CHAPTER 3: THE RELATIONAL DATA MODEL AND RELATIONAL DATABASE CONSTRAINTS

Answers to Selected Exercises

3.11 - Suppose each of the following Update operations is applied directly to the database of Figure 3.6. Discuss *all* integrity constraints violated by each operation, if any, and the different ways of enforcing these constraints:

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

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

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

(d) Insert < '677678989', null, '40.0' > into WORKS_ON.
(e) Insert < '453453453', 'John', M, '12-DEC-60', 'SPOUSE' > into DEPENDENT.

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

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

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

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

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

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

Answers:
(a) No constraint violations.

(b) Violates referential integrity because DNUM=2 and there is no tuple in the DEPARTMENT relation with DNUMBER=2. We may enforce the constraint by: (i) rejecting the insertion of the new PROJECT tuple, (ii) changing the value of DNUM in the new PROJECT tuple to an existing DNUMBER value in the DEPARTMENT relation, or (iii) inserting a new DEPARTMENT tuple with DNUMBER=2.

(c) Violates both the key constraint and referential integrity. Violates the key constraint because there already exists a DEPARTMENT tuple with DNUMBER=4. We may enforce this constraint by: (i) rejecting the insertion, or (ii) changing the value of DNUMBER in the new DEPARTMENT tuple to a value that does not violate the key constraint. Violates referential integrity because MGRSSN='943775543' and there is no tuple in the EMPLOYEE relation with SSN='943775543'. We may enforce the constraint by: (i) rejecting the insertion, (ii) changing the value of MGRSSN to an existing SSN value in EMPLOYEE, or (iii) inserting a new EMPLOYEE tuple with SSN='943775543'.

(d) Violates both the entity integrity and referential integrity. Violates entity integrity

because PNO, which is part of the primary key of WORKS_ON, is null. We may enforce this constraint by: (i) rejecting the insertion, or (ii) changing the value of PNO in the new WORKS_ON tuple to a value of PNUMBER that exists in the PROJECT relation. Violates referential integrity because ESSN='677678989' and there is no tuple in the EMPLOYEE relation with SSN='677678989'. We may enforce the constraint by: (i) rejecting the insertion, (ii) changing the value of ESSN to an existing SSN value in EMPLOYEE, or (iii) inserting a new EMPLOYEE tuple with SSN='677678989'.

(e) No constraint violations.

(f) No constraint violations.

(g) Violates referential integrity because several tuples exist in the WORKS_ON, DEPENDENT, DEPARTMENT, and EMPLOYEE relations that reference the tuple being deleted from EMPLOYEE. We may enforce the constraint by: (i) rejecting the deletion, or (ii) deleting all tuples in the WORKS_ON, DEPENDENT, DEPARTMENT, and EMPLOYEE relations whose values for ESSN, ESSN, MGRSSN, and SUPERSSN, respectively, is equal to'987654321'.

(h) Violates referential integrity because two tuples exist in the WORKS_ON relations that reference the tuple being deleted from PROJECT. We may enforce the constraint by: (i) rejecting the deletion, or (ii) deleting the tuples in the WORKS_ON relation whose value for PNO=1 (the value for the primary key PNUMBER for the tuple being deleted from PROJECT).

(i) No constraint violations.

(j) Violates referential integrity because the new value of SUPERSSN='943775543' and there is no tuple in the EMPLOYEE relation with SSN='943775543'. We may enforce the constraint by: (i) rejecting the deletion, or (ii) inserting a new EMPLOYEE tuple with SSN='943775543'.

(k) No constraint violations.

3.12 - Consider the AIRLINE relational database schema shown in Figure 3.8, which describes a database for airline flight information. Each FLIGHT is identified by a flight NUMBER, and consists of one or more FLIGHT_LEGs with LEG_NUMBERs 1, 2, 3, etc. Each leg has scheduled arrival and departure times and airports, and has many LEG_INSTANCEs--one for each DATE on which the flight travels. FARES are kept for each flight. For each leg instance, SEAT_RESERVATIONs are kept, as is the AIRPLANE used in the leg, and the actual arrival and departure times and airports. An AIRPLANE is identified by an AIRPLANE_ID, and is of a particular AIRPLANE_TYPE. CAN_LAND relates AIRPLANE_TYPEs to the AIRPORTs in which they can land. An AIRPORT is identified by an AIRPORT_CODE. Consider an update for the AIRLINE database to enter a reservation on a particular flight or flight leg on a given date.

(a) Give the operations for this update.

(b) What types of constraints would you expect to check?

