CHAPTER 8: SQL-99: SCHEMA DEFINITION, CONSTRAINTS, QUERIES, AND VIEWS

Answers to Selected Exercises

8. 7 Consider the database shown in Figure 1.2, whose schema is shown in Figure 2.1. What are the referential integrity constraints that should hold on the schema? Write appropriate SQL DDL statements to define the database.

Answer:

The following referential integrity constraints should hold (we use the notation:
R.(A1, ..., An) --> S.(B1, ..., Bn)
to represent a foreign key from the attributes A1, ..., An of R (the referencing relation) to S (the referenced relation)):
PREREQUISITE.(CourseNumber) --> COURSE.(CourseNumber)
PREREQUISITE.(PrerequisiteNumber) --> COURSE.(CourseNumber)
SECTION.(CourseNumber) --> COURSE.(CourseNumber)
GRADE_REPORT.(StudentNumber) --> STUDENT.(StudentNumber)
GRADE_REPORT.(SectionIdentifier) --> SECTION.(SectionIdentifier)
One possible set of CREATE TABLE statements to define the database is given below.

```
CREATE TABLE STUDENT ( Name VARCHAR(30) NOT NULL,
StudentNumber INTEGER NOT NULL,
Class CHAR NOT NULL,
Major CHAR(4),
PRIMARY KEY (StudentNumber) );
CREATE TABLE COURSE ( CourseName VARCHAR(30) NOT NULL,
CourseNumber CHAR(8) NOT NULL,
CreditHours INTEGER,
Department CHAR(4),
PRIMARY KEY (CourseNumber),
UNIQUE (CourseName) );
CREATE TABLE PREREQUISITE ( CourseNumber CHAR(8) NOT NULL,
PrerequisiteNumber CHAR(8) NOT NULL,
PRIMARY KEY (CourseNumber, PrerequisiteNumber),
FOREIGN KEY (CourseNumber) REFERENCES
COURSE (CourseNumber),
FOREIGN KEY (PrerequisiteNumber) REFERENCES
COURSE (CourseNumber) );
CREATE TABLE SECTION ( SectionIdentifier INTEGER NOT NULL,
CourseNumber CHAR(8) NOT NULL,
Semester VARCHAR(6) NOT NULL,
Year CHAR(4) NOT NULL,
Instructor VARCHAR(15),
PRIMARY KEY (SectionIdentifier),
FOREIGN KEY (CourseNumber) REFERENCES
COURSE (CourseNumber) );
CREATE TABLE GRADE_REPORT ( StudentNumber INTEGER NOT NULL,
SectionIdentifier INTEGER NOT NULL,
Grade CHAR,
PRIMARY KEY (StudentNumber, SectionIdentifier),
FOREIGN KEY (StudentNumber) REFERENCES
STUDENT (StudentNumber),
FOREIGN KEY (SectionIdentifier) REFERENCES
SECTION (SectionIdentifier) );
```

8. 8 Repeat Exercise 8.7, but use the AIRLINE schema of Figure 5.8.
Answer:

