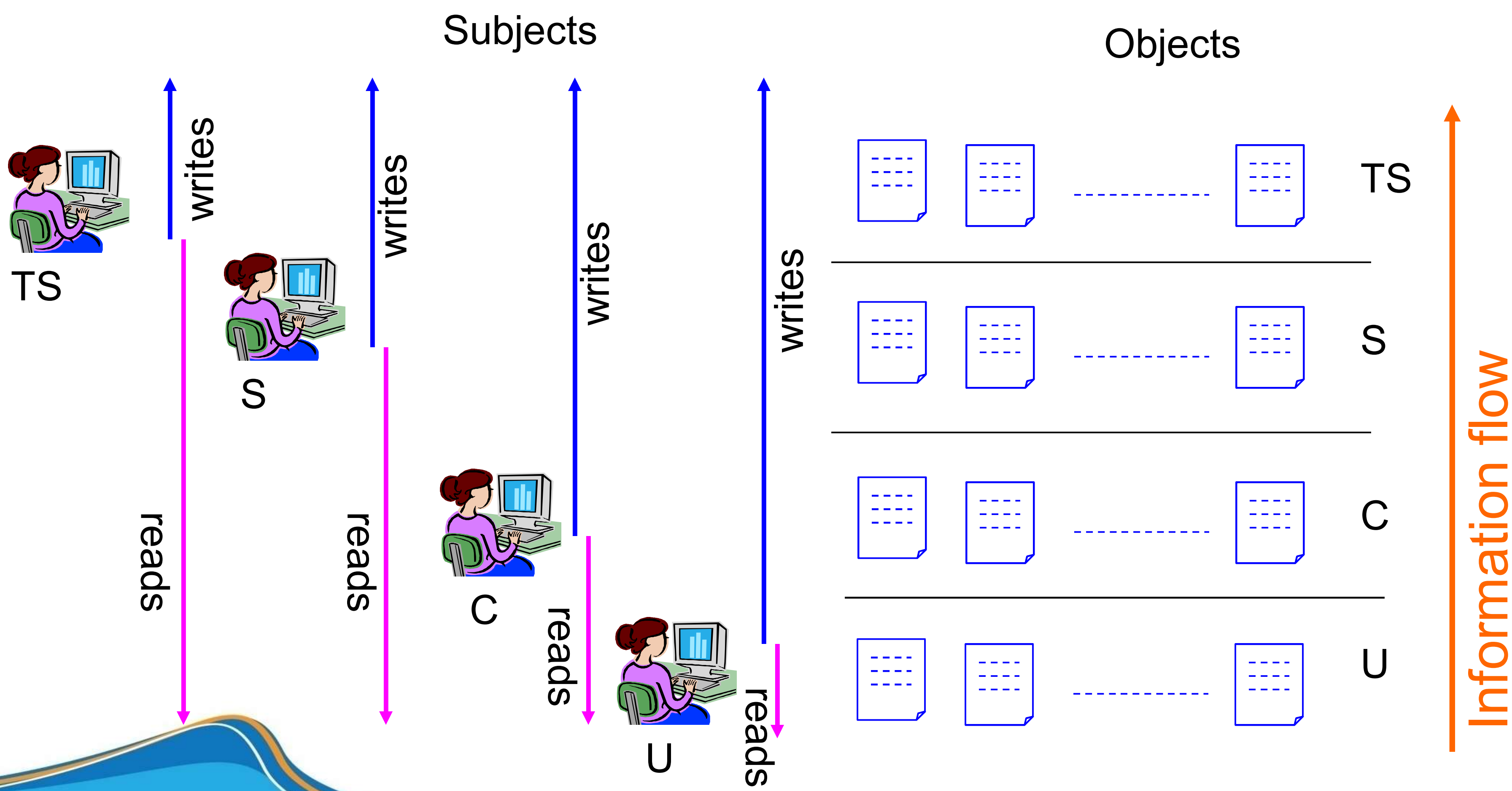
MAC

- Access control is based on the classification of subjects and data object.
- MAC is used in environments that need strict security control, such as government, military, ...
- MAC is installed in ORACLE Database.

MAC

- Object: tables, views, tuples.
- Subject: users, user programs.
- Security class (or level, or labels)
  - Top Secret (TS), Secret (S), Confidential (C), Unclassified (U), where TS > S > C > U
- Each subject and each object is classified into a class.
  - No read – up: Subject S can read object O if Class(S) >= Class(O).
  - No write – down: Subject S can write O if class(S) <= Class (O).
    - However, in fact, most DBMSs do not allow write ups, only write to the same levels object. Check this out with Oracle?

