Chapter 7: Complex SQL Retrieval Queries

1. Nested Queries, Joined Tables, Outer Joins

- **Nested Queries**: Subqueries used within SELECT, WHERE, or FROM clauses.
  - Example: SELECT * FROM Employees WHERE Salary > (SELECT AVG(Salary) FROM Employees);
- **Joined Tables**: Combines records from multiple tables.
  - **Syntax**: SELECT * FROM TableA INNER JOIN TableB ON TableA.id = TableB.id;
- **Outer Joins**: Includes rows even without matches.
  - **Left Outer Join**: All rows from the left table + matched from the right.
  - **Right Outer Join**: All rows from the right table + matched from the left.
  - Example: SELECT * FROM TableA LEFT JOIN TableB ON TableA.id = TableB.id;

2. Aggregate Functions & Grouping

- **Functions**: COUNT(), SUM(), AVG(), MIN(), MAX()
- **GROUP BY**: Organizes rows sharing a property into groups.
  - Example: SELECT Department, COUNT(*) FROM Employees GROUP BY Department;
- **Grouping with Conditions**: Using HAVING to filter groups.
  - Example: SELECT Department, AVG(Salary) FROM Employees GROUP BY Department HAVING AVG(Salary) > 50000;

3. SQL's 3-Valued Logic (True, False, Unknown)

- **True**: Condition met.
- **False**: Condition not met.
- **Unknown**: When NULL is involved.
  - Example: SELECT * FROM Employees WHERE column IS NULL;

4. Comparison Operators: IN, ANY, SOME

- **IN**: Checks if a value is in a specified list.
  - Example: SELECT * FROM Employees WHERE Department IN ('HR', 'Finance');
- **ANY/SOME**: True if any value from a subquery meets the condition.
  - Example: SELECT * FROM Products WHERE Price > ANY (SELECT Price FROM Electronics);

5. SQL EXISTS & NOT EXISTS

- **EXISTS**: Returns TRUE if the subquery has results.
  - Example: SELECT * FROM Employees WHERE EXISTS (SELECT 1 FROM Departments WHERE Employees.DeptID = Departments.ID);
- **NOT EXISTS**: Returns TRUE if the subquery has no results.
  - Example: SELECT * FROM Employees WHERE NOT EXISTS (SELECT 1 FROM Departments WHERE Employees.DeptID = Departments.ID);

6. SQL DISTINCT & UNIQUE

- **DISTINCT**: Removes duplicate rows in results.
  - Example: SELECT DISTINCT Department FROM Employees;
- **UNIQUE**: Enforces uniqueness in a column.
  - Example: CREATE TABLE Employees (ID INT UNIQUE, Name VARCHAR(50));

7. NATURAL JOIN & Outer Joins

- **NATURAL JOIN**: Joins tables on columns with the same name automatically.
  - Example: SELECT * FROM Employees NATURAL JOIN Departments;
- **Left & Right Outer Joins**: Includes non-matching rows from one table.

8. Built-in Aggregate Functions

- **Functions**: COUNT(), MAX(), MIN(), AVG(), SUM()
- **Example**: SELECT Department, MAX(Salary) FROM Employees GROUP BY Department;

Chapter 4: Enhanced Entity Relationship (EER) Model

1. EER Model Concepts

- **Subclasses & Superclasses**: Defines hierarchical relationships.
- **Inheritance**: Subclasses inherit attributes and relationships of superclasses.

2. Specialization & Generalization

- **Specialization**: Dividing an entity into sub-entities.
- **Generalization**: Combining sub-entities into a more general entity.
- **Single Inheritance**: Subclass inherits from one superclass.
- **Multiple Inheritance**: Subclass inherits from multiple superclasses.

3. Category (Union) Types

- **Category**: Represents a subset of a union of multiple entity types.
  - Example: Vehicle as a union of Car and Boat.

4. Attribute & Relationship Inheritance

- **Attributes**: Inherited by subclasses from their superclass.
- **Relationships**: Also inherited.

5. EER Diagrams: Subclasses & Specialization

- **EER Diagrams**: Show hierarchical structures.
- **Constraints**: Rules applied to subclasses.
  - **Disjointness**: An entity belongs to only one subclass (Disjoint) or can belong to multiple (Overlapping).
  - **Completeness**: Total (must be a member of a subclass) or Partial (may not be a member).