The following referential integrity constraints should hold:
FLIGHT_LEG.(FLIGHT_NUMBER) --> FLIGHT.(NUMBER)
FLIGHT_LEG.(DEPARTURE_AIRPORT_CODE) --> AIRPORT.(AIRPORT_CODE)
FLIGHT_LEG.(ARRIVAL_AIRPORT_CODE) --> AIRPORT.(AIRPORT_CODE)
LEG_INSTANCE.(FLIGHT_NUMBER, LEG_NUMBER) -->
FLIGHT_LEG.(FLIGHT_NUMBER, LEG_NUMBER)
LEG_INSTANCE.(AIRPLANE_ID) --> AIRPLANE.(AIRPLANE_ID)
LEG_INSTANCE.(DEPARTURE_AIRPORT_CODE) --> AIRPORT.(AIRPORT_CODE)
LEG_INSTANCE.(ARRIVAL_AIRPORT_CODE) --> AIRPORT.(AIRPORT_CODE)
FARES.(FLIGHT_NUMBER) --> FLIGHT.(NUMBER)
CAN_LAND.(AIRPLANE_TYPE_NAME) --> AIRPLANE_TYPE.(TYPE_NAME)
CAN_LAND.(AIRPORT_CODE) --> AIRPORT.(AIRPORT_CODE)
AIRPLANE.(AIRPLANE_TYPE) --> AIRPLANE_TYPE.(TYPE_NAME)
SEAT_RESERVATION.(FLIGHT_NUMBER, LEG_NUMBER, DATE) -->
LEG_INSTANCE.(FLIGHT_NUMBER, LEG_NUMBER, DATE)
One possible set of CREATE TABLE statements to define the database is given below.

```
CREATE TABLE AIRPORT ( AIRPORT_CODE CHAR(3) NOT NULL,
NAME VARCHAR(30) NOT NULL,
CITY VARCHAR(30) NOT NULL,
STATE VARCHAR(30),
PRIMARY KEY (AIRPORT_CODE) );
CREATE TABLE FLIGHT ( NUMBER VARCHAR(6) NOT NULL,
AIRLINE VARCHAR(20) NOT NULL,
WEEKDAYS VARCHAR(10) NOT NULL,
PRIMARY KEY (NUMBER) );
CREATE TABLE FLIGHT_LEG ( FLIGHT_NUMBER VARCHAR(6) NOT NULL,
LEG_NUMBER INTEGER NOT NULL,
DEPARTURE_AIRPORT_CODE CHAR(3) NOT NULL,
SCHEDULED_DEPARTURE_TIME TIMESTAMP WITH TIME ZONE,
ARRIVAL_AIRPORT_CODE CHAR(3) NOT NULL,
SCHEDULED_ARRIVAL_TIME TIMESTAMP WITH TIME ZONE,
PRIMARY KEY (FLIGHT_NUMBER, LEG_NUMBER),
FOREIGN KEY (FLIGHT_NUMBER) REFERENCES FLIGHT (NUMBER),
FOREIGN KEY (DEPARTURE_AIRPORT_CODE) REFERENCES
AIRPORT (AIRPORT_CODE),
FOREIGN KEY (ARRIVAL_AIRPORT_CODE) REFERENCES
AIRPORT (AIRPORT_CODE) );
CREATE TABLE LEG_INSTANCE ( FLIGHT_NUMBER VARCHAR(6) NOT NULL,
LEG_NUMBER INTEGER NOT NULL,
LEG_DATE DATE NOT NULL,
NO_OF_AVAILABLE_SEATS INTEGER,
AIRPLANE_ID INTEGER,
DEPARTURE_AIRPORT_CODE CHAR(3),
DEPARTURE_TIME TIMESTAMP WITH TIME ZONE,
ARRIVAL_AIRPORT_CODE CHAR(3),
ARRIVAL_TIME TIMESTAMP WITH TIME ZONE,
PRIMARY KEY (FLIGHT_NUMBER, LEG_NUMBER, LEG_DATE),
FOREIGN KEY (FLIGHT_NUMBER, LEG_NUMBER) REFERENCES
FLIGHT_LEG (FLIGHT_NUMBER, LEG_NUMBER),
FOREIGN KEY (AIRPLANE_ID) REFERENCES
AIRPLANE (AIRPLANE_ID),
FOREIGN KEY (DEPARTURE_AIRPORT_CODE) REFERENCES
AIRPORT (AIRPORT_CODE),
FOREIGN KEY (ARRIVAL_AIRPORT_CODE) REFERENCES
```

```
AIRPORT (AIRPORT_CODE) );
CREATE TABLE FARES ( FLIGHT_NUMBER VARCHAR(6) NOT NULL,
FARE_CODE VARCHAR(10) NOT NULL,
AMOUNT DECIMAL(8,2) NOT NULL,
RESTRICTIONS VARCHAR(200),
PRIMARY KEY (FLIGHT_NUMBER, FARE_CODE),
FOREIGN KEY (FLIGHT_NUMBER) REFERENCES FLIGHT (NUMBER) );
CREATE TABLE AIRPLANE_TYPE ( TYPE_NAME VARCHAR(20) NOT NULL,
MAX_SEATS INTEGER NOT NULL,
COMPANY VARCHAR(15) NOT NULL,
PRIMARY KEY (TYPE_NAME) );
CREATE TABLE CAN_LAND ( AIRPLANE_TYPE_NAME VARCHAR(20) NOT NULL,
AIRPORT_CODE CHAR(3) NOT NULL,
PRIMARY KEY (AIRPLANE_TYPE_NAME, AIRPORT_CODE),
FOREIGN KEY (AIRPLANE_TYPE_NAME) REFERENCES
AIRPLANE_TYPE (TYPE_NAME),
FOREIGN KEY (AIRPORT_CODE) REFERENCES
AIRPORT (AIRPORT_CODE) );
CREATE TABLE AIRPLANE ( AIRPLANE_ID INTEGER NOT NULL,
TOTAL_NUMBER_OF_SEATS INTEGER NOT NULL,
AIRPLANE_TYPE VARCHAR(20) NOT NULL,
PRIMARY KEY (AIRPLANE_ID),
FOREIGN KEY (AIRPLANE_TYPE) REFERENCES AIRPLANE_TYPE (TYPE_NAME) );
CREATE TABLE SEAT_RESERVATION ( FLIGHT_NUMBER VARCHAR(6) NOT NULL,
LEG_NUMBER INTEGER NOT NULL,
LEG_DATE DATE NOT NULL,
SEAT_NUMBER VARCHAR(4),
CUSTOMER_NAME VARCHAR(30) NOT NULL,
CUSTOMER_PHONE CHAR(12),
PRIMARY KEY (FLIGHT_NUMBER, LEG_NUMBER, LEG_DATE, SEAT_NUMBER),
FOREIGN KEY (FLIGHT_NUMBER, LEG_NUMBER, LEG_DATE) REFERENCES
LEG_INSTANCE (FLIGHT_NUMBER, LEG_NUMBER, LEG_DATE) );
```