MAC



Subjects
Objects
writes
writes
writes
writes
TS
S
C
U
reads
reads
reads
reads
TS
S
C
U
Information flow

MAC – Comment

- The principle of the data unit of the security object.
  - The entire database, or file, or columns or in each item.
- There is no automatic technique for assigning security labels.
- Many users request access simultaneously.
  - Because of the information flow policy, people with higher levels of security can not write to a data categories of lower level. For example, assume 2 subjects s1 and s2 with label (s1)> label (s2), data item d with label (d) = label (s2), and commercial rule states that writing data to d by s2 requires s1's approval. This is not suitable for commercial applications of MLS database technology.

MAC

- MAC is also called Multilevel security – MLS, applied to Multilevel Relational Model - MLR.
- The DBMS that satisfies the properties of multi-level security is designed based on the Bell and LaPadula platform models.

MLR

- In a Multilevel security model, data items and subjects have their own access levels, for example TS (Top Secret), S (Secret), U (Unclassified), etc., including classification and permission to use confidential information (clearance).
- The subject, when accessing data, is restricted by the mandatory access controls, the "no read up, no write down" model by Bell and LaPadula.

MLR

- A multi-level relation is described by two components :
  - R(A1,C1,…., An,Cn, TC) where:
    - Ai is a property in range Di.
    - Ci is a classification property for Ai; Its domain is a collection of access levels that can be associated with the value of Ai.
    - TC is a classification property for (TC=TUPLE-CLASS), is the highest access level for ci.
  - Classification property can not be null.

Name	**C_{Name}**	**Dept#**	**$C_{Dept\#}$**	**Salary**	**$C_{Dept\#}$**	**TC**
A	Low	Dept1	Low	100K	Low	Low
B	High	Dept2	High	200K	High	High
S	Low	Dept1	Low	150K	High	High

MLR

- An instance of the relation at level c contains all the data that the subject at class c sees. Therefore, it contains all the data that access level <= c.
- All elements with access level higher than c, or not comparable are hidden behind the null value.

Name	**C_{Name}**	**Dept#**	**$C_{Dept\#}$**	**Salary**	**$C_{Dept\#}$**	**TC**
Bob	Low	Dept1	Low	100K	Low	Low
Sam	Low	Dept1	Low	null	Low	Low

Low instance

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

High instance

MLR

- The required conditions :
  - A multi-level relation must satisfy the following conditions :
    - For each tuple in a multilevel relation, the primary key's attributes must have the same access level.
    - For each tuple in a multilevel relation, the access level associated with a property other than the PK must be greater than or equal to the access level of the primary key.
  - Keys and multiple instances :
    - In the standard relational DB model, each tuple is uniquely identified by its key.
    - When apply an access levels, there may be concurrent sets of equal values at key properties, but with different access levels, this phenomenon is called multiple instances.

MLR

- Polyinstantiation :
  - Occurs in the following two states :
    - *Invisible* Polyinstantiation : When a lower level user inserts data into a field that already contains data at a higher or incomparable level.
    - *Visible* Polyinstantiation : When a high level user inserts data into a field that already contains data at a lower level.

Name	C_{Name}	**Dept#**	$C_{Dept\#}$	**Salary**	$C_{Dept\#}$	**TC**
A	Low	Dept1	Low	100K	Low	Low
B	High	Dept2	High	200K	High	High
S	Low	Dept1	Low	150K	High	High
B	Low	Dept1	Low	100K	Low	Low

Tuples with name “B” are multi instance

MLR

- *Invisible* Polyinstantiation :
  - Suppose a user at a low level requires inserting data with the same primary key of a tuple that exists at a higher level; DBMS has three options :
    1. Inform the user that a tuple with the same primary key exists at a high security level and refuses to insert.
    2. Replace existing higher level tuple with the new inserted tuple at a lower level.
    3. Insert new tuple at lower level without changing existing tuple at higher level (ie multi-instance entity).
    - Choose 2) allows the low-level user to overwrite data that he does not see and thus loses the data integrity.
    - Choose 3) is a reasonable choice; because its importance is to introduce a Polyinstantiation entity.

MLR

- *Visible* Polyinstantiation :
  - Suppose a high-level user requires inserting data with the same primary key as a lower-level tuple; DBMS has three options :
    1. Inform the user that a tuple with the same primary key exists and refuses to insert.
    2. Replace existing tuple at a lower level with the new tuple inserted at a higher level.
    3. Insert new tuple at a higher level without changing existing tuple at lower level (ie Polyinstantiation entity).
- Choose 3) is a reasonable choice; because its importance is to introduce a multi-instance entity.

MLR

- 5 constraint:
  - Entity integrity
  - Polyinstantiation integrity
  - Data-borrow integrity
  - Foreign key integrity
  - Referential integrity
- 5 commands(insert, delete, select, update, UPLEVEL) manipulating in multi-level relations.

- *Ref: Ravi Sandhu, Fang Chen, The multilevel Relational (MLR) data Model, ACM, 1998.*

- The end of chapter 3.