SQL: Data Manipulation & Data Definition

Part two

Database Management - CIS 386 01 FA17

By: Dr. Aos Mulahuwaish

Subqueries

- Subqueries are where the results of an inner SELECT (or subselect) are used in the outer statement to help determine the contents of the final result

- Subselects typically appear in WHERE or HAVING clauses of SELECT statements, but can also be used in INSERT, UPDATE and DELETE statements

Subquery Rules

1. The ORDER BY clause may not be used in a subquery (although it may be used in the outermost SELECT statement)
2. The subquery SELECT list must consist of a single column name or expression, except for subqueries that use the keyword EXISTS
3. By default, column names in a subquery refer to the table name in the FROM clause of the subquery. It is possible to refer to a table in a FROM clause of an outer query by qualifying the column name.
4. When a subquery is one of the two operands involved in a comparison, the subquery must appear on the right-hand side of the comparison.

Subquery Example 1

Query - TURDY.CIS486.TURDY\Scott D. James - Untitled1*

```
USE [CIS486]
GO

/* Using a subquery with equality - List the staff who work at the branch at '163 Main Street' */


SELECT StaffNumber, FirstName, LastName, Position
FROM Staff_Table
WHERE BranchNumber = (SELECT BranchNumber
                      FROM Branch_Table
                      WHERE Street = '163 Main Street');
```

	StaffNumber	FirstName	LastName	Position
1	SG14	David	Ford	Supervisor
2	SG37	Ann	Beech	Assistant
3	SG5	Susan	Brand	Manager

Grids | Messages

Query batch completed. | TURDY (8.0) | TURDY\Scott D. James (52) | CIS486 | 0:00:00 | 3 rows | Ln 4, Col 94

Subquery Example 2

Query - TURDY.CIS486.TURDY\Scott D. James - Untitled1*

```
USE [CIS486]
GO

/* Using a subquery with an aggregate function - List all staff whose salary is greater than */
/* the average salaray, and show by how much their salary is greater than the average        */


SELECT StaffNumber, FirstName, LastName, Position, Salary - (SELECT AVG(Salary) FROM Staff_Table)
AS DifferenceInSalary
FROM Staff_Table
WHERE Salary > (SELECT AVG(Salary) FROM Staff_Table);
```

	StaffNumber	FirstName	LastName	Position	DifferenceInSalary
1	SG14	David	Ford	Supervisor	1000.000000
2	SG5	Susan	Brand	Manager	7000.000000
3	SL21	John	White	Manager	13000.000000

Grids | Messages

Query batch completed. | TURDY (8.0) | TURDY\Scott D. James (52) | CIS486 | 0:00:00 | 3 rows | Ln 11, Col 52

Subquery Example 3

Query - TURDY.CIS486.TURDY\Scott D. James - Untitled1*

```
/* Nested Subqueries - Using IN: List the properties that are handled by staff who work at the */
/*                               branch located at 163 Main Street                              */

SELECT PropertyNumber, Street, City, ZipCode, RentType, Rooms, MonthlyRent
FROM PropertyForRent_Table
WHERE StaffNumber IN (SELECT StaffNumber
                      FROM Staff_Table
                      WHERE BranchNumber = (SELECT BranchNumber
                                            FROM Branch_Table
                                            WHERE Street = '163 Main Street'));
```

	PropertyNumber	Street	City	ZipCode	RentType	Rooms	MonthlyRent
1	PG16	5 Novar Drive	Frankenmuth	48734	Apartment	4	450.00
2	PG21	18 Dale Road	Frankenmuth	48734	House	5	600.00
3	PG36	2 Manor Road	Frankenmuth	48734	Apartment	3	375.00

Grids | Messages

Query batch completed. | TURDY (8.0) | TURDY\Scott D. James (52) | CIS486 | 0:00:00 | 3 rows | Ln 7, Col 51

ANY/SOME and ALL

- ANY/SOME and ALL can be used in subqueries that product a single column of numbers
- If a subquery is prefaced with ALL, the condition will only be true if it is satisfied by all values produced by the subquery
- If ANY (or SOME) is used in a subquery, the condition will be true if it is satisfied by one or more values