8.9 Consider the LIBRARY relational database schema of Figure 6.14. Choose the appropriate action (reject, cascade, set to null, set to default) for each referential integrity constraint, both for DELETE of a referenced tuple, and for UPDATE of a primary key attribute value in a referenced tuple. Justify your choices.

Answer:

Below are possible choices. In general, if it is not clear which action to choose, REJECT should be chosen, since it will not permit automatic changes to happen (by update propagation) that may be unintended.
BOOK_AUTHORS.(BookId) --> BOOK.(BookId)
CASCADE on both DELETE or UPDATE (since this corresponds to a multi-valued attribute of BOOK (see the solution to Exercise 6.27); hence, if a BOOK is deleted, or the value of its BookId is updated (changed), the deletion or change is automatically propagated to the referencing BOOK_AUTHORS tuples)
BOOK.(PublisherName) --> PUBLISHER.(Name)
REJECT on DELETE (we should not delete a PUBLISHER tuple which has existing BOOK tuples that reference the PUBLISHER)
CASCADE on UPDATE (if a PUBLISHER's Name is updated, the change should be propagated automatically to all referencing BOOK tuples)
BOOK_LOANS.(BookId) --> BOOK.(BookId)
CASCADE on both DELETE or UPDATE (if a BOOK is deleted, or the value of its BookId is updated (changed), the deletion or change is automatically propagated to the referencing

BOOK_LOANS tuples) (Note: One could also choose REJECT on DELETE)
BOOK_COPIES.(BookId) --> BOOK.(BookId)
CASCADE on both DELETE or UPDATE (if a BOOK is deleted, or the value of its BookId is updated (changed), the deletion or change is automatically propagated to the referencing BOOK_COPIES tuples)
BOOK_LOANS.(CardNo) --> BORROWER.(CardNo)
CASCADE on both DELETE or UPDATE (if a BORROWER tuple is deleted, or the value of its CardNo is updated (changed), the deletion or change is automatically propagated to the referencing BOOK_LOANS tuples) (Note: One could also choose REJECT on DELETE, with the idea that if a BORROWER is deleted, it is necessary first to make a printout of all BOOK_LOANS outstanding before deleting the BORROWER; in this case, the tuples in BOOK_LOANS that reference the BORROWER being deleted would first be explicitly deleted after making the printout, and before the BORROWER is deleted)
BOOK_COPIES.(BranchId) --> LIBRARY_BRANCH.(BranchId)
CASCADE on both DELETE or UPDATE (if a LIBRARY_BRANCH is deleted, or the value of its BranchId is updated (changed), the deletion or change is automatically propagated to the referencing BOOK_COPIES tuples) (Note: One could also choose REJECT on DELETE)
BOOK_LOANS.(BranchId) --> LIBRARY_BRANCH.(BranchId)
CASCADE on both DELETE or UPDATE (if a LIBRARY_BRANCH is deleted, or the value of its BranchId is updated (changed), the deletion or change is automatically propagated to the referencing BOOK_LOANS tuples) (Note: One could also choose REJECT on DELETE)

