6

SQL: Views, Integrity Constraints and Access Control

Prerequisites for This Section

- **Readings:**
  - **Required:** Connolly and Begg, sections 3.4 and 6.4
  - **Required:** Connolly and Begg, sections 6.2 and 6.5
  - **Required:** Connolly and Begg, section 6.6

- **Assessments:**
  - Multiple-Choice Quiz 4

Section Objectives

In this section you will learn:

① Purpose, advantages and disadvantages of views.

② How to create and drop views using SQL.

③ The process of view resolution and under what conditions views are updatable.

④ Integrity control consists of constraints imposed in order to protect the database from becoming inconsistent.

⑤ How to use the CREATE and ALTER TABLE statements to define the integrity constraints.

⑥ Access control is built around the concepts of authorization identifiers, ownership, and privileges.

⑦ How to use the GRANT and REVOKE statements to pass on and revoke the privileges.

DreamHome Rental Database

The relational schema for part of DreamHome case study is:

- **Branch** (branchNo, street, city, postcode)
- **Staff** (staffNo, fName, lName, position, sex, DOB, salary, branchNo)
- **PropertyForRent** (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)
- **Client** (clientNo, fName, lName, telNo, prefType, maxRent)
- **PrivateOwner** (ownerNo, fName, lName, address, telNo)
- **Viewing** (clientNo, propertyNo, viewDate, comment)
- **Registration** (clientNo, branchNo, staffNo, dateJoined)

Agenda

1. Views
2. Creating and Dropping a View
3. Querying and Updating a View
4. Advantages and Disadvantages of Views
5. Integrity Constraints in SQL
6. Access Control

Views

- **View** is
  - **A virtual relation** that does not necessarily actually exist in the database but is produced upon request, at time of request.
  - **Dynamic result** of one or more relational operations operating on base relations to produce another relation.
- **Contents of a view** are defined as a query on one or more base relations or views.

Agenda

1. Views
2. Creating and Dropping a View
3. Querying and Updating a View
4. Advantages and Disadvantages of Views
5. Integrity Constraints in SQL
6. Access Control

CREATE VIEW

- The **CREATE VIEW** statement syntax:

CREATE VIEW ViewName [(newColumnName [,...])]

    AS *subselect*

    [**WITH [CASCADED | LOCAL] CHECK OPTION**]

- Can assign a name to each column in view.
  - If list of column names is specified, it must have same number of items as number of columns produced by *subselect*.
  - If omitted, each column takes name of corresponding column in *subselect*.
  - List must be specified if there is any ambiguity in a column name.
- The ***subselect*** is known as the defining query.

Different Views

① A **horizontal view** restricts a user’s access to selected rows of one or more tables

② A **vertical view** restricts a user’s access to selected columns of one or more tables

③ A **grouped view** is the view that the use of a subselect contains a GROUP BY clause.

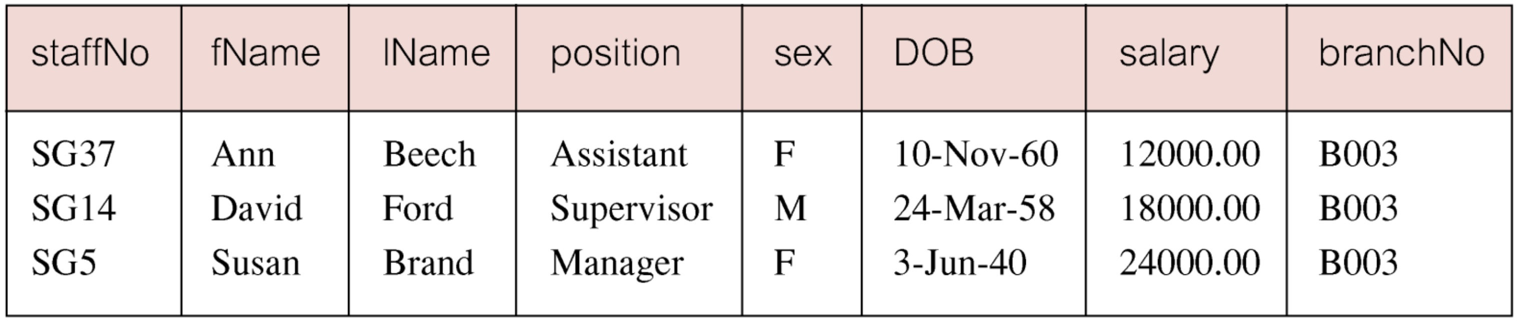
④ A **joined view** is the view that the use of a subselect contains multiple tables.

Example 6.3 - Create Horizontal View

- Create view so that manager at branch B003 can only see details for staff who work in his or her office.

```
CREATE VIEW Manager3Staff
    AS      SELECT *
            FROM   Staff
            WHERE branchNo = ‘B003’;
```

Table 6.3 Data for view Manager3Staff.



staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

Example 6.4 - Create Vertical View

- Create view of staff details at branch B003 excluding salaries.

```
CREATE VIEW Staff3
  AS SELECT staffNo, fName, lName, position, sex
     FROM     Staff
     WHERE  branchNo = ‘B003’;
```

Table 6.4 Data for view Staff3.

staffNo	fName	lName	position	sex
SG37	Ann	Beech	Assistant	F
SG14	David	Ford	Supervisor	M
SG5	Susan	Brand	Manager	F

Example 6.5 - Grouped and Joined Views

- Create view of staff who manage properties for rent, including branch number they work at, staff number, and number of properties they manage.

- **CREATE VIEW** StaffPropCnt (branchNo, staffNo, cnt)

```
AS  SELECT     s.branchNo, s.staffNo, COUNT(*)
    FROM       Staff s, PropertyForRent p
    WHERE      s.staffNo = p.staffNo
    GROUP BY s.branchNo, s.staffNo;
```

Table 6.5 Data for view StaffPropCnt.

branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

What happens?

- INSERT INTO Manager3Staff(staffNo, branchNo) VALUE (‘SG55’, ‘B003’);
- INSERT INTO Manager3Staff(staffNo, branchNo) VALUE (‘SG50’, ‘B002’);
- UPDATE Manager3Staff SET branchNo = ‘B007’ WHERE staffNo = ‘SG37’;
- SELECT COUNT(*) FROM Staff;
- SELECT COUNT(*) FROM Manager3Staff;

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Table 6.3 Data for view Manager3Staff.

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

```
CREATE VIEW Manager3Staff
    AS      SELECT *
            FROM    Staff
            WHERE branchNo = ‘B003’;
```

WITH CHECK OPTION

- **New rows** appear within view when insert/update on view cause them to satisfy **WHERE** condition.
- Rows that enter or leave a view are called ***migrating rows***.
- **WITH CHECK OPTION** ensures that if a row fails to satisfy **WHERE** clause of defining query, it is not **added** to underlying base table.

Example 6.6 - WITH CHECK OPTION

- **Example 6.6**

```
CREATE VIEW Manager3Staff
  AS       SELECT *
           FROM    Staff
           WHERE  branchNo = 'B003'
  WITH CHECK OPTION;
```

- **Cannot update** branch number of row B003 to B002 as this would cause row to migrate from view.
- Also **cannot insert** a row into view with a branch number that does not equal B003.

What happens?

- INSERT INTO Manager3Staff(staffNo, branchNo) VALUE (‘SG55’, ‘B003’);
- INSERT INTO Manager3Staff(staffNo, branchNo) VALUE (‘SG50’, ‘B002’);
- UPDATE Manager3Staff SET branchNo = ‘B007’ WHERE staffNo = ‘SG37’;
- SELECT COUNT(*) FROM Staff;
- SELECT COUNT(*) FROM Manager3Staff;

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Table 6.3 Data for view Manager3Staff.

