Chapter 1 The Worlds of Database Systems

There are no questions in this chapter.

Chapter 2 The Relational Model of Data

2.1 An Overview of Data Models

There are no questions in this section.

2.2 Basics of the Relational Model

Exercise 2.1

a)
Attributes of relation `Accounts` are `acctNo` , `type` and `balance`.
Attributes of relation `Customers` are `firstName` , `lastName`, `idNo` and `account`.
b)
For the relation `Accounts`:

- (12345, savings, 12000)
- (23456, checking, 1000)
- (34567, savings, 25)

For the relation `Customers`:

- (Robbie, Banks, 901-222 12345)
- (Lena, Hand, 805-333, 12345)
- (Lena, Hand, 805-333, 23456)

c)

- (12345, savings, 12000)
- (Robbie, Banks, 901-222 12345)

d)

- Accounts(acctNo, type, balance)
- Customers(firstName, lastName, idNo, account)

e)

- Accounts(acctNo: integer, type: string, balance: integer)
- Customers(firstName: string, lastName:string, idNo: string, account: integer)

f)

- Accounts(integer, string, integer)
- Customers(string, string, string, integer)

g)
Just swap the attributes (whatever you like).

- Accounts(type, accNo, balance)
- Customers(lastName, firstName, idNo, account)

Exercise 2.2

Student(id, firstName, lastName)
`id` is the key.

Exercise 2.3

It is a combinatorial problem. It is important how many attributes we have. Order of tuples doesn't matter.

a) $3 \times 2 = 6$

b) $4 \times 3 \times 2 \times 1 = 12$

c) $n!$

2.3 Defining a Relation Schema in SQL

Exercise 3.1

a)

```
1  CREATE TABLE Product(
2    maker VARCHAR(50),
3    model INT,
4    type VARCHAR(25),
5    PRIMARY_KEY(maker)
6  );
```

b)

```
CREATE TABLE PC(
  model INT,
  speed FLOAT,
  ram INT,
  hd INT,
  price FLOAT,
  PRIMARY_KEY(model)
);
```

c)

```
CREATE TABLE Laptop(
  model INT,
  speed FLOAT,
  ram INT,
  hd INT,
  screen FLOAT,
  price FLOAT,
  PRIMARY_KEY(model)
);
```

d)

```
CREATE TABLE Printer(
  model INT,
  color BOOLEAN,
  type VARCHAR(25),
  price FLOAT,
  PRIMARY_KEY(model)
);
```

e)

```
ALTER TABLE Printer DROP color
```

f)

```
ALTER TABLE Laptop ADD od VARCHAR(25) DEFAULT 'none'
```

Exercise 3.2

a)

```
CREATE TABLE classes(
  class VARCHAR(50) UNIQUE,
  type VARCHAR(50),
  country VARCHAR(50),
  numGums INT,
  bore FLOAT,
  displacement INT
);
```

b)

```
CREATE TABLE Ships(
  name VARCHAR(50),
  class VARCHAR(59),
  launched INT
);
```

c)

```
CREATE TABLE Battles(
  name VARCHAR(50) UNIQUE,
  data DATE
);
```

d)

```
CREATE TABLE Outcomes(
  ship VARCHAR(50),
  battle VARCHAR(50),
  result VARCHAR(50)
);
```

e)

```
ALTER TABLE Classes DROP bore;
```

f)

```
ALTER TABLE Ships ADD yard VARCHAR(50);
```

2.4 An Algebraic Query Language

Exercise 4.1

a)

$$R1 := \sigma_{speed \geq 3.00}(PC)$$
$$R2 := \pi_{model}(R1)$$

b)

$$R1 := \sigma_{hd \geq 100}(Laptop)$$
$$R2 := Product \bowtie (R2)$$
$$R3 := \pi_{maker}(R3)$$

c)

$$R1 := \sigma_{maker=B}(Product \bowtie PC)$$
$$R2 := \sigma_{maker=B}(Product \bowtie Laptop)$$
$$R3 := \sigma_{maker=B}(Product \bowtie Printer)$$
$$R4 := \pi_{model,price}(R1) \cup \pi_{model,price}(R2) \cup \pi_{model,price}(R3)$$

d)

$$R1 := \sigma_{color=true \text{ AND } type=layser}(Printer)$$
$$R2 := \pi_{model}(R1)$$

e)

$$R1 := \sigma_{type=laptop}(Product)$$
$$R2 := \sigma_{type=pc}(Product)$$
$$R3 := \pi_{maker}(R1) - \pi_{maker}(R2)$$

f)

$$R1 := \rho_{PC1}(PC)$$
$$R2 := \rho_{PC2}(PC)$$
$$R3 := R1 \bowtie_{PC1.hd=PC2.hd \text{ AND } PC1.model<>PC2.model} R2$$
$$R4 := \pi_{hd}(R3)$$

g)

The most important thing here is to notice that *a pair should be listed only once.* Here we use `PC1.model < PC2.model` to achieve this functionality.

$$R1 := \rho_{PC1}(PC)$$
$$R2 := \rho_{PC2}(PC)$$
$$R3 := R1 \bowtie_{PC1.speed=PC2.speed \text{ AND } PC1.ram=P2.ram \text{ AND } PC1.model<PC2.model} R2$$

h)

$$R1 := \pi_{model}(\sigma_{speed \geq 2.80}(PC)) \cup \pi_{model}(\sigma_{speed \geq 2.80}(Laptop))$$
$$R2 := \pi_{maker,model}(R1 \bowtie Product)$$
$$R3 := \rho_{R3(maker2,model2)}(R2)$$
$$R4 := R2 \bowtie_{maker=maker2 \text{ AND } model<>model2} R3$$
$$R5 := \pi_{maker}(R4)$$

i)

The most tricky thing is to understand how to find the way to get the maximum. In C, we could use a temp variable to record the current maximum in the loop. However, we cannot do that using relation algebra.

Here, I give an example, suppose we have an only one attribute relation A. For simplicity, I choose $A = (a_1, \cdots, a_n)^T$. I do the cartesian product. If there is a maximum n, we could get that for all $a_i, i = 1, \ldots, n, n \geq a_i$. Then idea comes.

$$
\begin{aligned}
R1 &:= \pi_{model,speed}(PC) \\
R2 &:= \pi_{model,speed}(Laptop) \\
R3 &:= R1 \cup R2 \\
R4 &:= \rho_{R4(model2,speed2)}(R3) \\
R5 &:= \pi_{model,speed}(R3 \bowtie_{speed<speed2} R4) \\
R6 &:= R3 - R5 \\
R7 &:= \pi_{maker}(R6 \bowtie Product)
\end{aligned}
$$

j)

$$
\begin{aligned}
R1 &:= \pi_{maker,speed}(PC \bowtie Product) \\
R2 &:= \rho_{R2(maker2,speed2)}(R1) \\
R3 &:= \rho_{R3(maker3,speed3)}(R1) \\
R4 &:= R1 \bowtie_{maker=maker2 \ \textbf{AND} \ speed<>speed2} R2 \\
R5 &:= R4 \bowtie_{maker=maker3 \ \textbf{AND} \ speed3<>speed2 \ \textbf{AND} \ speed3<>speed} R3 \\
R6 &:= \pi_{maker}(R5)
\end{aligned}
$$

k)

$$
\begin{aligned}
R1 &:= \pi_{maker,model}(PC \bowtie Product) \\
R2 &:= \rho_{R2(maker2,model2)}(R1) \\
R3 &:= \rho_{R3(maker3,model3)}(R1) \\
R4 &:= \rho_{R4(maker4,model4)}(R1) \\
R5 &:= R1 \bowtie_{maker=maker2 \ \textbf{AND} \ model<>model2} R2 \\
R6 &:= R3 \bowtie_{maker=maker3 \ \textbf{AND} \ model3<>model2 \ \textbf{AND} \ model3<>model} R5 \\
R7 &:= R4 \bowtie_{maker4=maker \ \textbf{AND} \ (model4=model \ \textbf{OR} \ model4=model2 \ \textbf{OR} \ model4=model3)} R6 \\
R8 &:= \pi_{maker}(R7)
\end{aligned}
$$

Exercise 4.2

It's a dirty job. I omit detail here. But do remember *a query tree is a tree data structure representing a relational algebra expression.*

Exercise 4.3

a)

$$R1 := \sigma_{bore \geq 16}(Classes)$$
$$R2 := \pi_{class,country}(R1)$$

b)

$$R1 := \sigma_{launched < 1921}(Ships)$$
$$R2 := \pi_{name}(R1)$$

c)

$$R1 := \sigma_{battle=DenmarkStrait \text{ AND } result=sunk}(Outcomes)$$
$$R2 := \pi_{ship}(R1)$$

d)

$$R1 := Classes \bowtie_{launched>1921 \text{ AND } displacement>35000} Ships$$
$$R2 := \pi_{name}(R1)$$

e)

$$R1 := \sigma_{battle=Guadalcanal}(Outcomes)$$
$$R2 := Ships \bowtie_{ship=name} R1$$
$$R3 := Classes \bowtie R2$$
$$R4 := \pi_{name,displacement,numGuns}(R3)$$

f)

$$R1 := \pi_{ship}(Outcomes)$$
$$R2 := \rho_{R2(name)}(R1)$$
$$R3 := \pi_{name}(Ships \cup R2)$$

g)

$$R1 := \pi_{class}(Classes)$$
$$R2 := \pi_{class}(\sigma_{class<>name}(Ships))$$
$$R3 := R1 - R2$$

h)

$$R1 := \pi_{country}(\sigma_{type=bb}(Classes))$$
$$R2 := \pi_{country}(\sigma_{type=bc}(Classes))$$
$$R3 := R1 \cap R2$$

i)

$$R1 := \pi_{ship,result,date}(Battles \bowtie_{battle=name} Outcomes)$$
$$R2 := \rho(R2(ship, result, date))(R1)$$
$$R3 := \pi_{ship}(R1 \bowtie_{ship=ship2 \textbf{ AND } result=damaged \textbf{ AND } date<date2} R2)$$

Exercise 4.4

Just like Exercise 4.2, I omit detail here.

Exercise 4.5

Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.

Exercise 4.6

Here we assume that there are two relations:

- $A(a_1, a_2, \ldots, a_n)$
- $B(b_1, b_2, \ldots, b_n)$

First, the $\cap$ operation is *monotone*. Assume that $C = A \cap B$. When A adds a tuple a_{n+1}, C may be not changed or become $C' = C \cup a_{n+1}$ such that $C \subseteq C'$. For other situations, the idea is the same.

Second, the σ operator is *monotone*. Assume that $C = \sigma_c(A)$. When A adds a tuple a_{n+1}, C may be not changed or become $C' = C \cup a_{n+1}$ such that $C \subseteq C'$.

Exercise 4.7

a)

- $\max = n + m$
- $\min = \max(n, m)$

b)

- max $= n \times m$
- min $= 0$

c)

- max $= n \times m$
- max $= 0$

d)

- max $= n$
- min $= 0$

Exercise 4.8

- $\pi_{1...n}(R \bowtie S)$
- $\pi_{1...n}(\sigma_{R.1=S.1...R.K=S.K}(R \times S))$
- $R \bowtie_{1,...k} (S)$

Exercise 4.9

$R - \pi_{1,...,n}(R \bowtie S)$

Exercise 4.10

An intuitive property of the division operator of the relational algebra is simply that it is the inverse of the cartesian product.