ANY/SOME and ALL Example 1

Query - TURDY.CIS486.TURDY\Scott D. James - Untitled1*

```
USE [CIS486]
GO

/* Use of ANY/SOME -- Find all staff whose salary is larager than the salary of at least one */
/*                    staff member at branch B003                                             */

SELECT StaffNumber, FirstName, LastName, Position, Salary
FROM Staff_Table
WHERE Salary > SOME (SELECT Salary
                     FROM Staff_Table
                     WHERE BranchNumber = 'B003');
```

	StaffNumber	FirstName	LastName	Position	Salary
1	SG14	David	Ford	Supervisor	18000.00
2	SG5	Susan	Brand	Manager	24000.00
3	SL21	John	White	Manager	30000.00

Grids | Messages

Query batch completed. | TURDY (8.0) | TURDY\Scott D. James (52) | CIS486 | 0:00:00 | 3 rows | Ln 11, Col 51

ANY/SOME and ALL Example 2

Query - TURDY.CIS486.TURDY\Scott D. James - Untitled1*

```
USE [CIS486]
GO

/* Use of ALL -- Find all staff whose salary is larager than the salary of every */
/*               staff member at branch B003                                      */

SELECT StaffNumber, FirstName, LastName, Position, Salary
FROM Staff_Table
WHERE Salary > ALL (SELECT Salary
                    FROM Staff_Table
                    WHERE BranchNumber = 'B003');
```

	StaffNumber	FirstName	LastName	Position	Salary
1	SL21	John	White	Manager	30000.00

Grids | Messages

Query batch completed. | TURDY (8.0) | TURDY\Scott D. James (52) | CIS486 | 0:00:00 | 1 rows | Ln 11, Col 50

EXISTS and NOT EXISTS

- For use only in subqueries
- EXISTS is true if there is at least one row returned by the subquery
- NOT EXISTS works in the opposite manner of EXISTS

EXISTS Example 1

Query - TURDY.CIS486.TURDY\Scott D. James - Untitled1*

```
/* EXISTS Example - Find all staff who work in a Saginaw branch */

SELECT StaffNumber, FirstName, LastName, Position
FROM Staff_Table s
WHERE EXISTS(SELECT *
             FROM Branch_Table b
             WHERE s.BranchNumber = b.BranchNumber AND City = 'Saginaw');

/* or:   SELECT StaffNumber, FirstName, LastName, Position
         FROM Staff_Table s, Branch_Table b
         WHERE s.BranchNumber = b.BranchNumber AND City = 'Saginaw'; */
```

	StaffNumber	FirstName	LastName	Position
1	SL21	John	White	Manager
2	SL41	Julie	Lee	Assistant

Grids | Messages

Query batch completed. | TURDY (8.0) | TURDY\Scott D. James (52) | CIS486 | 0:00:00 | 2 rows | Ln 12, Col 1

SQL Data Definition Language

- In this section we will be examining how to manipulate the high-level database entities using SQL. Objects that will be manipulated include:
  - **Tables**
  - **Columns**
  - **Indexes**
  - **Constraints**
  - **Views**
- We won't have as many detailed examples in this section since many DBs vary from the DDL standard

SQL Identifiers

- **Identifiers:** SQL identifiers are used to identify objects in the database, such as table names, view names, and columns. The characters that can be used in a user-defined SQL identifier must appear in **character set.** The following restrictions are imposed on an indenter:
  - Must start with a letter
  - Cannot contain spaces
  - Can be no longer than 128 characters
  - Can use A-Z, a-z and _ (underscore)

SQL Data Types

- **Boolean** – BOOLEAN (TRUE, FALSE or UNKNOWN as the null value)
- **Character** – CHAR or VARCHAR (variable length – saves space)
- **Bit** – BIT or BIT VARYING (creates bit strings)
- **Exact Numeric** – NUMERIC, DECIMAL, INTEGER or SMALLINT
- **Approximate Numeric** – FLOAT, REAL, DOUBLE PRECISION
- **Datetime** – DATE, TIME, TIMESTAMP
- **Interval** – INTERVAL
- **Large Objects** – CHARACTER LARGE OBJECT or BINARY LARGE OBJECT

SQL Data Types Declarations

- Address VARCHAR (30)
- Flags BIT(4)
- NUMERIC (5,3)
  - Could store –12.345 in this
- DECIMAL (7,2)
  - Can store up to 99,999.99 in this