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

```
CREATE VIEW Manager3Staff
      AS        SELECT *
                FROM    Staff
                WHERE branchNo = ‘B003’
WITH CHECK OPTION;
```

WITH LOCAL/CASCADE CHECK OPTION

- **LOCAL/CASCADE** apply to view hierarchies.
  - With **LOCAL**, any row insert/update on view and any view directly or indirectly defined on this view must not cause row to disappear from view unless row also disappears from derived view/table.
  - With **CASCADE** (default), any row insert/ update on this view and on any view directly or indirectly defined on this view must not cause row to disappear from the view.

Example - WITH LOCAL CHECK OPTION

- Now consider the following:

① **CREATE VIEW LowSalary**

AS SELECT * FROM Staff WHERE **salary > 9000**;

② **CREATE VIEW HighSalary**

AS SELECT * FROM **LowSalary**

WHERE **salary > 10000**;

WITH LOCAL CHECK OPTION;

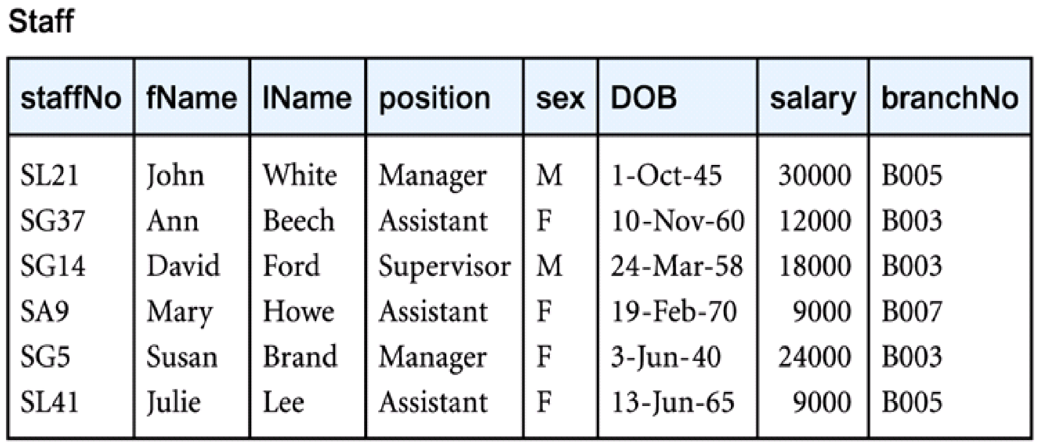
③ **CREATE VIEW** Manager3Staff

AS SELECT * FROM **HighSalary**

WHERE **branchNo = 'B003'**;

What happens?

```
UPDATE Manager3Staff
SET       salary = 9500
WHERE  staffNo = ‘SG37’;
```


Staff

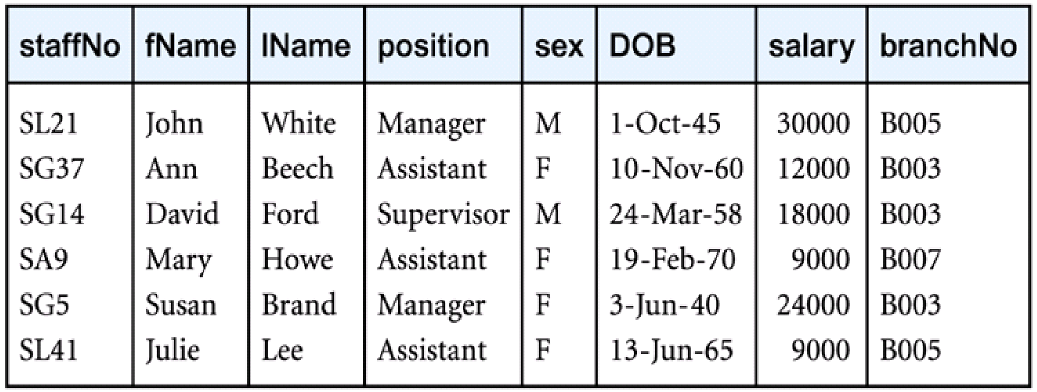
staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

- This update would fail: although update would cause row to disappear from HighSalary, row would not disappear from LowSalary.

What happens?

```
UPDATE Manager3Staff
SET    salary = 8000
WHERE  staffNo = 'SG37';
```

Staff



staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

- This update would succeed as row would no longer be part of LowSalary.

Example – WITH CASCADE CHECK OPTION

- Now consider the following:

① **CREATE VIEW LowSalary**

AS SELECT * FROM Staff WHERE **salary > 9000**;

② **CREATE VIEW HighSalary**

AS SELECT * FROM **LowSalary**

WHERE **salary > 10000**;

WITH CASCADE CHECK OPTION;

③ **CREATE VIEW** Manager3Staff

AS SELECT * FROM **HighSalary**

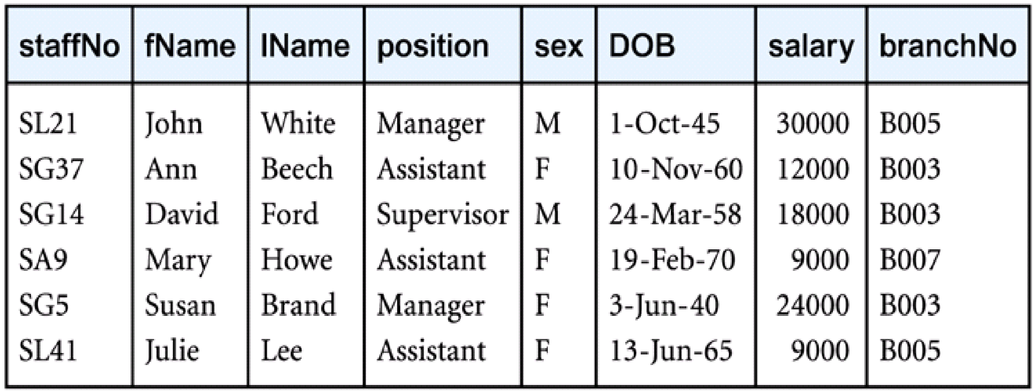
WHERE **branchNo = ‘B003’**;

What happens?

```
UPDATE Manager3Staff
SET        salary = 9500
WHERE  staffNo = ‘SG37’;
```

```
UPDATE Manager3Staff
SET        salary = 8000
WHERE  staffNo = ‘SG37’;
```

Staff



staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

- Setting salary to 9500 or 8000 would be rejected because row would disappear from HighSalary.
- To prevent anomalies like this, each view should be created using WITH CASCADE CHECK OPTION.

DROP VIEW

- Syntax:

```
DROP VIEW ViewName [RESTRICT | CASCADE]
```

- Causes definition of view to be deleted from database.
- For example:

```
DROP VIEW Manager3Staff;
```

- With **CASCADE**, all related dependent objects are deleted; i.e. any views defined on view being dropped.
- With **RESTRICT** (default), if any other objects depend for their existence on continued existence of view being dropped, command is rejected.

Agenda

1. Views
2. Creating and Dropping a View
3. Querying and Updating a View
4. Advantages and Disadvantages of Views
5. Integrity Constraints in SQL
6. Access Control

View Resolution

- With **view resolution**, any operations on view are automatically translated into operations on relations from which it is derived.
- Example: count number of properties managed by each member at branch B003.