Additional EER Concepts

1. Specialization & Generalization Lattice

- A hierarchical structure showing relationships among classes.

2. EER Conceptual Schema for University Database

- **Entities**: Student, Professor, Course.
- **Relationships**: Enrollment, Advises, Teaches.

3. Formal Definition of EER Model Concepts

- **Class**: A blueprint for creating entities.
- **Sub-class**: A more specific version of a class.
- **Specialization**: Process of defining sub-classes.
- **Generalization**: Process of defining a superclass.
- **Predicate Defined**: Subclass defined by a condition.
- **User Defined**: Subclass manually defined by the user.

4. UML Class Diagram from EER Model

- **Classes**: Represent entities.
- **Attributes**: Properties of classes.
- **Associations**: Show relationships between classes.

5. Aggregation & Association

- **Aggregation**: Represents a "part-of" relationship.
  - Example: A department consists of multiple employees.
- **Association**: General relationship between entities.
  - Example: Students enrolled in courses.

6. Knowledge Representations

- **EER Diagrams**: Used to graphically represent EER models.
- **Conceptual Schema**: Blueprint of the database structure.

DIAGRAMS

Entity-Relationship (ER) Model

a) **Definition**: The ER model is a basic conceptual data modeling approach used to define the data structure for a database in terms of entities, attributes, and relationships.
b) **Components**:
c) **Entities**: Objects or concepts, such as Student or Course.
d) **Attributes**: Properties of entities, like Name or Age.
e) **Relationships**: Connections between entities, like EnrolledIn between Student and Course.
f) **Diagrams**: ER diagrams use rectangles for entities, ovals for attributes, and diamonds for relationships.
g) **Purpose**: Used for simple and straightforward database modeling.

Enhanced Entity-Relationship (EER) Model

a) **Definition**: The EER model extends the ER model to support more complex database designs. It introduces additional concepts like inheritance, specialization, and generalization.
b) **Additional Features**:
c) **Subclasses and Superclasses**: EER supports hierarchical structures where entities can inherit attributes and relationships.
d) **Specialization and Generalization**: Concepts to define more specific entities (subclasses) from a general entity (superclass) or combine several subclasses into a general superclass.
e) **Category (Union) Types**: Allows an entity to represent a subset of a union of different entity types.
f) **Attribute and Relationship Inheritance**: Subclasses inherit attributes and relationships from superclasses.
g) **Diagrams**: EER diagrams are more complex and include additional symbols to represent inheritance and constraints.
h) **Purpose**: Used for advanced data modeling scenarios, particularly in systems with hierarchical and complex relationships.

Key Differences

Complexity:

a) **ER Model**: Simpler and suitable for basic database designs.
b) **EER Model**: More complex, designed to handle more sophisticated database structures.

Concepts:

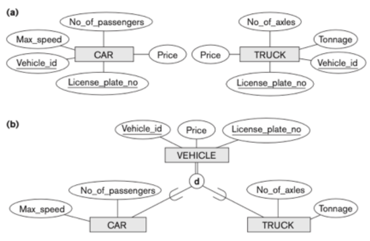
a) **ER Model**: Focuses on entities, attributes, and simple relationships.
b) **EER Model**: Extends the ER model to include inheritance, specialization, generalization, and more.

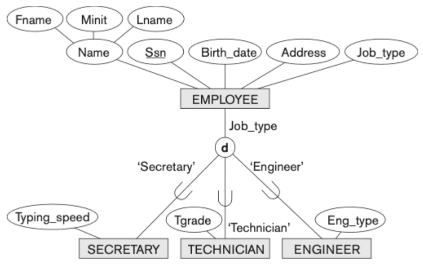
Use Cases:

a) **ER Model**: Used when modeling straightforward databases with no need for hierarchical relationships.
b) **EER Model**: Ideal for modeling databases that require more detailed and structured relationships, such as those involving inheritance or multiple levels of hierarchy.

