1.14. Consider Figure 1.2.

A. If the name of the ‘CS’ (Computer Science) Department changes to ‘CSSE’ (Computer Science and Software Engineering) Department and the corresponding prefix for the course number also changes, identify the columns in the database that would need to be updated.

```
COURSE.Department, COURSE.Course_number, SECTION.Course_number,
PREREQUISITE.Course_number, PREREQUISITE.Prequisite_number
```

B. Can you restructure the columns in the COURSE , SECTION , and PREREQUISITE tables so that only one column will need to be updated?

```
Create an additional unique course identifier associated with
each course number then replace all the Course_number columns
in the tables that have one with the unique identifier column.
```

2.15. Consider Figure 2.1. In addition to constraints relating the values of columns in one table to columns in another table, there are also constraints that impose restrictions on values in a column or a combination of columns within a table. One such constraint dictates that a column or a group of columns must be unique across all rows in the table. For example, in the STUDENT table, the Student_number column must be unique (to prevent two different students from having the same Student_number). Identify the column or the group of columns in the other tables that must be unique across all rows in the table.

```
COURSE.Course_number;
SECTION.Section_identifier;
GRADE_REPORT.Student_number, GRADE_REPORT.Section_identifier;
PREREQUISITE.Course_number, PREREQUISITE.Prerequisite_number
```

6.6. Repeat Exercise 6.5, but use the AIRLINE database schema of Figure 5.8.

6.5. 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.

```
CREATE TABLE AIRPORT (
 Airport_code varchar(10) NOT NULL,
 Name varchar(20) NOT NULL,
 City varchar(20) NOT NULL,
 State varchar(50),
 PRIMARY KEY (Airport_code)
);

CREATE TABLE FLIGHT (
 Flight_number varchar(10) NOT NULL,
 Airline varchar(20) NOT NULL,
 Weekdays varchar(10),
 PRIMARY KEY (Flight_number)
);

CREATE TABLE FLIGHT_LEG (
 Flight_number varchar(10) NOT NULL,
 Leg_number varchar(10) NOT NULL,
 Departure_airport_code varchar(10) NOT NULL,
 Scheduled_departure_time time NOT NULL,
 Arrival_airport_code varchar(10) NOT NULL,
 Scheduled_arrival_time time NOT NULL,
 PRIMARY KEY (Flight_number, Leg_number),
 CONSTRAINT FLIGHTNOFK FOREIGN KEY Flight_number
REFERENCES FLIGHT(Flight_number) ON DELETE REJECT ON UPDATE CASCADE
);

CREATE TABLE LEG_INSTANCE (
 Flight_number varchar(10) NOT NULL,
 Leg_number varchar(10) NOT NULL,
 Date date NOT NULL,
 Number_of_available_seats int NOT NULL,
 Airplane_id varchar(10) NOT NULL,
```

```
 Departure_airport_code varchar(10) NOT NULL,
 Departure_time time NOT NULL,
 Arrival_airport_code varchar(10) NOT NULL,
 Arrival_time time NOT NULL,
 PRIMARY KEY (Flight_number, Leg_number, Date),
 CONSTRAINT FLIGHTNOFK FOREIGN KEY Flight_number
REFERENCES FLIGHT(Flight_number) ON DELETE REJECT ON UPDATE CASCADE,
 CONSTRAINT FLIGHTLEGNOFK FOREIGN KEY Leg_number
REFERENCES FLIGHT_LEG(Leg_number) ON DELETE REJECT ON UPDATE CASCADE
);

CREATE TABLE FARE (
 Flight_number varchar(10) NOT NULL,
 Fare_code varchar(10) NOT NULL,
 Amount float NOT NULL,
 Restrictions varchar(10),
 PRIMARY KEY (Flight_number, Fare_code),
 CONSTRAINT FLIGHTNOFK FOREIGN KEY Flight_number
REFERENCES FLIGHT(Flight_number) ON DELETE REJECT ON UPDATE CASCADE
);

CREATE TABLE AIRPLANE_TYPE (
 Airplane_type_name varchar(10) NOT NULL,
 Max_seats int NOT NULL,
 Company varchar(20) NOT NULL,
 PRIMARY Key (Airplane_type_name)
);

CREATE TABLE CAN_LAND (
 Airplane_type_name varchar(10) NOT NULL,
 Airport_code varchar(10) NOT NULL,
 PRIMARY KEY (Airplane_type_name, Airport_code),
 CONSTRAINT AIRPLANETYPENAMEFK FOREIGN KEY Airplane_type_name
REFERENCES AIRPLANE_TYPE(Airplane_type_name) ON DELETE REJECT ON UPDATE
CASCADE,
 CONSTRAINT CANLANDAIRPORTCODEFK FOREIGN KEY (Airport_code) REFERENCES
AIRPORT(Airport_code) ON UPDATE CASCADE ON DELETE REJECT
);
```

```
CREATE TABLE AIRPLANE (
 Airplane_id varchar(10) NOT NULL,
 Total_number_of_seats int NOT NULL,
 Airplane_type varchar(10) NOT NULL,
 PRIMARY KEY (Airplane_id),
 CONSTRAINT AIRPLANEAIRPLANETYPEFK FOREIGN KEY (Airplane_type) REFERENCES
AIRPLANE_TYPE(Airplane_type_name) ON UPDATE CASCADE ON DELETE REJECT
);

CREATE TABLE SEAT_RESERVATION (
 Flight_number varchar(10) NOT NULL,
 Leg_number varchar(10) NOT NULL,
 Date date NOT NULL,
 Seat_number varchar(5) NOT NULL,
 Customer_name varchar(50) NOT NULL,
 Customer_phone varchar(15) NOT NULL,
 PRIMARY KEY (Flight_number, Leg_number, Date, Seat_number),
 CONSTRAINT SEATRESERVATIONFLIGHTNUMBERFK FOREIGN KEY (Flight_number)
REFERENCES FLIGHT(Flight_number) ON UPDATE CASCADE ON DELETE REJECT,
 CONSTRAINT SEATRESERVATIONLEGNUMBERFK FOREIGN KEY (Leg_number) REFERENCES
LEG_INSTANCE(Leg_number) ON UPDATE CASCADE ON DELETE REJECT,
 CONSTRAINT SEATRESERVATIONDATEFK FOREIGN KEY (Date) REFERENCES
LEG_INSTANCE(Date) ON UPDATE CASCADE ON DELETE REJECT
);
```