(c) Which of these constraints are key, entity integrity, and referential integrity constraints and which are not?

(d) Specify all the referential integrity constraints on Figure 3.8.

Answers:
(a) One possible set of operations for the following update is the following: INSERT <FNO,LNO,DT,SEAT_NO,CUST_NAME,CUST_PHONE> into SEAT_RESERVATION; MODIFY the LEG_INSTANCE tuple with the condition: (FLIGHT_NUMBER=FNO AND LEG_NUMBER=LNO AND DATE=DT) by setting NUMBER_OF_AVAILABLE_SEATS = NUMBER_OF_AVAILABLE_SEATS - 1; These operations should be repeated for each LEG of the flight on which a reservation is made. This assumes that the reservation has only one seat. More complex operations will be needed for a more realistic reservation that may reserve several seats at once.

(b) We would check that NUMBER_OF_AVAILABLE_SEATS on each LEG_INSTANCE of the flight is greater than 1 before doing any reservation (unless overbooking is permitted), and that the SEAT_NUMBER being reserved in SEAT_RESERVATION is available.

(c) The INSERT operation into SEAT_RESERVATION will check all the key, entity integrity, and referential integrity constraints for the relation. The check that NUMBER_OF_AVAILABLE_SEATS on each LEG_INSTANCE of the flight is greater than 1 does not fall into any of the above types of constraints (it is a general semantic integrity constraint).

(d) We will write a referential integrity constraint as R.A --> S (or R.(X) --> T) whenever attribute A (or the set of attributes X) of relation R form a foreign key that references the primary key of relation S (or T). FLIGHT_LEG.FLIGHT_NUMBER --> FLIGHT
FLIGHT_LEG.DEPARTURE_AIRPORT_CODE --> AIRPORT
FLIGHT_LEG.ARRIVAL_AIRPORT_CODE --> AIRPORT
LEG_INSTANCE.(FLIGHT_NUMBER,LEG_NUMBER) --> FLIGHT_LEG
LEG_INSTANCE.DEPARTURE_AIRPORT_CODE --> AIRPORT
LEG_INSTANCE.ARRIVAL_AIRPORT_CODE --> AIRPORT
LEG_INSTANCE.AIRPLANE_ID --> AIRPLANE
FARES.FLIGHT_NUMBER --> FLIGHT
CAN_LAND.AIRPLANE_TYPE_NAME --> AIRPLANE_TYPE
CAN_LAND.AIRPORT_CODE --> AIRPORT
AIRPLANE.AIRPLANE_TYPE --> AIRPLANE_TYPE
SEAT_RESERVATION.(FLIGHT_NUMBER,LEG_NUMBER,DATE) --> LEG_INSTANCE

3.13 - Consider the relation CLASS(Course#, Univ_Section#, InstructorName, Semester, BuildingCode, Room#, TimePeriod, Weekdays, CreditHours). This represents classes taught in a university with unique Univ_Section#. Give what you think should be various candidate keys and write in your own words under what constraints each candidate key would be valid.

Answer:
Possible candidate keys include the following (Note: We assume that the values of the Semester attribute include the year; for example "Spring/94" or "Fall/93" could be values for Semester):

1. {Semester, BuildingCode, Room#, TimePeriod, Weekdays} if the same room cannot be used at the same time by more than one course during a particular semester.
2. {Univ_Section#} if it is unique across all semesters.
3. {InstructorName, Semester} if an instructor can teach at most one course during each semester.
4. If Univ_Section# is not unique, which is the case in many universities, we have to

examine the rules that the university uses for section numbering. For example, if the sections of a particular course during a particular semester are numbered 1, 2, 3, ..., then a candidate key would be {Course#, Univ_Section#, Semester}. If, on the other hand, all sections (of any course) have unique numbers during a particular semester only, then the candidate key would be {Univ_Section#, Semester}.

3.14 - Consider the following six relations for an order-processing database application in a company:

CUSTOMER (Cust#, Cname, City)
ORDER (Order#, Odate, Cust#, Ord_Amt)
ORDER_ITEM (Order#, Item#, Qty)
ITEM (Item#, Unit_price)
SHIPMENT (Order#, Warehouse#, Ship_date)
WAREHOUSE (Warehouse#, City)

Here, Ord_Amt refers to total dollar amount of an order; Odate is the date the order was placed; Ship_date is the date an order (or part of an order) is shipped from the warehouse. Assume that an order can be shipped from several warehouses. Specify the foreign keys for this schema, stating any assumptions you make. What other constraints can you think of for this database?