SELECT employee_name FROM employees WHERE salary > (SELECT AVG(salary) FROM employees);
SELECT e.employee_name, d.department_name FROM employees e JOIN departments d ON e.department_id = d.department_id;
SELECT e.employee_name, d.department_name FROM employees e LEFT JOIN departments d ON e.department_id = d.department_id;
SELECT e.employee_name, d.department_name FROM employees e RIGHT JOIN departments d ON e.department_id = d.department_id;
SELECT department_id, SUM(salary) AS total_salary FROM employees GROUP BY department_id;
SELECT employee_name FROM employees WHERE manager_id IS NULL;
SELECT employee_name FROM employees WHERE department_id IN (SELECT department_id FROM departments WHERE department_name IN ('Sales', 'Marketing', 'IT'));
SELECT department_name FROM departments d WHERE EXISTS (SELECT 1 FROM employees e WHERE e.department_id = d.department_id);
SELECT department_name FROM departments d WHERE NOT EXISTS (SELECT 1 FROM employees e WHERE e.department_id = d.department_id);
SELECT DISTINCT job_title FROM employees;
SELECT salary FROM employees GROUP BY salary HAVING COUNT(*) = 1;
SELECT * FROM employees NATURAL JOIN departments;
SELECT e.employee_name, d.department_name FROM employees e LEFT JOIN departments d ON e.department_id = d.department_id;
SELECT e.employee_name, d.department_name FROM employees e RIGHT JOIN departments d ON e.department_id = d.department_id;
SELECT department_id, MAX(salary) AS max_salary, MIN(salary) AS min_salary, AVG(salary) AS avg_salary, COUNT(*) AS num_employees FROM employees GROUP BY department_id;
SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY
Unknown value. A person's date of birth is not known, so it is represented by NULL in the database. An example of the other case of unknown would be NULL for a person's home phone because it is not known whether or not the person has a home phone.
Unavailable or withheld value. A person has a home phone but does not want it to be listed, so it is withheld and represented as NULL in the database.
Not applicable attribute. An attribute LastCollegeDegree would be NULL for a person who has no college degrees because it does not apply to that person.
Nested queries are queries within other queries. They allow you to filter data based on conditions defined in a subquery. This is useful for complex filtering scenarios where you need to retrieve data based on results from another query. For example, you can find all customers who have placed orders exceeding a certain amount by using a nested query to filter orders and then join that result with the customer table. **Inner Join:** Returns only rows where there is a matching value in both tables. Useful for finding related data when both tables have matching entries. **Left Join:** Returns all rows from the left table and matching rows from the right table. Useful when you want to include all data from one table even if there's no corresponding data in the other. **Right Join:** Similar to a left join, but it returns all rows from the right table and matching rows from the left table. **Full Outer Join:** Returns all rows from both tables, regardless of whether there's a matching row in the other table. Useful when you want to display all possible combinations of data from both tables. **Aggregate** functions (COUNT, SUM, AVG, MIN, MAX, etc.) calculate values based on groups of rows. The GROUP BY clause is used to specify how data should be grouped before applying the aggregate function. This is useful for summarizing data, such as finding the average sale price per product or the total number of orders per customer. **Triggers** are stored procedures that automatically execute in response to specific events in a database (like insert, update, or delete). They are used to maintain data integrity, enforce business rules, and automate tasks. **Assertions** are database constraints that define rules data must follow. They are checked when data is inserted, updated, or deleted. If an assertion fails, the operation is rolled back. Assertions are more static and define general rules about the data itself. Views are virtual tables that present a subset of data from one or more base tables. They are useful for simplifying complex queries, restricting access to sensitive data, and providing different perspectives on the same data. Views can be updateable or not updateable. Updateable views allow you to modify the underlying base tables through the view. Schema change commands (CREATE, ALTER, DROP, TRUNCATE) modify the structure of the database itself. These commands are used to define new tables, add or remove columns, alter constraints, create or drop indexes, and even delete entire tables and views.
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');
SELECT DISTINCT Essn FROM WORKS_ON WHERE (Pno, Hours) IN (SELECT Pno, Hours FROM WORKS_ON WHERE Essn = '123456789');
SELECT Fname, Lname FROM EMPLOYEE WHERE EXISTS (SELECT * FROM DEPENDENT WHERE Ssn = Essn) AND EXISTS (SELECT * FROM DEPARTMENT WHERE Ssn = Mgr_ssn);
SELECT Fname, Lname FROM EMPLOYEE WHERE NOT EXISTS ((SELECT Pnumber FROM PROJECT WHERE Dnum = 5) EXCEPT (SELECT Pno FROM WORKS_ON WHERE Ssn = Essn));
SELECT Lname, Fname FROM EMPLOYEE WHERE NOT EXISTS (SELECT * FROM WORKS_ON B
WHERE (B.Pno IN (SELECT Pnumber FROM PROJECT WHERE Dnum = 5) AND NOT EXISTS (SELECT * FROM WORKS_ON C WHERE C.Essn = Ssn AND C.Pno = B.Pno)));
SELECT Pnumber, Pname, COUNT (*) FROM PROJECT, WORKS_ON WHERE Pnumber = Pno GROUP BY Pnumber, Pname HAVING COUNT (*) > 2;
SELECT Pnumber, Pname, COUNT (*) FROM PROJECT, WORKS_ON, EMPLOYEE WHERE Pnumber = Pno AND Ssn = Essn AND Dno = 5 GROUP BY Pnumber, Pname;
SELECT Dno, COUNT (*) FROM EMPLOYEE WHERE Salary>40000 AND Dno IN (SELECT Dno FROM EMPLOYEE GROUP BY Dno HAVING COUNT (*) > 5) GROUP BY Dno;
WITH BIGDEPTS (Dno) AS (SELECT Dno FROM EMPLOYEE GROUP BY Dno HAVING COUNT (*) > 5) SELECT Dno, COUNT (*) FROM EMPLOYEE WHERE Salary>40000 AND Dno IN BIGDEPTS GROUP BY Dno;
WITH RECURSIVE SUP_EMP (SupSsn, EmpSsn) AS (SELECT SupervisorSsn, Ssn FROM EMPLOYEE UNION SELECT E.Ssn, S.SupSsn FROM EMPLOYEE AS E, SUP_EMP AS S WHERE E.SupervisorSsn = S.EmpSsn) SELECT* FROM SUP_EMP;

