NRI INSTITUTE OF INFORMATION SCIENCE & TECHNOLOGY BHOPAL

DEPARTMENT OF INFORMATION TECHNOLOGY

LAB MANUAL

DBMS LAB
(IT-405)

INFORMATION TECHNOLOGY (IT)

Experiment # 1

Objective: To learn the concept of E-R diagram.

Problem statement: Design an E-R diagram for the university database.

Underlying concept: Entity, relationship, attributes, E-R Modeling.

Entity-Relationship Model

The entity-relationship (E-R) data model is based on a perception of a real world that consists of a collection of basic objects, called *entities*, and of *relationships* among these objects. An entity is a "thing" or "object" in the real world that is distinguishable from other objects. For example, each person is an entity, and bank accounts can be considered as entities.

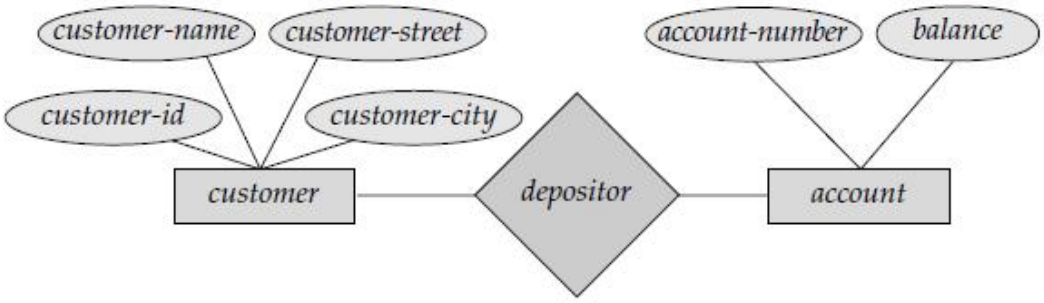
Entities are described in a database by a set of **attributes**. For example, the attributes *account-number* and *balance* may describe one particular account in a bank, and they form attributes of the *account* entity set. Similarly, attributes *customer-name*, *customer-street* address and *customer-city* may describe a *customer* entity. An extra attribute *customer-id* is used to uniquely identify customers (since it may be possible to have two customers with the same name, street address, and city). A unique customer identifier must be assigned to each customer. In the United States, many enterprises use the social-security number of a person (a unique number the U.S. government assigns to every person in the United States) as a customer identifier.

A **relationship** is an association among several entities. For example, a *depositor* relationship associates a customer with each account that she has. The set of all entities of the same type and the set of all relationships of the same type are termed an **entity set** and **relationship set**, respectively.

The overall logical structure (schema) of a database can be expressed graphically by an *E-R diagram*, which is built up from the following components:

- **Rectangles**, which represent entity sets
- **Ellipses**, which represent attributes
- **Diamonds**, which represent relationships among entity sets
- **Lines**, which link attributes to entity sets and entity sets to relationships

Each component is labeled with the entity or relationship that it represents. As an illustration, consider part of a database banking system consisting of customers and of the accounts that these customers have. Figure 1.2 shows the corresponding E-R diagram.



