Chapter 5

More SQL: Complex Queries, Triggers, Views, and Schema Modification
Chapter 5 Outline

- More Complex SQL Retrieval Queries
- Specifying Constraints as Assertions and Actions as Triggers
- Views (Virtual Tables) in SQL
- Schema Change Statements in SQL
More Complex SQL Retrieval Queries

- Additional features allow users to specify more complex retrievals from database:
  - Nested queries, joined tables, outer joins, aggregate functions, and grouping
Comparisons Involving NULL and Three-Valued Logic

- Meanings of NULL
  - Unknown value
  - Unavailable or withheld value
  - Not applicable attribute

- Each individual NULL value considered to be different from every other NULL value

- SQL uses a three-valued logic:
  - TRUE, FALSE, and UNKNOWN
Comparisons Involving NULL and Three-Valued Logic (cont’d.)

<table>
<thead>
<tr>
<th>Table 5.1</th>
<th>Logical Connectives in Three-Valued Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td><strong>AND</strong></td>
</tr>
<tr>
<td></td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
</tr>
<tr>
<td></td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>(b)</td>
<td><strong>OR</strong></td>
</tr>
<tr>
<td></td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
</tr>
<tr>
<td></td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>(c)</td>
<td><strong>NOT</strong></td>
</tr>
<tr>
<td></td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
</tr>
<tr>
<td></td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>
Comparisons Involving NULL and Three-Valued Logic (cont’d.)

- SQL allows queries that check whether an attribute value is NULL
  - IS or IS NOT NULL

Query 18. Retrieve the names of all employees who do not have supervisors.

Q18: SELECT Fname, Lname
     FROM EMPLOYEE
     WHERE Super_ssn IS NULL;
Nested Queries, Tuples, and Set/Multiset Comparisons

- **Nested queries**
  - Complete select-from-where blocks within WHERE clause of another query
- **Outer query**
- **Comparison operator** \( \text{IN, NOT IN, EXISTS, NOT EXISTS} \)
  - Compares value \( v \) with a set (or multiset) of values \( V \)
  - Evaluates to \( \text{TRUE} \) if \( v \) is one of the elements in \( V \)
Nested Queries (cont’d.)

Q4A: SELECT DISTINCT Pnumber
    FROM PROJECT
    WHERE Pnumber IN
    ( SELECT Pnumber
        FROM PROJECT, DEPARTMENT, EMPLOYEE
        WHERE Dnum=Dnumber AND
        Mgr_ssn=Ssn AND Lname=‘Smith’ )
    OR
    Pnumber IN
    ( SELECT Pno
        FROM WORKS_ON, EMPLOYEE
        WHERE Essn=Ssn AND Lname=‘Smith’ );
Nested Queries (cont’d.)

- Use tuples of values in comparisons
  - Place them within parentheses

```sql
SELECT DISTINCT Essn
FROM WORKS_ON
WHERE (Pno, Hours) IN (SELECT Pno, Hours
FROM WORKS_ON
WHERE Essn='123456789');
```
Nested Queries (cont’d.)

- Use other comparison operators to compare a single value \( v \)
  - \( = \) \( \text{ANY} \) (or \( = \) \( \text{SOME} \)) operator
    - Returns \( \text{TRUE} \) if the value \( v \) is equal to some value in the set \( V \) and is hence equivalent to \( \text{IN} \)
  - Other operators that can be combined with \( \text{ANY} \) (or \( \text{SOME} \)): >, >=, <, <=, and <>
Nested Queries with ALL

Select Fname, Lname
From Employee E
Where E.Salary > ALL ( Select E2.Salary
                      From Employee E2
                      Where E2.Dno = 5 );

Select E1.Fname, E1.Lname
From Employee E1
Where E1.Salary > ( Select MAX(E2.Salary)
                      From Employee E2
                      Where E2.Dno = 5 );
Nested Queries with ALL

Select E1.Fname, E1.Lname
From Employee E1
Where E1.Salary < ALL ( Select E2.Salary
From Employee E2
Where E2.Dno = 5 );

Select E1.Fname, E1.Lname
From Employee E1
Where E1.Salary < ( Select MIN(E2.Salary)
From Employee E2
Where E2.Dno = 5 );
Nested Queries with Any

Select E1.Fname, E1.Lname
From Employee E1
Where E1.Salary > Any ( Select E2.Salary
From Employee E2
Where E2.Dno = 5 );

Select E1.Fname, E1.Lname
From Employee E1
Where E1.Salary < Any ( Select E2.Salary
From Employee E2
Where E2.Dno = 5 );
Nested Queries (cont’d.)

