

## Chapter 6: Relational Algebra

1. *Exercise 6.13.* What is meant by a safe expression in relational calculus?

An expression in relational calculus is *safe* if all the tuples returned by the expression appear in the relations or tuples used in the expression. Such expression is “safe” because it can only return a finite set of tuples as its result.

2. *Exercise 6.18.* Consider the LIBRARY relational database schema shown in Figure 6.14, which is used to keep track of books, borrowers, and book loans. Referential integrity constraints are shown as directed arcs in Figure 6.14, as in the notation of Figure 5.7. Write down ~~relational expressions~~ *relational algebra expressions* for the following queries:

- a. How many copies of the book titled *The Lost Tribe* are owned by the library branch whose name is ‘Sharpstown’?

$\mathbf{A} \rightarrow \sigma_{Title='The\ Lost\ Tribe'}(\mathbf{BOOK}) \bowtie_{Book\_id} \mathbf{BOOK\_COPIES}$   
 $\mathbf{B} \rightarrow \sigma_{Branch\_name='Sharpstown'}(\mathbf{LIBRARY\_BRANCH})$   
 $\mathbf{Answer} \rightarrow \Pi_{No\_of\_copies}(\mathbf{A} \bowtie_{Branch\_id} \mathbf{B})$

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

$\mathbf{LazyCardNumbers} \rightarrow \Pi_{Card\_no}(\mathbf{BORROWER}) - \Pi_{Card\_no}(\mathbf{BOOK\_LOANS})$   
 $\mathbf{Answer} \rightarrow \Pi_{Name}(\mathbf{LazyCardNumbers} \bowtie_{Card\_no} \mathbf{BORROWER})$

- d. For each book that is loaned out from the Sharpstown branch and whose *Due\_date* is today, retrieve the book title, the borrower’s name, and the borrower’s address.

$\mathbf{A} \rightarrow \sigma_{Branch\_name='Sharpstown'}(\mathbf{LIBRARY\_BRANCH})$   
 $\mathbf{B} \rightarrow \sigma_{Due\_date=today}(\mathbf{BOOK\_LOANS}) \bowtie_{Book\_id} \mathbf{BOOK}$   
 $\mathbf{C} \rightarrow \Pi_{Card\_no, Title}(\mathbf{A} \bowtie_{Branch\_id} \mathbf{B})$   
 $\mathbf{Answer} \rightarrow \Pi_{Title, Name, Address}(\mathbf{C} \bowtie_{Card\_no} \mathbf{BORROWER})$

3. *Exercise 6.26.* Specify queries c, d, f of Exercise 6.18 in both tuple and domain relational calculus.

- b. Retrieve the names of all borrowers who do not have any books checked out.

Tuple Relational Calculus :  
 $\{b.Name \mid \mathbf{BORROWER}(b) \wedge (\forall l)(\mathbf{BOOK\_LOANS}(l) \wedge b.Card\_no \neq l.Card\_no)\}$

Domain Relational Calculus:  
 $\{\langle b \rangle \mid \langle abcd \rangle \in \mathbf{BORROWER} \wedge (\forall g)(\langle efghi \rangle \in \mathbf{BOOK\_LOANS} \wedge a \neq g)\}$

- c. For each book that is loaned out from the Sharpstown branch and whose `Due_date` is today, retrieve the book title, the borrower's name, and the borrower's address.

Tuple Relational Calculus :

$$\{b.Title, br.Name, br.Address \mid \text{BOOK}(b) \wedge \text{BORROWER}(br) \\ \wedge (\exists bl)(\text{BOOK\_LOANS}(bl) \wedge bl.Card\_no = br.Card\_no \\ \wedge (\exists lbr)(\text{LIBRARY\_BRANCH}(lbr) \wedge lbr.Branch\_name = 'Sharpstown' \wedge bl.Branch\_id = \\ lbr.Branch\_id) \\ )\}$$

Domain Relational Calculus:

$$\{ \langle bef \rangle \mid \langle abc \rangle \in \text{BOOK} \wedge (\exists d)(\langle defg \rangle \in \text{BORROWER} \\ \wedge (\exists ij)(\langle hijkl \rangle \in \text{BOOK\_LOANS} \wedge d = j \wedge \\ (\exists mn)(\langle mno \rangle \in \text{LIBRARY\_BRANCH} \wedge n = 'Sharpstown' \wedge i = m) \\ ) \\ )\}$$

- f. **Question Withdrawn.** Check the newsgroup posting '*How to do aggregates in DRC*'.