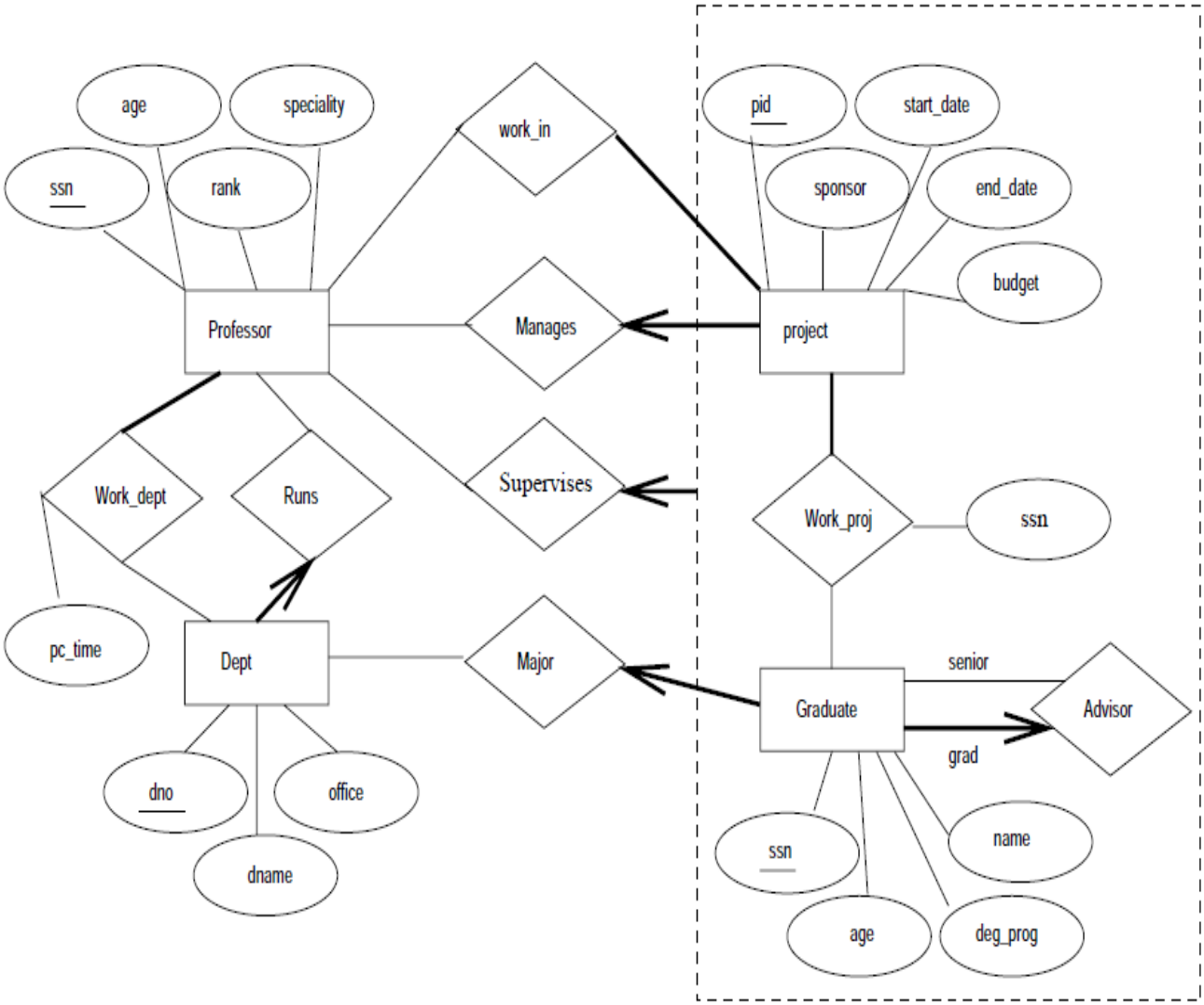
The E-R diagram indicates that there are two entity sets, *customer* and *account*, with attributes as outlined earlier. The diagram also shows a relationship *depositor* between customer and account. In addition to entities and relationships, the E-R model represents certain constraints to which the contents of a database must conform. One important constraint is **mapping cardinalities**, which express the number of entities to which another entity can be associated via a relationship set. For example, if each account must belong to only one customer, the E-R model can express that constraint.

Exercise:
Consider the following information about a university database:

1. Professors have an SSN, a name, an age, a rank, and a research specialty.
2. Projects have a project number, a sponsor name (e.g., NSF), a starting date, an ending date, and a budget.
3. Graduate students have an SSN, a name, an age, and a degree program (e.g., M.S. or Ph.D.).
4. Each project is managed by one professor (known as the project's principal investigator).
5. Each project is worked on by one or more professors (known as the project's co-investigators).
6. Professors can manage and/or work on multiple projects.
7. Each project is worked on by one or more graduate students (known as the project's research assistants).
8. When graduate students work on a project, a professor must supervise their work on the project. Graduate students can work on multiple projects, in which case they will have a (potentially different) supervisor for each one.
9. Departments have a department number, a department name, and a main office.
10. Departments have a professor (known as the chairman) who runs the department.
11. Professor work in one or more departments and for each department that they work in, a time percentage is associated with their job.
12. Graduate students have one major department in which they are working on their degree.
13. Each graduate student has another, more senior graduate student (known as a student advisor) who advises him or her on what courses to take.

Design and draw an ER diagram that captures the information about the university.
Use only the basic ER model here; that is, entities, relationships, and attributes.
Be sure to indicate any key and participation constraints.

Solution:


age
speciality
ssn
rank
work_in
pid
start_date
sponsor
end_date
budget
Professor
Manages
project
Work_dept
Runs
Supervises
Work_proj
ssn
pc_time
Dept
Major
senior
Graduate
Advisor
grad
dno
office
dname
ssn
name
age
deg_prog

Experiment # 2

Objective: To learn the concept of ternary relationship.

Problem statement: Design a relational database for the ternary relationship.

Underlying concept: Primary Key, Foreign key.

A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely an entity in the entity set. For example, the *customer-id* attribute of the entity set *customer* is sufficient to distinguish one *customer* entity from another. Thus, *customer-id* is a superkey. Similarly, the combination of *customer-name* and *customer-id* is a superkey for the entity set *customer*. The *customer-name* attribute of *customer* is not a superkey, because several people might have the same name. The concept of a superkey is not sufficient for our purposes, since, as we saw, a superkey may contain extraneous attributes. If *K* is a superkey, then so is any superset of *K*. We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called **candidate keys**.