- Avoid potential errors and ambiguities
  - Create tuple variables (aliases) for all tables referenced in SQL query

Query 16. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

Q16: ```sql
SELECT E.Fname, E.Lname
FROM EMPLOYEE AS E
WHERE E.Ssn IN (
  SELECT Essn
  FROM DEPENDENT AS D
  WHERE E.Fname = DDEPENDENT_name
  AND E.Sex = D.Sex
);```
Correlated Nested Queries

- **Correlated** nested query
  - Evaluated once for each tuple in the outer query
  - Nested subquery evaluated multiple times: Repeated for # of tuples in the outer table
The EXISTS and UNIQUE Functions in SQL

- **EXISTS function**
  - Check whether the result of a correlated nested query is empty or not

- **EXISTS and NOT EXISTS**
  - Typically used in conjunction with a correlated nested query

- **SQL function UNIQUE(Q)**
  - Returns **TRUE** if there are no duplicate tuples in the result of query Q
IN / EXISTS

Select * From Employee E Where E.ssn IN (Select D.Essn From Dependent D);

Select * From Employee E Where EXISTS (Select D.Essn From Dependent D);
Exists with Correlated SubQuery

Select * 
From Employee E 
Where Exists (Select D.essn 
From Dependent D);

Select * 
From Employee E 
Where Exists (Select D.essn 
From Dependent D 
Where E.ssn = D.essn);
Exists/Not Exists
with Correlated SubQuery

Select * 
From Employee E 
Where Exists (Select D.essn 
  From Dependent D 
  Where E.ssn = D.essn); 

Select * 
From Employee E 
Where Not Exists (Select D.essn 
  From Dependent D 
  Where E.ssn = D.essn);
Exists AND Not Exists with Correlated SubQuery

Select *  
From Employee E  
Where Exists (Select Dp.essn  
From Dependent Dp  
Where E.ssn = Dp.essn)  
And  
Not Exists (Select D.mgrssn  
From Department D  
Where E.ssn = D.mgrssn);
IN with Correlated SubQuery

\[
\text{Select} \quad * \\
\text{From} \quad \text{Employee E} \\
\text{Where} \quad \text{E.ssn} \quad \text{IN} \quad (\text{Select} \quad \text{D.Essn} \\
\text{From} \quad \text{Dependent D});
\]

\[
\text{Select} \quad * \\
\text{From} \quad \text{Employee E} \\
\text{Where} \quad \text{E.ssn} \quad \text{IN} \quad (\text{Select} \quad \text{D.Essn} \\
\text{From} \quad \text{Dependent D} \\
\text{Where} \quad \text{E.ssn} = \text{D.Essn});
\]
IN/NOT IN with Correlated SubQuery

Select * 
From Employee E 
Where E.Dno IN (Select D.Dnumber 
From Department D 
Where E.ssn = D.mgrssn);

Select * 
From Employee E 
Where E.Dno NOT IN (Select D.Dnumber 
From Department D 
Where E.ssn = D.mgrssn);
Explicit Sets and Renaming of Attributes in SQL

- Can use explicit set of values in WHERE clause
- Use qualifier AS followed by desired new name
  - Rename any attribute that appears in the result of a query

Q&A: SELECT E.Lname AS Employee_name, S.Lname AS Supervisor_name FROM EMPLOYEE AS E, EMPLOYEE AS S WHERE E.Super_ssn=S.Ssn;
Joined Tables in SQL and Outer Joins

- **Joined table**
  - Permits users to specify a table resulting from a join operation in the FROM clause of a query

- **The FROM clause in Q1A**
  - Contains a single joined table

```
Q1A:  SELECT  Fname, Lname, Address
      FROM     (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
      WHERE    Dname='Research';
```
Joined Tables in SQL and Outer Joins (cont’d.)

- Specify different types of join
  - NATURAL JOIN
  - Various types of OUTER JOIN

- NATURAL JOIN on two relations R and S
  - No join condition specified
  - Implicit EQUIJOIN condition for each pair of attributes with same name from R and S
Natural Join

Select  *
From   Employee E, Department D
Where  E.dno = D.Dnumber

Returns:
Matching Tuples on E.dno = D.Dnumber with All the Employee Columns and All the Department Columns
Joined Tables in SQL and Outer Joins (cont’d.)