8.10 Write appropriate SQL DDL statements for declaring the LIBRARY relational database schema of Figure 6.14. Specify appropriate keys and referential triggered actions.

Answer: One possible set of CREATE TABLE statements is given below:

```
CREATE TABLE BOOK ( BookId CHAR(20) NOT NULL,
Title VARCHAR(30) NOT NULL,
PublisherName VARCHAR(20),
PRIMARY KEY (BookId),
FOREIGN KEY (PublisherName) REFERENCES PUBLISHER (Name) ON UPDATE
CASCADE );
CREATE TABLE BOOK_AUTHORS ( BookId CHAR(20) NOT NULL,
AuthorName VARCHAR(30) NOT NULL,
PRIMARY KEY (BookId, AuthorName),
FOREIGN KEY (BookId) REFERENCES BOOK (BookId)
ON DELETE CASCADE ON UPDATE CASCADE );
CREATE TABLE PUBLISHER ( Name VARCHAR(20) NOT NULL,
Address VARCHAR(40) NOT NULL,
Phone CHAR(12),
PRIMARY KEY (Name) );
CREATE TABLE BOOK_COPIES ( BookId CHAR(20) NOT NULL,
BranchId INTEGER NOT NULL,
No_Of_Copies INTEGER NOT NULL,
PRIMARY KEY (BookId, BranchId),
FOREIGN KEY (BookId) REFERENCES BOOK (BookId)
ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (BranchId) REFERENCES BRANCH (BranchId)
ON DELETE CASCADE ON UPDATE CASCADE );
CREATE TABLE BORROWER ( CardNo INTEGER NOT NULL,
Name VARCHAR(30) NOT NULL,
Address VARCHAR(40) NOT NULL,
```

```
Phone CHAR(12),
PRIMARY KEY (CardNo) );
CREATE TABLE BOOK_LOANS ( CardNo INTEGER NOT NULL,
BookId CHAR(20) NOT NULL,
BranchId INTEGER NOT NULL,
DateOut DATE NOT NULL,
DueDate DATE NOT NULL,
PRIMARY KEY (CardNo, BookId, BranchId),
FOREIGN KEY (CardNo) REFERENCES BORROWER (CardNo)
ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (BranchId) REFERENCES LIBRARY_BRANCH (BranchId)
ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (BookId) REFERENCES BOOK (BookId)
ON DELETE CASCADE ON UPDATE CASCADE );
CREATE TABLE LIBRARY_BRANCH ( BranchId INTEGER NOT NULL,
BranchName VARCHAR(20) NOT NULL,
Address VARCHAR(40) NOT NULL,
PRIMARY KEY (BranchId) );
```

8.11 Write SQL queries for the LIBRARY database queries given in Exercise 6.18.

Answer:

Below, we give one possible SQL query for each request. Other queries are also possible.

(a) How many copies of the book titled The Lost Tribe are owned by the library branch whose name is "Sharpstown"?

```
SELECT NoOfCopies
FROM ( (BOOK NATURAL JOIN BOOK_COPIES ) NATURAL JOIN
LIBRARY_BRANCH )
WHERE Title='The Lost Tribe' AND BranchName='Sharpstown'
```

(b) How many copies of the book titled The Lost Tribe are owned by each library branch?

```
SELECT BranchName, NoOfCopies
FROM ( (BOOK NATURAL JOIN BOOK_COPIES ) NATURAL JOIN
LIBRARY_BRANCH )
WHERE Title='The Lost Tribe'
```

(c) Retrieve the names of all borrowers who do not have any books checked out.

```
SELECT Name
FROM BORROWER B
WHERE NOT EXIST ( SELECT *
FROM BOOK_LOANS L
WHERE B.CardNo = L.CardNo )
```