2.5 Constraints on Relations

Exercise 5.1

a)

$$\sigma_{speed<2.00 \text{ AND } price>500} = \emptyset$$

b)

$$\sigma_{screen<15.4 \text{ AND } hd<100 \text{ AND } price\geq1000}(Laptop) = \emptyset$$

c)

$$\pi_{maker}(\sigma_{type=pc}(Product)) \cap \pi_{maker}(\sigma_{type=laptop}(Product)) = \emptyset$$

d)

$$\begin{aligned}
R1 &:= \pi_{maker,model,speed}(Product \bowtie PC) \\
R2 &:= \pi_{maker,speed}(Product \bowtie Laptop) \\
R3 &:= \pi_{model}(R1 \bowtie_{R1.maker=R2.maker \textbf{ AND } R1.speed>R2.speed} R2) \\
R4 &:= \pi_{model}(R3) = \emptyset
\end{aligned}$$

e)

$$\begin{aligned}
R1 &:= \pi_{ram,price}(PC) \\
R2 &:= \pi_{ram,price}(Laptop)
\end{aligned}$$

And we can get constraints such that

$$R1 \bowtie_{R2.ram>R1.ram \textbf{ AND } R2.price\leq R1.price} R2 = \emptyset$$

Exercise 5.2

a)

$$\pi_{Class}(\sigma_{bore>16}(Classes)) = \emptyset$$

b)

$$\pi_{Class}(\sigma_{bore>16 \textbf{ AND } numGuns>9}(Classes)) = \emptyset$$

c)

$$\pi_{Class}(\sigma_{ship>2}(Outcome)) = \emptyset$$

d)

$$\pi_{country}(\sigma_{type=battlecruiser}(Classes)) \cap \pi_{country}(\sigma_{type=battleship}(Classes)) = \emptyset$$

e)

$$R1 := \sigma_{numGuns>9}(Classes)$$
$$R2 := \sigma_{numGuns<9}(Classes)$$
$$R3 := (R1 \bowtie Ships) \bowtie Battles$$
$$R4 := (R2 \bowtie Ships) \bowtie \sigma_{result=sunk}(Battles)$$
$$R5 := \pi_{battlename}(R3) \bowtie \pi_{battlename}(R4) = \emptyset$$

Exercise 5.3

$$\pi_{v_1,v_2,\ldots,v_n}(A_1, A_2, \ldots, A_n) \subseteq \pi_{v_1,v_2,\ldots,v_n}(B_1, B_2, \ldots, B_n)$$

Exercise 5.4

If we have 3 relational-algebra expressions for example $E1$, $E2$ and $E3$ we can not express them like $E1 = E2 = E3$. Thus, the answer is no.

Chapter 3 Design theory for Relational Databases

3.1 Functional Dependencies

Exercise 1.1

The FD's:

- Social Security number → name
- area code → state
- street address, city, state → ZIP code
- personal number → name

Possible keys are any attributes on the left hand side of the arrows.

Exercise 1.2

Let x, y, z are coordinates of the molecule and v_x, v_y, v_z are the velocities of the corresponding coordinates.

The FD's:

- $x, y, z \rightarrow v_x$
- $x, y, z \rightarrow v_y$
- $x, y, z \rightarrow v_z$

x, y, z and v_x, v_y, v_z are the keys because with these combination only we can get molecule in the closed surface.

Exercise 1.3

a)

Apart A_1 there are $n-1$ attributes to be added. For each attributes, there are tow choices: whether to include it or not. Due to the reason that set is unordered. Hence

$$|A_1| = 2^{n-1}$$

b)

$$|A_1 \cup A_2| = |A_1| + |A_2| - |A_1 \cap A_2| = 2^{n-1} + 2^{n-1} - 2^{n-2}$$

c)

$$|A_1, A_2| + |A_3, A_4| - |A_1, A_2, A_3, A_4| = 2^{n-2} + 2^{n-2} - 2^{n-4}$$

d)

$$|A_1, A_2| + |A_1, A_3| - |A_1, A_2, A_3| = 2^{n-2} + 2^{n-2} - 2^{n-3}$$

3.2 Rules About Functional Dependencies

Exercise 2.1

a)

First, we need to find all the closures of the subsets of the attributes.

$$\begin{aligned}
\{A\}^+ = \{A\}, \{B\}^+ = \{B\}, \{C\}^+ = \{A, C, D\}, \{D\}^+ = \{A, D\} \\
\{AB\}^+ = \{A, B, C, D\}, \{AC\}^+ = \{A, C, D\}, \{AD\}^+ = \{A, D\} \\
\{BC\}^+ = \{A, B, C, D\}, \{BD\}^+ = \{A, B, C, D\}, \{CD\}^+ = \{A, C, D\} \\
\{ABC\}^+ = \{A, B, D\}^+ = \{BCD\}^+ = \{A, B, C, D\}, \{ACD\}^+ = \{A, C, D\}
\end{aligned}$$

The answer is obvious.

b)

From a), it is easy to get that the keys are:

$$AB, BC \textbf{ AND } BD$$

c)

The superkeys are all those that contain one of those three keys. The proper superkeys are:

$$ABC, ABD, BCD \textbf{ AND } ABCD$$

Exercise 2.2

i)

It is obvious that $\{A\}^+ = \{A, B, C, D\}$. So there is no need to calculate the closures of subsets which contain A. Use this idea, we can simplify the calculation.

$$\begin{aligned} \{A\_\ \_\ \_\}^+ &= \{A, B, C, D\} \\ \{B\ \_\ \_\}^+ &= \{B, C, D\} \\ \{C\}^+ &= \{C\}, \{D\}^+ = \{D\}, \{CD\}^+ = \{C, D\} \end{aligned}$$

The keys are:

$$A$$

The superkeys are:

$$AB, AC, AD, ABC, ABD, ACD \textbf{ AND } ABCD$$

ii)

$$\begin{aligned} \{A\}^+ = \{A\}, \{B\}^+ = \{B\}, \{C\}^+ = \{C\}, \{D\}^+ = \{D\} \\ \{AB\}^+ = \{A, B, C, D\}, \{AC\}^+ = \{A, C\}, \{AD\}^+ = \{A, B, C, D\} \\ \{BC\}^+ = \{A, B, C, D\}, \{BD\}^+ = \{B, D\}, \{CD\}^+ = \{A, B, C, D\} \\ \{ABC\}^+ = \{A, B, D\}^+ = \{BCD\}^+ = \{ACD\}^+ = \{A, B, C, D\} \end{aligned}$$

The keys are:

$$AB, AD, BC \textbf{ AND } CD$$

The superkeys are:

$$ABC, ABD, BCD, ACD \textbf{ AND } ABCD$$

iii)

$$\{A\_\ \_\ \_\}^+ = \{A, B, C, D\}$$
$$\{B\_\ \_\ \_\}^+ = \{A, B, C, D\}$$
$$\{C\_\ \_\ \_\}^+ = \{A, B, C, D\}$$
$$\{D\_\ \_\ \_\}^+ = \{A, B, C, D\}$$

The keys are:

$$A, B, C \textbf{ AND } D$$

The superkeys are:

$$\text{Subset}\{A, B, C, D\} - \text{keys}$$

Exercise 2.3

a)

$$\{A_1, A_2, \cdots, A_n\}^+ = \{A_1, A_2, \cdots, A_n, B\}$$

And we can easily get:

$$\{A_1, A_2, \cdots, A_n, C\}^+ = \{A_1, A_2, \cdots, A_n, B\}$$

Thus proved.

b)

It can be easily proved by trivial FD.

c)

$$C_1C_2 \cdots C_k \to D \Rightarrow B_1B_2 \cdots B_mE_1E_2 \cdots E_j$$

Use the augmentation rule.

$$A_1A_2 \cdots A_n \to B_1B_2 \cdots B_m \Rightarrow A_1A_2 \cdots A_nE_1E_2 \cdots E_j \to B_1B_2 \cdots B_mE_1E_2 \cdots E_j$$

Use the transition rule.

$$A_1A_2 \cdots A_nE_1E_2 \cdots E_j \to D$$

d)

$$\{A_1, A_2, \cdots, A_n\}^+ = \{A_1, A_2, \cdots, A_n, B_1, B_2 \cdots B_m\}$$

$$\{C_1, C_2, \cdots, C_k\}^+ = \{C_1, C_2, \cdots, C_k, D_1, D_2 \cdots D_j\}$$

Using augmentation and combing rules, thus proved.

Exercise 2.4

a)

Attribute A represents Social Security Number and B represented a person's name.

b)

Attribute A represents Social Security Number and B represented a person's gender, C represents a person's name.

c)

Attribute A represents latitude and B represents longitude, C represents a point on the world map.

Exercise 2.5

Consider the relationship $R(A, B, C, D)$. Suppose there exists a non-trivial dependency $A \to B$. Then:

- There exist an attribute B that can be functionally determined by all other attributes.
- For some value of A the value of B can be functionally determined

It is given that no attribute can be functionally determined by all other attributes. This is contradiction.

Exercise 2.6

First, we convert $X \subseteq Y$ to $Y = X + S$. It is obvious that $\{Y\}^+ = \{X\}^+ + \{S\}^+$. Thus

$$X^+ \subseteq Y^+$$

Exercise 2.7

We prove it by concept, when we find the closure of X, it must go to the end. So for $(X^+)^+$, it can't find any suitable attributes.

Exercise 2.8

a)

$$A \to A$$
$$B \to B$$
$$C \to C$$
$$D \to D$$

b)

$$ABC \to D$$

c)

$$A \to B$$
$$ABC \to D$$

Exercise 2.9

- $A \to B, B \to A, B \to C, C \to B$
- $A \to B, B \to C, C \to A$
- $B \to A, A \to C, C \to B$
- $C \to A, A \to B, B \to C$
- $A \to C, B \to A, C \to A, C \to B$
- $C \to A, A \to B, B \to A, B \to C$
- $C \to B, C \to A, B \to C, A \to B$
- $B \to C, B \to A, A \to B, C \to B$
- $A \to B, A \to C, B \to A, C \to B$

Exercise 2.10

a)

$$\{A\}^+ = \{A\}, \{B\}^+ = \{B\}, \{C\}^+ = \{A, C, E\}$$
$$\{AB\}^+ = \{A, B, C, D, E\}, \{BC\}^+ = \{A, B, C, E\}$$
$$\{AC\}^+ = \{A, C, E\}, \{ABC\}^+ = \{A, B, C, D, E\}$$

So we can find the FD in S: $AB \to C$ and $C \to A$.

b)

$$\{A\}^+ = \{A, D\}, \{B\}^+ = \{B\}, \{C\}^+ = \{C\}$$
$$\{AB\}^+ = \{A, B, D, E\}, \{BC\}^+ = \{B, C\}$$
$$\{AC\}^+ = \{A, C, E\}, \{ABC\}^+ = \{A, B, C, D, E\}$$

From the above closure, A and C are the only attributes presents in S. So we can find FD in S: $A \to B, C \to B$.

c)

$$\{A\}^+ = \{A\}, \{B\}^+ = \{B\}, \{C\}^+ = \{C\}$$
$$\{AB\}^+ = \{A, B, D\}, \{BC\}^+ = \{A, B, C, D, E\}$$
$$\{AC\}^+ = \{A, B, C, D, E\}, \{ABC\}^+ = \{A, B, C, D, E\}$$

For $\{AB\}^+$, A and B are the only attributes presents in S.For $\{AC\}^+$ and $\{BC\}^+$, A, B and C are the only attributes. So we can find FD in S: $A \to B, B \to A, C \to B, C \to A$.

d)

$$\{A\_\ \_\}^+ = \{A, B, C\}$$
$$\{B\_\ \_\}^+ = \{A, B, C\}$$
$$\{C\_\ \_\}^+ = \{A, B, C\}$$

So we can find FD in S: $A \to B, B \to C, C \to A$.

Exercise 2.11

We have relation schema $S(A, B, C, D)$ with

$$A \to B$$
$$B \to C$$
$$B \to D$$

We can compute $\{A\}^+ = \{A, B, C, D\}$. So we can get

$$A \to B$$
$$B \to C$$

Gives $A \to C$. Thus, if FD F follows from the given FD's, then F can be proved from the given FD's using Armstrong's axioms.

3.3 Design of Relational Database Schemas

Exercise 3.1

a)

$$\{AB\}^+ = \{A, B, C, D\}$$
$$\{C\}^+ = \{A, C, D\}$$
$$\{D\}^+ = \{A, D\}$$

The second and the third FD's violate the BCNF.

By choosing FD $C \to D$, we split the relation into $R_1(A, C, D)$ and $R_2(B, C)$

Next, we need to find FD's in R_1 and R_2. There are only two attributes in R_2, so R_2 is in BCNF.

$$\{A\}^+ = \{A\}, \{C\}^+ = \{A, C, D\}, \{D\}^+ = \{A, D\}$$
$$\{AC\}^+ = \{A, C, D\}, \{AD\}^+ = \{A, D\}$$
$$\{CD\}^+ = \{A, C, D\}$$

So we can find that the FD's in R_1 are: $C \to D, D \to A$. D is not a superkey. So again, the FD $D \to A$ is not in BCNF and we decompose R_1 into $R_3(A, D)$ and $R_4(C, D)$.

b)

We simply compute $\{B\}^+ = \{B, C, D\}$. Since A is not in the closure, there is a violation of the BCNF.

