WITH CHECK OPTION

- Rows exist in a view because they satisfy WHERE condition of defining query.
- If a row changes and no longer satisfies condition, it disappears from the view.
- 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 prohibits a row migrating out of the view.

WITH CHECK OPTION

- LOCAL/CASCADED 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 CASCADED (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 7.6 - WITH CHECK OPTION

```
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.

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 7.6 - WITH CHECK OPTION

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

- Also cannot insert a row into view with a branch number that does not equal B003.

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 7.6 - WITH CHECK OPTION

- Now consider the following:

```
CREATE VIEW LowSalary
 AS  SELECT * FROM Staff WHERE salary > 9000;
CREATE VIEW HighSalary
  AS  SELECT * FROM LowSalary
              WHERE salary > 10000
  WITH LOCAL CHECK OPTION;
CREATE VIEW Manager3Staff
  AS SELECT * FROM HighSalary
              WHERE branchNo = 'B003';
```

Example 7.6 - WITH CHECK OPTION

```
UPDATE Manager3Staff
SET salary = 9500
WHERE staffNo = 'SG37';
```

- This update would fail: although update would cause row to disappear from HighSalary, row would not disappear from LowSalary.
- However, if update tried to set salary to 8000, update would succeed as row would no longer be part of LowSalary.

Example 7.6 - WITH CHECK OPTION

- If HighSalary had specified WITH CASCADED CHECK OPTION, 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 CASCADED CHECK OPTION.

Advantages of Views

- Data independence
- Currency
- Improved security
- Reduced complexity
- Convenience
- Customization
- Data integrity

Disadvantages of Views

- Update restriction
- Structure restriction
- Performance

View Materialization

- View resolution mechanism may be slow, particularly if view is accessed frequently.
- 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 Maintenance

- 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;
```

staffNo
SG37
SG14

View Materialization

- If insert row into PropertyForRent with rent ≤400 then view would be unchanged.
- If insert row for property PG24 at branch B003 with staffNo = SG19 and rent = 550, then row would appear in materialized view.
- If insert 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.
- If delete property PG24, row should be deleted from materialized view.
- If delete property PG54, then row for PG37 should not be deleted (because of existing property PG21).

Transactions

- SQL defines transaction model based on COMMIT and ROLLBACK.
- Transaction is logical unit of work with one or more SQL statements guaranteed to be atomic with respect to recovery.
- An SQL transaction automatically begins with a *transaction-initiating* SQL statement (e.g., SELECT, INSERT).
- Changes made by transaction are not visible to other concurrently executing transactions until transaction completes.

Transactions

- Transaction can complete in one of four ways:

  - COMMIT ends transaction successfully, making changes permanent.

  - ROLLBACK aborts transaction, backing out any changes made by transaction.

  - For programmatic SQL, successful program termination ends final transaction successfully, even if COMMIT has not been executed.

  - For programmatic SQL, abnormal program end aborts transaction.

Immediate and Deferred Integrity Constraints

- Do not always want constraints to be checked immediately, but instead at transaction commit.
- Constraint may be defined as INITIALLY IMMEDIATE or INITIALLY DEFERRED, indicating mode the constraint assumes at start of each transaction.
- In former case, also possible to specify whether mode can be changed subsequently using qualifier [NOT] DEFERRABLE.
- Default mode is INITIALLY IMMEDIATE.

Immediate and Deferred Integrity Constraints

- **SET CONSTRAINTS statement used to set mode for specified constraints for current transaction:**

SET CONSTRAINTS

{ALL | constraintName [, . . .]}

{DEFERRED ¦ IMMEDIATE}

Access Control - 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.
- Owner is only person who may know about it.

Privileges

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

SELECT	Retrieve data from a table.
INSERT	Insert new rows into a table.
UPDATE	Modify rows of data in a table.
DELETE	Delete rows of data from a table.
REFERENCES	Reference columns of named table in integrity constraints.
USAGE	Use domains, collations, character sets, and translations.

Privileges

- Can restrict INSERT/UPDATE/REFERENCES to named columns.
- 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

```
GRANT   {PrivilegeList | ALL PRIVILEGES}
ON      ObjectName
TO      {AuthorizationIdList | PUBLIC}
[WITH GRANT OPTION]
```

- *PrivilegeList* consists of one or more of above privileges separated by commas.
- ALL PRIVILEGES grants all privileges to a user.

GRANT

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

Example 7.7 - GRANT

Give the user with authorization identifier Manager all privileges on the Staff table.

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

Example 7.8 - GRANT

Give users Personnel and Director the privileges SELECT and UPDATE on column salary of the Staff table.