Integrity Features

- In this section, we examine the facilities provided by the SQL standard for integrity control, **integrity control** consists of constraints that we wish to impose in order to protect the database from becoming inconsistent. There is five types of integrity constraint:

- **Requiring Data** – Any column can be defined with NOT NULL – forcing data to be supplied

```
position VARCHAR(10)      NOT NULL
```

- **Domain Constraints** – The CHECK clause can be added to a table or to a column

a) CHECK

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

b) CREATE DOMAIN

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

For example:

```
CREATE DOMAIN SexType AS CHAR
             CHECK (VALUE IN ('M', 'F'));
sex          SexType  NOT NULL
```

Integrity Features (cont'd)

- ***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
```

Integrity Features (cont'd)

- **Entity Integrity** – the PRIMARY KEY clause can be used to specify the columns that make up the unique, non-null values for each row – alternate keys can be specified using the UNIQUE keyword
  - SQL will not allow duplicate PRIMARY KEYs
  - ISO standard supports FOREIGN 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)

- **Referential Integrity** – The FOREIGN KEY … REFERENCES clauses check the links between a child table containing a foreign key to the parent table with the candidate key
  - SQL rejects any INSERT or UPDATE operation that attempts to create a foreign key without an associated candidate key
  - ISO standard supports definition of FKs with FOREIGN KEY clause in CREATE and ALTER TABLE:

    FOREIGN KEY(branchNo) REFERENCES Branch

Integrity Features (Referential Actions cont'd)

- Any referential integrity clause may also specify a referential action to be taken place upon an UPDATE or DELETE (assuming a deletion occurs):
  - **CASCADE** – Delete the row from the parent table and automatically delete the matching rows in the child table. Since these deleted rows may themselves has a candidate key that is used as a foreign key in another table, the foreign key rules for these tables are triggered, and so on in a cascading action
  - **SET NULL** – Delete the row from the parent table and set the foreign key values(s) in the child table to NULL. This will only work if the foreign key columns do not have the NOT NULL qualifier.
  - **SET DEFAULT** – Delete the row from the parent table and set each component of the foreign key in the child table to the specified default value. This will only work if DEFAULT values have been specified.
  - **NO ACTION** – Reject the delete operation from the parent table. This is the default setting if no referential action has been specified.
  - **Examples:**

FOREIGN KEY (staffNo) REFERENCES Staff ON DELETE SET NULL

FOREIGN KEY (ownerNo) REFERENCES Owner ON UPDATE CASCADE

Integrity Features (cont'd)

- **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 staffNo
                       FROM PropertyForRent
                       GROUP BY staffNo
                       HAVING COUNT(*) > 100))
```

Main SQL DDL Statements

- SQL DDL allows database objects such as schemas, domains, tables, views, and indexes to be created and destroyed.
- Main SQL DDL statements are:
  - **CREATE SCHEMA** – builds a database (DROP SCHEMA removes)
  - **CREATE DOMAIN** – builds a data domain, like an enum (ALTER/DROP available)
  - **CREATE TABLE** – builds a table (ALTER/DROP available)
  - **CREATE VIEW** – builds a view of one or more tables (DROP available)
- Not standard, but most DBs support CREATE/DROP INDEX

Creating a Database Examples

- **CREATE SCHEMA** [Name | AUTHORIZATION CreatorIdentifier]
  - `CREATE SCHEMA TestDB AUTHORIZATION Smith;`

- **DROP SCHEMA** Name [**RESTRICT** | **CASCADE**]

Creating a Domain Example

- **CREATE DOMAIN** Name [**AS**] DataType [**DEFAULT** DefaultOption] [**CHECK** (CheckCondition)]
  - `CREATE DOMAIN SexType AS CHAR`
    `DEFAULT 'M'`
    `CHECK (VALUE IN ('M','F')`

Creating a Table Example

CREATE TABLE TableName

{(ColumnName DataType [**NOT NULL**][**UNIQUE**][**DEFAULT** DefaultOption][**CHECK** CheckCondition][,…]}

[**PRIMARY KEY** (ListOfColumns),]

{[**UNIQUE** (ListOfColumns),][,…]}

{[**FOREIGN KEY** (ListOfForeignKeyColumns)

REFERENCES ParentTableName [(ListOfCandidateKeyColumns)],

[**MATCH** [**PARTIAL** | **FULL**]