## Chapter 8: SQL

4. *Exercise 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.

I will use the book's syntax (as in sections 8.1.2 and 8.2).

```
CREATE TABLE BOOK (
  Book_id INT NOT NULL,
  Title VARCHAR(50) NOT NULL,
  Publisher_name VARCHAR(50),
  PRIMARY KEY (Book_id),
  FOREIGN KEY (Publisher_name)
    REFERENCES PUBLISHER (Name)
    ON DELETE SET NULL
    ON UPDATE CASCADE
);
```

```
CREATE TABLE BOOK_AUTHORS (
  Book_Id INT NOT NULL,
  Author_name VARCHAR(50) NOT NULL,
  PRIMARY KEY (Book_id, Author_name),
  FOREIGN KEY (Book_id)
    REFERENCES BOOK (Book_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

```
CREATE TABLE PUBLISHER (
  Name VARCHAR(50) NOT NULL,
```

```
    Address VARCHAR(100),  
    Phone VARCHAR(10),  
    PRIMARY KEY (Name)  
);
```

```
CREATE TABLE BOOK_COPIES (  
    Book_id INT NOT NULL,  
    Branch_id INT NOT NULL,  
    No_of_copies INT NOT NULL,  
    PRIMARY KEY (Book_id, Branch_id),  
    FOREIGN KEY (Book_id)  
        REFERENCES BOOK (Book_id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (Branch_id)  
        REFERENCES LIBRARY_BRANCH (Branch_id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE  
);
```

```
CREATE TABLE BOOK_LOANS (  
    Book_id INT NOT NULL,  
    Branch_id INT NOT NULL,  
    Card_no INT NOT NULL,  
    PRIMARY KEY (Book_id, Branch_id, Card_no),  
    FOREIGN KEY (Book_id)  
        REFERENCES BOOK (Book_id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (Branch_id)  
        REFERENCES LIBRARY_BRANCH (Branch_id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (Card_no)  
        REFERENCES BORROWER (Card_no)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE  
);
```

```
CREATE TABLE LIBRARY_BRANCH (  
    Branch_id INT NOT NULL,  
    Branch_name VARCHAR(50) NOT NULL,  
    Address VARCHAR(100),  
    PRIMARY KEY (Branch_id)  
);
```

```
CREATE TABLE BORROWER (  
    Card_no INT NOT NULL,  
    Name VARCHAR(50) NOT NULL,  
    Address VARCHAR(100),
```

```

Phone VARCHAR(10),
PRIMARY KEY (Card_no)
);

```

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

- a. How many copies of the book titled *The Lost Tribe* are owned by the library branch whose name is 'Sharpstown'?

```

SELECT No_of_copies FROM BOOK_COPIES WHERE
    BOOK_COPIES.Book_id IN
        (SELECT Book_id FROM BOOK WHERE Title='The Lost Tribe')
AND
    BOOK_COPIES.Branch_id IN
        (SELECT Branch_id FROM LIBRARY_BRANCH
        WHERE Branch_name = 'Sharpstown')
;

```

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

```

SELECT Name from BORROWER WHERE
    BORROWER.Card_no NOT IN
        (SELECT Card_no from BOOK_LOANS)
;

```

- d. For each book that is loaned out from the Sharpstown branch and whose **Due\_date** is today, retrieve the book title, the borrower's name, and the borrower's address.