SELECT staffNo, cnt

FROM StaffPropCnt

WHERE branchNo = ‘B003’

ORDER BY staffNo;

View Resolution

- The process of **view resolution** merges the query with the defining query of the StaffPropCnt view:

① The view column names in the **SELECT** list are translated into their corresponding column names in the defining query:

② View names in the **FROM** clause are replaced with the corresponding FROM lists of the defining query:

③ The **WHERE** clause from the user query is combined with the **WHERE** clause of the defining query using the logical operator **AND**:

④ The **GROUP BY** and **HAVING** clauses are copied from the defining query:

⑤ The **ORDER BY** clause is copied from the user query with the view column name translated into the defining query column name:

View Resolution

```
SELECT staffNo, cnt
FROM    StaffPropCnt
WHERE branchNo = ‘B003’
ORDER BY staffNo;
```

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
        AS   SELECT      s.branchNo, s.staffNo, COUNT(*)
             FROM          Staff s, PropertyForRent p
             WHERE       s.staffNo = p.staffNo
             GROUP BY s.branchNo, s.staffNo;
```

```
SELECT     s.staffNo AS staffNo, COUNT(*) AS cnt
FROM         Staff s, PropertyForRent p
WHERE      s.staffNo = p.staffNo AND branchNo = ‘B003’
GROUP BY s.branchNo, s.staffNo
ORDER BY s.staffNo;
```

Restrictions on View

- The ISO standard imposes several important restrictions on the creation and use of views, although there is considerable variation among dialects.

① If a column in the view is based on an aggregate function, the column may appear only in **SELECT** and **ORDER BY** clauses of queries that access the view. In particular , such a column may not be used in a **WHERE** clause and may not be argument to an aggregate function in any query based on the view.

Example 1: **SELECT COUNT** (cnt) **FROM** StaffPropCnt;

Example 2: **SELECT** * **FROM** StaffPropCnt **WHERE** cnt > 2;

② A grouped view may never be joined with a base table or a view.

Updatable View

- If view is updated then base table(s) will reflect change.

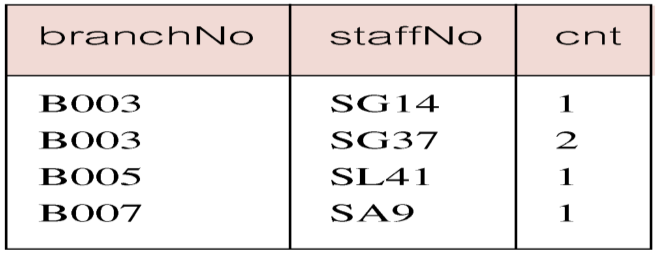
- For view to be updatable, DBMS must be able to trace any row or column back to its row or column in the source table.

Example 1 - Updatable View

- If we tried to insert record showing that at branch B003, SG5 manages 2 properties:

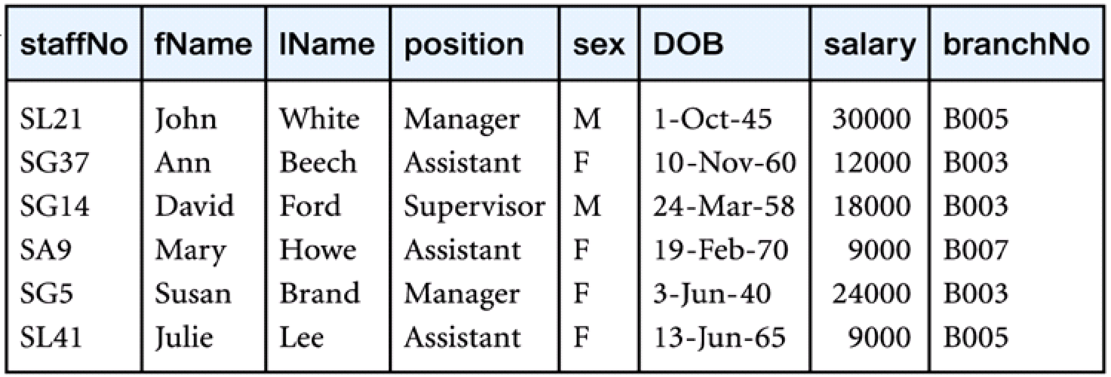
INSERT INTO StaffPropCnt
VALUES ('B003', 'SG5', 2);

Table 6.5 Data for view StaffPropCn



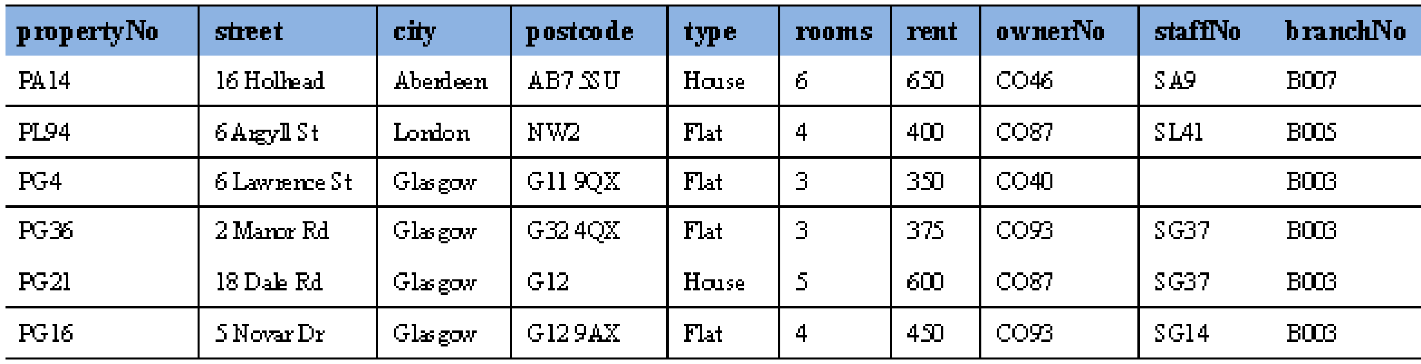
branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

Staff



staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

PropertyForRent



propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

?

Example 1 - Updatable View

- Have to insert 2 records into PropertyForRent showing which properties SG5 manages. However, do not know which properties they are; i.e. do not know primary keys.

Table 6.5 Data for view StaffPropCn

branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Example 2 - Updatable View

- **CREATE VIEW** StaffPropList (branchNo, staffNo, propertyNo)

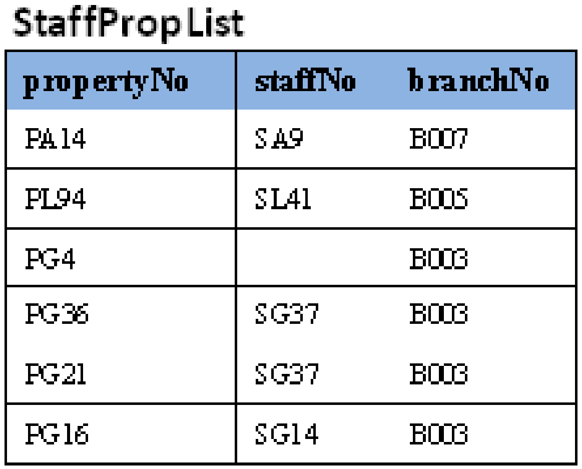
  AS SELECT branchNo, staffNo, propertyNo

  FROM PropertyForRent;

- Now try to insert the record:

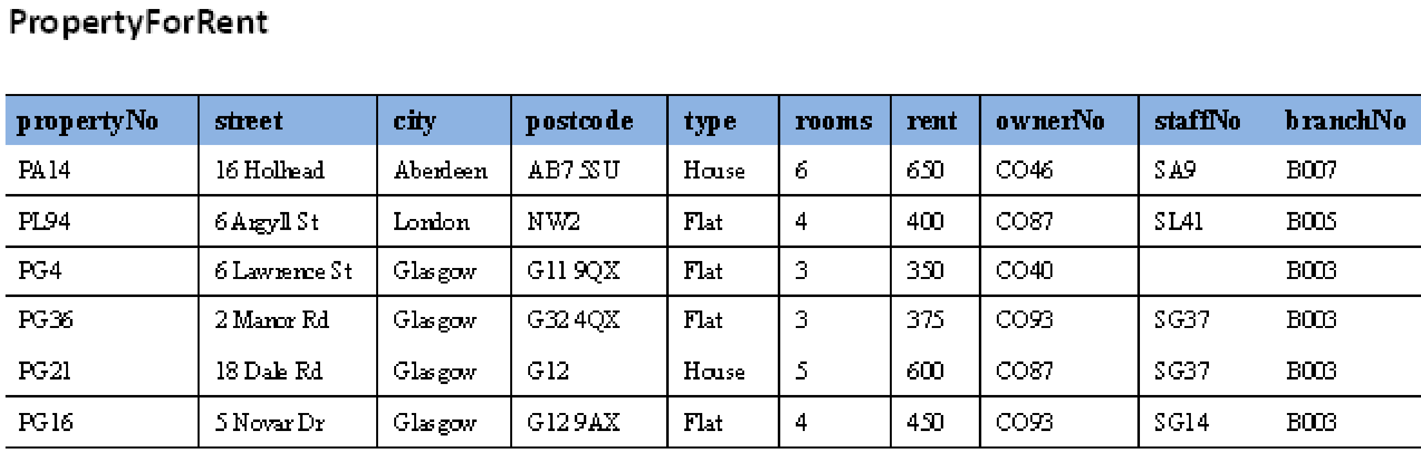
  INSERT INTO StaffPropList

  VALUES (‘B003’, ‘SG5’, ‘PG19’);


StaffPropList

propertyNo	staffNo	branchNo
PA14	SA9	B007
PL94	SL41	B005
PG4		B003
PG36	SG37	B003
PG21	SG37	B003
PG16	SG14	B003

?


PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Example 2 - Updatable View

- **CREATE TABLE** PropertyForRent (