- **Inner join**
  - Default type of join in a joined table
  - Tuple is included in the result only if a matching tuple exists in the other relation

- **LEFT OUTER JOIN**
  - Every tuple in left table must appear in result
  - If no matching tuple
    - Padded with NULL values for attributes of right table
# Left Outer Join

Select D.dnumber, D.dname, E.dno, E.ssn
From Department D **LEFT OUTER JOIN** Employee E
On D.dnumber = E.dno;

<table>
<thead>
<tr>
<th>Dnumber</th>
<th>dname</th>
<th>dno</th>
<th>ssn</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Headquarter</td>
<td>1</td>
<td>888665555</td>
</tr>
<tr>
<td>4</td>
<td>Admin</td>
<td>4</td>
<td>987654321</td>
</tr>
<tr>
<td>4</td>
<td>Admin</td>
<td>4</td>
<td>987987987</td>
</tr>
<tr>
<td>4</td>
<td>Admin</td>
<td>4</td>
<td>999887777</td>
</tr>
<tr>
<td>5</td>
<td>Research</td>
<td>5</td>
<td>123456789</td>
</tr>
<tr>
<td>5</td>
<td>Research</td>
<td>5</td>
<td>333445555</td>
</tr>
<tr>
<td>5</td>
<td>Research</td>
<td>5</td>
<td>453453453</td>
</tr>
<tr>
<td>5</td>
<td>Research</td>
<td>5</td>
<td>666884444</td>
</tr>
<tr>
<td>7</td>
<td>Automation</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Outer Join

Select *
From Employee E
LEFT OUTER JOIN Dependent D
ON D.essn = E.ssn AND D.relationship = 'spouse';

Returns:
All the columns of E with all the columns of D for the matching ssn with essn
+
All the columns of E with Null padded columns of D for the unmatching ssn with essn.
Joined Tables in SQL and Outer Joins (cont’d.)

- **RIGHT OUTER JOIN**
  - Every tuple in right table must appear in result
  - If no matching tuple
    - Padded with NULL values for the attributes of left table

- **FULL OUTER JOIN**

- Can nest join specifications
Aggregate Functions in SQL

- Used to summarize information from multiple tuples into a single-tuple summary

- **Grouping**
  - Create subgroups of tuples before summarizing

- **Built-in aggregate functions**
  - `COUNT`, `SUM`, `MAX`, `MIN`, and `AVG`

- Functions can be used in the `SELECT` clause or in a `HAVING` clause
Aggregate Functions in SQL (cont’d.)

- NULL values discarded when aggregate functions are applied to a particular column.

Query 20. Find the sum of the salaries of all employees of the ‘Research’ department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```sql
Q20: SELECT SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
     FROM (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
     WHERE Dname='Research';
```

Queries 21 and 22. Retrieve the total number of employees in the company (Q21) and the number of employees in the ‘Research’ department (Q22).

```sql
Q21: SELECT COUNT (*)
     FROM EMPLOYEE;

Q22: SELECT COUNT (*)
     FROM EMPLOYEE, DEPARTMENT
     WHERE DNO=DNUMBER AND DNAME='Research';
```
Grouping: The GROUP BY and HAVING Clauses

- **Partition** relation into subsets of tuples
  - Based on **grouping attribute(s)**
  - Apply function to each such group independently
  - \(# \text{ of group} = \# \text{ of unique values in Group by Col}\)

- **GROUP BY** clause
  - Specifies grouping attributes
  - Should appear in Select clause as well

- If NULLs exist in grouping attribute
  - Separate group created for all tuples with a NULL value in grouping attribute
Aggregate Function

Select R.a
From R
Where R.b > (Select COUNT (*)
   From S
   Where R.c = S.c)
Aggregate Function in Having

Select R.a, R.b
From R Left Outer Join S
ON R.c = S.c
Group by R.a, R.b
Having R.b > MAX (S.b)
Aggregation with Group BY

Select   E.dno, Count(*)
From     Employee E, Department D
Where    E.dno = D.Dnumber
Group By E.dno;
Will return:
Dno       Count(E.ssn)
-----------  ------------
  1         1
  4         3
  5         4
Group By with Aggregation

Select D.dnumber, Count(E.dno)
From Department D Left Outer Join Employee E
ON D.Dnumber = E.dno

Group By D.dnumber;

Will return:

<table>
<thead>
<tr>
<th>Dnumber</th>
<th>Count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>
Group By with Aggregation

For Empty group,
Count(*) = 0,
MAX, MIN, AVE, SUM return NULL

Select D.dnumber, MAX(E.salary)
From Department D LEFT OUTER JOIN Employee E
ON D.Dnumber = E.dno
Group By D.dnumber;

Will return:

<table>
<thead>
<tr>
<th>Dnumber</th>
<th>MAX(E.salary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>55000</td>
</tr>
<tr>
<td>4</td>
<td>43000</td>
</tr>
<tr>
<td>5</td>
<td>40000</td>
</tr>
<tr>
<td>7</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Grouping: The GROUP BY and HAVING Clauses (cont’d.)

- **HAVING clause**
  - Provides a condition on the summary information

---

**Query 28.** For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $40,000.

Q28:  