It is possible that several distinct sets of attributes could serve as a candidate key. Suppose that a combination of *customer-name* and *customer-street* is sufficient to distinguish among members of the *customer* entity set. Then, both *{customer-id}* and *{customer-name, customer-street}* are candidate keys. Although the attributes *customerid* and *customer-name* together can distinguish *customer* entities, their combination does not form a candidate key, since the attribute *customer-id* alone is a candidate key.

We shall use the term **primary key** to denote a candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. A key (primary, candidate, and super) is a property of the entity set, rather than of the individual entities. Any two individual entities in the set are prohibited from having the same value on the key attributes at the same time. The designation of a key represents a constraint in the real-world enterprise being modeled.

A **foreign key** is a key used to link two tables together. This is sometimes called a referencing key. Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table. The relationship between two tables matches the Primary Key in one of the tables with a Foreign Key in the second table.

Exercise:

Suppose that we have a ternary relationship R between entity sets A, B, and C such that A has a key constraint and total participation and B has a key constraint; these are the only constraints. A has attributes a1 and a2, with a1 being the key; B and C are similar. R has no descriptive attributes. Write SQL statements that create tables corresponding to this information so as to capture as many of the constraints as possible.

Solution:
The following SQL statements create the corresponding relations.

```
CREATE TABLE A ( a1 CHAR(10),
a2 CHAR(10),
b1 CHAR(10),
c1 CHAR(10),
PRIMARY KEY (a1),
UNIQUE (b1),
FOREIGN KEY (b1) REFERENCES B,
FOREIGN KEY (c1) REFERENCES C )
CREATE TABLE B ( b1 CHAR(10),
b2 CHAR(10),
PRIMARY KEY (b1) )
CREATE TABLE C ( b1 CHAR(10),
c2 CHAR(10),
PRIMARY KEY (c1) )
```

The first SQL statement folds the relationship R into table A and thereby guarantees the participation constraint.

Experiment # 3

Objective: To learn how to convert an E-R diagram into a relational database.

Problem statement: Design a relational database for the ER diagram designed in exercise 1 by properly specifying the primary key and foreign key.

Underlying concept: Primary Key, Foreign key.

Solution:

The following SQL statements create the corresponding relations.

1. CREATE TABLE Professors (profssn CHAR(10),
name CHAR(64),
age NUMBER,
rank NUMBER,
speciality CHAR(64),
PRIMARY KEY (profssn))

2. CREATE TABLE Depts (dno NUMBER,
dname CHAR(64),
office CHAR(10),
PRIMARY KEY (dno))

3. CREATE TABLE Runs (dno NUMBER,
profssn CHAR(10),
PRIMARY KEY (dno, profssn),
FOREIGN KEY (profssn) REFERENCES Professors,
FOREIGN KEY (dno) REFERENCES Depts)

4. CREATE TABLE Work Dept (dno NUMBER,
profssn CHAR(10),
pc time NUMBER,
PRIMARY KEY (dno, profssn),
FOREIGN KEY (profssn) REFERENCES Professors,
FOREIGN KEY (dno) REFERENCES Depts)

Observe that we would need check constraints or assertions in SQL to enforce the rule that Professors work in at least one department.

5. CREATE TABLE Project (pid NUMBER,
sponsor CHAR(32),
start date DATE,
end date DATE,
budget FLOAT,

PRIMARY KEY (pid))

6. CREATE TABLE Graduates (grad ssn CHAR(10),
age NUMBER,
name CHAR(64),
deg prog CHAR(32),
major NUMBER,
PRIMARY KEY (grad ssn),
FOREIGN KEY (major) REFERENCES Depts)

Note that the Major table is not necessary since each Graduate has only one major and so this can be an attribute in the Graduates table.

7. CREATE TABLE Advisor (senior ssn CHAR(10),
grad ssn CHAR(10),
PRIMARY KEY (senior ssn, grad ssn),
FOREIGN KEY (senior ssn)
REFERENCES Graduates (grad ssn),
FOREIGN KEY (grad ssn) REFERENCES Graduates)

8. CREATE TABLE Manages (pid NUMBER,
profssn CHAR(10),
PRIMARY KEY (pid, profssn),
FOREIGN KEY (profssn) REFERENCES Professors,
FOREIGN KEY (pid) REFERENCES Projects)

9. CREATE TABLE Work In (pid NUMBER,
profssn CHAR(10),
PRIMARY KEY (pid, profssn),
FOREIGN KEY (profssn) REFERENCES Professors,
FOREIGN KEY (pid) REFERENCES Projects)

Observe that we cannot enforce the participation constraint for Projects in the Work In table without check constraints or assertions in SQL.