```
propertyNo   PNumber        NOT NULL, ….
rooms        PRooms         NOT NULL        DEFAULT 4,
rent         PRent          NOT NULL,       DEFAULT 600,
ownerNo      OwnerNumber    NOT NULL,
staffNo      StaffNumber    Constraint StaffNotHandlingTooMuch ….
branchNo     BranchNumber   NOT NULL,
PRIMARY KEY (propertyNo),
FOREIGN KEY (staffNo) REFERENCES Staff
  ON DELETE SET NULL ON UPDATE CASCADE ….);
```

In PropertyForRent all columns except postcode/staffNo are not allowed nulls. However, have no way of giving remaining non-null columns values.

View Updatability

- ISO specifies that a view is updatable if and only if:

①**DISTINCT** is not specified.

②Every element in **SELECT** list of defining query is a column name and no column appears more than once.

③**FROM** clause specifies only one table, excluding any views based on a join, union, intersection or difference.

④**No nested SELECT** referencing outer table.

⑤**No GROUP BY or HAVING** clause.

⑥Every row added through view must not violate integrity constraints of base table.

Agenda

Advantages of Views

① Data independence

② Currency

③ Improved security

④ Reduced complexity

⑤ Convenience

⑥ Customization

⑦ Data integrity

Disadvantages of Views

① Update restriction

② Structure restriction

③ Performance

View Materialization

- With **view materialization**, the view is stored as a **temporary table**, which is maintained as the underlying base tables are updated.
  - View materialization stores view as temporary table when view is first queried.
  - Thereafter, queries based on materialized view can be faster than recomputing view each time.
- Difficulty is maintaining the currency of view while base tables(s) are being updated.

View Materialization

- **View maintenance** aims to apply only those changes necessary to keep view current.
- Consider following view:

```
CREATE VIEW StaffPropRent(staffNo)
AS SELECT DISTINCT staffNo
     FROM PropertyForRent
     WHERE branchNo = 'B003' AND
           rent > 400;
```

Table 6.8 Data for view StaffPropRent.

staffNo
SG37
SG14

Example--View Materialization

- Decide whether to insert the row into the materialized view
  - If insert a row into PropertyForRent with **rent ≤400**, then view would be unchanged
  - If insert a row for property PG24 at branch B003 with **staffNo = SG19** and **rent = 550**, then row would appear in materialized view.
  - If insert a row for property PG54 at branch B003 with **staffNo = SG37** and **rent = 450**, then no new row would need to be added to materialized view.

Table 6.8 Data for view StaffPropRent.

staffNo
SG37
SG14

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Example--View Materialization

Table 6.8 Data for view StaffPropRent.

staffNo
SG37
SG14

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

StaffPropRent

staffNo
SG37
SG14
SG19

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003
PG24						550		SG19	B003
PG54						450		SG37	B003

Example-- View Materialization

- Decide whether to delete or retain the row in the materialized view.
  - If delete property PG24, then the row should be deleted from materialized view.
  - If delete property PG54, then row for SG37 should not be deleted (because of existing property PG21).

StaffPropRent

staffNo
SG37
SG14
SG19

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003
PG24						550		SG19	B003
PG54						450		SG37	B003

Agenda

Integrity Enhancement Feature

- Consider five types of integrity constraints:

  ① Required data

  Example: position **VARCHAR(10) NOT NULL**

  ② Domain constraints

  ③ Entity integrity

  ④ Referential integrity

  ⑤ General constraints.

② Domain Constraints

- **CHECK**

```
sex  CHAR  NOT NULL
     CHECK (sex IN (‘M’, ‘F’))
```

- **CREATE DOMAIN**

```
CREATE DOMAIN DomainName [AS] dataType
[DEFAULT defaultOption]
[CHECK (searchCondition)]
```

For example:

```
CREATE DOMAIN SexType AS CHAR
          CHECK (VALUE IN (‘M’, ‘F’));
sex       SexTypeNOT NULL
```

②Domain Constraints

- ***searchCondition*** can involve a table lookup:

```
CREATE DOMAIN BranchNo AS CHAR(4)
CHECK (VALUE IN (SELECT branchNo
                        FROM Branch));
```

- Domains can be removed using DROP DOMAIN:

```
DROP DOMAIN DomainName
  [RESTRICT | CASCADE]
```

③ *Entity Integrity*

- Primary key of a table must contain a unique, non-null value for each row.
- ISO standard supports **PRIMARY KEY** clause in CREATE and ALTER TABLE statements:

```
PRIMARY KEY(staffNo)