(d) For each book that is loaned out from the "Sharpstown" branch and whose DueDate is today, retrieve the book title, the borrower's name, and the borrower's address.

```
SELECT B.Title, R.Name, R.Address
FROM BOOK B, BORROWER R, BOOK_LOANS BL, LIBRARY_BRANCH LB
WHERE LB.BranchName='Sharpstown' AND LB.BranchId=BL.BranchId AND
BL.DueDate='today' AND BL.CardNo=R.CardNo AND BL.BookId=B.BookId
```

(e) For each library branch, retrieve the branch name and the total number of books loaned out from that branch.

```
SELECT L.BranchName, COUNT(*)
FROM BOOK_COPIES B, LIBRARY_BRANCH L
WHERE B.BranchId = L.BranchId
GROUP BY L.BranchName
```

(f) Retrieve the names, addresses, and number of books checked out for all borrowers who have more than five books checked out.

```
SELECT B.CardNo, B.Name, B.Address, COUNT(*)
FROM BORROWER B, BOOK_LOANS L
WHERE B.CardNo = L.CardNo
GROUP BY B.CardNo
HAVING COUNT(*) > 5
```

(g) For each book authored (or co-authored) by "Stephen King", retrieve the title and the number of copies owned by the library branch whose name is "Central".

```
SELECT TItle, NoOfCopies
FROM ( ( (BOOK_AUTHORS NATURAL JOIN BOOK) NATURAL JOIN
BOOK_COPIES)
NATURAL JOIN LIBRARY_BRANCH)
WHERE Author_Name = 'Stephen King' and BranchName = 'Central'
```

8.12 How can the key and foreign key constraints be enforced by the DBMS? Is the enforcement technique you suggest difficult to implement? Can the constraint checks be executed in an efficient manner when updates are applied to the database?

Answer:

One possible technique that is often used to check efficiently for the key constraint is to create an index on the combination of attributes that form each key (primary or secondary). Before inserting a new record (tuple), each index is searched to check that no value currently exists in the index that matches the key value in the new record. If this is the case, the record is inserted successfully.

For checking the foreign key constraint, an index on the primary key of each referenced relation will make this check relatively efficient. Whenever a new record is inserted in a referencing relation , its foreign key value is used to search the index for the primary key of the referenced relation, and if the referenced record exists, then the new record can be successfully inserted in the referencing relation.

For deletion of a referenced record, it is useful to have an index on the foreign key of each referencing relation so as to be able to determine efficiently whether any records reference the record being deleted.

If the indexes described above do not exist, and no alternative access structure (for example, hashing) is used in their place, then it is necessary to do linear searches to check for any of the above constraints, making the checks quite inefficient.

8.13 Specify the queries of Exercise 6.16 in SQL. Show the result of each query if applied to the COMPANY database of Figure 5.6.

Answers:

In SQL, as in most languages, it is possible to specify the same query in multiple ways. We will give one or more possible specification for each query.

(a) Retrieve the names of employees in department 5 who work more than 10 hours per week on the 'ProductX' project.

```
SELECT LNAME, FNAME
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE DNO=5 AND SSN=ESSN AND PNO=PNUMBER AND PNAME='ProductX' AND HOURS>10
```

Another possible SQL query uses nesting as follows:

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE DNO=5 AND SSN IN ( SELECT ESSN
FROM WORKS_ON
WHERE HOURS>10 AND PNO IN ( SELECT PNUMBER
FROM PROJECT
WHERE PNAME='ProductX' ) )
```

Result:

LNAME FNAME
Smith John
English Joyce

(b) List the names of employees who have a dependent with the same first name as themselves.