6.7. Consider the LIBRARY relational database schema shown in Figure 6.6. Choose the appropriate action (reject, cascade, set to NULL , set to default) for each referential integrity constraint, both for the deletion of a referenced tuple and for the update of a primary key attribute value in a referenced tuple. Justify your choices.

BOOK

Note: The name of a publisher may change, but the name of the publisher associated with the book (printed in the front few pages and used for academic citations) should stay the same no matter what. So, if there really are any cases in which a publisher's name changes, then there should be another table associating a Publisher_id with one or more Publisher_names. This would allow you to consolidate the contact info in one

entry while also help preserve the correct reference information for each book.

BOOK's foreign key restraint on Publisher_name should say ON UPDATE CASCADE, because the book should remain associated with the publisher's contact info (see note above for why this is broken due to the necessity of academic citation consistency), and ON DELETE REJECT, because a book needs to be associated with a publisher (again, this depends on what the library is using the publisher data for, citational data or for contact info).

BOOK_AUTHORS

Note: Similar to the Publisher_name in BOOK, it is wrong to be changing author names for books because it violates the consistency of citational stuff, but this all depends on what the library is trying to do here. Mark Twain, for example, is also named Samuel Clemens for some of this books, so really there should be a more sophisticated structure here to account for all these possibilities.

BOOK_AUTHORS foreign key constraint on book_id should say ON UPDATE CASCADE since if the Book_id in BOOK were to change the author should still be associated with the corresponding book and ON DELETE CASCADE, since the author of a book we don't have isn't necessary.