10. CREATE TABLE Supervises (profssn CHAR(10),
grad ssn CHAR(10),
pid NUMBER,
PRIMARY KEY (profssn, grad ssn, pid),
FOREIGN KEY (profssn) REFERENCES Professors,
FOREIGN KEY (grad ssn) REFERENCES Graduates,
FOREIGN KEY (pid) REFERENCES Projects)

Note that we do not need an explicit table for the Work Proj relation since every time a Graduate works on a Project, he or she must have a Supervisor.

Experiment # 4

Objective: To learn different clauses used to insert information using INSERT INTO statement.

Problem statement: Insert at least three tuples in each table of the university database.

Underlying concept: INSERT INTO clause.

The syntax for the **INSERT statement** when inserting a single record using the **VALUES keyword** is:

```
INSERT INTO table
(column1, column2, ... )
VALUES
(expression1, expression2, ... );
```

Or the syntax for the Oracle **INSERT statement** when inserting multiple records **using a SELECT statement** is:

```
INSERT INTO table
(column1, column2, ... )
SELECT expression1, expression2, ...
FROM source_table
WHERE conditions;
```

Parameters or Arguments

table is the table to insert the records into.

column1, *column2* are the columns in the *table* to insert values.

expression1, *expression2* are the values to assign to the columns in the table. So *column1* would be assigned the value of *expression1*, *column2* would be assigned the value of *expression2*, and so on.

source_table is the source table when inserting data from another table.

conditions are conditions that must be met for the records to be inserted.

Experiment # 5

Objective: To learn different clauses used to find information in a SELECT statement.

Problem statement: Find the information given below for the university database.

Underlying concept: various clauses with SELCT.

Exercise: Consider the following relations:
Student(snum: integer, sname: string, major: string, level: string, age: integer)
Class(name: string, meets at: string, room: string, fid: integer)
Enrolled(snum: integer, cname: string)
Faculty(fid: integer, fname: string, deptid: integer)

The meaning of these relations is straightforward; for example, Enrolled has one record per student-class pair such that the student is enrolled in the class.

Write the following queries in SQL. No duplicates should be printed in any of the answers.

1. Find the names of all Juniors (level = JR) who are enrolled in a class taught by I. Teach.
2. Find the age of the oldest student who is either a History major or enrolled in a course taught by I. Teach.
3. Find the names of all classes that either meet in room R128 or have five or more students enrolled.
4. Find the names of all students who are enrolled in two classes that meet at the same time.
5. Find the names of faculty members who teach in every room in which some class is taught.
6. Find the names of faculty members for whom the combined enrollment of the courses that they teach is less than five.
7. For each level, print the level and the average age of students for that level.
8. For all levels except JR, print the level and the average age of students for that level.
9. For each faculty member that has taught classes only in room R128, print the faculty member's name and the total number of classes she or he has taught.
10. Find the names of students enrolled in the maximum number of classes.
11. Find the names of students not enrolled in any class.
12. For each age value that appears in Students, find the level value that appears most often. For example, if there are more FR level students aged 18 than SR, JR, or SO students aged 18, you should print the pair (18, FR).

Answer: The answers are given below:

1. SELECT DISTINCT S.Sname
FROM Student S, Class C, Enrolled E, Faculty F

WHERE S.snum = E.snum AND E.cname = C.name AND C.fid = F.fid AND
F.fname = ‘I.Teach’ AND S.level = ‘JR’

2. SELECT MAX(S.age)
FROM Student S
WHERE (S.major = ‘History’)
OR S.snum IN (SELECT E.snum
FROM Class C, Enrolled E, Faculty F
WHERE E.cname = C.name AND C.fid = F.fid
AND F.fname = ‘I.Teach’)

3. SELECT C.name
FROM Class C
WHERE C.room = ‘R128’
OR C.name IN (SELECT E.cname
FROM Enrolled E
GROUP BY E.cname
HAVING COUNT (*) >= 5)

4. SELECT DISTINCT S.sname
FROM Student S
WHERE S.snum IN (SELECT E1.snum
FROM Enrolled E1, Enrolled E2, Class C1, Class C2
WHERE E1.snum = E2.snum AND E1.cname <> E2.cname
AND E1.cname = C1.name
AND E2.cname = C2.name AND C1.meets at = C2.meets at)

5. SELECT DISTINCT F.fname
FROM Faculty F
WHERE NOT EXISTS ((SELECT *
FROM Class C)
EXCEPT
(SELECTC1.room
FROM Class C1
WHERE C1.fid = F.fid))

6. SELECT DISTINCT F.fname
FROM Faculty F
WHERE 5 > (SELECT COUNT (E.snum)
FROM Class C, Enrolled E
WHERE C.name = E.cname
AND C.fid = F.fid)