By choosing FD $B \to C$, we split the relation into $R_1(B, C, D)$ and $R_2(A, B)$.

Next, we need to find FD's in R_1 and R_2. There are only two attributes in R_2, so R_2 is in BCNF.

$$\{B\}^+ = \{B, C, D\}$$
$$\{C\}^+ = \{C\}$$
$$\{D\}^+ = \{D\}$$
$$\{CD\}^+ = \{C, D\}$$

So we can find the FD's in R_1 are: $B \to C$ and $B \to D$.

c)

$$\{AB\}^+ = \{A, B, C, D\}$$
$$\{BC\}^+ = \{A, B, C, D\}$$
$$\{CD\}^+ = \{A, B, C, D\}$$
$$\{AD\}^+ = \{A, B, C, D\}$$

No BCNF violations, Done.

d)

$$\{A\}^+ = \{A, B, C, D\}$$
$$\{B\}^+ = \{A, B, C, D\}$$
$$\{C\}^+ = \{A, B, C, D\}$$
$$\{D\}^+ = \{A, B, C, D\}$$

No BCNF violations, Done.

e)

$$\{AB\}^+ = \{A, B, C, D\}$$
$$\{DE\}^+ = \{B, C, D, E\}$$
$$\{B\}^+ = \{B, D\}$$

All three FD's violate BCNF.

By choosing FD $AB \to C$, we split the relation into $R_1(A, B, C, D)$ and $R_2(A, B, E)$.

For R_2:

$$\{A\}^+ = \{A\}, \{B\}^+ = \{B\}, \{E\}^+ = \{E\}$$
$$\{AB\}^+ = \{A, B\}, \{BE\}^+ = \{B, E\}$$
$$\{AE\}^+ = \{A, E\}, \{ABE\}^+ = \{A, B, E\}$$

R_2 doesn't violate BCNF.

For R_1:

$$\{A\}^+ = \{A\}, \{B\}^+ = \{B, D\}, \{C\}^+ = \{C\}, \{D\}^+ = \{D\}$$
$$\{AB\}^+ = \{A, B, C, D\}$$

So we can find the FD's in R_1 are: $AB \to C$,$AB \to D$ and $B \to D$.

Since $B \to D$ violates BCNF. So we again decompose R_1 into $R_3(B, D)$ and $R_4(A, B, C)$. Done.

f)

For simplicity, we only give one answer here:

$$R_1(A), R_2(A, B), R_3(A, C), R_4(A, D), R_5(D, E)$$

Exercise 3.2

For $A \to B$ and $A \to C$:

$$\{A\}^+ = A, B, C$$

We decompose R into $R_1(A, B, C)$ and $R_2(A, D)$

For $A \to BC$, the $\{A\}^+$ won't be changed. So we can get the same result.

Exercise 3.3

Well, from Exercise 3.2, we can know that $\{A\}^+$ will never be changed.So we can get the same result.

Exercise 3.4

Factory	**Model**	**Year**
Nokia	3310	2000
Samsung	s3	2012
Samsung	s20	2020

3.4 Decomposition: The Good, Bad, and Ugly

Exercise 4.1

A	B	C	D	E
a	b	c	d_1	e_1
a_1	b	c	d	e_2
a	b_1	c	d_2	e

a)

The result of chasing test is below:

A	B	C	D	E
a	b	c	d_1	e_1
a	b	c	d	e_1
a	b_1	c	d_2	e

Therefore, the decomposition is not lossless.

b)

The result of chasing test is below:

A	B	C	D	E
a	b	c	d	e
a	b	c	d	e_2
a	b_1	c	d_2	e

The first row has no subscripted symbols and the decomposition is lossless

c)

The result of chasing test is below:

A	B	C	D	E
a	b	c	d	e
a_1	b	c	d	e_2
a	b_1	c	d_1	e

The first row has no subscripted symbols and the decomposition is lossless

d)

A	B	C	D	E
a	b	c	d_1	e
a_1	b	c	d	e_2
a	b_1	c	d_1	e

Therefore, the decomposition is not lossless.

Exercise 4.2

a)

There is no relation that contain BE so this is not dependency preserving. $B \rightarrow E$ is not preserved.

b)

There is relation $\{A, C, E\}$ that contains ACE so $AC \rightarrow E$ is dependency preserving.

There is relation $\{B, C, D\}$ that contains BCD so $BC \rightarrow D$ is dependency preserving.

c)

$A \rightarrow D$ and $D \rightarrow E$ are not preserved.

d)

None of the FD's is preserved.

3.5 Third Normal Form

Exercise 5.1

a)

We first find all the keys.

$$\{AB\}^+ = \{A, B, C, D\}$$
$$\{BD\}^+ = \{A, B, C, D\}$$

Thus we can get the primes are $\{A, B, D\}$. For $AB \to C$, AB is the key itself. And for $C \to D$ and $D \to A$, the right side consists of prime attributes. It is in 3NF.

b)

We first find all the keys.

$$\{AB\}^+ = \{A, B, C, D\}$$

Thus we can get the primes are $\{A, B\}$. For $B \to C$ and $B \to D$. B is not the superkey. And neither the rights sides consist of prime attributes. Thus it violates 3NF.

Make relation $R_1(A, B)$ and $R_2(B, C, D)$ in which B is candidate key.

c)

We first find all the keys.

$$\{AB\}^+ = \{A, B, C, D\}$$
$$\{BC\}^+ = \{A, B, C, D\}$$
$$\{CD\}^+ = \{A, B, C, D\}$$
$$\{AD\}^+ = \{A, B, C, D\}$$

Thus we can get the primes are $\{A, B, C, D\}$. In all FDs, left side is a key. It is in 3NF.

d)

We first find all the keys.

$$\{A\}^+ = \{A, B, C, D\}$$
$$\{B\}^+ = \{A, B, C, D\}$$
$$\{C\}^+ = \{A, B, C, D\}$$
$$\{D\}^+ = \{A, B, C, D\}$$

In all FDs, left side is a key. It is 3NF.

e)

We first find all the keys.

$$\{ABE\}^+ = \{A, B, C, D, E\}$$

Thus we can get the primes are $\{A, B, E\}$. For $B \to C$ It obviously violates 3NF

Make relation $R_1(A, B, C, D)$, $R_2(D, E, C)$ and $R_3(A, B, E)$

f)

For simplicity, we only give one answer here:

$R_1(A, B, C, D)$ and $R_2(D, E, B)$

Exercise 5.2

a)

Key for the relation Courses is HS.

b)

FD's are already minimal basis, as no attributes can be removed from the left side, right side is having only single attributes and none of the FDcan be removed as given in the question.

c)

By using the 3NF synthesis algorithm, the final set of relations obtained will be CT, HRC, HTR, HSR, CSG . All the relations are in BCNF.

Exercise 5.3

a)

Key for the relation Courses is IS.

b)

The given set of FD's are the minimal basis.

c)

The final set of decomposed relations is SD, IB, ISQ and BO.

Exercise 5.4

A	B	C	D	E
a	b	c	d_1	e_1
a	b_2	c_2	d	e_2
a	b	c_3	d_3	e

For $AB \to C$ and $C \to B$, we can get

A	B	C	D	E
a	b	c	d_1	e_1
a	b_2	c_2	d	e_2
a	b	c	d_3	e

For $A \to D$, we can get

A	B	C	D	E
a	b	c	d	e_1
a	b_2	c_2	d	e_2
a	b	c	d	e

This decomposition is lossless.

Exercise 5.5

Need help

3.6 Multivalued Dependencies

Exercise 6.1

we can pair the B-value from any tuple with the value of the remaining attribute C from any other tuple.

Thus the tuples (a, b_1, c_2), (a, b_1, c_3), (a, b_2, c_1), (a, b_2, c_3), (a, b_3, c_1), and a, b_3, c_2 are also in R.

Exercise 6.2

a)

From the description, we can get the following FD.

$$\begin{aligned} S &\to NB \\ CS &\to CN \\ CS &\to CB \\ AS &\to AM \end{aligned}$$

Also, we can get the following MVD.

$$\begin{aligned} S &\twoheadrightarrow NB \\ S &\twoheadrightarrow AS \\ S &\twoheadrightarrow am \end{aligned}$$

b)

- $R_1(S, CS, A)$
- $R_2(S, N, B)$
- $R_2(CS, CN, B)$
- $R_4(AS, M)$

Exercise 6.3

a)

Left side is not a key and holds more than one MVD so it violates 4NF.

We could decompose R into $R_1(A, B)$, $R_2(A, C)$ and $R_3(A, D)$.

b)

For $B \twoheadrightarrow CD$, left side is not a key. So it violates 4NF.

We could decompose R into $R_1(A, B)$, $R_2(B, C, D)$.

c)

For $B \twoheadrightarrow D$, left side is not a key. It violates 4NF.

We could decompose R into $R_1(A, B, C)$, $R_2(B, D)$.

d)

We could easily calculate $\{A\}^+ = A, B, C, D, E$. So all the left sides are keys. so it is in 4NF.

Exercise 6.4

We have five attributes $(name, street, city, title, year)$

- City is not determined by other four attributes as there can be same street address with different city.
- Year is not determined as there can be different address of one star.
- Title is not determines as with two same movie, different address is present.
- Street is not determined as with two same title, street is different.
- Name is not determined as with two same year and same movie star have different address.

None of the five attributes is determined by other four.

3.7 An Algorithm for Discovering MVD's

This section needs community help.

Exercise 7.1

Exercise 7.2

Exercise 7.3

Exercise 7.4

Chapter 4 High-Level Database Models

4.1 The Entity/Relationship Model

This section needs community help.

Exercise 1.1

Exercise 1.2

Exercise 1.3

Exercise 1.4

Exercise 1.5

Exercise 1.6

Exercise 1.7

Exercise 1.8

Exercise 1.9

Exercise 1.10

4.2 Design Principles

This section needs community help.

Exercise 2.1

Exercise 2.2

Exercise 2.3

Exercise 2.4

Exercise 2.5

Exercise 2.6

Exercise 2.7

4.3 Constraints in the E/R Model

This section needs community help.

Exercise 3.1

Exercise 3.2

Exercise 3.3

4.4 Weak Entity Sets

This section needs community help.

Exercise 4.1

Exercise 4.2

Exercise 4.3

Exercise 4.4

This section needs community help.

4.5 From E/R Diagrams to Relational Designs

Exercise 5.1

Exercise 5.2

Exercise 5.3

Exercise 5.4

4.6 Converting Subclass Structures to Relations

This section needs community help.

Exercise 6.1

Exercise 6.2

Exercise 6.3

Exercise 6.4

4.7 Unified Modeling Language

This section needs community help.

Exercise 7.1

Exercise 7.2

Exercise 7.3

Exercise 7.4

Exercise 7.5

Exercise 7.6

Exercise 7.7

Exercise 7.8

Exercise 7.9

Exercise 7.10

4.8 From UML Diagrams to Relations

This section needs community help.

Exercise 8.1

Exercise 8.2

Exercise 8.3

4.9 Object Definition Language

This section needs community help.

Exercise 9.1

Exercise 9.2

Exercise 9.3

Exercise 9.4

Exercise 9.5

Exercise 9.6

Exercise 9.7

Exercise 9.8

4.10 From ODL Designs to Relational Designs

This section needs community help.