PUBLISHER

The PUBLISHER table does not have any foreign keys. So there are no constraints to create.

BOOK COPIES

BOOK_COPIES foreign key constraint on Book_id should say ON UPDATE CASCADE since if the Book_id in BOOK changes the number of copies of the given book in a given branch should still be associated with the same book in the BOOK table. It should also say ON DELETE CASCADE since if the Book_id is deleted from the BOOK table that means the book is no longer in the collection of any library so all copies of the book should be removed.

The foreign key constraint on Branch_id should be ON UPDATE CASCADE since if the Branch_id changes in the LIBRARY_BRANCH table the Branch_id of the book should still be associated with that branch. If the book were being changed to a different branch then the Branch_id in the BOOK_COPIES table should be changed for that book which would not affect what Branch_id’s in the LIBRARY_BRANCH table. If a Branch_id in LIBRARY_BRANCH were to be deleted that would mean that library branch had

shutdown. This means that either the books for that library should have been sold or moved to a different branch. If they were moved to a different branch then the Branch_id of the book should be changed or the No_of_copies of the book should be incremented if the other branch already has the book. This should happen before the branch is deleted. If the books were sold then they should be removed from the database anyway. So Branch_id constraint should be ON DELETE CASCADE.

BOOK_LOANS

BOOK_LOANS foreign key constraint on Book_id should say ON UPDATE CASCADE since if the Book_id in BOOK changes, the book is still checked out and should be referenced appropriately in BOOK_LOANS. It should also say ON DELETE CASCADE, since if the Book_id is deleted from the BOOK table, that means the book is no longer in the collection so it doesn't matter if a borrower has this book checked out (although that is a strange circumstance to occur, and it depends on what the library is actually trying to do with this database).

BOOK_LOANS foreign key constraint on Branch_id should say ON UPDATE CASCADE since if the Branch_id changes, the BOOK_LOAN tuple should continue to reference the appropriate branch. ON DELETE CASCADE, because if a branch is closed down, then the loaned out books probably don't matter anymore (again, this depends on what the library is actually trying to do with this database or the books should already have been associated with a new branch).

BOOK_LOANS foreign key constraint on Card_no should say ON UPDATE CASCADE since if the borrower's card number changes (due to a replacement card or some such) the borrowed book should still refer to the same borrower. ON DELETE REJECT, since a library shouldn't be deleting the account of a borrower who hasn't returned all their books (but of course this entirely depends on the library's policies and the intended use of this database).

LIBRARY_BRANCH

The LIBRARY_BRANCH table does not have any foreign keys. So there are no constraints to create.

BORROWER

The BORROWER table does not have any foreign keys. So there are no constraints to create.

6.8. Write appropriate SQL DDL statements for declaring the LIBRARY relational database schema of Figure 6.6. Specify the keys and referential triggered actions.

```
CREATE TABLE BOOK (
 Book_id varchar(17) NOT NULL,
 Title varchar(100) NOT NULL,
 Publisher_name varchar(20) NOT NULL,
 PRIMARY KEY Book_id
 CONSTRAINT BOOKPUBLISHERNAMEFK FOREIGN KEY Publisher_name REFERENCES
PUBLISHER(Publisher_name) ON DELETE REJECT ON UPDATE CASCADE
);

CREATE TABLE BOOK_AUTHORS (
 Book_id varchar(17) NOT NULL,
 Author_name varchar(50) NOT NULL,
 PRIMARY KEY (Book_id, Author_name),
 CONSTRAINT BOOKAUTHORSBOOKIDFK FOREIGN KEY (Book_id) REFERENCES
BOOK(Book_id) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE PUBLISHER (
 Name varchar(20) NOT NULL,
 Address varchar(100) NOT NULL,
 Phone varchar(15) NOT NULL,
 PRIMARY KEY Name
);

CREATE TABLE BOOK_COPIES (
 Book_id varchar(17) NOT NULL,
 Branch_id varchar(20) NOT NULL,
 No_of_copies int NOT NULL,
 PRIMARY KEY (Book_id, Branch_id),
 CONSTRAINT BOOKCOPIESBRANCHIDFK FOREIGN KEY (Branch_id) REFERENCES
LIBRARY_BRANCH(Branch_id) ON DELETE CASCADE ON UPDATE CASCADE,
 CONSTRAINT BOOKCOPIESBOOKIDFK FOREIGN KEY (Book_id) REFERENCES
BOOK(Book_id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