```
SELECT LNAME, FNAME
FROM EMPLOYEE, DEPENDENT
WHERE SSN=ESSN AND FNAME=DEPENDENT_NAME
```

Another possible SQL query uses nesting as follows:

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE EXISTS ( SELECT *
FROM DEPENDENT
WHERE FNAME=DEPENDENT_NAME AND SSN=ESSN )
```

Result (empty):

LNAME FNAME

(c) Find the names of employees that are directly supervised by 'Franklin Wong'.

```
SELECT E.LNAME, E.FNAME
FROM EMPLOYEE E, EMPLOYEE S
WHERE S.FNAME='Franklin' AND S.LNAME='Wong' AND E.SUPERSSN=S.SSN
```

Another possible SQL query uses nesting as follows:

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE SUPERSSN IN ( SELECT SSN
FROM EMPLOYEE
WHERE FNAME='Franklin' AND LNAME='Wong' )
```

Result:

LNAME FNAME
Smith John
Narayan Ramesh
English Joyce

(d) For each project, list the project name and the total hours per week (by all employees) spent on that project.

```
SELECT PNAME, SUM (HOURS)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNAME
```

Result:
PNAME SUM(HOURS)
ProductX 52.5
ProductY 37.5
ProductZ 50.0
Computerization 55.0
Reorganization 25.0
Newbenefits 55.0

(e) Retrieve the names of employees who work on every project.

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE NOT EXISTS ( SELECT PNUMBER
FROM PROJECT
WHERE NOT EXISTS ( SELECT *
FROM WORKS_ON
WHERE PNUMBER=PNO AND ESSN=SSN ) )
```

Result (empty):
LNAME FNAME

(f) Retrieve the names of employees who do not work on any project.

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE NOT EXISTS ( SELECT *
FROM WORKS_ON
WHERE ESSN=SSN )
```

Result (empty):
LNAME FNAME

(g) For each department, retrieve the department name, and the average salary of employees working in that department.

```
SELECT DNAME, AVG (SALARY)
FROM DEPARTMENT, EMPLOYEE
WHERE DNUMBER=DNO
GROUP BY DNAME
```

Result:
DNAME AVG(SALARY)
Research 33250
Administration 31000
Headquarters 55000

(h) Retrieve the average salary of all female employees.

```
SELECT AVG (SALARY)
FROM EMPLOYEE
WHERE SEX='F'
```

Result:

AVG(SALARY)
31000

(i) Find the names and addresses of employees who work on at least one project located in Houston but whose department has no location in Houston.

```
SELECT LNAME, FNAME, ADDRESS
FROM EMPLOYEE
WHERE EXISTS ( SELECT *
FROM WORKS_ON, PROJECT
WHERE SSN=ESSN AND PNO=PNUMBER AND PLOCATION='Houston' )
AND
NOT EXISTS ( SELECT *
FROM DEPT_LOCATIONS
WHERE DNO=DNUMBER AND DLOCATION='Houston' )
```

Result:
LNAME FNAME ADDRESS
Wallace Jennifer 291 Berry, Bellaire, TX

(j) List the last names of department managers who have no dependents.

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE EXISTS ( SELECT *
FROM DEPARTMENT
WHERE SSN=MGRSSN )
AND
NOT EXISTS ( SELECT *
FROM DEPENDENT
WHERE SSN=ESSN )
```

Result:
LNAME FNAME
Borg James

8.14 Specify the following additional queries on the database of Figure 5.5 in SQL. Show the query results if applied to the database of Figure 5.6.

(a) For each department whose average employee salary is more than $30000, retrieve the department name and the number of employees working for that department.

(b) Suppose we want the number of *male* employees in each department rather than all employees (as in Exercise 8.14a). Can we specify this query in SQL? Why or why not?

Answers:

(a)

```
SELECT DNAME, COUNT (*)
FROM DEPARTMENT, EMPLOYEE
WHERE DNUMBER=DNO
GROUP BY DNAME
HAVING AVG (SALARY) > 30000
```

Result:
DNAME DNUMBER COUNT(*)
Research 5 4
Administration 4 3

Headquarters 1 1

(b) The query may still be specified in SQL by using a nested query as follows (not all implementations may support this type of query):

```
SELECT DNAME, COUNT (*)
FROM DEPARTMENT, EMPLOYEE
WHERE DNUMBER=DNO AND SEX='M' AND DNO IN ( SELECT DNO
FROM EMPLOYEE
GROUP BY DNO
HAVING AVG (SALARY) > 30000 )
GROUP BY DNAME
```

Result:
DNAME DNUMBER COUNT(*)
Research 5 3
Administration 4 1
Headquarters 1 1

8.15 Specify the updates of Exercise 5.10 using the SQL update commands.

Answers:

Below, we show how each of the updates may be specified in SQL. Notice that some of these updates violate integrity constraints as discussed in the solution to Exercise 5.10, and hence should be rejected if executed on the database of Figure 5.6.