PRIMARY KEY(clientNo, propertyNo)
```

- Can only have one PRIMARY KEY clause per table.
- Can still ensure uniqueness for alternate keys using **UNIQUE**:

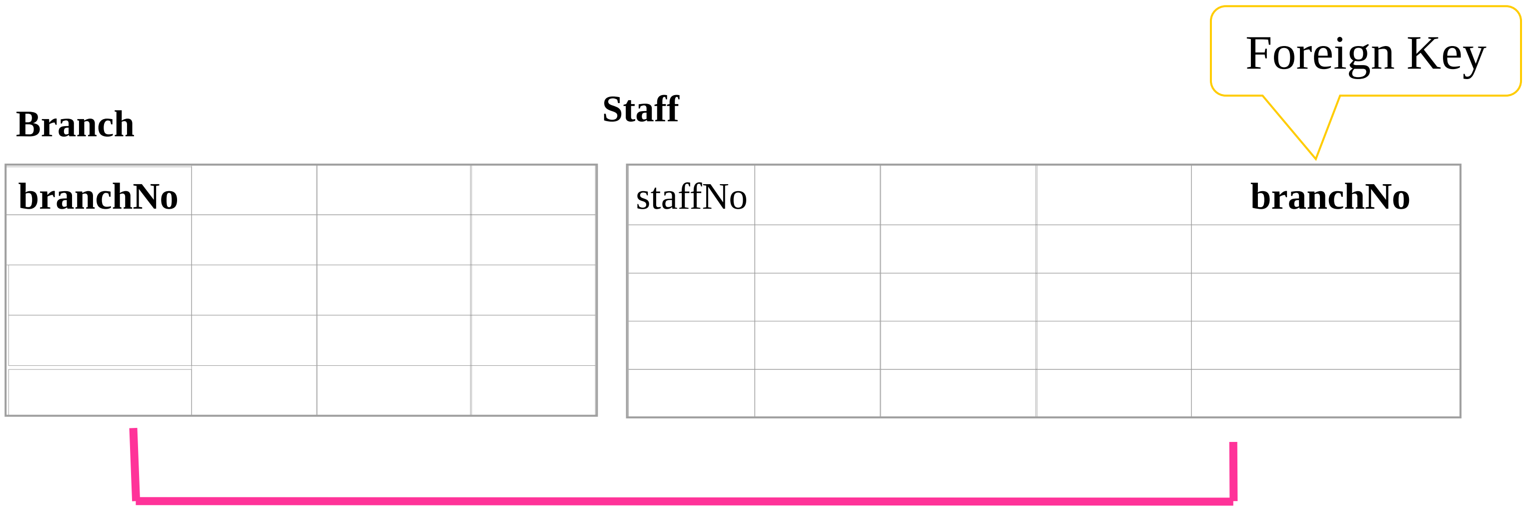
```
UNIQUE(telNo)
```

④Referential Integrity

- ISO standard supports definition of FKs with FOREIGN KEY clause in CREATE and ALTER TABLE Staff:

  FOREIGN KEY(branchNo) **REFERENCES** Branch

- The modify operations might violate referential integrity:
  - **INSERT/UPDATE** in **child table Staff**
  - **UPDATE/DELETE** in **parent table Branch**



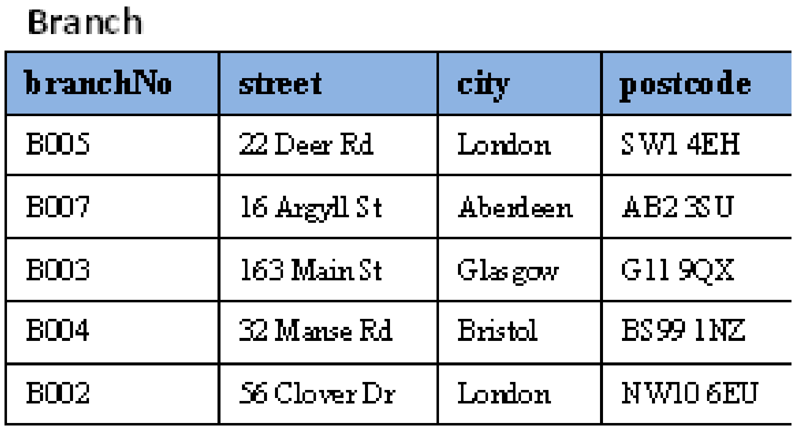
④Referential Integrity

- Example: Change BranchNo ‘B003’ to ‘B010’.

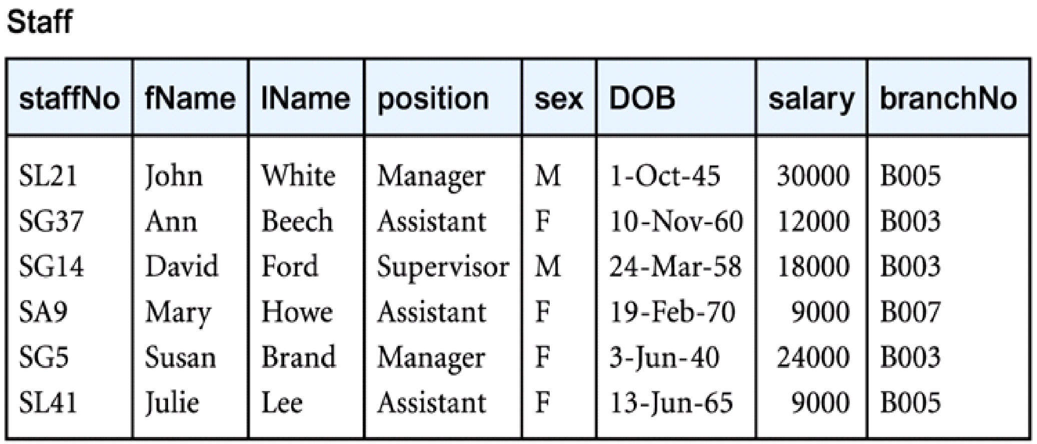
```
UPDATE  Branch
SET        branchNo = ‘B010’
WHERE  branch = ‘B003’;
```

```
UPDATE  Staff
SET        branchNo = ‘B010’
WHERE  branch = ‘B003’;
```

- What happens?


Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU


Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

④ *Referential Integrity*

- **Solution:** Extend the FK's definition
- **Different actions** to the child table and the parent table while modification invalids the referential integrity
  - Any **INSERT/UPDATE** attempting to create FK value in **child table** without matching CK value in parent is **rejected**.
  - Action taken attempting to **UPDATE/DELETE a CK** value in **parent table** with matching rows in child is dependent on <u>referential action</u> specified using ON UPDATE and ON DELETE subclauses:

    ① **CASCADE**

    ② **SET NULL**

    ③ **SET DEFAULT**

    ④ **NO ACTION**

④Referential Integrity

- **Syntax:**

  FOREIGN KEY(listOfForeignKeyColumns)
  REFERENCES ParentTable Name[(listOfCandidateKeyColumns)]
  [ON UPDATE referentialAction**]** **[ON DELETE** referentialAction**]**

- **Referential Actions**

  - **CASCADE**: Delete row from parent and delete matching rows in child, and so on in cascading manner.
  - **SET NULL**: Delete row from parent and set FK column(s) in child to NULL. Only valid if FK columns are NOT NULL.
  - **SET DEFAULT**: Delete row from parent and set each component of FK in child to specified default. Only valid if DEFAULT specified for FK columns.
  - **NO ACTION**: Reject delete from parent. Default.

④Referential Integrity

- For example: define a foreign key branchNo in table Staff.
  - **FOREIGN KEY** (branchNo) **REFERENCES** Branch
    ON UPDATE CASCADE ON DELETE SET NULL
- What happens for the following statements:

```
DELETE
FROM    Staff
WHERE  branchNo='B005';
```

```
DELETE
FROM    Branch
WHERE  branchNo='B005';
```

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

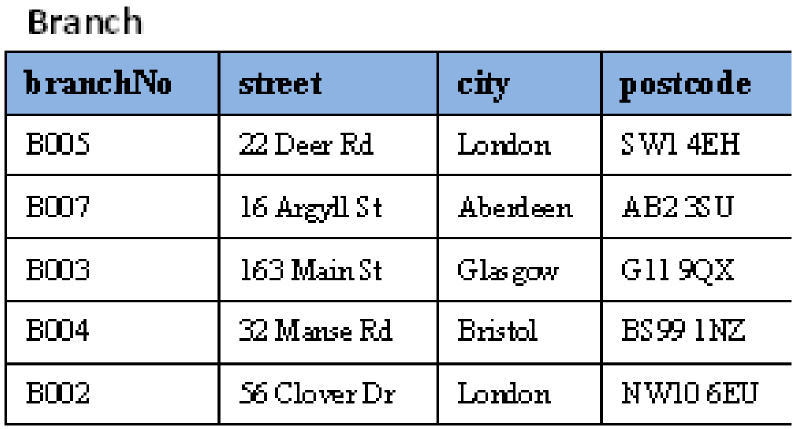
staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

④Referential Integrity

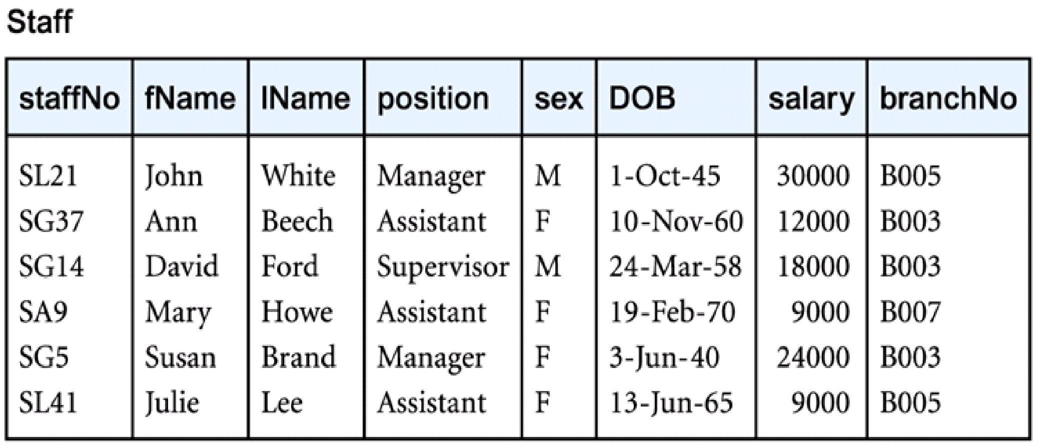
- Example: Change BranchNo ‘B003’ to ‘B010’.

