|  |  |
| --- | --- |
| **Homework 4** | **Basic SQL** |
| **Due Wed, Oct 7 at 11:30 pm** | **Objectives:** Practice of Basic SQL |

**CS 4347: Database Systems Alex Lundin aml140830**

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

(a) What are the referential integrity constraints that should hold on the schema?

The following referential integrity constraints should hold (we use the notation:

**R.(A1, ..., An) --> S.(B1, ..., Bn)**

R is the referencing relation with a foreign key containing attributes (A1, ..., An)

S is the referenced relation with a primary key containing attributes (B1, …, Bn)

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)

(b) Write appropriate SQL DDL statements to define the database.

­­ CREATE TABLE AIRPORT

(Airport\_code INT NOT NULL,

Name VARCHAR(40) NOT NULL,

City VARCHAR(40) NOT NULL,

State VARCHAR(2) NOT NULL,

PRIMARY KEY(Airport\_code));

CREATE TABLE FLIGHT

(Flight\_number INT NOT NULL,

Airline VARCHAR(40) NOT NULL,

Weekdays VARCHAR(40) NOT NULL,

PRIMARY KEY(Flight\_number));

CREATE TABLE FLIGHT\_LEG

(Flight\_number INT NOT NULL,

Leg\_number INT NOT NULL,

Departure\_airport\_code INT NOT NULL,

Scheduled\_departure\_time TIMESTAMP(6) NOT NULL,

Arrival\_airport\_code INT NOT NULL,

Scheduled\_arrival\_time TIMESTAMP(6) NOT NULL,

PRIMARY KEY(Flight\_number, Leg\_number),

FOREIGN KEY(Flight\_number) REFERENCES FLIGHT(Flight\_number),

UNIQUE(Leg\_number));

CREATE TABLE LEG\_INSTANCE

(Flight\_number INT NOT NULL,

Leg\_number INT NOT NULL,

Date1 DATE NOT NULL,

Number\_of\_avaiable\_seats INT NOT NULL,

Airplane\_id INT NOT NULL,

Departure\_airport\_code INT NOT NULL,

Departure\_time TIMESTAMP(6) NOT NULL,

Arrival\_airport\_code INT NOT NULL,

Arrival\_time TIMESTAMP(6) NOT NULL,

PRIMARY KEY(Flight\_number, Leg\_number, Date1),

FOREIGN KEY(Flight\_number) REFERENCES FLIGHT(Flight\_number),

FOREIGN KEY(Leg\_number) REFERENCES FLIGHT\_LEG(Leg\_number));

CREATE TABLE FARE

(Flight\_number INT NOT NULL,

Fare\_code INT NOT NULL,

Amount DECIMAL(10,2) NOT NULL,

Restrictions VARCHAR(40) NOT NULL,

PRIMARY KEY(Flight\_number,Fare\_code),

FOREIGN KEY(Flight\_number) REFERENCES FLIGHT(Flight\_number),

UNIQUE(Fare\_code));

CREATE TABLE AIRPLANE\_TYPE

(Airplane\_type\_name VARCHAR(40) NOT NULL,

Max\_seats INT NOT NULL,

Company VARCHAR(40) NOT NULL,

PRIMARY KEY(Airplane\_type\_name));

CREATE TABLE CAN\_LAND

(Airport\_code INT NOT NULL,

Airplane\_type\_name VARCHAR(40) NOT NULL,

PRIMARY KEY(Airport\_code,Airplane\_type\_name),

FOREIGN KEY(Airport\_code) REFERENCES AIRPORT(Airport\_code),

FOREIGN KEY(Airplane\_type\_name) REFERENCES AIRPLANE\_TYPE(Airplane\_type\_name));

CREATE TABLE AIRPLANE

(Airplane\_id INT NOT NULL,

Total\_number\_of\_seats INT NOT NULL,

Airplane\_type VARCHAR(40) NOT NULL,

PRIMARY KEY(Airplane\_id));

CREATE TABLE SEAT\_RESERVATION

(Flight\_number INT NOT NULL,

Leg\_number INT NOT NULL,

Date1 DATE NOT NULL,

Seat\_number INT NOT NULL,

Customer\_name VARCHAR(40) NOT NULL,

Customer\_phone VARCHAR(40) NOT NULL,

PRIMARY KEY(Flight\_number,Leg\_number, Date1, Seat\_number),

FOREIGN KEY(Flight\_number) REFERENCES FLIGHT(Flight\_number),

FOREIGN KEY(Leg\_number) REFERENCES FLIGHT\_LEG(Leg\_number),

UNIQUE(Seat\_number));

**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 PUBLISHER

(Name VARCHAR(20) NOT NULL,

Address VARCHAR(40) NOT NULL,

Phone CHAR(12),

PRIMARY KEY (Name) );

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 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));

**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?

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.

**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 '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' )

**6.13.** 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’, ‘cs4390’, 3, ‘cs’>.

d. Delete the record for the student whose name is ‘Smith’ and whose student number is 17.

(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

**6.16.** Write SQL statements to create a table EMPLOYEE\_BACKUP to back up the EMPLOYEE table shown in Figure 5.6.

INSERT INTO EMPLOYEE\_BACKUP VALUES ( SELECT \* FROM EMPLOYEE )