[**ON UPDATE** ReferentialAction]

[**ON DELETE** ReferentialAction]][,…]}

{[**CHECK** (CheckCondition)][,…]}

A Moderate Create Table Example

```
CREATE DOMAIN OwnerNumber AS VARCHAR(5)
   CHECK (VALUE IN (SELECT OwnerNo FROM PrivateOwner_Table));
CREATE DOMAIN StaffNumber AS VARCHAR(5)
   CHECK (VALUE IN (SELECT StaffNo FROM Staff_Table));
CREATE DOMAIN BranchNumber AS CHAR(4)
   CHECK (VALUE IN (SELECT BranchNo FROM Branch_Table));
CREATE DOMAIN PropertyNumber AS VARCHAR(5);
CREATE DOMAIN Street AS VARCHAR(25);
CREATE DOMAIN City As VARCHAR(15);
CREATE DOMAIN ZipCode AS VARCHAR(10);
CREATE DOMAIN PropertyType AS VARCHAR(9)
   CHECK (VALUE IN ('Apartment','House'));
CREATE DOMAIN PropertyRooms AS SMALLINT
   CHECK (VALUE BETWEEN 1 AND 15);
CREATE DOMAIN PropertyRent AS DECIMAL(6,2)
   CHECK (VALUE BETWEEN 0 AND 9999.99);
```

Create Table Example (cont'd)

```
CREATE TABLE PropertyForRent_Table (
  PropertyNo  PropertyNumber    NOT NULL,
  StreetAddr  Street            NOT NULL,
  CityAddr    City              NOT NULL,
  ZipCodeAddr ZipCode           NOT NULL,
  Type        PropertyType      NOT NULL DEFAULT 'House',
  Rooms       PropertyRooms     NOT NULL DEFAULT 4,
  OwnerNo     OwnerNumber       NOT NULL,
  StaffNo     StaffNumber
    CONSTRAINT StaffNotHandlingTooMuch
        CHECK (NOT EXISTS (SELECT StaffNo
                           FROM PropertyForRent_Table
                            GROUP BY StaffNo
                             HAVING COUNT(*) > 100)),
  BranchNo      BranchNumber    NOT NULL ,
```

Create Table Example (cont'd)

```
  PRIMARY KEY (PropertyNo),
  FOREIGN KEY (StaffNo) REFERENCES Staff_Table
        ON DELETE SET NULL
        ON UPDATE CASCADE,
FOREIGN KEY (OwnerNo) REFERENCES Owner_Table
        ON DELETE NO ACTION
        ON UPDATE CASCADE,
FOREIGN KEY (BranchNo) REFERENCES Branch_Table
        ON DELETE NO ACTION
        ON UPDATE CASCADE);
```

Changing a Table Definition

- **The ALTER TABLE statement can be used for:**
  - Adding a new column to a table
  - Dropping a column from a table
  - Adding a new table constraint
  - Dropping a table constraint
  - Setting a default for a column
  - Dropping a default for a column

ALTER TABLE Examples

- **Example:** change the Staff table by removing the default of 'Assistant' for the position column and setting the default for the sex column to female ('F'):

```
ALTER TABLE Staff
   ALTER Position DROP DEFAULT;
ALTER TABLE Staff
  ALTER Sex SET DEFAULT 'F';
```

- **Example:** change the PropertyForRent table by removing the constraint that staff are not allowed to handle more than 100 properties at a time. Change the Client table by adding a new column representation the preferred number of rooms:

```
ALTER TABLE PropertyForRent
  DROP CONSTRAINT StaffNotHandlingTooMuch;
ALTER TABLE Client
  ADD PrefNoRooms PropertyRooms;
```

Removing a Table

- **Syntax:**
  - DROP TABLE TableName [RESTRICT | CASCADE]

- **Example:**
  - `DROP TABLE PropertyForRent_Table;`

Creating an Index

- INDEXES are structures that provide quicker access to the rows of a table based on the values of one or more columns
- **Syntax:**
  - CREATE [UNIQUE] INDEX IndexName ON TableName (ColumnName [ASC | DESC][,…])
  - DROP INDEX IndexName
- **Examples:**
  - `CREATE INDEX IdxStaffNo ON Staff_Table (StaffNo);`
  - `DROP INDEX StaffNo;`

Views

View

Dynamic result of one or more relational operations operating on base relations to produce another relation.