```
UPDATE  Branch
SET     branchNo = ‘B010’
WHERE  branch = ‘B003’;
```

- What happens?


Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU


Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

⑤ General Constraints

- Could use CHECK/UNIQUE in CREATE and ALTER TABLE.
- Similar to the CHECK clause, also have:

  CREATE ASSERTION AssertionName
  CHECK (searchCondition)

- Example:

```
CREATE ASSERTION StaffNotHandlingTooMuch
CHECK (NOT EXISTS  (SELECT *
                    FROM PropertyForRent
                    GROUP BY staffNo
                    HAVING COUNT(*) > 100))
```

Transaction

- A **transaction** is a logical unit of work consisting of one or more SQL statements that is guaranteed to be **atomic** with respect to **recovery**.
- The ISO standard defines a transaction model based on two statements:
  - A **COMMIT** statement ends the transaction successfully, making the database changes permanent.
  - A **ROLLBACK** statement abort the transaction, backing out all changed made by the transaction.

Immediate and Deferred Integrity Constraints

- In some situations, we do not want integrity constraints to be checked immediately, that is after every SQL statement has been executed, but instead at transaction commit.
- The **SET CONSTRAINTS** statement is to set the mode for specified constraints for the current transaction.
- The format of the statement is:

SET CONSTRAINTS

{**ALL** | constraintName [,…]} {**DEFERRED** | **IMMEDIATE**}

- The default mode is **IMMEDIATE**.

Agenda

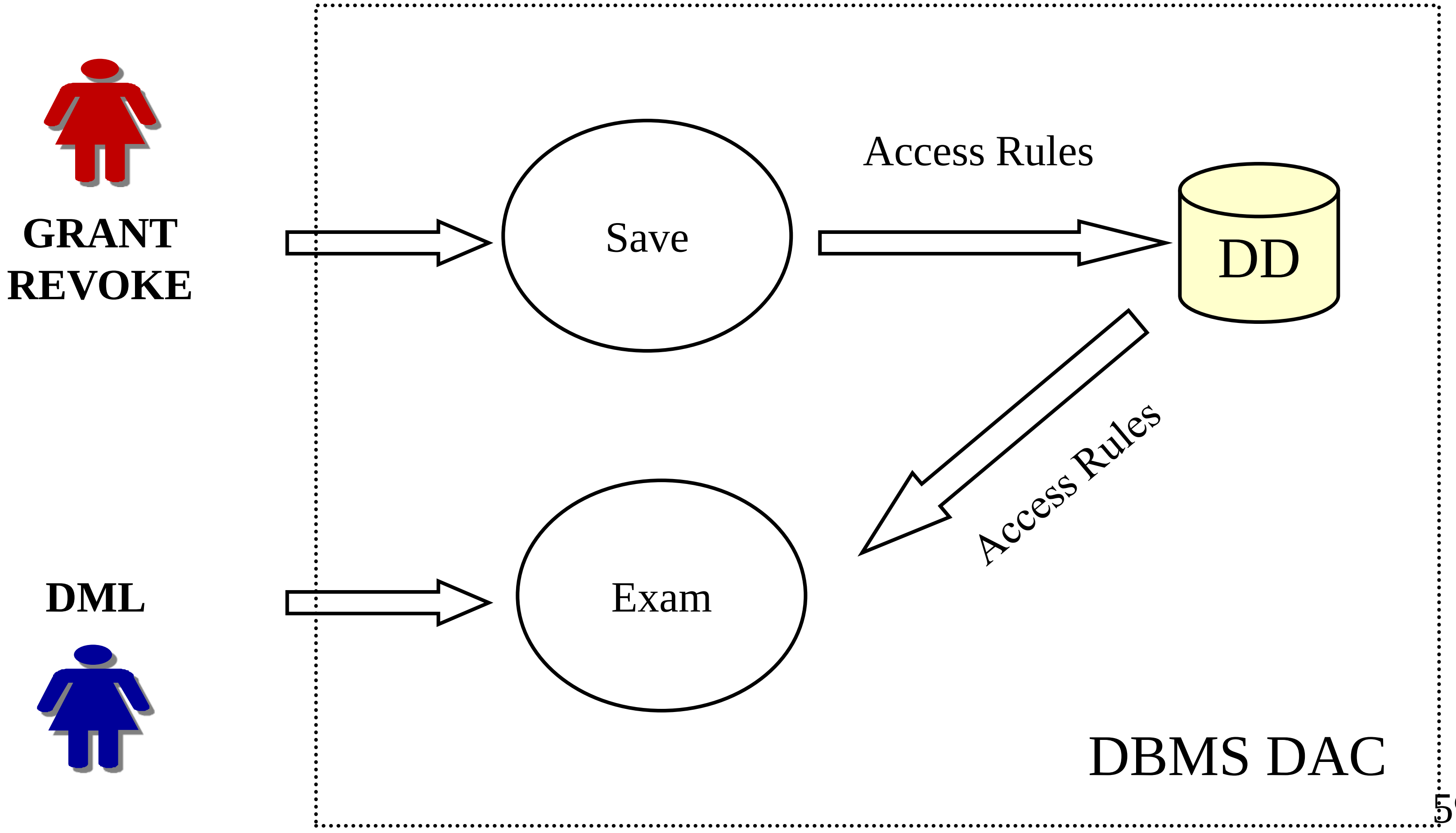
1. Views
2. Creating and Dropping a View
3. Querying and Updating a View
4. Advantages and Disadvantages of Views
5. Integrity Constraints in SQL
6. Access Control

Access Control

- Modern DBMSs typically provide one or both of the following authorization mechanisms:
  - Discretionary Access Control（**DAC**): each user is given appropriate access rights (or *privileges*) on specific database objects.
  - Mandatory Access Control (**MAC**): each database object is assigned a certain *classification level* and each subject is given a designed *clearance level*.

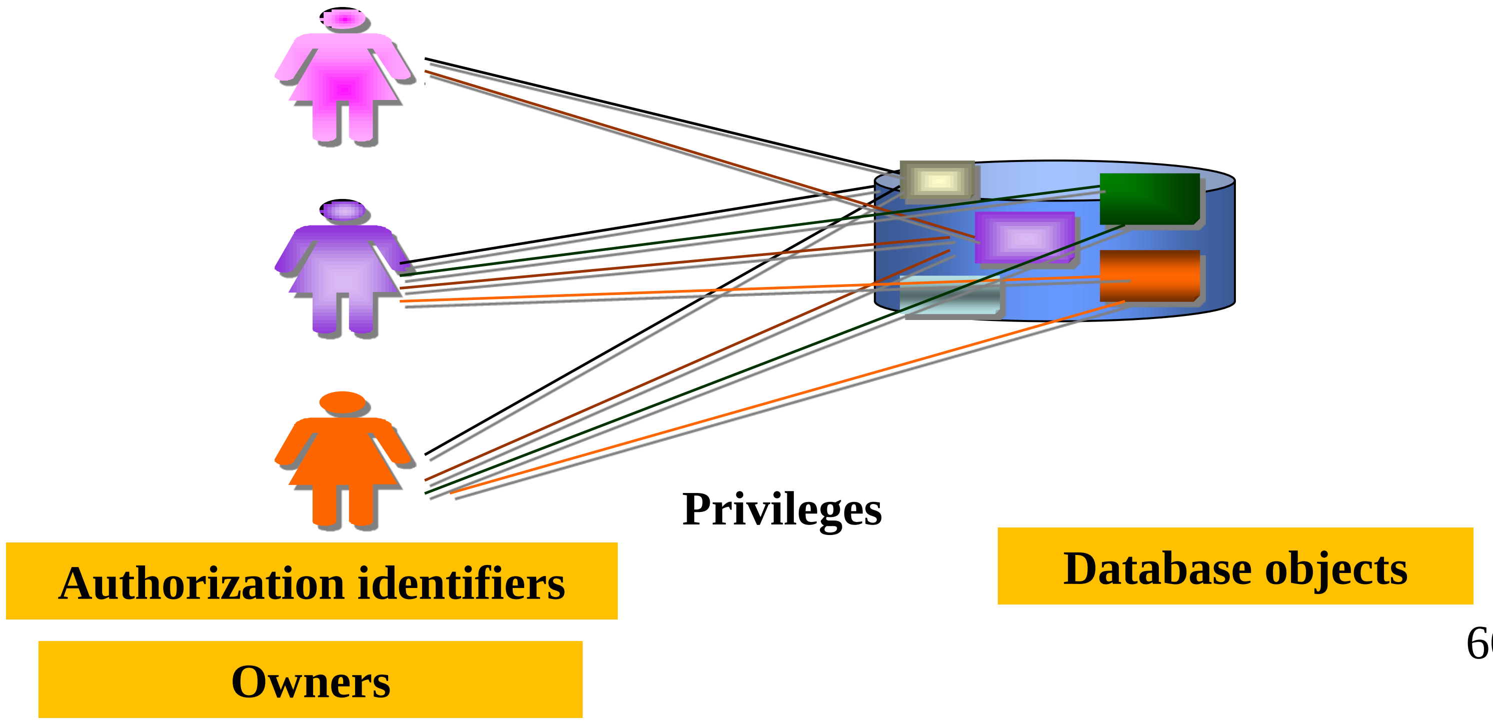
Discretionary Access Control