Exercise 10.1

Exercise 10.2

Exercise 10.3

Exercise 10.4

Chapter 5 Algebraic and Logical Query Languages

5.1 Relational Operations on Bags

Exercise 1.1

For simplicity, we assume that R has r tuples, S has t tuples and T has t tuples.

a)

For a bag:

$$\text{Tuples}((R \cup S) \cup T) = (r + s) + t = r + s + t$$
$$\text{Tuples}(R \cup (S \cup T)) = r + (s + t) = r + s + t$$

For a set:

Each set can have a tuple only once. If each set has a common tuple then the result will have the single occurrence.

b)

$$\text{Tuples}((R \cap S) \cap T) = \min(\min(r, s), t)$$
$$\text{Tuples}(R \cap (S \cap T)) = \min(r, \min(s, t))$$

Bags are essentially sets that allow the appearance of a tuple more than once. In this case the operations on a bag and set yield the same results.

c)

We let $R = \{a, b\}, S = \{b, c\}, T = \{c, d\}$.

$$(R \bowtie S) \bowtie T = (\{a, b\} \bowtie \{b, c\}) \bowtie \{c, d\} = \{a, b, c, d\}$$
$$R \bowtie (S \bowtie T) = \{a, b\} \bowtie (\{b, c\} \bowtie \{c, d\}) = \{a, b, c, d\}$$

d)

For a bag:

Suppose a tuple t occurs n and m times in R and S respectively. The union of these two bags in the bag $R \cup S$, tuple t would appear $n + m$ times. The union of these two bags in the bag $S \cup R$ tuple t would appear $m + n$ times. Both sides of the relation yield the same result.

For a set:

In a set a tuple can only appear at most one time. Tuple t might appear in set R and S 1 or 0 times. The combinations of number of occurrences for tuple t in R and S respectively are $(0, 0), (0, 1), (1, 0)$, and $(1, 1)$. The union of either one of these combinations on the right or left side of the relation would yield the same result.

e)

For a bag:

Suppose a tuple t occurs n and m times in R and S respectively. The intersection of these two bags in the bag $R \cap S$, tuple t would appear $\min(n, m)$ times. The intersection of these two bags in the bag $S \cap R$ tuple t would appear $\min(n, m)$ times. Both sides of the relation yield the same result.

For a set:

In a set a tuple can only appear at most one time. Tuple t might appear in set R and S 1 or 0 times. The combinations of number of occurrences for tuple t in R and S respectively are $(0, 0), (0, 1), (1, 0)$, and $(1, 1)$. The intersection of either one of these combinations on the right or left side of the relation would yield the same result.

f)

We let $R = \{a, b\}, S = \{b, c\}$.

$$R \bowtie S = \{a, b\} \bowtie \{b, c\} = \{a, b, c\}$$
$$S \bowtie R = \{b, c\} \bowtie \{a, b\} = \{a, b, c\}$$

g)

On left side we have projection and on the right we have two projections. Union of the right side is the same result as the left side. They have same condition (L) to project.

h)

For a bag:

Let $R = \{x\}, S = \{x\}, T = \{x\}$

$$R \cup (S \cap T) = \{x\} \cup (\{x\} \cap \{x\}) = \{x, x\}$$
$$(R \cup S) \cap (R \cup T) = (\{x\} \cup \{x\}) \cap (\{x\} \cup \{x\}) = \{x, x\}$$

For a set:

From set theory we can know that it holds for set.

i)

Does **not** hold for sets and bags. Left side when the query executes can not be the same with the right side. On right side we have two expressions and those two can have some common tuples, nothing else. Let's take into account that conditions C and D are not the same.

Exercise 1.2

a)

Let $R = \{x\}, S = \{x, x\}, T = \{x\}$

$$(R \cap S) - T) = (\{x\} \cap \{x, x\}) - \{x\} = \emptyset$$
$$R \cap (S - T) = \{x\} \cap (\{x, x\} - \{x\}) = \{x\}$$

b)

Let $R = \{x\}, S = \{x, x\}, T = \{x\}$

$$R \cap (S \cup T) = \{x\} \cap (\{x, x\} \cup \{x\}) = \{x\}$$
$$(R \cap S) \cup (R \cap T) = (\{x\} \cup \{x, x\}) \cap (\{x\} \cup \{x\}) = \{x, x\}$$

c)

It holds for sets because **selection** on the left side with condition OR will give the same result with the **union** and situation on the right side.

Bags are unordered collection of elements **with elements**. Because of that, bags behave differently. And this example does **not** hold for bags.

5.2 Extended Operators of Relational Algebra

Exercise 2.1

a)

$$\pi_{A+B,A^2,B^2}(R) = \{(1,0,1),(5,4,9),(1,0,1),(6,4,16),(7,9,16)\}$$

b)

$$\pi_{B+1,C-1}(S) = \{(1,0),(3,3),(3,4),(4,3),(1,1),(4,3)\}$$

c)

$$\tau_{B,A}(R) = \{(0,1),(0,1),(2,3),(2,4),(3,4)\}$$

d)

$$\tau_{B,C}(S) = \{(0,1),(0,2),(2,4),(2,5),(3,4),(3,4)\}$$

e)

$$\delta(R) = \{(0,1),(2,3),(2,4),(3,4)\}$$

f)

$$\delta(S) = \{(0,1),(2,4),(2,5),(3,4),(0,2)\}$$

g)

$$\gamma_{A,\mathbf{SUM}(B)}(R) = \{(0,2),(2,7),(3,4)\}$$

h)

$$\gamma_{B,\mathbf{AVG}(C)}(R) = \{(0,1.5),(2,4.5),(3,4)\}$$

i)

$$\gamma_A(R) = \{(0),(2),(3)\}$$

j)

$$R \bowtie S = \{(2,3,4)\}$$
$$\gamma_{A,\mathbf{MAX}(C)}(R \bowtie S) = \{(2,4)\}$$

k)

$$R \overset{\circ}{\bowtie}_L S = \{(0,1,\perp),(0,1,\perp),(2,3,4),(2,4,\perp),(3,4,\perp)\}$$

l)

$$R \overset{\circ}{\bowtie}_R S = \{(\bot, 0, 1), (\bot, 2, 4), (\bot, 2, 5), (2, 3, 4), (\bot, 0, 2), (\bot, 3, 4)\}$$

m)

Combine above k) and l)

n)

$$\begin{aligned} R \overset{\circ}{\bowtie}_{R.B<S.B} S = \{&(0, 1, 2, 4), (0, 1, 2, 5), (0, 1, 3, 4), (0, 1, 3, 4), \\ &(0, 1, 2, 4), (0, 1, 2, 5), (0, 1, 3, 4), (0, 1, 3, 4), \\ &(2, 3, \bot, \bot), (2, 4, \bot, \bot), (3, 4, \bot, \bot), \\ &(\bot, \bot, 0, 1), (\bot, \bot, 0, 2)\} \end{aligned}$$

Exercise 2.2

a)

We use R to represent the relation. For $\delta(R)$, there is no duplicates. So for $\delta(\delta(R))$, the effect is none. So $\delta(R) = \delta(\delta(R))$. And it is *idempotent*

b)

It is *idempotent*, because when we repeat the projection it will yield the same relation.

c)

The result of $\sigma_C(R)$ is a relation where meets C. Repeating the selection will yield the same results because the relation already satisfy C. So it is *idempotent*.

d)

The result is a relation whose schema consists of the grouping attributes and the aggregated attributes. It is *not idempotent*.

e)

The result is sorted list of tuples based on some attributes L. It is *not idempotent*.

Exercise 2.3

We could use classical relational algebra by the following operations.

$$\begin{aligned} R1 &:= \rho_{A1,B1}(R) \\ R2 &:= R1 \bowtie_{A=A1} R \\ R3 &:= \pi_{A,A1}(R2) \end{aligned}$$

5.3 A Logic for Relations

Exercise 3.1

Example of the Datalog rule which is finite with the head relation: Datalog rule we have: $R(x, y)$ **AND** $z = z$. The expression here is $z = z$. The actual datalog rule are the head predicate has finite relation if the predicates of relational sub goals have finite relation. The head relation for the above example is finite and hence it violates the condition. This violation is irrespective of the finite relations that are assigned to the relational predicates.

5.4 Relational Algebra and Datalog

Exercise 4.1

a)

$$U(a, b, c) \leftarrow R(a, b, c)$$
$$U(a, b, c) \leftarrow S(a, b, c)$$

b)

$$I(a, b, c) \leftarrow R(a, b, c) \textbf{ AND } S(a, b, c)$$

c)

$$D(a, b, c) \leftarrow R(a, b, c) \textbf{ AND NOT } S(a, b, c)$$

d)

$$X(a, b, c) \leftarrow R(a, b, c) \textbf{ AND NOT } T(a, b, c)$$
$$X(a, b, c) \leftarrow S(a, b, c) \textbf{ AND NOT } T(a, b, c)$$

e)

$$X(a, b, c) \leftarrow R(a, b, c) \textbf{ AND NOT } S(a, b, c)$$
$$Y(a, b, c) \leftarrow R(a, b, c) \textbf{ AND NOT } T(a, b, c)$$
$$D(a, b, c) \leftarrow X(a, b, c) \textbf{ AND } Y(a, b, c)$$

f)

$$P(a, b) \leftarrow R(a, b, c)$$

g)

$$X(a,b) \leftarrow R(a,b,c)$$
$$Y(b,c) \leftarrow S(a,b,c)$$
$$Y(a,b) \leftarrow Y(b,c)$$
$$Z(a,b) \leftarrow X(a,b) \textbf{ AND } Y(a,b)$$

Exercise 4.2

a)

$$A(x,y,z) \leftarrow R(x,y,z) \textbf{ AND } x = y$$

b)

$$A(x,y,z) \leftarrow R(x,y,z) \textbf{ AND } x < y \textbf{ AND } y < z$$

c)

$$A(x,y,z) \leftarrow R(x,y,z) \textbf{ AND}(\ x < y \textbf{ OR } y < z)$$

d)

$$A(x,y,z) \leftarrow R(x,y,z) \textbf{ AND } x! = y$$

e)

$$A(x,y,z) \leftarrow R(x,y,z) \textbf{ AND } x > y \textbf{ OR } y > z$$
$$A(x,y,z) \leftarrow R(x,y,z) \textbf{ AND } x < y \textbf{ OR } y > z$$

f)

$$A(x,y,z) \leftarrow R(x,y,z) \textbf{ AND } x > y \textbf{ OR } y > z$$
$$A(x,y,z) \leftarrow R(x,y,z) \textbf{ AND } x > z \textbf{ OR } y > z$$

Exercise 4.3

a)

$$J(a,b,c,d) \leftarrow R(a,b,c) \textbf{ AND } S(b,c,d)$$

b)

$$J(b, c, d, e) \leftarrow S(b, c, d) \textbf{ AND } T(d, e)$$

c)

$$J(a, b, c, d, e) \leftarrow R(a, b, c) \textbf{ AND } T(d, e) \textbf{ AND } S(b, c, d) \textbf{ AND } T(d, e)$$

Exercise 4.4

Well, the process is the same as Exercise 4.2 above. Just add an extra **AND** operation. Omit here.

Exercise 4.5

a)

$$\pi_{x,y}(Q \bowtie R)$$

b)

$$\pi_{x,y}(Q)$$

c)

$$\pi_{x,y}(Q \bowtie_{x<y} R)$$

Chapter 6 The Database Language SQL

6.1 Simple Queries in SQL

Exercise 1.1

1. If A and B are two different attributes, there must be comma between them.
2. Because of that, B is an alias for A, because B is the second name, if it was that the second name was A, in that case A would be an alias for B. Of course, we count that between them there is no punctuation.

Exercise 1.2

a)

```
SELECT address
FROM Studio
WHERE name = 'MGM studios';
```

b)

```
SELECT birthdate
FROM MovieStar
WHERE name = 'Sandra Bullock';
```

c)

If you interpret the question as asking only that `Lover` appear as a substring, the the following is OK:

```
SELECT starName
FROM StarsIn
WHERE movieYear = 1980 OR movieTitle LIKE '%Love%';
```

However, another reasonable interpretation is that we want the word `Love` as a word by itself. The query above returns stars of a Movie *The Cook, the Thief, His Wife, and Her Lover*. To identify only titles that have `Lover` as a word by itself, either at the beginning, the middle, or the end as the entire title, we need to use four patterns. The following query works;

```
SELECT starName
FROM StarsIn
WHERE movieYear = 1980 OR
      movieTitle LIKE 'Love %' OR
      movieTitle LIKE '% Love %' OR
      movieTitle LIKE '% Love' OR
      movieTitle = 'Love';
```

d)

```
SELECT cert#
FROM MovieExec
WHERE netWorth >= 10000000
```

e)

This example is the same as c).

```
SELECT name
FROM MovieStar
WHERE gender = 'M' OR
      address LIKE 'Malibu %' OR
      address LIKE '% Malibu %' OR
      address LIKE '% Malibu' OR
      address = 'Malibu';
```

Exercise 1.3

a) Any tuple with a equal to 10 or b equal to 20.

b) Any tuple with a equal to 10 and b equal to 20.

c) Any tuple except a is `NULL`.

d) Any tuple with a equal to b except NULL.

e) Any tuple with $a \leq b$ except NULL.