(a) Insert < 'Robert', 'F', 'Scott', '943775543', '21-JUN-42', '2365 Newcastle Rd, Bellaire, TX', M, 58000, '888665555', 1 > into EMPLOYEE.

```
INSERT INTO EMPLOYEE
VALUES ('Robert', 'F', 'Scott', '943775543', '21-JUN-42', '2365 Newcastle Rd, Bellaire, TX',
M, 58000, '888665555', 1)
```

(b) Insert < 'ProductA', 4, 'Bellaire', 2 > into PROJECT.

```
INSERT INTO PROJECT
VALUES ('ProductA', 4, 'Bellaire', 2)
```

(c) Insert < 'Production', 4, '943775543', '01-OCT-88' > into DEPARTMENT.

```
INSERT INTO DEPARTMENT
VALUES ('Production', 4, '943775543', '01-OCT-88')
```

(d) Insert < '677678989', null, '40.0' > into WORKS_ON.

```
INSERT INTO WORKS_ON
VALUES ('677678989', NULL, '40.0')
```

(e) Insert < '453453453', 'John', M, '12-DEC-60', 'SPOUSE' > into DEPENDENT.

```
INSERT INTO DEPENDENT
VALUES ('453453453', 'John', M, '12-DEC-60', 'SPOUSE')
```

(f) Delete the WORKS_ON tuples with ESSN= '333445555'.

```
DELETE FROM WORKS_ON
WHERE ESSN= '333445555'
```

(g) Delete the EMPLOYEE tuple with SSN= '987654321'.

```
DELETE FROM EMPLOYEE
WHERE SSN= '987654321'
```

(h) Delete the PROJECT tuple with PNAME= 'ProductX'.

```
DELETE FROM PROJECT
WHERE PNAME= 'ProductX'
```

(i) Modify the MGRSSN and MGRSTARTDATE of the DEPARTMENT tuple with DNUMBER=
5 to '123456789' and '01-OCT-88', respectively.

```
UPDATE DEPARTMENT
SET MGRSSN = '123456789', MGRSTARTDATE = '01-OCT-88'
WHERE DNUMBER= 5
```

(j) Modify the SUPERSSN attribute of the EMPLOYEE tuple with SSN= '999887777' to '943775543'.

```
UPDATE EMPLOYEE
SET SUPERSSN = '943775543'
WHERE SSN= '999887777'
```

(k) Modify the HOURS attribute of the WORKS_ON tuple with ESSN= '999887777' and PNO= 10 to '5.0'.

```
UPDATE WORKS_ON
SET HOURS = '5.0'
WHERE ESSN= '999887777' AND PNO= 10
```

8.16 Specify the following queries in SQL on the database schema of Figure 1.2.

(a) Retrieve the names of all senior students majoring in 'COSC' (computer science).

(b) Retrieve the names of all courses taught by professor King in 85 and 86.

(c) For each section taught by professor King, retrieve the course number, semester, year, and number of students who took the section.

(d) Retrieve the name and transcript of each senior student (Class=5) majoring in COSC. Transcript includes course name, course number, credit hours, semester, year, and grade for each course completed by the student.

(e) Retrieve the names and major departments of all straight A students (students who have a grade of A in all their courses).

(f) Retrieve the names and major departments of all students who do not have any grade of A in any of their courses.