- **SQL** supports only discretionary access control(**DAC)** through the **GRANT** and **REVOKE** statements.



Discretionary Access Control

- The mechanism is based on the concepts of
  - **Authorization identifiers**
  - **Ownership**
  - **Privileges**.



Authorization Identifiers and Ownership

- Authorization identifier is normal SQL identifier used to establish identity of a user.
  - Usually has an associated password.
  - Used to determine which objects user may reference and what operations may be performed on those objects.
- Each object created in SQL has an owner, as defined in AUTHORIZATION clause of schema to which object belongs.
  - The owner is initially the only person who may know of the existence of the object and, consequently, perform any operations on the object.

Privileges

- Actions user permitted to carry out on given base table or view:

SELECT	retrieve data from a table.
DELETE	delete rows of data from a table.
INSERT	insert new rows into a table.
UPDATE	modify rows of data in a table.
REFERENCES	reference columns of named table in integrity constraints.
USAGE	use domains, collations, character sets, and translations.

Privileges

- **Owner** of table must grant other users the necessary privileges using GRANT statement.
- To **create view**, user must have **SELECT privilege** on all tables that make up view and **REFERENCES privilege** on the named columns.

GRANT

- The **GRANT** statement syntax:

  GRANT {*PrivilegeList* | **ALL PRIVILEGES**}

  ON *ObjectName*

  TO {AuthorizationIdList | **PUBLIC**}

  [**WITH GRANT OPTION**]

- **ALL PRIVILEGES** grants all six privileges to a user.
- *ObjectName* can be a base table, view, domain, character set, collation or translation.
- **PUBLIC** allows access to be granted to all present and future authorized users.
- **WITH GRANT OPTION** allows privileges to be passed on.

GRANT

- *PrivilegeList* consists of one or more of the following privileges separated by commas.
  - ① **SELECT**
  - ② **DELETE**
  - ③ **INSERT** [(columnName[,…])]
  - ④ **UPDATE** [(columnName[,…])]
  - ⑤ **REFERENCES** [(columnName[,…])]
  - ⑥ **USAGE**
- GRANT can restrict INSERT/UPDATE/REFERENCES to named columns.

Example 6.7/8 - GRANT

- Give Manager full privileges to Staff table.