```
CREATE TABLE BOOK_LOANS (
 Book_id varchar(17) NOT NULL,
 Branch_id varchar(20) NOT NULL,
 Card_no int NOT NULL,
 Date_out date NOT NULL,
 Due_date date NOT NULL,
 PRIMARY KEY (Book_id, Branch_id, Card_no),
 CONSTRAINT BOOKLOANSBOOKIDFK FOREIGN KEY (Book_id) REFERENCES BOOK(Book_id)
ON DELETE CASCADE ON UPDATE CASCADE,
 CONSTRAINT BOOKLOANSBRANCHIDFK FOREIGN KEY (Branch_id) REFERENCES
LIBRARY_BRANCH(Branch_id) ON DELETE CASCADE ON UPDATE CASCADE,
 CONSTRAINT BOOKLOANSCARDNOFK FOREIGN KEY (Card_no) REFERENCES
BORROWER(Card_no) ON DELETE REJECT ON UPDATE CASCADE
);

CREATE TABLE LIBRARY_BRANCH (
 Branch_id varchar(20) NOT NULL,
 Branch_name varchar(20) NOT NULL,
 Address varchar(100) NOT NULL,
 PRIMARY KEY (Branch_id)
);

CREATE TABLE BORROWER (
 Card_no int NOT NULL,
 Name varchar(50) NOT NULL,
 Address varchar(100) NOT NULL,
 Phone varchar(15) NOT NULL,
 PRIMARY KEY Card_no,
);
```

6.9. 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 efficiently when updates are applied to the database?

```
When performing an update or delete in the database the
affected rows in the database could be determined. Once we know
what rows in the database will be affected by the update or
```

deletion we can check the list of constraints and see if any apply to the rows we are updating or deleting. Once we determine if any constraints affect the update you can proceed with the update. If any of the constraints are violated during the updated we can throw an error and roll back the update or deletion.

This does not seem particularly difficult to implement since once the database is defined you could probably have a lookup table for the constraints. You could then recursively apply your update to the database checking if any update breaks a constraint. If an update or deletion breaks a constraint at any point we can backtrack restoring the database to its previous state.

This may not be very efficient if there are a lot of items being updated and/or a large number of constraints. For example, if we update one row that has a value used as a foreign key for a large number of rows in another table then we would need to update all of the items in the other table and check the corresponding constraints for every item that needs to be updated in that table.

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

A. Retrieve the names of all senior students majoring in ‘cs’ (computer science).

```
SELECT Name FROM STUDENT WHERE Major = 'CS';
```

B. Retrieve the names of all courses taught by Professor King in 2007 and 2008.

```
SELECT Course_name
FROM COURSE
INNER JOIN SECTION
ON COURSE.Course_number = SECTION.Course_number
WHERE Instructor = 'King' AND (Year = '07' OR Year = '08');
```

C. For each section taught by Professor King, retrieve the course number, semester, year, and number of students who took the section.

```
SELECT SECTION.Course_number, SECTION.Semester, SECTION.Year,
COUNT(*)
FROM SECTION
INNER JOIN GRADE_REPORT
ON GRADE_REPORT.Section_identifer = SECTION.Section_identifier
WHERE Instructor = 'King'
GROUP_BY GRADE_REPORT.Section_identifier;
```

D. Retrieve the name and transcript of each senior student (Class = 4) majoring in CS. A transcript includes course name, course number, credit hours, semester, year, and grade for each course completed by the student.