7. SELECT S.level, AVG(S.age)
FROM Student S
GROUP BY S.level

8. SELECT S.level, AVG(S.age)
FROM Student S
WHERE S.level <> ‘JR’
GROUP BY S.level

9. SELECT F.fname, COUNT(*) AS CourseCount
FROM Faculty F, Class C
WHERE F.fid = C.fid
GROUP BY F.fid, F.fname
HAVING EVERY (C.room = ‘R128’)

10. SELECT DISTINCT S.sname
FROM Student S
WHERE S.snum IN (SELECT E.snum
FROM Enrolled E
GROUP BY E.snum
HAVING COUNT (*) >= ALL (SELECT COUNT (*)
FROM Enrolled E2
GROUP BY E2.snum))

11. SELECT DISTINCT S.sname
FROM Student S
WHERE S.snum NOT IN (SELECT E.snum
FROM Enrolled E)

12. SELECT S.age, S.level
FROM Student S
GROUP BY S.age, S.level,
HAVING S.level IN (SELECT S1.level
FROM Student S1
WHERE S1.age = S.age
GROUP BY S1.level, S1.age
HAVING COUNT (*) >= ALL (SELECT COUNT (*)
FROM Student S2
WHERE s1.age = S2.age
GROUP BY S2.level, S2.age))

Experiment # 6

Objective: To learn different clauses used to find information in a SELECT statement.

Problem statement: Find the information given below for the airline database.

Underlying concept: various clauses with SELCT.

Exercise: The following relations keep track of airline flight information:

Flights(flno: integer, from: string, to: string, distance: integer,
departs: time, arrives: time, price: real)
Aircraft(aid: integer, aname: string, cruisingrange: integer)
Certified(eid: integer, aid: integer)
Employees(eid: integer, ename: string, salary: integer)

Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft, and only pilots are certified to fly. Write each of the following queries in SQL.

1. Find the names of aircraft such that all pilots certified to operate them have salaries more than 80,000.
2. For each pilot who is certified for more than three aircraft, find the eid and the maximum cruising range of the aircraft for which she or he is certified.
3. Find the names of pilots whose salary is less than the price of the cheapest route from Los Angeles to Honolulu.
4. For all aircraft with cruising range over 1000 miles, find the name of the aircraft and the average salary of all pilots certified for this aircraft.
5. Find the names of pilots certified for some Boeing aircraft.
6. Find the aids of all aircraft that can be used on routes from Los Angeles to Chicago.
7. Identify the routes that can be piloted by every pilot who makes more than $100,000.
8. Print the enames of pilots who can operate planes with cruisingrange greater than 3000 miles but are not certified on any Boeing aircraft.
9. A customer wants to travel from Madison to New York with no more than two changes of flight. List the choice of departure times from Madison if the customer wants to arrive in New York by 6 p.m.
10. Compute the difference between the average salary of a pilot and the average salary of all employees (including pilots).
11. Print the name and salary of every non pilot whose salary is more than the average salary for pilots.
12. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles.
13. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles, but on at least two such aircrafts.
14. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles and who are certified on some Boeing aircraft.

Answer: The answers are given below:

1. SELECT DISTINCT A.aname
FROM Aircraft A
WHERE A.Aid IN (SELECT C.aid
FROM Certified C, Employees E
WHERE C.eid = E.eid AND
NOT EXISTS (SELECT *
FROM Employees E1
WHERE E1.eid = E.eid AND E1.salary < 80000))

2. SELECT C.eid, MAX (A.cruisingrange)
FROM Certified C, Aircraft A
WHERE C.aid = A.aid
GROUP BY C.eid
HAVING COUNT (*) > 3

3. SELECT DISTINCT E.ename
FROM Employees E
WHERE E.salary < (SELECT MIN (F.price)
FROM Flights F
WHERE F.from = 'Los Angeles' AND F.to = 'Honolulu')

4. Observe that aid is the key for Aircraft, but the question asks for aircraft names; we deal with this complication by using an intermediate relation Temp:

SELECT Temp.name, Temp.AvgSalary
FROM (SELECT A.aid, A.aname AS name,
AVG (E.salary) AS AvgSalary
FROM Aircraft A, Certified C, Employees E
WHERE A.aid = C.aid AND
C.eid = E.eid AND A.cruisingrange > 1000
GROUP BY A.aid, A.aname) AS Temp

5. SELECT DISTINCT E.ename
FROM Employees E, Certified C, Aircraft A
WHERE E.eid = C.eid AND
C.aid = A.aid AND
A.aname LIKE 'Boeing%'

6. SELECT A.aid
FROM Aircraft A
WHERE A.cruisingrange > (SELECT MIN (F.distance)
FROM Flights F

```
WHERE F.from = ‘Los Angeles’ AND F.to = ‘Chicago’ )
```