Exercise 1.4

It's simple.

```
SELECT *
FROM Movies
WHERE length > 0;
```

6.2 Query Involving More Than One Relation

Exercise 2.1

a)

```
SELECT name
FROM StarsIn, MovieStar
WHERE gender = 'M' AND name = starName
      AND movieTitle = 'Titanic';
```

b)

```
SELECT starName
FROM StarsIn, Movies
WHERE gender = 'M' AND name = starName
      AND year = 1995
      AND studioName = 'MGM';
```

c)

```
SELECT presC#
FROM Studio, MovieExec
WHERE Studio.name = 'MGM' AND presC# = cert#
```

d)

```
SELECT Movie1.title
FROM Movies Movie1, Movies Movie2
WHERE Movie2.title = 'Gone With the Wind'
      AND Movie1.length > Movie2.length
```

e)

```
SELECT Exec1.name
FROM MovieExec Exec1, MovieExec Exec2
WHERE Exec2.name = 'Merv Griffin'
      AND Exec1.netWorth > Exec2.netWorth
```

Exercise 2.2

A systematic way to handle this problem is to create a tuple variable for every $R_i, i = 1, 2, \ldots, n$, whether we need to or not. That is, the FROM clause is

```
FROM R1 AS T1, R2 AS T2, ...
```

Now, build the WHERE clause from C by replacing every reference to some attribute A of R_i by $T_i.A$. Also, build the SELECT clause from list of attributes L by replacing every attribute A of R_i by $T_i.A$.

Exercise 2.3

```
SELECT L
FROM R1 NATURAL JOIN R2, ...
ON R1.column = R2.column AND ...
WHERE C;
```

6.3 Subqueries

Exercise 3.1

```
SELECT title
FROM Movies Old
WHERE Movies.year < Old.year AND Movies.title = Old.title;
```

Exercise 3.2

```
SELECT *
      (SELECT R1 FROM L) AS a
      (SELECT R2 FROM L) AS b
      ...
FROM L;
```

Exercise 3.3

a)

```
SELECT name, address
FROM MovieStar
WHERE gender = 'F' AND (name address) IN
  (SELECT name, address
   FROM MovieExec
   WHERE netWorth > 1 100000000);
```

b)

```sql
SELECT name, address
FROM MovieStar
WHERE (name address) NOT IN
  (SELECT name, address
   FROM MovieExec);
```

Exercise 3.4

We could user `EXCEPT` to replace `EXISTS`.

Exercise 3.5

a)

- Studio.name
- Studio.address
- presC#
- MovieExec.name
- MovieExec.address
- cert#
- netWorth

b)

- movieTitle
- movieYear
- starName
- name
- ADDRESS
- gender
- birthdate

c)

- movieTitle
- movieYear
- starName

- name
- ADDRESS
- gender
- birthdate

Exercise 3.6

```
SELECT *
FROM (PC FULL NATURAL OUTER JOIN Laptop
      FULL NATURAL JOIN Printer) AS allProduct
      LEFT OUTER JOIN Product ON Product.model = allProduct.model
```

Exercise 3.7

a)

```
SELECT *
FROM R.key = S.key
```

b)

```
SELECT *
FROM R, S
```

c)

```
SELECT *
FROM R, S
where C
```

6.4 Full-Relation Operations

Exercise 4.1

Need help

Exercise 4.2

Need help

Exercise 4.3

```
SELECT starName, MIN(year) AS minYear, COUNT(title) AS ctTitle
FROM starsIn
WHERE ctTitle > 3 GROUP BY starName;
```

Exercise 4.4

It is possible! In the γ, we need to produce all the aggregations that the `HAVING` clause uses. Then, we can follow the γ by a σ that eliminated from the result of the γ the tuples that correspond to the groups that the `HAVING` would eliminate. Finally, we use π to get rid of the extra aggregations that were used only by the `HAVING` clause.

6.5 Database Modifications

This are no questions in this section

6.6 Transactions in SQL

Exercise 6.1

a)

```
SET TRANSACTION READ ONLY
  ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
SELECT model, price
FROM PC
WHERE speed = providedSpeed, ram = providedRAM
COMMIT
```

b)

```
SET TRANSACTION READ WRITE
  ISOLATION LEVEL SERIALIZABLE
BEGIN TRANSACTION
DELETE FROM PC
WHERE PC.model = providedModel
DELETE FROM Product
WHERE Product.model = providedModel
COMMIT
```

c)

```
SET TRANSACTION READ WRITE
  ISOLATION LEVEL SERIALIZABLE
BEGIN TRANSACTION
UPDATE PC
SET price = price - 100
WHERE PC.model = providedModel
COMMIT
```

d)

```
SET TRANSACTION READ WRITE
  ISOLATION LEVEL SERIALIZABLE
```

```
BEGIN TRANSACTION
UPDATE PC AS P
SET (maker, model, speed, ram, hd, price)
WHERE IF(maker=P.maker AND model=P.model AND speed=P.speed AND
         ram=P.ram AND hd=P.hd AND price=P.price)
      PRINT 'Error. There is model like that.'
      COMMIT
COMMIT
```

Exercise 6.2

Here, I only give the answer for the first problem, it is easy to give an example for other situations. When looking up the PC, we may UPDATE it at the same time.

Exercise 6.3

Here, I only give the answer for the first problem, it is easy to give an example for other situations. When the ISOLATION LEVEL is READ UNCOMMITTED, we may look up the dirty data.

Exercise 6.4

- *Serializable*: With a lock-based concurrency control serializability requires read and write locks to be released at the end of the transaction.When using non-lock based concurrency control, no locks are acquired; however, if the system detects a write collision among several concurrent transactions, only one of them is allowed to commit. Here the Transaction T,has to acquire locks before reading and writing. So, there will be no impact of other processes running.
- *Repeatable reads*: With a lock-based concurrency control this keeps read and write locks until the end of the transaction. Write skew is possible at this isolation level, a situation where two writes are allowed to the same column in a row by two different writers, resulting in the row having data that is a mix of the two transactions. Here the Transaction T, has to acquire locks before reading and writing. So, there will be no impact of other processes running.
- *Read committed*: A lock-based concurrency control this keeps write locks until the end of the transaction, but read locks are released as soon as the SELECT operation is performed. Read committed is an isolation level that guarantees that any data read is committed at the moment it is read. It simply restricts the reader from seeing any intermediate, uncommitted, 'dirty' read. Here the data read by Transaction T will always be committed so there should be no issue with other process.
- *Read uncommitted*: This is the lowest isolation level. In this level, dirty reads are allowed, so one transaction may see not-yet-committed changes made by other transactions. Here it could be a situation where data updated by other transactions are not available and the Transaction T keeps running for ever

Chapter 7 Constraints and Triggers

7.1 Keys and Foreign Keys

Exercise 1.1

a)

```
  FOREIGN KEY (producerC#) REFERENCES MovieExec(cert#);
```

b)

```
  FOREIGN KEY (producerC#) REFERENCES MovieExec(cert#)
  ON UPDATE SET NULL  -- if MOvie exec cert# is changed, set to null
                      -- in practice this should not be used
                      -- since it will lose referential integrity
                      -- ideally, it should be CASCADE
  ON DELETE SET NULL; -- if Movie exec does not exist
```

c)

```
  FOREIGN KEY (producerC#) REFERENCES MovieExec(cert#)
    ON UPDATE CASCADE
    ON DELETE CASCADE;
```

d)

```
  FOREIGN KEY (movieTitle, movieYear)
    REFERENCES Movies(title, year);
-- by default, UPDATE and CASCADE are restrict
```

e)

```
  FOREIGN KEY (starName) REFERENCES MovieStar(name)
    ON DELETE CASCADE;
```

Exercise 1.2

No.

Assume the foreign key constraint (a) in relation R that references relation S.

Thus:

- R must contain the attribute(s) a
- a is a candidate key of S
- if a tuple in R contains a, a exists in S.

Therefore, we can't enforce that every Movie has a tuple in StarsIn because:

- Movies does not contain the candidate key attributes of StarsIn

Exercise 1.3

- Product: model
- PC: model
- Laptop: model
- Printer: model

In PC, Laptop and Printer, we need a foreign key constraint:

```
CREATE TABLE Product(
  ...
  PRIMARY KEY (model)
);
CREATE TABLE PC(
  ...
  PRIMARY KEY (model),
  FOREIGN KEY (model) REFERENCES Product(model)
);
CREATE TABLE Laptop(
  ...
  PRIMARY KEY (model),
  FOREIGN KEY (model) REFERENCES Product(model)
);
CREATE TABLE Printer(
  ...
  PRIMARY KEY (model),
  FOREIGN KEY (model) REFERENCES Product(model)
);
```

7.2 Constraints on Attributes and Tuples

Exercise 2.1

The answers below are attribute constraints, defined after the corresponding `<type>` of the attribute.

a)

```
  year <type> CHECK (year >= 1915)
```

b)

```
  length <type> CHECK (length >= 60 AND length <= 250)
```

c)

```
  studioName <type> CHECK
    (studioName in('Disney', 'Fox', 'MGM', 'Paramount'))
```

Exercise 2.2

The answers below are attribute constraints, defined after the corresponding `<type>` of the attribute.

a) In relation `Laptop`:

```
speed <type> CHECK (speed >= 2.0 )
```

b) In relation `Printer`:

```
type <type> CHECK (type IN ('laser','ink-jet', 'bubble-jet'))
```

c) In relation `Product`:

```
type <type> CHECK (type IN ('PC','laptop', 'printer'))
```

d) In relation `Product`. This checks that it exists in at least one of them, not that it exists in exactly one of them. This is one potential solution:

```
model <type> CHECK (EXISTS (select model from Printers P where
                            P.model = model)) or
                    EXISTS (select model from Laptops L where
                            L.model = model)) or
                    EXISTS (select model from Printers P where
                            P.model = model)))
```

Exercise 2.3

The answers below are tuple-based constraints:

a) Assume the function $year(date)$ extracts the year of a given $date$.

As a tuple constraint in `MovieStar`:

```
CHECK  (year(birthdate) < ALL (SELECT movieyear
                               FROM StarsIn
                               WHERE starName = name))
```

and as a tuple constraint in `StarsIn`:

```
CHECK  (movieyear > (SELECT year(birthdate)
                     FROM StarsIn
                     WHERE starName = name))
```

b) as a tuple constraint, in relation `Studios`:

```
UNIQUE(address)
```

c) as a tuple constraints:

For `MovieStar`:

```
CHECK (name NOT IN (SELECT name FROM MovieExec))
```

For MovieExec:

```
CHECK (name NOT IN (SELECT name FROM MovieStar))
```

d) as tuple constraint in Studio:

```
CHECK (EXISTS (SELECT * FROM Movies where studioName = name))
```

e) as a tuple constraint, in Movies

```
CHECK(
  -- it not a president
   (producerC# NOT IN (SELECT presC# FROM Studio))
   OR --otherwise the studio must the same than the president
   (studioName IN (SELECT name from Studio where presC# = producerC
  #))
   )
```

Exercise 2.4

a) as a tuple constraint in PC:

```
CHECK(speed >= 2.0  -- either 2.0 or more
      OR
      price <= 600)  -- or at most 600
```

b) as a tuple constraint in Laptop:

```
CHECK(
  (screen >= 15) -- screen is 15 inches or more
  OR
  (hd >= 40 OR price < 1000) -- or its HD is at least 40
                             -- or price < 1000
);
```

Exercise 2.5

a)

```
CHECK(bore > 16)
```

b)

```
CHECK(numGuns < 9 -- at most 9 guns
      OR
      bore < 14 ) -- or its bore must be no longer than 14
```

c)

In Ships:

```
  CHECK(launched < ALL (SELECT date
                        FROM Battles B JOIN Outcomes
                        ON (battle = B.name
                            and ship = name))
```