Answers:

(a)

```
SELECT Name
FROM STUDENT
WHERE Major='COSC'
```

(b)

```
SELECT CourseName
FROM COURSE, SECTION
WHERE COURSE.CourseNumber=SECTION.CourseNumber AND Instructor='King'
AND (Year='85' OR Year='86')
```

Another possible SQL query uses nesting as follows:

```
SELECT CourseName
```

```
FROM COURSE
WHERE CourseNumber IN ( SELECT CourseNumber
FROM SECTION
WHERE Instructor='King' AND (Year='85' OR Year='86') )

(c) SELECT CourseNumber, Semester, Year, COUNT(*)
FROM SECTION, GRADE_REPORT
WHERE Instructor='King' AND SECTION.SectionIdentifier=GRADE_REPORT.SectionIdentifier
GROUP BY CourseNumber, Semester, Year

(d) SELECT Name, CourseName, C.CourseNumber, CreditHours, Semester, Year, Grade
FROM STUDENT ST, COURSE C, SECTION S, GRADE_REPORT G
WHERE Class=5 AND Major='COSC' AND ST.StudentNumber=G.StudentNumber AND
G.SectionIdentifier=S.SectionIdentifier AND S.CourseNumber=C.CourseNumber

(e) SELECT Name, Major
FROM STUDENT
WHERE NOT EXISTS ( SELECT *
FROM GRADE_REPORT
WHERE StudentNumber= STUDENT.StudentNumber AND NOT(Grade='A'))

(f) SELECT Name, Major
FROM STUDENT
WHERE NOT EXISTS ( SELECT *
FROM GRADE_REPORT
WHERE StudentNumber= STUDENT.StudentNumber AND Grade='A' )
```

8.17 Write SQL update statements to do the following on the database schema shown in Figure 1.2.

(a) Insert a new student <'Johnson', 25, 1, 'MATH'> in the database.

(b) Change the class of student 'Smith' to 2.

(c) Insert a new course <'Knowledge Engineering','COSC4390', 3,'COSC'>.

(d) Delete the record for the student whose name is 'Smith' and student number is 17.

Answers:

```
(a) INSERT INTO STUDENT
VALUES ('Johnson', 25, 1, 'MATH')

(b) UPDATE STUDENT
SET CLASS = 2
WHERE Name='Smith'

(c) INSERT INTO COURSE
VALUES ('Knowledge Engineering','COSC4390', 3,'COSC')

(d) DELETE FROM STUDENT
WHERE Name='Smith' AND StudentNumber=17
```

8.18 no answer provided

8.21 In SQL, specify the following queries on the database specified in Figure 5.5 using the concept of nested queries and the concepts described in this chapter.
a. Retrieve the names of all employees who work in the department that has the employee with the highest salary among all employees.
b. Retrieve the names of all employees whose supervisor's supervisor has '888665555' for Ssn.
c. Retrieve the names of employees who make at least $10,000 more than the employee who is paid the least in the company.

8.22 Consider the EMPLOYEE table's constraint EMPSUPERFK as specified in Figure 8.2 is changed to read as follows:

```
CONSTRAINT EMPSUPERFK
  FOREIGN KEY (SUPERSSN) REFERNCES EMPLOYEE(SSN)
     ON DELETE CASCADE ON UPDATE CASCADE,
```

Answer the following questions:
a. What happens when the following command is run on the database state shown in Figure 5.6?

```
DELETE EMPLOYEE WHERE LNAME = 'Borg'
```

b. Is it better to CASCADE or SET NULL in case of EMPSUPERFK constraint ON DELETE?

8.23 Write SQL statements to create a table EMPLOYEE_BACKUP backup of EMPLOYEE table shown in Figure 5.5 and 5.6.

Answers
8.21

```
a) SELECT LNAME FROM EMPLOYEE WHERE DNO =
        ( SELECT DNO FROM EMPLOYEE WHERE SALARY =
                ( SELECT MAX(SALARY) FROM EMPLOYEE) )
b) SELECT LNAME FROM EMPLOYEE WHERE SUPERSSN IN
( SELECT SSN FROM EMPLOYEE WHERE SUPERSSN = '888665555' )
c) SELECT LNAME FROM EMPLOYEE WHERE SALARY >= 10000 +
( SELECT MIN(SALARY) FROM EMPLOYEE)
```

8.22
a) The James E. Borg entry is deleted from the table, and each employee with him as a supervisor is also (and their supervisees, and so on). In total, 8 rows are deleted and the table is empty.
b) It is better to SET NULL, since an employee is not fired (DELETED) when their supervisor is deleted. Instead, their SUPERSSN should be SET NULL so that they can later get a new supervisor.

8.23

```
INSERT INTO EMPLOYEE_BACKUP VALUES ( SELECT * FROM EMPLOYEE )
```