7.
```
SELECT DISTINCT F.from, F.to
FROM Flights F
WHERE NOT EXISTS ( SELECT *
FROM Employees E
WHERE E.salary > 100000
AND
NOT EXISTS (SELECT *
FROM Aircraft A, Certified C
WHERE A.cruisingrange > F.distance
AND E.eid = C.eid
AND A.aid = C.aid) )
```

8.
```
SELECT DISTINCT E.ename
FROM Employees E
WHERE E.eid IN ( ( SELECT C.eid
FROM Certified C
WHERE EXISTS ( SELECT A.aid
FROM Aircraft A
WHERE A.aid = C.aid
AND A.cruisingrange > 3000 )
AND
NOT EXISTS ( SELECT A1.aid
FROM Aircraft A1
WHERE A1.aid = C.aid
AND A1.aname LIKE ‘Boeing%’ ))
```

9.
```
SELECT F.departs
FROM Flights F
WHERE F.flno IN ( ( SELECT F0.flno
FROM Flights F0
WHERE F0.from = ‘Madison’ AND F0.to = ‘New York’
AND F0.arrives < ‘18:00’ )
UNION
( SELECT F0.flno
FROM Flights F0, Flights F1
WHERE F0.from = ‘Madison’ AND F0.to <> ‘New York’
AND F0.to = F1.from AND F1.to = ‘New York’
AND F1.departs > F0.arrives
AND F1.arrives < ‘18:00’ )
UNION
( SELECT F0.flno
FROM Flights F0, Flights F1, Flights F2
WHERE F0.from = ‘Madison’
AND F0.to = F1.from
```

```
AND F1.to = F2.from
AND F2.to = ‘New York’
AND F0.to <> ‘New York’
AND F1.to <> ‘New York’
AND F1.departs > F0.arrives
AND F2.departs > F1.arrives
AND F2.arrives < ‘18:00’ ))
```

```
10. SELECT Temp1.avg - Temp2.avg
FROM (SELECT AVG (E.salary) AS avg
FROM Employees E
WHERE E.eid IN (SELECT DISTINCT C.eid
FROM Certified C )) AS Temp1,
(SELECT AVG (E1.salary) AS avg
FROM Employees E1 ) AS Temp2
```

```
11. SELECT E.ename, E.salary
FROM Employees E
WHERE E.eid NOT IN ( SELECT DISTINCT C.eid
FROM Certified C )
SQL: Queries, Constraints, Triggers 53
AND E.salary > ( SELECT AVG (E1.salary)
FROM Employees E1
WHERE E1.eid IN
( SELECT DISTINCT C1.eid
FROM Certified C1 ) )
```

```
12. SELECT E.ename
FROM Employees E, Certified C, Aircraft A
WHERE C.aid = A.aid AND E.eid = C.eid
GROUP BY E.eid, E.ename
HAVING EVERY (A.cruisingrange > 1000)
```

```
13. SELECT E.ename
FROM Employees E, Certified C, Aircraft A
WHERE C.aid = A.aid AND E.eid = C.eid
GROUP BY E.eid, E.ename
HAVING EVERY (A.cruisingrange > 1000) AND COUNT (*) > 1
```

```
14. SELECT E.ename
FROM Employees E, Certified C, Aircraft A
WHERE C.aid = A.aid AND E.eid = C.eid
GROUP BY E.eid, E.ename
HAVING EVERY (A.cruisingrange > 1000) AND ANY (A.aname = ’Boeing’)
```

Experiment # 7

Objective: To learn different clauses used to find information in a SELECT statement.

Problem statement: Find the information given below for the insurance database.

Underlying concept: various clauses with SELCT.

Exercise:

Consider the insurance database, where the primary keys are underlined.

person (driver-id, name, address)
car (license, model, year)
accident (report-number, date, location)
owns (driver-id, license)
participated (driver-id, car, report-number, damage-amount)

Construct the following SQL queries for this relational database.

a. Find the total number of people who owned cars that were involved in accidents in 1989.

b. Find the number of accidents in which the cars belonging to "John Smith" were involved.

c. Add a new accident to the database; assume any values for required attributes.

d. Delete the Mazda belonging to "John Smith".

e. Update the damage amount for the car with license number "AABB2000" in the accident with report number "AR2197" to $3000.

Answer: Note: The participated relation relates drivers, cars, and accidents.

a. Find the total number of people who owned cars that were involved in accidents in 1989.