Exercise 2.5

For both situations, the `gender` cannot be `NULL`

7.3 Modifications of Constraints

Exercise 3.1

a)

```
  ALTER TABLE Movie ADD CONSTRAINT MovieKey
    PRIMARY KEY(title, year);
```

b)

```
  ALTER TABLE MovieExec ADD CONSTRAINT P1
    CHECK(name IN (SELECT producerC# FROM Movie));
```

c)

```
  ALTER TABLE Movie ADD CONSTRAINT P2
    CHECK(length >= 60 AND length <= 250);
```

d)

```
  ALTER TABLE MovieStar ADD CONSTRAINT P3
    CHECK(name NOT IN (SELECT name FROM MovieExec));
  ALTER TABLE MovieExec ADD CONSTRAINT P4
    CHECK(name NOT IN (SELECT name FROM MovieStar));
```

e)

```
  ALTER TABLE Studio ADD CONSTRAINT P5
    UNIQUE(address);
```

Exercise 3.2

a)

```
  ALTER TABLE Classes ADD CONSTRAINT P1
    PRIMARY KEY(class, country);
```

b)

```
ALTER TABLE Battles ADD CONSTRAINT P2
  CHECK(name IN (SELECT battle FROM Outcomes));
```

c)

```
ALTER TABLE Ships ADD CONSTRAINT P3
  CHECK(name in (SELECT ship FROM Outcomes));
```

d)

```
ALTER TABLE Classes ADD CONSTRAINT P4
  CHECK(numGuns <= 14);
```

e)

```
ALTER TABLE Battle ADD CONSTRAINT P5
  CHECK(date NOT IN (SELECT launched FROM Ships));
```

7.4 Assertions

Exercise 4.1

a)

```
CREATE ASSERTION no_maker CHECK(
  NOT IN ((SELECT maker FROM Product NATURAL JOIN PC)
          INTERSECT
          (SELECT maker FROM Product NATURAL JOIN Laptop))
);
```

b)

```
CREATE ASSERTION speed_better CHECK(
  NOT EXISTS (PC JOIN Laptop ON PC.model = Laptop.model AND
                            Laptop.speed < PC.speed )
);
```

c)

```
CREATE ASSERTION more_memory_more_price CHECK(
  NOT EXISTS (PC JOIN Laptop ON PC.model = Laptop.model AND
                                Laptop.ram > PC.ram AND
                                Laptop.price <= PC.price )
);
```

d)

```
CREATE ASSERTION check_integrity CHECK(
  NOT EXISTS((SELECT model FROM PC)
              INTERSECT
             (SELECT model FROM Laptop)
```

```
                UNION
                 ((SELECT model FROM Laptop)
                   INTERSECT
                  (SELECT model FROM Printer)
                 )
  );
```

Exercise 4.2

a)

```
  CREATE ASSERTION class_no_more_than_2_sheep CHECK(
    NOT EXISTS (2 >= ALL (SELECT COUNT(*)
                 FROM Ships
                 GROUP BY class)
  )
```

b)

```
  CREATE ASSERTION noCountry CHECK(NOT EXISTS (
    (SELECT type FROM Classes WHERE type=bb)
    INTERSECT
    (SELECT type FROM Classes WHERE type=bc))
  );
```

c)

```
  CREATE ASSERTION sunkCheck CHECK(
    SELECT *
    FROM ((Classes NATURAL JOIN Ships) NATURAL JOIN
          Battles WHERE numGuns > 9)
         INTERSECT
         ((SELECT * FROM Classes NATURAL JOIN
           Ships WHERE numGuns < 9) NATURAL JOIN Battles WHERE result
    ='sunk')
  );
```

d)

```
  CREATE ASSERTION launched CHECK(
    NOT IN(SELECT class
           FROM Ships
           WHERE class='Tennessee')
           INTERSECT
           (1921 > ALL (SELECT launched FROM class))
  );
```

e)

```
CREATE ASSERTION nameShip CHECK(
  SELECT name, class
  FROM Ships S, Classes C
  WHERE S.name=C.class
);
```

Exercise 4.3

I think the Exercise 11 should be Example 11, which is

```
CREATE ASSERTION RichPres CHECK (NOT EXISTS
  (SELECT Studio.name
  FROM Studio, MovieExec
  WHERE presC# = cert# AND netWorth < 10000000
  )
);
```

It is easy to convert the assertion to tuple-based constraints:

```
CHECK (NOT EXISTS (
  SELECT Studio.name
  FROM Studio, MovieExec
  WHERE presC# = cert# AND netWorth < 10000000
  )
);
```

7.5 Triggers

Exercise 5.1

```
CREATE TRIGGER AvgNetWorthTrigger
AFTER UPDATE OF netWorth ON MovieExec
REFERENCING
  OLD TABLE AS OldStuff,
  NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN (500000 > (SELECT AVG(netWorth) FROM MovieExec))
  DELETE FROM MovieExec
  WHERE (name, address, cert# , netWorth) IN NewStuff;
  INSERT INTO MovieExec
  (SELECT * FROM Old Stuff);
```

Exercise 5.2

a)

```
CREATE TRIGGER noLowerPrice
AFTER UPDATE OF price ON PC
REFERENCING
  OLD AS OldTuple
  NEW AS NewTuple
FOR EACH ROW
WHEN (OldTuple.price << NewTuple.price)
  UPDATE PC
  SET price = OldTuple.price
  WHERE speed = OldTuple.speed;
```

b)

```
CREATE TRIGGER NewPrinterTrigger
AFTER INSERT ON Printer
REFERENCING
  NEW ROW AS NewRow,
  NEW TABLE AS NewStuff
FOR EACH ROW
WHEN (NOT EXISTS (SELECT * FROM Product
  WHERE Product.model = NewRow.model))
  DELETE FROM Printer
  WHERE (model, color, type, price) IN NewStuff;
```

c)

```
CREATE TRIGGER averagePrice
INSTEAD OF
UPDATE OF price OF Laptop
INSERT ON Laptop
DELETE ON Laptop
REFERENCING
  OLD_TABLE AS OldStuff
  NEW_TABLE AS NewStuff
WHEN(1500 <= ALL (
      SELECT AVG(price)
      FROM (Laptop EXCEPT OldStuff) UNION NewStuff, Product
      WHERE Laptop.model = Product.model
      GROUP BY maker ))
  DELETE FROM Laptop
  WHERE (model, speed, ram, hd, screen, price) IN OldStuff
  INSERT INTO Laptop
  (SELECT * FROM NewStuff);
```

d)

```
BEFORE UPDATE ON PC  FOR EACH ROW
BEGIN
  IF NEW.hd $>=$ 100*(NEW.ram)
  THEN
```

```
      SIGNAL SQLSTATE '45000'
      SET MESSAGE\_TEXT = 'Cannot add or update row: HardDisk
                          should be at least 100 times more than RAM
  .';
      END IF;
  END;
```

e)

```
  BEFORE INSERT ON PRODUCT
  BEGIN
    IF NEW.model IN (SELECT DISTINCT model FROM PC
                     UNION ALL
                     SELECT DISTINCT model FROM Printer
                     UNION ALL
                    SELECT DISTINCT model FROM Laptop)
    THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE\_TEXT = 'Cannot add or update row: Model number
   already assigned.
                          Please try with another model number';
    END IF;
  END;
```

Exercise 5.3

Need help

Exercise 5.4

Need help

Chapter 8 Views and Indexes

8.1 Virtual Views

Exercise 1.1

a)

```
  CREATE VIEW RichExec AS
    SELECT name, address, cert#, netWorth
    FROM MovieExec
    WHERE netWorth >= 10000000;
```

b)

```
CREATE VIEW StudioPres AS
  SELECT name, address, cert#
  FROM MovieExec, Studio
  WHERE cert# = presC#
```

c)

```
CREATE VIEW
ExecutiveStar(name, address, gender, birthdate, cert#, netWorth) AS
  SELECT star.name, star.address, star.gender, start.birthdate,
         exec.cert#, exec.netWorth
  FROM MovieStar star, MovieExec exec
  WHERE star.name = exec.name AND star.address = exec.address
```

Exercise 1.2

a)

```
SELECT name
FROM ExecutiveStar
WHERE gender = 'F'
```

b)

```
SELECT RichExec.name
FROM RichExec, StudioPres
WHERE RichExec.name = StudioPres.name
```

c)

```
SELECT ExecutiveStar.name
FROM ExecutiveStar, StudioPres
WHERE ExecutiveStar.netWorth >= 50000000 AND
      ExecutiveStar.name = StudioPres.name
```

8.2 Modifying Views

Exercise 2.1

a) is updatable because it has only one table in the FROM clause. Other two queries are not because they save two tables in FROM clause.

Exercise 2.2

a) Yes, this view is updatable.

b)

```
CREATE TRIGGER DisneyComedyInsert
INSTEAD OF INSERT ON DisneyComedies
```

```
  REFERENCING NEW ROW AS NewRow
  FOR EACH ROW
  INSERT INTO Movies(title, year, length, studioName, genre)
  VALUES(NewRow.title, NewRow.year, NewRow.length,
        'Disney', 'comedy');
```

c)

```
  CREATE TRIGGER DisneyComedyUpdate
  INSTEAD OF UPDATE ON DisneyComedies
  REFERENCING NEW ROW AS NewRow
  FOR EACH ROW
  UPDATE Movies SET length NewRow.length
  where title = NewRow.title AND year = NewRow.year
        AND studioName = 'Disney' AND genre = 'comedy';
```

Exercise 2.3

a) No, this view isn't updatable.

b)

```
  CREATE TRIGGER NewPCInsert
  INSTEAD OF INSERT ON NewPC
  REFERENCING NEW ROW AS NewRow
  FOR EACH ROW
  INSERT INTO Product VALUES(NewRow.maker, NewRow.model, 'pc')
  INSERT INTO PC VALUES(NewRow.model, NewRow.speed, NewRow.ram,
                        NewRow.hd, NewRow.price);
```

c)

```
  CREATE TRIGGER NewPCUpdatePrice
  INSTEAD OF UPDATE ON NewPC
  REFERENCING NEW ROW AS NewRow
  FOR EACH ROW
  UPDATE PC SET price NewRow.price
  WHERE model = NewRow.model
```

d)

```
  CREATE TRIGGER NewPCDelete
  INSTEAD OF DELETE ON NewPC
  REFERENCING OLD ROW AS OldRow
  FOR EACH ROW
  DELETE FROM Product WHERE model = OldRow.model
  DELETE FROM PC WHERE model = OldRow.model
```

8.3 Indexes in SQL

Exercise 3.1

a)

```
CREATE INDEX StudioNameIndex ON Studio(name);
```

b)

```
CREATE INDEX MovieExecAddress ON MovieExec(address);
```

c)

```
CREATE INDEX MovieInfoIndex on Movies(genre, length);
```

8.4 Calculating the Best Indexes to Create

Exercise 4.1

Action	No Index	Star Index	Movie Index	Both Indexes
Q1	100	4	100	4
Q2	100	100	4	4
Q3	2	4	4	6
Average	1	2	3	4

Exercise 4.2

Need community help.

8.5 Materialized Views

This whole section needs community help.

Exercise 5.1

Exercise 5.2

Exercise 5.3

Exercise 5.4

Chapter 9 SQL in a Server Environment

9.1 The Three-Tier Architecture

There are no questions in this section.

9.2 The SQL Environment

There are no questions in this section.