```sql
SELECT Dnumber, COUNT (*)
FROM DEPARTMENT, EMPLOYEE
WHERE Dnumber=Dno AND Salary>40000 AND
(SELECT Dno
FROM EMPLOYEE
GROUP BY Dno
HAVING COUNT(*) > 5)
```
Select E.dno, Count(*)
From Employee E
Group By E.dno;
Will return:
Dno  Count(*)
-----  ---------------
1      1
4      3
5      4

SubQ of Q28: Select E.dno
From Employee E
Group By E.dno
Having COUNT(*) > 2;

Dno
-----
4
5

Q28: Select D.dnumber, Count(*)
From Employee E, Department D
Where D.Dnumber = E.dno and E.salary > 40000 and And
(SubQ);

Dnumber  Count(*)
--------  ------------
      1
Group By with Having

Select D.dnumber, Count(*)
From Employee E, Department D,
( Select E.dno
     From Employee E
     Group By E.dno
     Having COUNT(*) > 2) as Temp(high_dno)
Where D.dnumber = E.dno and E.salary > 40000 and Temp.high_dno = e.dno
Group by D.dnumber;
Discussion and Summary of SQL Queries

```
SELECT <attribute and function list>
FROM <table list>
[ WHERE <condition> ]
[ GROUP BY <grouping attribute(s)> ]
[ HAVING <group condition> ]
[ ORDER BY <attribute list> ];
```
Specifying Constraints as Assertions and Actions as Triggers

- **CREATE ASSERTION**
  - Specify additional types of constraints outside scope of built-in relational model constraints

- **CREATE TRIGGER**
  - Specify automatic actions that database system will perform when certain events and conditions occur
Specifying General Constraints as Assertions in SQL

- **CREATE ASSERTION**
  - Specify a query that selects any tuples that violate the desired condition
  - Use only in cases where it is not possible to use `CHECK` on attributes and domains

```sql
CREATE ASSERTION SALARY_CONSTRAINT
CHECK ( NOT EXISTS ( SELECT *
    FROM EMPLOYEE E, EMPLOYEE M,
    DEPARTMENT D
    WHERE E.Salary>M.Salary
    AND E.Dno=D.Dnumber
    AND D.Mgr_ssn=M.Ssn ) )
```
What is checked in SubQ

NOT EXISTS ( Select * 
    From Employee E, Employee M, Department D 
    Where E.salary > M.salary and 
    E.dno = D.Dnumber and D.mgr_ssn = M.ssn);

1. To get each dept manager’s info:
   Select * 
   From Department D, Employee M 
   Where D.mgr_ssn = M.ssn;

2. To get salary of each employee who is working for the dept whose manager is mgr_ssn to compare
   Select * 
   From Employee E, Employee M, Department D 
   Where E.salary > M.salary and 
   E.dno = D.Dnumber and D.mgr_ssn = M.ssn;)

Addison-Wesley
is an imprint of
PEARSON
Copyright © 2011 Ramez Elmasri and Shamkant Navathe
SubQuery in Select

Select     tbls.owner, tbls.table_name,  
            (Select count(column_name) as total_columns  
            From all_tab_columns cols  
            Where cols.owner = tbls.owner and  
            cols.table_name =  
            tbls.table_name) Subquery2

From all_tables tbls;
Must return a single value.
Usually an aggregate function:
Min, Max, Ave, Sum, Count without Group By
Introduction to Triggers in SQL

- **CREATE TRIGGER** statement
  - Used to monitor the database
- Typical trigger has three components:
  - Event(s)
  - Condition
  - Action
Views (Virtual Tables) in SQL

- Concept of a view in SQL
  - Single table derived from other tables
  - Considered to be a virtual table
Specification of Views in SQL

- **CREATE VIEW** command
  - Give table name, list of attribute names, and a query to specify the contents of the view