Note: this is not the same as the total number of accidents in 1989. We must count people with several accidents only once.

```
select count (distinct name)
from accident, participated, person
where accident.report-number = participated.report-number
        and participated.driver-id = person.driver-id
        and date between date '1989-00-00' and date '1989-12-31'
```

b. Find the number of accidents in which the cars belonging to "John Smith" were involved.

```
select count (distinct *)
from accident
where exists
        (select *
        from participated, person
        where participated.driver-id = person.driver-id
        and person.name = 'John Smith'
        and accident.report-number = participated.report-number)
```

c. Add a new accident to the database; assume any values for required attributes. assume "Jones" owns one Toyota. First we must find the license of the given car. Then the participated and accident relations must be updated in order to both record the accident and tie it to the given car. We assume values "Berkeley" for location, '2001-09-01' for date and date, 4007 for reportnumber and 3000 for damage amount.

```
insert into accident
values (4007, '2001-09-01', 'Berkeley')


insert into participated
        select o.driver-id, c.license, 4007, 3000
        from person p, owns o, car c
        where p.name = 'Jones' and p.driver-id = o.driver-id and
                o.license = c.license and c.model = 'Toyota'
```

d. Delete the Mazda belonging to "John Smith".

Since model is not a key of the car relation, we can either assume that only one of John Smith's cars is a Mazda, or delete all of John Smith's Mazdas (the query is the same). Again assume name is a key for person.

```
delete car
where model = 'Mazda' and license in
        (select license
        from person p, owns o
        where p.name = 'John Smith' and p.driver-id = o.driver-id)
```

Note: The owns, accident and participated records associated with the Mazda still exist.

e. Update the damage amount for the car with license number "AABB2000" in the accident with report number "AR2197" to $3000.

```
update participated
set damage-amount = 3000
where report-number = "AR2197" and driver-id in
        (select driver-id
        from owns
        where license = "AABB2000")
```

Experiment # 8

Objective: To learn different clauses used to find information in a SELECT statement.

Problem statement: Find the information given below for the employee database.

Underlying concept: various clauses with SELCT.

Exercise:

Consider the employee database, where the primary keys are underlined.

employee (employee-name, street, city)
works (employee-name, company-name, salary)
company (company-name, city)
manages (employee-name, manager-name)

Give an expression in SQL for each of the following queries.

a. Find the names of all employees who work for First Bank Corporation.

b. Find the names and cities of residence of all employees who work for First Bank Corporation.

c. Find the names, street addresses, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000.

d. Find all employees in the database who live in the same cities as the companies for which they work.

e. Find all employees in the database who live in the same cities and on the same streets as do their managers.

f. Find all employees in the database who do not work for First Bank Corporation.

g. Find all employees in the database who earn more than each employee of Small Bank Corporation.

h. Assume that the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.

i. Find all employees who earn more than the average salary of all employees of their company.

j. Find the company that has the most employees.

k. Find the company that has the smallest payroll.

l. Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

Answer:

a. Find the names of all employees who work for First Bank Corporation.

```
select employee-name
from works
where company-name = 'First Bank Corporation'
```

b. Find the names and cities of residence of all employees who work for First Bank Corporation.

```
select e.employee-name, city
from employee e, works w
where w.company-name = 'First Bank Corporation' and
```

w.employee-name = e.employee-name

c. Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000.

If people may work for several companies, the following solution will only list those who earn more than $10,000 per annum from “First Bank Corporation” alone.

```
select *
from employee
where employee-name in
        (select employee-name
        from works
        where company-name = ’First Bank Corporation’ and salary ¿ 10000)
```

As in the solution to the previous query, we can use a join to solve this one also.

d. Find all employees in the database who live in the same cities as the companies for which they work.

```
select e.employee-name
from employee e, works w, company c
where e.employee-name = w.employee-name and e.city = c.city
        and w.company -name = c.company -name
```

e. Find all employees in the database who live in the same cities and on the same streets as do their managers.

```
select P.employee-name
from employee P, employee R, manages M
where P.employee-name = M.employee-name and
        M.manager-name = R.employee-name and
        P.street = R.street and P.city = R.city
```

f. Find all employees in the database who do not work for First Bank Corporation.

The following solution assumes that all people work for exactly one company.

```
select employee-name
from works
where company-name = ’First Bank Corporation’
```

If one allows people to appear in the database (e.g. in employee) but not appear in works, or if people may have jobs with more than one company, the solution is slightly more complicated.

```
select employee-name
from employee
where employee-name not in
        (select employee-name
        from works
        where company-name = ’First Bank Corporation’)
```

g. Find all employees in the database who earn more than every employee of Small Bank Corporation.

The following solution assumes that all people work for at most one company

```
select employee-name
from works
where salary > all
        (select salary
```

```
        from works
        where company-name = 'Small Bank Corporation')
```

If people may work for several companies and we wish to consider the total earnings of each person, the problem is more complex. It can be solved by using a nested subquery, but we illustrate below how to solve it using the **with** clause.

```
with emp-total-salary as
        (select employee-name, sum(salary) as total-salary
        from works
        group by employee-name
        )
select employee-name
from emp-total-salary
where total-salary > all
        (select total-salary
        from emp-total-salary, works
        where works.company-name = 'Small Bank Corporation' and
        emp-total-salary.employee-name = works.employee-name
        )
```

h. Assume that the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.