9.3 The SQL/Host-Language Interface

Exercise 3.1

a)

```
int main() {
  EXEC SQL BEGIN DECLARE SECTION;
  char maker[10];
  int model;
  int speed;
  int price;
  EXEC SQL END DECLARE SECTION;
  EXEC SQL WHENEVER SQLERROR GOTO query_error;
  EXEC SQL WHENEVER NOT FOUNT GOTO bad_number;
  printf("Enter the price for PC:"); scanf_s("%d", &price);
  EXEC SQL DECLARE c CURSOR FOR EXEC SQL SELECT Product.maker,
  Product.model,
  Product.speed FROM Product,
  PC WHERE Product.model = PC.model ORDER BY abs(Product.model - PC
 .model) EXEC SQL OPEN CURSOR c;
  while (1) {
    EXEC SQL FETCH c INTO: maker,
    :model,
    :speed;
    if (NOT FOUND) break;
  }
  EXCEL SQL CLOSE CURSOR c;
  query_error;
  printf("SQL error: %ld", sqlca->sqlcode);
  exit();
  bad_number;
  printf("Invalid order number");
  exit()
}
```

b)

```
int main() {
  EXEC SQL BEGIN DECLARE SECTION;
  char maker[10];
  int model;
  int speed;
  int ram;
  int hd;
```

```
    int screen;
    int price;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL WHENEVER SQLERROR GOTO query_error;
    EXEC SQL WHENEVER NOT FOUND GOTO bad_number;
    printf("Enter the minimum Speed, RAM, Hard disk, Screen size of
   the PC");
    scanf_s("%d %d %d %d, &price, &ram, &hd, &speed");
    EXEC SQL DECLARE c CURSOR FOR EXEC SQL SELECT Product.maker,
    Laptop.model,
    Laptop.speed,
    Laptop.ram,
    Laptop.hd,
    Laptop.price,
    Laptop.screen FROM Product,
    PC WHERE PC.model = Laptop.model ORDER BY abs(PC.price - Laptop.
   price);
    EXEC SQL OPEN CURSOR c;
    while (1) {
      EXEC SQL FETCH c INTO: maker,
      :model,
      :speed,
      :ram,
      :hd,
      :price,
      :screen;
      IF(NOT FOUND) break
    }
    EXEC SQL CLOSE CURSOR c;
    query_error;
    printf("SQL error:  % ld ", sqlca->sqlcode);
    exit();
    bad_number;
    printf("Invalid order number ");
    exit()
  }
```

c)

```
  int main() {
    EXEC SQL BEGIN DECLARE SECTION;
    char maker[10];
    int model;
    int speed;
    int ram;
    int hd;
    int screen;
    int price;
    EXEC SQL END DECLARE SECTION;
```

```
    EXEC SQL WHENEVER SQLERROR GOTO query_error;
    EXEC SQL WHENEVER NOT FOUND GOTO bad_number;
    printf("Enter the manufacturer of the PC");
    scanf_s("%s, &maker");
    EXEC SQL DECLARE c CURSOR FOR EXEC SQL SELECT Laptop.model,
    Product.type,
    Laptop.screen,
    Laptop.ram,
    Laptop.hd,
    Laptop.price,
    Laptop.screen FROM Product,
    PC,
    Laptop WHERE Product.model = Pc.model AND Laptop.model;
    EXEC SQL OPEN CURSOR c;
    while (1) {
      EXEC SQL FETCH c INTO: model,
      :type,
      :speed,
      :ram,
      :hd,
      :price,
      :screen;
      IF(NOT FOUND) break
    }
    EXEC SQL CLOSE CURSOR c;
    query_error;
    printf("SQL error:  % ld ", sqlca ->sqlcode);
    exit();
    bad_number;
    printf("Invalid order number ");
    exit()
  }
```

d)

```
  int main() {
    EXEC SQL BEGIN DECLARE SECTION;
    char maker[10];
    int model;
    int pmodel;
    int speed;
    int price;
    EXEC SQL END DECLARE SECTION EXEC SQL WHENEVER SQLERROR GOTO
   query_error;
    EXEC SQL WHENEVER NOT FOUND GOTO bad_number;
    printf("Enter the budget, minimum speed of the PC:");
    scanf_s("%d %d, &price, &speed");
    EXEC SQL DECLARE c CURSOR FOR EXEC SQL SELECT Laptop.model,
   Printer.model FROM Product, Laptop, Printer, PC WHERE PC.model
```

```
    <=<=price;
13   EXEC SQL OPEN CURSOR c;
14   while (1) {
15     EXEC SQL FETCH c INTO :model, :pmodel;
16     if(NOT FOUND) break;
17   }
18   EXCEL SQL CLOSE CURSOR c;
19   query_error;
20   printf("SQL error: %ld", sqlca->sqlcode);
21   exit();
22   bad_number;
23   printf("Invalid order number");
24   exit();
25 }
```

e)

```
1  int main() {
2    EXEC SQL BEGIN DECLARE SECTION;
3    char maker[10];
4    int model;
5    int speed;
6    int ram;
7    int hd;
8    int price;
9    EXEC SQL END DECLARE SECTION EXEC SQL WHENEVER SQLERROR GOTO
     query_error;
10   EXEC SQL WHENEVER NOT FOUND GOTO bad_number;
11   printf("Enter the manufacturer, speed, ram, hard disk, price of
     the PC: ");
12   scanf_s("%s %d %d %d %d, &maker, &model, &speed, &ram, &hd, &
     price");
13   EXEC SQL DECLARE c CURSOR FOR EXEC SQL SELECT Product.maker,
     Laptop.model, Laptop.speed, Laptop.ram, Laptop.hd, Laptop.price
     FROM Product, PC, Laptop WHERE PC.model=model;
14   EXEC SQL OPEN CURSOR c;
15   while (1) {
16     EXEC SQL FETCH c INSERT INTO Product (maker, model, type)
     VALUES (:maker, :model, :") INSERT INTO PC (model, speed, ram, hd,
      screen, price) VALUES (:model, :speed, :ram, :hd:", :price);
17     if(NOT FOUND) bad_order;
18   }
19   EXCEL SQL CLOSE CURSOR c;
20   query_error;
21   printf("SQL error: %ld", sqlca->sqlcode);
22   exit();
23   bad_number;
24   printf("Invalid order number");
```

```
    exit();
  }
```

Exercise 3.2

a)

```
int main() {
  EXEC SQL BEGIN DECLARE SECTION;
  int bore;
  int numGuns;
  EXEC SQL END DECLARE SECTION;
  EXEC SQL WHENEVER SQLERROR GOTO query_error;
  EXEC SQL WHENEVER NOT FOUND GOTO bad_number;
  printf("Enter the bore and number of Guns");
  scanf_s("%d%d, &bore, &numGuns");
  EXEC SQL DECLARE c CURSOR FOR EXEC SQL SELECT R1 AS SELECT (
 numGuns*bore^3) AS firepower FROM Classes, R2 AS SELECT min(
 firepower) minimum FROM R1, SELECT Classes FROM R1 WHERE firepower
  = (SELECT minimum FROM R2);
  EXEC SQL OPEN CURSOR c;
  while(1){
    EXEC SQL FETCH c INTO :bore, :numGuns;
    IF(NOT FOUND) break
  }
  EXEC SQL CLOSE CURSOS c;
  query_error;
  printf("SQL error: "%ld, sqlca->sqlcode");
  exit();
  bad_number;
  printf("Invalid order number");
  exit();
}
```

b)

```
int main() {
  EXEC SQL BEGIN DECLARE SECTION;
  string name;
  string class;
  string country;
  string outcomes;
  string battle;
  EXEC SQL END DECLARE SECTION;
  EXEC SQL WHENEVER SQLERROR GOTO query_error;
  EXEC SQL WHENEVER NOT FOUND GOTO bad_number;
  printf("Enter the name of battle");
  scanf_s("%s, &name");
```

```
13      EXEC SQL DECLARE c CURSOR FOR EXEC SQL SELECT R1 AS SELECT class,
         name AS ship, country FROM Classes NATURAL JOIN Ships, R2 AS
        SELECT battle, result, country FROM Outcomes NATURAL JOIN R1, R3
        AS (SELECT count(*) AS COUNT, country FROM R2 WHERE result=? AND
        battle=? GROUP BY country) SELECT country FROM R3 WHERE COUNT="",
        SELECT(max(COUNT) FROM R3);
14      EXEC SQL OPEN CURSOR c;
15      while (1) {
16        EXEC SQL FETCH c INTO name:, :class, :country, :outcomes, :
        battle;
17        IF(NOT FOUND) break
18      }
19      EXEC SQL CLOSE CURSOS c;
20      query_error;
21      printf("SQL error: "%ld, sqlca->sqlcode");
22      exit();
23      bad_number;
24      printf("Invalid order number");
25      exit();
26    }
```

c)

```
1   int main() {
2     EXEC SQL BEGIN DECLARE SECTION;
3     int numGuns;
4     int bore;
5     int displacement;
6     string class;
7     string country;
8     string type;
9     string name;
10    string class;
11    int launched;
12    EXEC SQL END DECLARE SECTION;
13    EXEC SQL WHENEVER SQLERROR GOTO query_error;
14    EXEC SQL WHENEVER NOT FOUND GOTO bad_number;
15    printf("Enter the name of class and information about tuple of
      Classes");
16    scanf_s("%s%s%s%d%d%d, &class, &country, &type, &numGuns, &bore,
      &displacement");
17    printf("Enter the name, class and launched of the ships");
18    scanf_s("%s%s%d, &name,&class, &launched");
19    EXEC SQL DECLARE c CURSOR FOR EXEC SQL SELECT INSERT INTO Classes
       VALUES(?,?,?,?,?,?), INSERT INTO Ships VALUES(?,?,?);
20    EXEC SQL OPEN CURSOR c;
21    while(1) {
22      EXEC SQL FETCH c INTO name:, :class, :country, :numGuns, :bore,
```

```
    :displacement, :type, :name, :class, :launched;
      IF(NOT FOUND) break
    }
    EXEC SQL CLOSE CURSOS c;
    query_error;
    printf("SQL error: "%ld, sqlca->sqlcode");
    exit();
    bad_number;
    printf("Invalid order number");
    exit();
  }
```

d)

```
int main() {
  EXEC SQL BEGIN DECLARE SECTION;
  int launched;
  string battle;
  string name;
  string date;
  EXEC SQL END DECLARE SECTION;
  EXEC SQL WHENEVER SQLERROR GOTO query_error;
  EXEC SQL WHENEVER NOT FOUND GOTO bad_number;
  printf("Enter the name of ships that were in battle before they
   were launched");
  scanf_s("%s%s%d%s, &name,&battle, &launched, &date");
  EXEC SQL DECLARE c CURSOR FOR EXEC SQL SELECT SELECT name.s,
  launched.s, date.b FROM Ships.s, Battles.b WHERE date<launced;
  EXEC SQL OPEN CURSOR c;
  while(1) {
    EXEC SQL FETCH c INTO name:, :date, :name, :battle, :launched;
    IF(NOT FOUND) break
  }
  EXEC SQL CLOSE CURSOS c;
  query_error;
  printf("SQL error: "%ld, sqlca->sqlcode");
  exit();
  bad_number;
  printf("Invalid order number");
  exit();
}
```

9.4 Stored Procedures

Exercise 4.1

a)

```
CREATE FUNCTION Movie (n CHAR(20), w INT) RETURN INT
  SELECT name, netWorth
  FROM Studio, MovieExec
  WHERE name=n AND netWorth=w;
```

b)

```
CREATE FUNCTION nameAddress(n CHAR(20), a CHAR(20)) RETURN BOOLEAN
  IF(SELECT name.S, name.E
     FROM MovieStar.S, MovieExec.E
     WHERE n=name.S AND NOT (n=name.E)) RETURN 1;
  ELSEIF 2 <= (SELECT name.S,name.E
               FROM MovieStar.S, MovieExec.E
               WHERE n=name.E AND NOT (n=name.S) )
  ELSEIF 3 <= (SELECT name.S, name.E
               FROM MovieStar.S, MovieExec.E
               WHERE n=name.E AND n=name.S)
  ELSEIF 4 <= (SELECT name.S, name.E
               FROM MovieStar.S, MovieExec.E
               WHERE NOT(n=name.E) AND NOT(n=name.S))
  END IF;
```

c)

```
CREATE FUNCTION sName (s CHAR(20)) RETURN CHAR
  IF(SELECT *
     FROM Movies
     WHERE studioName=s
     GROUP BY length
     HAVING COUNT(*)=2)
  THEN RETURN sName;
  ELSE IF (SELECT *
           FROM Movies
           WHERE studioName=s
           GROUP BY length
           HAVING COUNT(*)=1)
       THEN RETURN 'There is no second longest';
       END IF;
```

d)

```
CREATE FUNCTION star (s CHAR(20)) RETURN INT
  IF(SELECT MIN(year) AS lowestYear, length
     FROM Movies
     WHERE length>120)
  RETURN s;
  ELSE RETURN 0;
  END IF;
```

e)

```
CREATE FUNCTION address (a CHAR(50)) RETURN CHAR
  IF (SELECT name
      FROM MovieStar
      WHERE address=a
      GROUP BY name
      HAVING COUNT(address)=1)
  RETURN a;
  ELSEIF SELECT name
         FROM MovieStar
         WHERE address=a
         GROUP BY name
         HAVING COUNT(address)=0 OR HAVING COUNT(address)>1
         RETURN NULL;
         END IF;
```

f)

```
CREATE FUNCTION nameStar (n CHAR(50)) RETURN CHAR
  (DELETE name FROM MovieStar WHERE n=name);
  UNION (DELETE title, movieTitle
        FROM StarsIn, Movies
        WHERE n=starName AND n=title);
```