Answer:
Strictly speaking, a foreign key is a *set* of attributes, but when that set contains only one attribute, then that attribute itself is often informally called a foreign key. The schema of this question has the following five foreign keys:

1. the attribute Cust# of relation ORDER that references relation CUSTOMER,
2. the attribute Order# of relation ORDER_ITEM that references relation ORDER,
3. the attribute Item# of relation ORDER_ITEM that references relation ITEM,
4. the attribute Order# of relation SHIPMENT that references relation ORDER, and
5. the attribute Warehouse# of relation SHIPMENT that references relation WAREHOUSE.

We now give the queries in relational algebra:

a. $\text{SHIP\_W2} \leftarrow \sigma_{\text{Warehouse\#} = \text{'W2'}}(\text{SHIPMENT})$
$\text{RESULT} \leftarrow \pi_{\text{Order\#, Ship\_date}}(\text{SHIP\_W2})$

b. $\text{ORDERS\_JL} \leftarrow \text{ORDER} * \sigma_{\text{Cname} = \text{'Jose Lopez'}}(\text{CUSTOMER})$
$\text{RESULT} \leftarrow \pi_{\text{Order\#, Warehouse\#}}(\text{ORDERS\_JL} * \text{SHIPMENT})$

c. $\text{BY\_CUSTNUM} \leftarrow {}_{\text{Cust\#}}\mathcal{F}_{\text{COUNT Order\#, AVERAGE Ord\_Amt}}(\text{ORDER})$
$\text{RESULT}(\text{CUSTNAME, \#OFORDERS, AVG\_ORDER\_AMT}) \leftarrow$
$\pi_{\text{Cname, COUNT\_Order\#, AVERAGE\_Ord\_Amt}}(\text{BY\_CUSTNUM} * \text{CUSTOMER})$

d. $\text{TIMELY\_SHIPPED} \leftarrow \sigma_{\text{Ship\_date} \leq \text{Odate} + 30}(\text{ORDER} * \text{SHIPMENT})$
$\text{RESULT} \leftarrow \pi_{\text{Order\#}}(\text{ORDER}) - \pi_{\text{Order\#}}(\text{TIMELY\_SHIPPED})$

The above query lists all orders for which no "timely" shipment was made, including orders for which no shipment was ever made. It is instructive to compare the above query with the one below that lists those orders for which at least one "late" shipment was made.

$$\text{LATE\_SHIPPED} \leftarrow \text{Ship\_date} > \text{Odate} + 30\,(\text{ORDER} * \text{SHIPMENT})$$
$$\text{RESULT} \leftarrow \pi_{\text{Order\#}}\,(\text{LATE\_SHIPPED})$$

e. $$\text{NEW\_YORK} \leftarrow \pi_{\text{Warehouse\#}}\,(\text{City} = \text{'New York'}\,(\text{WAREHOUSE}))$$
$$\text{RESULT} \leftarrow \pi_{\text{Order\#, Warehouse\#}}\,(\text{SHIPMENT}) \div \text{NEW\_YORK}$$

3.15 - Consider the following relations for a database that keeps track of business trips of salespersons in a sales office:

SALESPERSON (SSN, Name, Start_Year, Dept_No)
TRIP (SSN, From_City, To_City, Departure_Date, Return_Date, Trip_ID)
EXPENSE (Trip_ID, Account#, Amount)

Specify the foreign keys for this schema, stating any assumptions you make.

Answer:
The schema of this question has the following two foreign keys:
1. the attribute SSN of relation TRIP that references relation SALESPERSON, and
2. the attribute Trip_ID of relation EXPENSE that references relation TRIP.
In addition, the attributes Dept_No of relation SALESPERSON and Account# of relation EXPENSE are probably also foreign keys referencing other relations of the database not mentioned in the question. We now give the queries in relational algebra:

a. $$\text{COSTLY\_TRIPS} \leftarrow \text{SUM\_Amount} > 2000\,(\text{Trip\_Id}\ \mathcal{F}\ \text{SUM Amount}\,(\text{EXPENSE}))$$
$$\text{RESULT} \leftarrow \text{TRIP} * \pi_{\text{Trip\_Id}}\,(\text{COSTLY\_TRIPS})$$

b. $$\text{RESULT} \leftarrow \pi_{\text{SSN}}\,(\text{To\_City} = \text{'Honolulu'}\,(\text{TRIP}))$$

c. $$\text{RESULT} \leftarrow \mathcal{F}_{\text{SUM Amount}}\,(\text{SSN} = \text{'234-56-7890'}\,(\text{TRIP}) * \text{EXPENSE})$$

3.16 - Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course:

STUDENT (SSN, Name, Major, Bdate)
COURSE (Course#, Quarter, Grade)
ENROLL (SSN, Course#, Quarter, Grade)
BOOK_ADOPTION (Course#, Quarter, Book_ISBN)
TEXT (Book_ISBN, Book_Title, Publisher, Author)

Specify the foreign keys for this schema, stating any assumptions you make.