```
SELECT *
FROM (
  SELECT GRADE_REPORT.Grade, GRADE_REPORT.Section_identifier
  FROM GRADE_REPORT
  INNER JOIN (
    SELECT Student_number, Name
    FROM STUDENT
    WHERE Class = '4' AND Major = 'CS'
  ) AS Students
  ON Students.Student_number = GRADE_REPORT.Student_number
) AS Students_with_grades
INNER JOIN (
  SELECT COURSE.Course_name, COURSE.Course_number, COURSE.Credit_hours,
SECTION.Semester, SECTION.Year
  FROM COURSE
  INNER JOIN SECTION
  ON COURSE.Course_number = SECTION.Course_number
) AS Classes
ON Students_with_grades.Section_identifier = Classes.Section_identifier
```

7.5. Specify the following queries on the database in Figure 5.5 in SQL. Show the query results if each query is applied to the database state in Figure 5.6.

A. For each department whose average employee salary is more than $30,000, retrieve the department name and the number of employees working for that department.

```
SELECT Dname, EmpCount
FROM (
  SELECT Dname, Dno
  FROM DEPARTMENT
) AS Dnames
```

```
INNER JOIN (
  SELECT Dno, EmpCount
  FROM (
    SELECT Dno, COUNT(*) AS EmpCount
    FROM EMPLOYEE
    GROUP_BY Dno
  ) AS Dno_EmployeeCounted
  INNER JOIN (
    SELECT Dno, AVG(Salary) AS AvgSal
    FROM EMPLOYEE
    GROUP_BY Dno
    FILTER AvgSal > 30000
  ) AS Dno_AvgSalaries
  ON Dno_EmployeeCounted.Dno = Dno_AvgSalaries.Dno
) AS EmpsCounted
ON Dnames.Dno = EmpsCounted.Dno
```

B. Suppose that we want the number of male employees in each department making more than $30,000, rather than all employees (as in Exer- cise 7.5a). Can we specify this query in SQL? Why or why not?

```
SELECT Dname, EmpCount
FROM (
  SELECT Dname, Dno
  FROM DEPARTMENT
) AS Dnames
INNER JOIN (
  SELECT Dno, EmpCount
  FROM (
    SELECT Dno, COUNT(*) AS EmpCount
    FROM EMPLOYEE
    GROUP_BY Dno
  ) AS Dno_EmployeeCounted
  INNER JOIN (
    SELECT Dno, AVG(Salary) AS AvgSal
    FROM EMPLOYEE
    GROUP_BY Dno
    FILTER AvgSal > 30000 AND Sex='M'
  ) AS Dno_AvgSalaries
  ON Dno_EmployeeCounted.Dno = Dno_AvgSalaries.Dno
) AS EmpsCounted
ON Dnames.Dno = EmpsCounted.Dno
```

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

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

```
SELECT Name, Major
FROM STUDENT
WHERE Student_number NOT IN (
  SELECT Student_number
  FROM GRADE_REPORT
  WHERE Grade != 'A'
);
```

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

```
SELECT Name, Major
FROM STUDENT
WHERE Student_number NOT IN {
  SELECT Student_number
  FROM GRADE_REPORT
  WHERE Grade = 'A'
);
```

7.7. In SQL, specify the following queries on the database in Figure 5.5 using the concept of nested queries and other 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.

```
SELECT Fname, Minit, Lname
FROM EMPLOYEE
WHERE Dno = (
  SELECT Dno
  FROM EMPLOYEE
  WHERE Salary = (
    SELECT MAX(Salary) AS MaxSalary
    FROM EMPLOYEE
  )
);
```

B. Retrieve the names of all employees whose supervisor's supervisor has '888665555' for Ssn.

```
SELECT Fname, Minit, Lname
FROM EMPLOYEE
WHERE Super_ssn IN (
  SELECT Ssn
  FROM EMPLOYEE
  WHERE Super_ssn = '888665555'
);
```

C. Retrieve the names of employees who make at least $10,000 more than the employee who is paid the least in the company.

```
SELECT (Fname, Minit, Lname)
FROM EMPLOYEE
WHERE Salary > 10000 + (SELECT MIN(Salary) FROM EMPLOYEE);
```