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

Example 7.9 - GRANT Specific Privileges to PUBLIC

Give all users the privilege SELECT on the Branch table.

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

REVOKE

- REVOKE takes away privileges granted with GRANT.

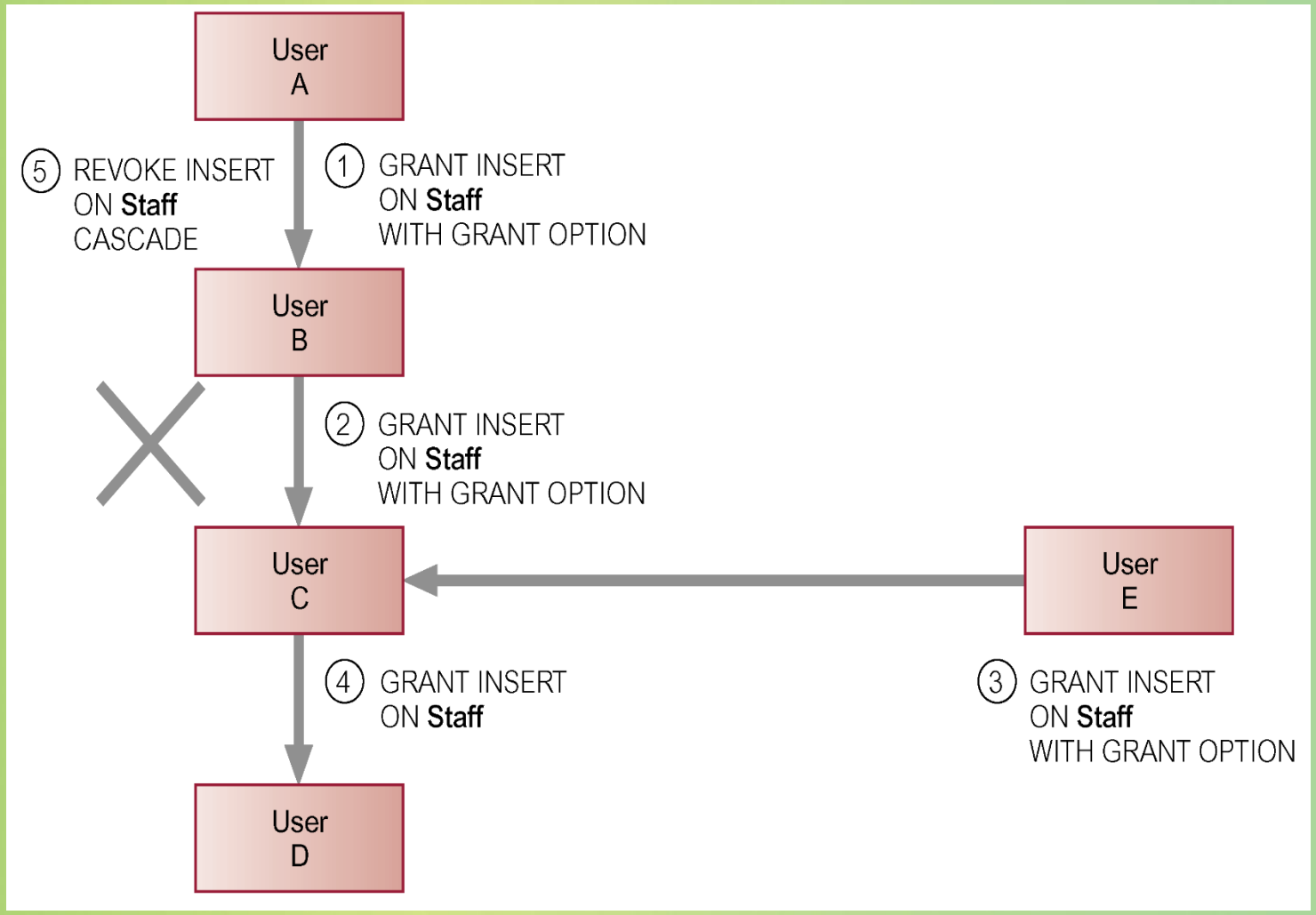
```
REVOKE [GRANT OPTION FOR]
        {PrivilegeList | ALL PRIVILEGES}
ON ObjectName
FROM {AuthorizationIdList | PUBLIC}
        [RESTRICT | CASCADE]
```

- 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.
- REVOKE fails if it results in an abandoned object, such as a view, unless the CASCADE keyword has been specified.
- Privileges granted to this user by other users are not affected.

REVOKE



Example 7.10/11 - 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;
```