Answer:
The schema of this question has the following four foreign keys:
3. the attribute SSN of relation ENROLL that references relation STUDENT,
4. the attribute Course# in relation ENROLL that references relation COURSE,
5. the attribute Course# in relation BOOK_ADOPTION that references relation COURSE, and
6. the attribute Book_ISBN of relation BOOK_ADOPTION that references relation TEXT.
We now give the queries in relational algebra:

d. $\text{COURSES\_JS\_W99} \leftarrow \pi_{\text{Name} = \text{'John Smith'}}(\text{STUDENT}) * \pi_{\text{Quarter} = \text{'W99'}}(\text{ENROLL})$
$\text{RESULT} \leftarrow F_{\text{COUNT Course\#}}(\text{COURSES\_JS\_W99})$

e. $\text{CS\_ADOPTIONS} \leftarrow \pi_{\text{Course\#, Book\_ISBN}}(\pi_{\text{Dept} = \text{'CS'}}(\text{COURSE}) * \text{BOOK\_ADOPTION})$
$\text{BOOK\_COUNT} \leftarrow {}_{\text{Course\#}}F_{\text{COUNT Book\_ISBN}}(\text{CS\_ADOPTIONS})$
$\text{COURSES\_NEEDED} \leftarrow \pi_{\text{Course\#}}(\pi_{\text{COUNT\_Book\_ISBN} > 2}(\text{BOOK\_COUNT}))$
$\text{RESULT} \leftarrow \pi_{\text{Course\#, Book\_ISBN, Book\_Title}}(\text{COURSES\_NEEDED} * \text{BOOK\_ADOPTION} * \text{TEXT})$

f. $\text{DEPT\_PUBS} \leftarrow \pi_{\text{Dept, Publisher}}(\text{COURSE} * \text{BOOK\_ADOPTION} * \text{TEXT})$
$\text{RESULT} \leftarrow \pi_{\text{Dept}}(\text{COURSE}) - \pi_{\text{Dept}}(\pi_{\text{Publisher} \neq \text{'BC Publishing'}}(\text{DEPT\_PUBS}))$

It is interesting to observe that, contrary to intuition, the above expression does not employ the DIVISION operation, despite the fact that the query involves the word *all*. If, however, the query were to list any department that has adopted *all* books

published by 'BC Publishing', then the solution would be the following expression involving DIVISION:

$\text{DEPT\_BOOKS} \leftarrow \pi_{\text{Dept, Book\_ISBN}}(\text{COURSE} * \text{BOOK\_ADOPTION})$
$\text{BOOKS\_BY\_BC} \leftarrow \pi_{\text{Book\_ISBN}}(\pi_{\text{Publisher} = \text{'BC Publishing'}}(\text{TEXT}))$
$\text{RESULT} \leftarrow \text{DEPT\_BOOKS} \div \text{BOOKS\_BY\_BC}$

3.18 - Database design often involves decisions about the storage of attributes. For example a Social Security Number can be stored as a one attribute or split into three attributes (one for each of the three hyphen-deliniated groups of numbers in a Social Security Number—XXX-XX-XXXX). However, Social Security Number is usually stored in one attribute. The decision is usually based on how the database will be used. This exercise asks you to think about specific situations where dividing the SSN is useful.

Answer:
~~a.~~a. We need the area code (also know as city code in some countries) and perhaps the country code (for dialing international phone numbers).
~~b.~~b. I would recommend storing the numbers in a separate attribute as they have their own independent existence. For example, if an area code region were split into two regions, it would change the area code associated with certain numbers, and having area code in a separate attribute will make it is easier to update the area code attribute by itself.
~~c.~~c. I would recommend splitting first name, middle name, and last name into different attributes as it is likely that the names may be sorted and/or retrieved by the last name, etc.

Formatted: Bullets and Numbering

~~d.~~d. In general, if the each attribute has an independent logical existence based on the application, it would make sense to store it in a separate column otherwise there is no clear advantage. For example, SSN need not be split into its component unless we are using the subsequences to make deductions about validity, geography, etc. In the two cases above, it made logical and business sense to split the attributes.