The simplest solution uses the **contains** comparison which was included in the original System R Sequel language but is not present in the subsequent SQL versions.

```
select T.company-name
from company T
where (select R.city
        from company R
        where R.company-name = T.company-name)
   contains
        (select S.city
        from company S
        where S.company-name = 'Small Bank Corporation')
```

Below is a solution using standard SQL.

```
select S.company-name
from company S
where not exists ((select city
                   from company
                   where company-name = 'Small Bank Corporation')
                  except
                   (select city
                   from company T
                   where S.company-name = T.company-name))
```

i. Find all employees who earn more than the average salary of all employees of their company.
The following solution assumes that all people work for at most one company.

```
select employee-name
from works T
where salary > (select avg (salary)
                from works S
                where T.company-name = S.company-name)
```

j. Find the company that has the most employees.

```
select company-name
from works
group by company-name
having count (distinct employee-name) >= all
    (select count (distinct employee-name)
    from works
    group by company-name)
```

k. Find the company that has the smallest payroll.

```
select company-name
from works
group by company-name
having sum (salary) <= all (select sum (salary)
                            from works
                            group by company-name)
```

l. Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

```
select company-name
from works
group by company-name
having avg (salary) > (select avg (salary)
                       from works
                       where company-name = 'First Bank Corporation')
```

Experiment # 9

Objective: To learn different clauses used to update information.

Problem statement: Modify the information given below for the employee database.

Underlying concept: various clauses with UPDATE.

Exercise:

Consider the employee database.

employee (employee-name, street, city)
works (employee-name, company-name, salary)
company (company-name, city)
manages (employee-name, manager-name)

Give an expression in SQL for each of the following queries.
a. Modify the database so that Jones now lives in Newtown.
b. Give all employees of First Bank Corporation a 10 percent raise.
c. Give all managers of First Bank Corporation a 10 percent raise.
d. Give all managers of First Bank Corporation a 10 percent raise unless the salary becomes greater than $100,000; in such cases, give only a 3 percent raise.
e. Delete all tuples in the works relation for employees of Small Bank Corporation.
Answer: The solution for part 0.a assumes that each person has only one tuple in the employee relation. The solutions to parts 0.c and 0.d assume that each person works for at most one company.
a. Modify the database so that Jones now lives in Newtown.

```
update employee
set city = 'Newton'
where person-name = 'Jones'
```

b. Give all employees of First Bank Corporation a 10-percent raise.

```
update works
set salary = salary * 1.1
where company-name = 'First Bank Corporation'
```

c. Give all managers of First Bank Corporation a 10-percent raise.

```
update works
set salary = salary * 1.1
where employee-name in (select manager-name
                        from manages)
                    and company-name = 'First Bank Corporation'
```

d. Give all managers of First Bank Corporation a 10-percent raise unless the salary becomes greater than $100,000; in such cases, give only a 3-percent raise.

```
update works T
set T.salary = T.salary * 1.03
where T.employee-name in (select manager-name
                                from manages)
                    and T.salary * 1.1 > 100000
```

```
    and T.company-name = 'First Bank Corporation'

    update works T
    set T.salary = T.salary * 1.1
    where T.employee-name in (select manager-name
                              from manages)
          and T.salary * 1.1 <= 100000
          and T.company-name = 'First Bank Corporation'
```

SQL-92 provides a **case** operation (see Exercise 4.11), using a more concise solution is:-

```
    update works T
    set T.salary = T.salary *
          (case
                when (T.salary * 1.1 > 100000) then 1.03
                else 1.1
          )
    where T.employee-name in (select manager-name
                              from manages) and
                              T.company-name = 'First Bank Corporation'
```

e. Delete all tuples in the works relation for employees of Small Bank Corporation.

```
    delete works
    where company-name = 'Small Bank Corporation'
```

Experiment # 10

Objective: To learn constraints on database updation.

Problem statement: Write **check** conditions for the employee database.

Underlying concept: Check clause.

Exercise:

Write **check** conditions for the employee database schema to ensure that:

a. Every employee works for a company located in the same city as the city in which the employee lives.

b. No employee earns a salary higher than that of his manager.

Answer:

a. check condition for the *works* table:-

```
check((employee-name, company-name) in
        (select e.employee-name, c.company-name
        from employee e, company c
        where e.city = c.city
        )
        )
```

b. check condition for the *works* table:-

```
check(
        salary < all
        (select manager-salary
        from (select manager-name, manages.employee-name as emp-name,
                salary as manager-salary
                from works, manages
                where works.employee-name = manages.manager-name)
        where employee-name = emp-name
        )
        )
```