- Virtual relation that does not necessarily actually exist in the database but is produced upon request, at time of request.

Views

- Contents of a view are defined as a query on one or more base relations.
- With **view resolution,** any operations on view are automatically translated into operations on relations from which it is derived.
- With **view materialization,** the view is stored as a temporary table, which is maintained as the underlying base tables are updated.

SQL - CREATE VIEW

```
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*.

SQL - CREATE VIEW

- List must be specified if there is any ambiguity in a column name.
- The *subselect* is known as the **defining query.**
- WITH CHECK OPTION ensures that if a row fails to satisfy WHERE clause of defining query, it is not added to underlying base table.
- Need SELECT privilege on all tables referenced in subselect and USAGE privilege on any domains used in referenced columns.

Example 7.3 - Create Horizontal View

- **Example:** 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';
```

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 7.4 - Create Vertical View

- **Example:** 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';
```

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

Example 7.5 - Grouped and Joined Views

- **Example:** 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;
```

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

SQL - DROP VIEW

- 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.

View Resolution

- **Example:** count number of properties managed by each member at branch B003.

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

(a) View column names in **SELECT** list are translated into their corresponding column names in the defining query:

```
SELECT s.staffNo As staffNo, COUNT(*) As cnt
```

(b) View names in **FROM** are replaced with corresponding FROM lists of defining query:

```
FROM Staff s, PropertyForRent p
```

View Resolution

(c) **WHERE** from user query is combined with WHERE of defining query using AND:

```
WHERE s.staffNo = p.staffNo AND branchNo = 'B003'
```

(d) **GROUP BY and HAVING** clauses copied from defining query:

```
GROUP BY s.branchNo, s.staffNo
```

(e) **ORDER BY** copied from query with view column name translated into defining query column name

```
ORDER BY s.staffNo
```

(f) Final merged query is now executed to produce the result:

```
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 Views

SQL imposes several restrictions on creation and use of views.

(a) If column in view is based on an aggregate function:

- Column may appear only in SELECT and ORDER BY clauses of queries that access view.
- Column may not be used in WHERE nor be an argument to an aggregate function in any query based on view.

- **For example, following query would fail:**

```
SELECT COUNT(cnt)
FROM StaffPropCnt;
```

- **Similarly, following query would also fail:**

```
SELECT *
FROM StaffPropCnt
WHERE cnt > 2;
```

Restrictions on Views

(b) Grouped view may never be joined with a base table or a view.

- **For example,** StaffPropCnt view is a grouped view, so any attempt to join this view with another table or view fails.

View Updatability

- All updates to base table reflected in all views that encompass base table.
- Similarly, may expect that if view is updated then base table(s) will reflect change.
- However, consider again view StaffPropCnt.
- If we tried to insert record showing that at branch B003, SG5 manages 2 properties:

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

- 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!

View Updatability

- However, consider again view StaffPropCnt.
- **If we tried to insert record showing that at branch B003, SG5 manages 2 properties:**

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

- 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!
- **If change definition of view and replace count with actual property numbers:**

```
CREATE VIEW StaffPropList (branchNo,
                           staffNo, propertyNo)
AS SELECT s.branchNo, s.staffNo, p.propertyNo
    FROM Staff s, PropertyForRent p
    WHERE s.staffNo = p.staffNo;
```

View Updatability

- **Now try to insert the record:**

```
INSERT INTO StaffPropList
VALUES ('B003', 'SG5', 'PG19');
```

- Still problem, because 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:

  - DISTINCT is not specified.

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

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

  - No nested SELECT referencing outer table.

  - No GROUP BY or HAVING clause.

  - Also, every row added through view must not violate integrity constraints of base table.

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

Transactions

- New transaction starts with next transaction-initiating statement.
- SQL transactions cannot be nested.
- SET TRANSACTION configures transaction:

```
SET TRANSACTION
    [READ ONLY | READ WRITE] |
    [ISOLATION LEVEL READ UNCOMMITTED |
    READ COMMITTED|REPEATABLE READ |SERIALIZABLE ]
```

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.
- 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.
- 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 7.7/8 - 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 7.9 - GRANT Specific Privileges to PUBLIC

Give all users SELECT on Branch table.

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

REVOKE

- REVOKE takes away privileges granted with GRANT.

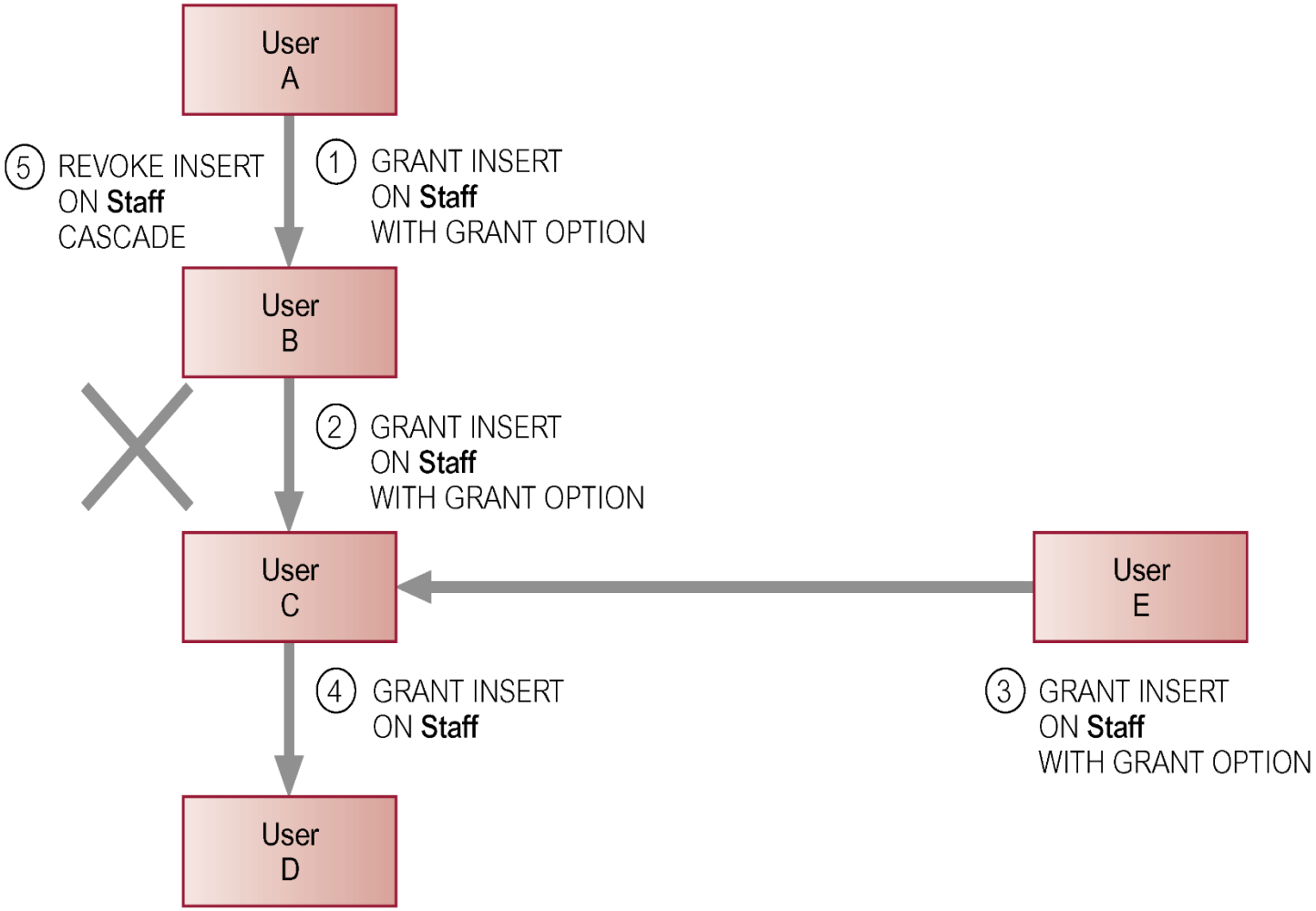
```
REVOKE [GRANT OPTION FOR]
  {PrivilegeList | ALL PRIVILEGES}
ON ObjectName
FROM {AuthorizationIdList | PUBLIC}
    [RESTRICT | CASCADE]
```

- 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


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

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;
```