```

SELECT Title, Name, Address FROM
    BOOK,
    (
        SELECT Book_id, Name, Address from BORROWER, BOOK_LOANS
        WHERE BORROWER.Card_no = BOOK.Card_no AND
        BOOK_LOANS.Due_date = today AND
        Branch_id IN(
            SELECT Branch_id from LIBRARY_BRANCH WHERE
            Branch_name = 'Sharpstown')
    ) R
WHERE R.Book_id = BOOK.Book_id
;

```

6. *Exercise 8.14.* Specify the following additional queries on the database of Figure 5.5 in SQL. Show the query results if each query is applied to the database of 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, COUNT(*) AS number_of_employees
FROM DEPARTMENT, EMPLOYEE
WHERE DEPARTMENT.Dnumber = EMPLOYEE.Dno
AND
EMPLOYEE.Dno IN(
    SELECT Dno from EMPLOYEE

```

```

GROUP BY DNo
HAVING AVG(Salary) > 30000
)
GROUP BY Dname
;
For Dno = 5, average salary is  $\frac{30000+40000+38000+25000}{4} = 33250$ , which is > 30000.
For Dno = 4, average salary is  $\frac{25000+43000+25000}{3} = 31000$ , which is > 30000.
For Dno = 1, average salary is  $\frac{55000}{1} = 55000$ , which is > 30000.

```

So result is as follows:

Dno	number_of_employees
5	4
4	3
1	1

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

```

Select DName, COUNT(*) AS number_of_employees
FROM DEPARTMENT, EMPLOYEE
WHERE DEPARTMENT.Dnumber = EMPLOYEE.Dno
AND
EMPLOYEE.Dno IN(
SELECT Dno from EMPLOYEE
WHERE EMPLOYEE.Sex='Male'
GROUP BY DNo
HAVING AVG(Salary) > 30000
)
GROUP BY Dname
;

```

For Dno = 5, average male salary is  $\frac{30000+40000+38000}{3} \approx 39333$ , which is > 30000.  
For Dno = 4, average male salary is  $\frac{25000}{1} = 25000$ , which is < 30000. Ignore.  
For Dno = 1, average male salary is  $\frac{55000}{1} = 55000$ , which is > 30000.

So result is as follows:

Dno	number_of_employees
5	3
1	1

7. *Exercise 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.

```

INSERT INTO STUDENT (Name, Student_Number, Class, Major)
VALUES('Johnson',25,1,'Math')
;

```

- b. Change the class of student 'Smith' to 2.

```

UPDATE STUDENT
  SET Class = 2
  WHERE Name = 'Smith'
;

```

- c. Insert a new course, <'Knowledge Engineering', 'CS4930', 3, 'CS'>.

```

INSERT INTO COURSE (Course_name, Course_number, Credit_hours, Department)
  VALUES('Knowledge Engineering', 'CS4930', 3, 'CS')
;

```

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

```

DELETE FROM STUDENT
  WHERE Name='Smith' AND Student_number = 17
;

```

8. *Exercise 8.24.* Specify the following views in SQL on the COMPANY database schema shown in Figure 5.5.

- b. A view that has the employee name, supervisor name, and employee salary for each employee who works in the 'Research' department.

```

CREATE VIEW my_view (Employee_Fname, Supervisor_Fname, Employee_salary)
AS
SELECT E1.Fname, E2.FName, E1.Salary
  FROM EMPLOYEE E1, EMPLOYEE E2, DEPARTMENT D
  WHERE
    D.Dname='Research' AND
    D.Dnumber=E1.Dno AND E1.Super_ssn = E2.Ssn
;

```

- d. A view that has the project name, controlling department name, number of employees, and total hours worked per week on the project for each project *with more than one employee working on it*.

```

CREATE VIEW my_view
(Pname, Controlling_DName, Number_Of_Employees, Total_Hours_Per_Week)
AS
SELECT Pname, Dname, COUNT(*), SUM(Hours)
  FROM PROJECT P, DEPARTMENT D, WORKS_ON W
  WHERE P.Pnumber=W.Pno AND P.Dnum=D.Dnumber
  GROUP BY Pname, Dname
  HAVING COUNT(*) > 1
;