Generalization. (a) Two entity types, CAR and TRUCK.
(b) Generalizing CAR and TRUCK into the superclass VEHICLE.





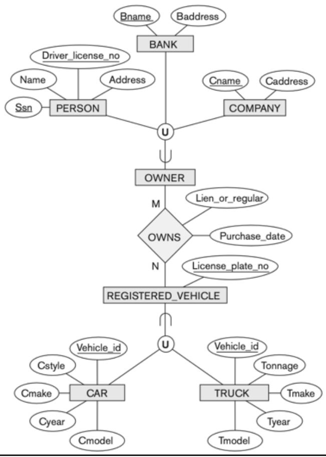
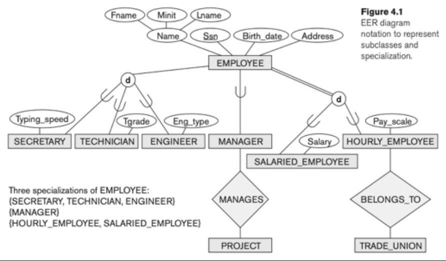


Figure 4.8
Two categories (union types): OWNER and REGISTERED_VEHICLE.



Three specializations of EMPLOYEE:
{SECRETARY, TECHNICIAN, ENGINEER}
{MANAGER}
{HOURLY_EMPLOYEE, SALARIED_EMPLOYEE}

Figure 4.1
EER diagram notation to represent subclasses and specialization.

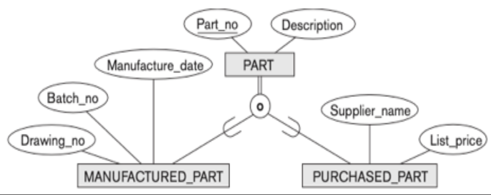


Figure 4.5
EER diagram notation for an overlapping (nondisjoint) specialization.

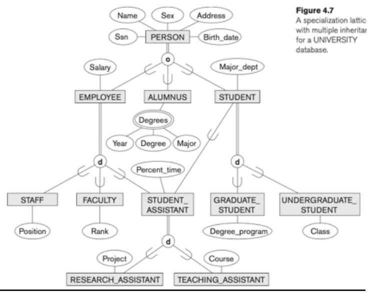


Figure 4.7
A specialization lattic
with multiple inheritar
for a UNIVERSITY
database.