3.19 - Consider a STUDENT relation in a UNIVERSITY database with the following attributes (Name, SSN, Local_phone, Address, Cell_phone, Age, GPA). Note that the cell phone may be from a different city and state (or province) from the local phone. A possible tuple of the relation is shown below:

Name	SSN	LocalPhone	Address	CellPhone	Age	GPA
George Shaw William Edwards	123-45-6789	555-1234	123 Main St., Anytown, CA 94539	555-4321	19	3.75

a. Identify the critical missing information from the LocalPhone and CellPhone attributes as shown in the example above. (Hint: How do call someone who lives in a different state or province?)
b. Would you store this additional information in the LocalPhone and CellPhone attributes or add new attributes to the schema for STUDENT?
c. Consider the Name attribute. What are the advantages and disadvantages of splitting this field from one attribute into three attributes (first name, middle name, and last name)?
d. What general guideline would you recommend for deciding when to store information in a single attribute and when to split the information.

Answer:
a. A combination of first name, last name, and home phone may address the issue assuming that there are no two students with identical names sharing a home phone line. It also assumes that every student has a home phone number. Another solution may be to use first name, last name, and home zip code. This again has a potential for duplicates, which would be very rare within one university. An extreme solution is to use a combination of characters from last name, major, house number etc.

b. If we use name in a primary key and the name changes then the primary key changes. Changing the primary key is acceptable but can be inefficient as any references to this key in the database need to be appropriately updated, and that can take a long time in a large database. Also, the new primary key must remain unique. [Footnote: Name change is an example of where our database must be able to model the natural world. In this case, we recognize that the name change can occur regardless of whether it is due to marriage, or a consequence of a religious and/or spiritual conversion, or for any other reason.]

c. The challenge of choosing an invariant primary key from the natural data items leads to the concept of generated keys, also known as surrogate keys. Specifically, we can use surrogate keys instead of keys that occur naturally in the database. Some database professionals believe that it is best to use keys that are uniquely generated by the database, for example each row may have a primary key that is generated in the sequence of creation of rows (tuples). There are many advantages and disadvantages that are often been argued in design sessions. The main advantage is that it gives us an invariant key without any worries about choosing a unique primary key. The main disadvantages of surrogate keys are that they do not have a business meaning (making some aspects of database management challenging) and that they are slightly less efficient (because they require another pass when inserting a row because the key often needs to be returned to the application after a row is

inserted).

3.20 - Recent changes in privacy laws have disallowed organizations from using SSN to identify individuals unless certain restrictions are satisfied. As a result, most US universities cannot use SSNs as primary keys (except for financial data). In practice, StudentID, a unique ID, a unique identifier, assigned to every student, is likely to be used as the primary key rather than SSN since StudentID is usable across all aspects of the system.

a. Some database designers are reluctant to use generated keys (also known as *surrogate* keys) for primary keys (such as StudentID) because they are artificial. Can you propose any natural choices of keys that can be used to store the student record in a UNIVERSITY database?
b. Suppose that you were able to guarantee uniqueness of a natural key that included last name. Are you guaranteed that the last name will not change during the lifetime of the database? If the last name last name can change, what solutions can you propose for creating a primary key that still includes last name but remains unique?
c. What are the advantages and disadvantages of using generated (surrogate) keys?

Answer:
a. By keeping the name attributes separated, we allow the possibility of looking these pieces of their name. In a practical use, it is not likely that the user will know the correct primary key for a given student and so we must consider how a user will locate the correct row without this information. If we were to collapse the name into a single attribute, then we have complicated any sort of “lookup by name” query; such a query would then require partial string matching and any results might not disambiguate between FirstName and LastName. Therefore, a practical system should allow name searches by FirstName and LastName; we must leave MiddleInitial separated still to avoid ambiguities from combining these pieces together.

b. A single attribute Phone# would no longer suffice if a student were able to have multiple phone numbers. We could possibly have multiple rows for a single student to allow this to happen, but then we have violated key principles of database design (e.g. having redundant data). A better solution would be to include the additional attributes HomePhone, CellPhone, and OfficePhone and allow the possibility of these attributes to have no value. Again, this is not most desirable because most students will not have all three of these attributes, and we will have many valueless key/attribute pairs. An excellent solution would be add an additional relation Phone# (SSN, Type, Number) while removing PhoneNumber from the Student relationship. This new relationship would allow the one-to-many relationship from students to phone numbers without creating redundant data or wasting space on sparse, valueless attributes.