```

9. Consider the following view, DEPT\_SUMMARY, defined on the COMPANY database of Figure 5.6:

```

CREATE VIEW  DEPT_SUMMARY (D, C, Total_s, Average_s)
AS SELECT   Dno, Count(*), SUM (Salary), AVG(Salary)
FROM       EMPLOYEE
GROUP BY   Dno;

```

State which of the following queries and updates would be allowed on the view. If a query or update would be allowed, show what the corresponding query or update on the base relations would look like, and give its result when applied to the database of Figure 5.6.

b. **SELECT** D, C  
**FROM** DEPT\_SUMMARY  
**WHERE** Total\_s > 100000;

Only for Dno=5, Total\_s > 100000. So only D=5, C= 4 will be listed.

c. **UPDATE** DEPT\_SUMMARY  
**SET** D=3  
**WHERE** D=4;

This update could be allowed but Dno cannot be updated globally (namely, in the tables DEPARTMENT, DEPT\_LOCATIONS and PROJECT) without complete knowledge of the database state.

## Chapter 10: Functional Dependencies and Normalization

10. *Exercise 10.6.* Why can we not infer a functional dependency automatically from a particular relation state?

We cannot, in general, determine a functional dependency just by looking at a relation state because it may not hold for future relation states. (We may, however, be able to determine some functional dependencies that do not hold.)

11. *Exercise 10.18.* Prove or disprove the following inference rules for functional dependencies. A proof can be made either by a proof of argument or by using inference rules IR1 through IR3. A disproof should be performed by demonstrating a relation instance that satisfies the conditions and functional dependencies in the left-hand side of the inference rule but does not satisfy the dependencies in the right-hand side.

a.  $\{W \rightarrow Y, X \rightarrow Z\} \models \{WX \rightarrow Y\}$

$WX \rightarrow YX$  Augmentation on  $W \rightarrow Y$   
 $WX \rightarrow Y$  Decomposition

c.  $\{X \rightarrow Y, X \rightarrow W, WY \rightarrow Z\} \models \{X \rightarrow Z\}$

(1)  $XY \rightarrow WY$  Augmentation on  $X \rightarrow Y$   
(2)  $WY \rightarrow Z$  Given  
(3)  $XY \rightarrow Z$  Transitive rule on (1) and (2)  
(4)  $X \rightarrow Y$  Given  
(5)  $XX \rightarrow Z$  Pseudoinverse on (3) and (4)  
(6)  $X \rightarrow Z$  Transitive rule on  $X \rightarrow XX$  and (5)

e.  $\{X \rightarrow Z, Y \rightarrow Z\} \models \{X \rightarrow Y\}$

This inference is false. Consider the following relation instance :

X	Y	Z
a	b	c
a	d	c
e	b	c
g	h	d
i	h	d
g	k	d

Here  $X \rightarrow Z$  holds because whenever  $X = a$ , we have  $Z = c$  and whenever  $X = g$ , we have  $Z = d$ . Also,  $Y \rightarrow Z$  holds because whenever  $Y = b$ , we have  $Z = c$  and whenever  $Y = h$ , we have  $Z = d$ . And so on. However notice that  $X \rightarrow Y$  is false – look at the first two tuples above.

12. *Exercise 10.20.* Consider the relation schema EMP\_DEPT in Figure 10.3(a) and the following set  $G$  of functional dependencies on EMP\_DEPT:  $G = \{Ssn \rightarrow \{Ename, Bdate, Address, Dnumber\}, Dnumber \rightarrow \{Dname, Dmgr\_ssn\}\}$ . Calculate the closures  $\{Ssn\}^+$  and  $\{Dnumber\}^+$  with respect to  $G$ .