```sql
V1: CREATE VIEW WORKS_ON1
    AS SELECT Fname, Lname, Pname, Hours
    FROM EMPLOYEE, PROJECT, WORKS_ON
    WHERE Ssn=Essn AND Pno=Pnumber;

V2: CREATE VIEW DEPT_INFO
    AS SELECT Dname, COUNT (*), SUM (Salary)
    FROM DEPARTMENT, EMPLOYEE
    WHERE Dnumber=Dno
    GROUP BY Dname;
```
Altering Existing View

Replace View Existing_View_Name As
Select… From… Where…;

Replace View Works_On1 As
Select  Fname, Lname, Pno, Pname, Hours
From    Employee, Works_On, Project
Where   Ssn = Essn and Pno = Pnumber
Specification of Views in SQL (cont’d.)

- Specify SQL queries on a view
- View always up-to-date
  - Responsibility of the DBMS and not the user
- **DROP VIEW** command
  - Dispose of a view
View Implementation, View Update, and Inline Views

- Complex problem of efficiently implementing a view for querying
- **Query modification** approach
  - Modify view query into a query on underlying base tables
  - Disadvantage: inefficient for views defined via complex queries that are time-consuming to execute
View Implementation

- **View materialization approach**
  - Physically create a temporary view table when the view is first queried
  - Keep that table on the assumption that other queries on the view will follow
  - Requires efficient strategy for automatically updating the view table when the base tables are updated
View Implementation (cont’d.)

- **Incremental update strategies**
  - DBMS determines what new tuples must be inserted, deleted, or modified in a materialized view table
View Update and Inline Views

- Update on a view defined on a single table without any aggregate functions
  - Can be mapped to an update on underlying base table
- View involving joins
  - Often not possible for DBMS to determine which of the updates is intended
View Update and Inline Views (cont’d.)

- Clause `WITH CHECK OPTION`
  - Must be added at the end of the view definition if a view is to be updated

- In-line view
  - Defined in the `FROM` clause of an SQL query
Schema Change Statements in SQL

- **Schema evolution commands**
  - Can be done while the database is operational
  - Does not require recompilation of the database schema
The DROP Command

- **DROP command**
  - Used to drop named schema elements, such as tables, domains, or constraint

- Drop behavior options:
  - **CASCADE** and **RESTRICT**

- Example:
  - `DROP SCHEMA COMPANY CASCADE;`
The ALTER Command

- **Alter table actions** include:
  - Adding or dropping a column (attribute)
  - Changing a column definition
  - Adding or dropping table constraints

- **Example:**
  
  ```sql
  ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job VARCHAR(12);
  ```

- **To drop a column**
  - Choose either **CASCADE** or **RESTRICT**
The ALTER Command (cont’d.)

- Change constraints specified on a table
  - Add or drop a named constraint

```sql
ALTER TABLE COMPANY.EMPLOYEE
DROP CONSTRAINT EMPSUPERFK CASCADE;
```
GRANT

SQL GRANT is a command used to provide access or privileges on the database objects to the users.

The Syntax for the GRANT command is:

```
GRANT privilege_name
ON object_name
TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];
```

*privilege_name* is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, and SELECT.

*object_name* is the name of an database object like TABLE, VIEW, STORED PROC and SEQUENCE.

*user_name* is the name of the user to whom an access right is being granted.

*PUBLIC* is used to grant access rights to all users.

*ROLES* are a set of privileges grouped together.

*WITH GRANT OPTION* - allows a user to grant access rights to other users.
REVOKE

The REVOKE command removes user access rights or privileges to the database objects.

The Syntax:

```
REVOKE privilege_name
    ON object_name
    FROM {user_name | PUBLIC | role_name}
```
COMMIT

Use the COMMIT statement to end your current transaction and make permanent all changes performed in the transaction.

A transaction is a sequence of SQL statements that Oracle Database treats as a single unit.

This statement also erases all savepoints in the transaction and releases transaction locks.
Until you commit a transaction:

You can see any changes you have made during the transaction by querying the modified tables, but other users cannot see the changes.

After you commit the transaction, the changes are visible to other users' statements that execute after the commit.

You can roll back (undo) any changes made during the transaction with the ROLLBACK statement.
RollBack

The ROLLBACK statement is the inverse of the COMMIT statement. It undoes some or all database changes made during the current transaction.
Summary

- Complex SQL:
  - Nested queries, joined tables, outer joins, aggregate functions, grouping

- `CREATE ASSERTION` and `CREATE TRIGGER`

- Views
  - Virtual or derived tables