```
GRANT ALL PRIVILEGES
ON        Staff
TO        Manager WITH GRANT OPTION;
```

- Give users Personnel and Director SELECT and UPDATE on column salary of Staff.

```
GRANT SELECT, UPDATE (salary)
ON        Staff
TO        Personnel, Director;
```

Example 6.9 - GRANT Specific Privileges to PUBLIC

- Give all users SELECT on Branch table.

```
GRANT  SELECT
ON     Branch
TO     PUBLIC;
```

Restrict SELECT to Named Columns

- Give all users SELECT on column staffNo, fName, lName of the Staff table.

```
GRANT  SELECT
ON          Staff (staffNo, fName, lName )
TO          PUBLIC;
```

?

```
CREATE VIEW Staff_list
 AS
   SELECT staffNo, fName, lName
   FROM    Staff;
```

```
GRANT  SELECT
 ON         Staff_list
 TO         PUBLIC;
```

Restrict SELECT to Specific Rows

- Give all users SELECT on the Staff table, but only to the staff who works at branch ‘B003’.

```
CREATE VIEW Branch3Staff
 AS
   SELECT *
   FROM    Staff;
```

```
GRANT  SELECT
 ON       Branch3Staff
 TO       PUBLIC;
```

REVOKE

- REVOKE takes away privileges granted with GRANT.

REVOKE [GRANT OPTION FOR]

{PrivilegeList | ALL PRIVILEGES}

ON ObjectName

FROM {AuthorizationIdList | PUBLIC}

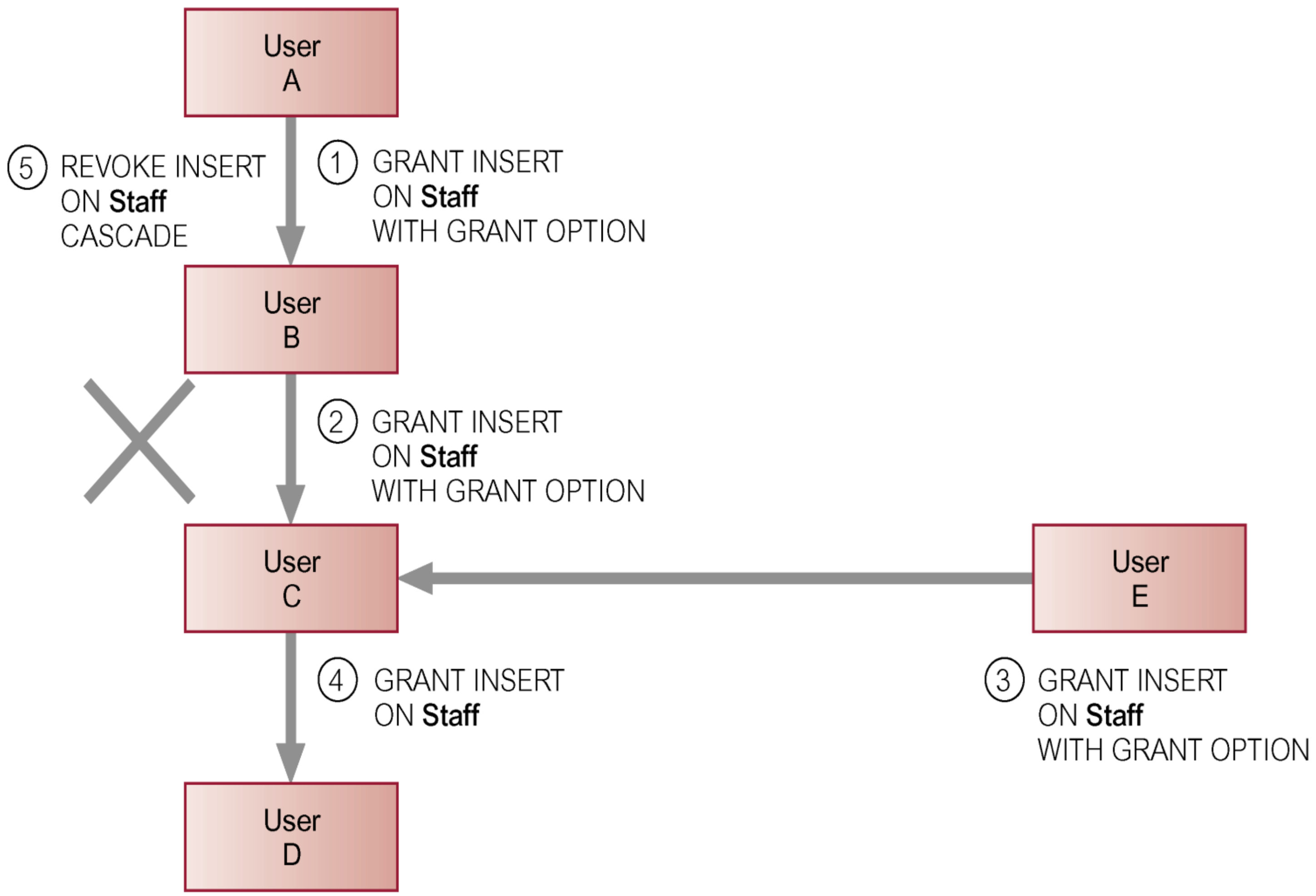
[**RESTRICT | CASCADE**]

- **ALL PRIVILEGES** refers to all privileges granted to a user by user revoking privileges.

REVOKE

- GRANT OPTION FOR allows privileges passed on via WITH GRANT OPTION of GRANT to be revoked separately from the privileges themselves.
- Privileges granted to this user by other users are not affected by this REVOKE statement.

REVOKE


User A
⑤ REVOKE INSERT ON **Staff** CASCADE
① GRANT INSERT ON **Staff** WITH GRANT OPTION
User B
② GRANT INSERT ON **Staff** WITH GRANT OPTION
User C
User E
④ GRANT INSERT ON **Staff**
③ GRANT INSERT ON **Staff** WITH GRANT OPTION
User D

Example 6.10/11 - REVOKE Specific Privileges

- Revoke privilege SELECT on Branch table from all users.

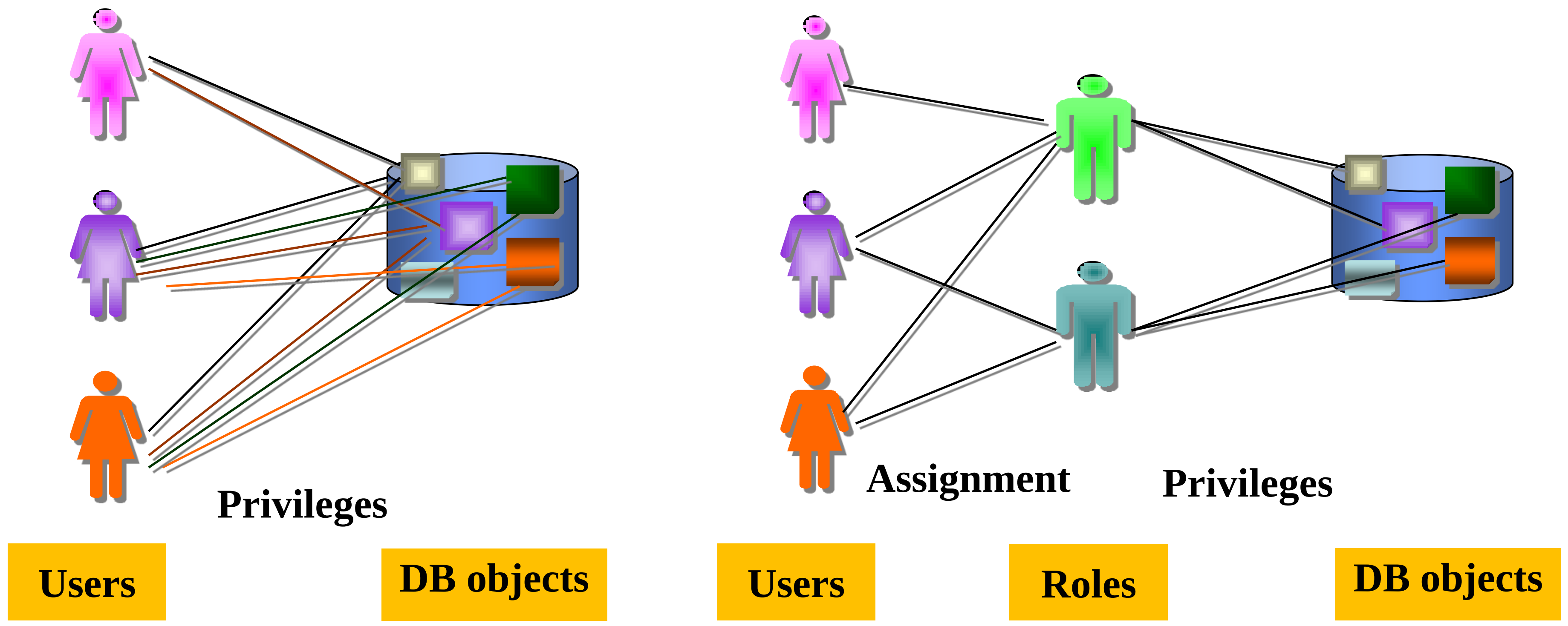
```
REVOKE SELECT
ON        Branch
FROM      PUBLIC;
```

- Revoke all privileges given to Director on Staff table.

```
REVOKE ALL PRIVILEGES
ON        Staff
FROM      Director;
```

Role Based Access Control (RBAC)

- Role based access control (RBAC) is a method of access security that is based on a person’s role within a business.
- A role is a set of privileges.



Section Objectives

In this section you will learn:

① Purpose of views.

② How to create and drop views using SQL.

③ How the DBMS performs operations on views.

④ Under what conditions views are updatable.

⑤ Advantages and disadvantages of views.

⑥ Purpose of integrity enhancement feature of SQL.

⑦ How to define integrity constraints using SQL.

⑧ How to use the integrity enhancement feature in the CREATE and ALTER TABLE statements.

⑨ How to use the GRANT and REVOKE statements as a level of security.

Questions?

Assignments

- **Exercise 3: part Ⅲ**

Prerequisites for Next Section

- **Readings:**
  - **Required:** Connolly and Begg, sections E.1.1 and E.2 (in third edition, sections 21.1.1, 21.2.1, and 21.2.2).
- **Assessments:**
  - Multiple-Choice Quiz 4