Use the closure algorithm discussed in class:

$$\begin{aligned} \{Ssn\}^+ &= \{Ssn\} \cup \{Ename\} \cup \{Bdate\} \cup \{Address\} \cup \{Dnumber\} \cup \{Dname\} \cup \{Dmgr\_ssn\} \\ &= \{Ssn, Ename, Bdate, Address, Dnumber, Dname, Dmgr\_ssn\} \end{aligned}$$

$$\begin{aligned} \{Dnumber\}^+ &= \{Dnumber\} \cup \{Dname\} \cup \{Dmgr\_ssn\} \\ &= \{Dnumber, Dname, Dmgr\_ssn\} \end{aligned}$$

13. *Exercise 10.24.* Prove that any relation with two attributes is in BCNF.

Review definition of BCNF in section 10.5. Let  $R$  be a relation with two attribute  $A$  and  $B$ , and consider all possible functional dependencies.

If both  $A \rightarrow B$  and  $B \rightarrow A$  holds, then both  $A$  and  $B$  are superkeys of  $R$  and thus  $R$  is in BCNF.

If only  $A \rightarrow B$  holds, then  $A$  is the superkey and thus  $R$  is still in BCNF.

Similarly, when only  $B \rightarrow A$  holds, then  $B$  is the superkey and thus  $R$  is in BCNF.

If none of the non-trivial functional dependencies hold, then the superkey is the set  $\{A, B\}$  and  $R$  is in BCNF.

Therefore, any relation with two attributes is in BCNF.

14. *Exercise 10.32.* Consider the following relation:

CAR\_SALE(Car#, Date\_sold, Salesman#, Commission%, Discount\_amt)

Assume that a car may be sold by multiple salesmen, and hence Car#, Salesman# is the primary key. Additional dependencies are

Date\_sold  $\rightarrow$  Discount\_amt and

Salesman#  $\rightarrow$  Commission%

Based on the given primary key, is this relation in 1NF, 2NF, or 3NF? Why or why not? How would you successively normalize it completely?

**1NF**  $\checkmark$  No set valued attributes.

**2NF** **X** The non-prime attribute Commission% is not *full functional dependent* on the candidate key {Car#, Salesman#} since Salesman#  $\rightarrow$  Commission% holds.

**3NF** **X** Since not in **2NF**. (Also, Date\_sold  $\rightarrow$  Commission% holds, but Date\_sold is neither a superkey nor is Commission% a prime attribute)

To normalize, we could break down the CAR\_SALE relation into the following relations: {Car#, Salesman#, Date\_sold}, {Salesman#, Commission%} and {Date\_sold, Discount\_amt}.

15. *Exercise 10.36*. Consider the following relation:

R (Doctor#, Patient#, Date, Diagnosis, Treat\_code, Charge)

In the above relation, a tuple describes a visit of a patient to a doctor along with a treatment code and daily charge. Assume that diagnosis is determined (uniquely) for each patient by a doctor. Assume that each treatment code has a fixed charge (regardless of patient). Is this relation in 2NF? Justify your answer and decompose if necessary. Then argue whether further normalization to 3NF is necessary, and if so, perform it.

The FDs are Treat\_code  $\rightarrow$  Charge, and {Doctor#, Patient#, Date}  $\rightarrow$  {Diagnosis, Treat\_code, Charge}.

**1NF**  $\checkmark$  No set valued attributes.

**2NF**  $\checkmark$  All non-prime attributes (namely, Diagnosis, Treat\_code and Charge) is *full functional dependent* on the candidate key {Doctor#, Patient#, Date}.

**3NF** **X** Treat\_code  $\rightarrow$  Charge holds but Treat\_code is neither a superkey nor is Charge a prime-attribute.

Since we are given that treatment code alone uniquely determines the charge, it's better to normalize further. To normalize, we could break down the relation R into the following relations: {Doctor#, Patient#, Date, Diagnosis, Treat\_code}, and {Treat\_code, Charge}.

However, if the charge of a treatment code could be changed (very likely), we will have no way to determine how much a patient was charged on a given date in the past. To fix this, we could store the amount charged using the following relations : {Doctor#, Patient#, Date, Diagnosis, Treat\_code, Amount\_charged}, and {Treat\_code, Charge}. The second relation could be used to determine the charge for future visits.