Exercise 4.2

a)

```
CREATE FUNCTION priceP (p INT) RETURN INT
  IF(SELECT MIN(price)
     FROM PC
     WHERE price=p); RETURN INT;
  END IF;
```

b)

```
CREATE FUNCTION productModel (m CHAR(20), n INT)
  IF(SELECT price
     FROM PC, Laptop, Printer
     WHERE m=maker AND n=model); RETURN price;
  END IF;
```

c)

```
CREATE FUNCTION takeALL (m INT, s INT, r INT, h INT, p INT ) RETURN
  CHAR
  INSERT INTO PC VALUES(m, s, r, h, p)
  IF(SELECT model, speed, ram, hd, price
     FROM PC
     WHERE model=m, speed=s, ram=r, hd=h, price=p); RETURN 'Error';
  ELSE model=model+1;
```

d)

```
CREATE FUNCTION morePrice (p INT) RETURN INT
  IF(SELECT price
     FROM Laptop, PC, Printer
     WHERE p=price
     HAVING COUNT(*)
     GROUP BY Laptop, PC, Printer;) RETURN INT;
  END IF;
```

Exercise 4.3

a)

```
CREATE FUNCTION firepower (c CHAR(50)) RETURN INT
  IF(SELECT class, numGuns, bore
     FROM Classes WHERE c=class); RETURN numGuns*bore^3;
```

b)

```
CREATE FUNCTION battleName (n CHAR(20)) RETURN
  IF(SELECT name, country
     FROM Battles
     WHERE n=name
     HAVING COUNT (*)=2
     GROUP BY country); RETURN n;
  ELSEIF(SELECT name, country
         FROM Battles
         WHERE n=name
         HAVING COUNT(*)>2 OR HAVING COUNT(*)<2
         GROUP BY country; RETURN NULL;
```

c)

```
CREATE FUNCTION takeALL (c CHAR(20), t CHAR(20), c CHAR(20), n INT,
  b INT, d INT)
  INSERT INTO Classes VALUES(c, t, c, n, b, d);
  INSERT INTO Ships WHERE c=class;
```

d)

```
CREATE FUNCTION shipBattle (n CHAR(20))
  IF(SELECT date.B, launched.S
     FROM Battles.B, Ships.s
     WHERE date<launched;) RETURN date=0, launched=0;
  ENDIF;
```

Exercise 4.4

$$\begin{aligned}\sum_{i=1}^{n}(x_i - \bar{x}^2/n) &= \frac{1}{n}\sum_{i=1}^{n}(x_i^2 - 2\bar{x}x_i + \bar{x}^2) \\ &= \frac{1}{n}\left(\sum x_i^2 - \sum 2\bar{x}x_i + \sum \bar{x}^2\right) \\ &= \frac{1}{n}\left(\sum x_i^2 - 2\bar{x}(n\bar{x}) + n\bar{x}^2\right) \\ &= \frac{1}{n}\left(\sum x_i^2 - n\bar{x}^2\right)\end{aligned}$$

9.5 Using a Call-Level Interface

This section needs community help.

Exercise 5.1

Exercise 5.2

9.6 JDBC

This section needs community help.

Exercise 6.1

Exercise 6.2

9.7 PHP

This section needs community help.

Exercise 7.1

Exercise 7.2

Exercise 7.3

Chapter 10 Advanced Topics in Relational Databases

10.1 Security and User Authorization in SQL

This section needs community help.

Exercise 1.1

Exercise 1.2

Exercise 1.3

10.2 Recursion in SQL

This section needs community help.

Exercise 2.1

Exercise 2.2

Exercise 2.3

Exercise 2.4

10.3 The Object-Relational Model

This section needs community help.

Exercise 3.1

10.4 User-Defined Types in SQL

This section needs community help.

Exercise 4.1

10.5 Operations on Object-Relational Data

This section needs community help.

Exercise 5.1

Exercise 5.2

Exercise 5.3

10.6 On-Line Analytic Processing

This section needs community help.

Exercise 6.1

Exercise 6.2

10.7 Data Cubes

This section needs community help.

Exercise 7.1

Exercise 7.2

Exercise 7.3

Exercise 7.4

Exercise 7.5

Exercise 7.6

Exercise 7.7

Chapter 11 High-Level Database Models

11.1 Semistructured Data

Exercise 1.1

Need community help

Exercise 1.2

UML is structured data model and its schema dependent and is less flexible. On the other side, semistructured data is more flexible than structured data but less. As far as scalability in structured data it is very difficult to scale DB schema, on the other hand in semistructured data it's scaling is simpler than structured data. But essential difference between this two models is:

UML is based on Relational database table and semi-structured model is based on XML/RDF.

11.2 XML

Exercise 2.1

Need community help.

Exercise 2.2

```
<tuples>
  <tuple id="Tuple 1"> ... </tuple>
  <elements>
    <element id="Element 1"> ... </element>
    <element id="Element 2"> ... </element>
    ...
  </elements>
  ...
</tuples>
```

Exercise 2.3

An element with no content is said to be empty. In XML, you can indicate an empty element like this: or you can use an empty tag, like this (this sort of element syntax is called self-closing).

Exercise 2.4

Need community help.

11.3 Document Type Definition

This section needs community help.

Exercise 3.1

Exercise 3.2

11.4 XML Schema

This section needs community help

Exercise 4.1

Exercise 4.2

Exercise 4.3

Chapter 12 Programming Languages for XML

This section needs community help.

12.1 XPath

Exercise 1.1

Exercise 1.2

This section needs community help.

12.2 XQuery

Exercise 2.1

Exercise 2.2

Exercise 2.3

Exercise 2.4

This section needs community help.

12.3 Extensible Stylesheet Language

Exercise 3.1

Exercise 3.2

Chapter 13 Secondary Storage Management

13.1 The Memory Hierarchy

Exercise 1.1

a)

$$\frac{18 \log_2 \frac{10^6}{250}}{12} = 18$$

b)

$$\frac{18 \log_2 1000}{12} = 15$$

c)

$$\frac{18 \log_2 \frac{1000}{6}}{12} = 12$$

d)

$$250 \times 2^5 \approx 8\textbf{TB}$$
$$1 \times 2^5 = 32\textbf{GB}$$
$$6 \times 2^5 = 192\textbf{GHz}$$

Exercise 1.2

$$3 \times 2^{\frac{3600}{18}} = 4.82 \times 10^{60}\textbf{GHz}$$

13.2 Disks

Exercise 2.1

a)

$$10 \times 100000 \times 1000 \times 1024 \approx \ 1\textbf{Tbytes}$$

b)

$$1000 \times 1024 \times 8/(3.5 \times 3.14 \times 0.8) \ approx 0.89$$

c)

$$1 + 100000 \times 0.0002 = 21$$

d)

$$10000 \textbf{ rpm} \rightarrow 167 \textbf{ rps} \rightarrow 6 \textbf{ ms per rotation}$$

e)

$$6 \times \frac{64 \times 0.288^\circ + 63 \times 0.072^\circ}{360^\circ} = 0.3828 \textbf{ ms}$$

f)

$$1 + (0.0002 \times 100000/3) = 7.67 \textbf{ ms}$$

g)

$$6/2 = 3 \textbf{ ms}$$

Exercise 2.2

We first need to calculate the average number of cylinders:

$$\frac{\frac{8192^2}{2} + \frac{57344^2}{2}}{65536} = 25600$$

And then we could calculate:

$$1 + 0.00025 \times 25600 + 4.17 + 0.13 = 11.7 \textbf{ ms}$$

Exercise 2.3

When the head of the disk is located at the first cylinder, average distance moved by head is

$$\frac{1 + 2 + 3 + \cdots + 65535}{65536}$$

When the head of the disk is located at the second cylinder, average distance moved by head is:

$$\frac{1 + 1 + 2 + 3 + \cdots + 65534}{65536}$$

Thus for any distance when the head is located at first cylinder all the sums are 0 and sum of distances for all cylinders will be $(1+2+3+\cdots+N-1)$. For any distance, when head is located at second cylinder all the sums are 1 and sum distances for all cylinders will be $(1+2+3+\cdots+N-2)$.

This expression is expressed as the following.

$$\frac{(x-1)(x-1+1)}{2N} + \frac{(N-x)(N-x+1)}{2N}$$

And we calculate the average:

$$\begin{aligned}
&\frac{1}{N}\sum_{x=1}^{N}\frac{(x-1)x}{2N} + \frac{(N-x)(N-x+1)}{2N} \\
&= \frac{1}{2N^2}\sum_{x=1}^{N}((x-1)x + (N-x)(N-x+1)) \\
&= \frac{1}{2N^2}\sum_{x=1}^{N}(2x^2 - (2N+2)x + n + N^2) \\
&= \frac{1}{2N^2}(2\sum_{x=1}^{N}x^2 - (2N+2)\sum_{x=1}^{N}x + N^2 + N^3) \\
&= \frac{N^2-1}{3N}
\end{aligned}$$

Exercise 2.4

The density function to find probability is

$$p(x) = \frac{3}{5} + \frac{4}{5}x$$

he average distance between two sectors:

$$\int_0^1 \int_0^1 p(xp(y)|x-y|dxdy)$$

Exercise 2.5

$$4.17 + 0.25 + (2 \times 8.33) + 3.73 = 24.81 \textbf{ ms}$$

13.3 Accelerating Access to Secondary Storage

Exercise 3.1

From the Section 13.2, we can know that the average latency and transfer time is 3.8 and 0.5

a)

- Request 1: cylinder 8000, since the disk head already on cylinder 8000 there is no seek time. The access time is 4.3ms. And the completion time is 4.3ms
- Request 2: cylinder 48000, the seek time is $1 + (0.00025 \times 40000) = 11$ms, And the completion time is $11 + 4.3 + 4.3 = 19.6$ms
- Request 3: cylinder 40000, the seek time is $1 + (0.00025 \times 8000) = 3$ms And the completion time is $3 + 4.3 + 19.6 = 26.9$ms
- Request 4: cylinder 4000, the seek time is $1 + (0.00025 \times 36000) = 10$ms And the completion time is $10 + 4.3 + 26.9 = 41.3$ms

b)

We need to only calculate the seek time.

- Request 1: cylinder 8000, since the disk head already on cylinder 8000 there is no seek time. The completion time is 4.3ms
- Request 2: cylinder 48000, the seek time is $1 + (0.00025 \times 40000) = 11$ms And the completion time is $11 + 4.3 + 4.3 = 19.6$ms
- Request 3: cylinder 4000, the seek time is $1 + (0.00025 \times 44000) = 12$ms
- And the completion time is $12 + 4.3 + 19.6 = 35.9$ms
- Request 4: cylinder 40000, the seek time is $1 + (0.00025 \times 36000) = 10$ms And the completion time is $10 + 4.3 + 35.9 = 50.2$ms

Exercise 3.2

We use the data from the Section 13.2

a)

Recall that the Megatron 747 has a transfer time (in milliseconds) of 0.25, and an average rotational latency of 4.17. If the average seek moves 1/3 of the way across half of the 16,384 tracks, the average seek time for the Megatron 747 is $1 + .001 \times (16384/6) = 3.73.1 + .001 \times (16384/6) = 3.73$. Thus, the answer is one block per $0.25 + 4.17 + 3.73 = 8.15$ ms, or 123 blocks per second, on each disk. Thus, the system can read 246 blocks per second.

b)

For a "regular" Megatron 747, the average latency is $0.25+4.17+6.46 = 10.880.25+4.17+6.46 = 10.88$ ms. However, if we have two, mirrored disks, each can be handling a request at the same time, or one read per 5.44 milliseconds. That gives us 184 blocks per second.

c)

Restricting each disk to half the tracks means that if there is not always a queue of waiting requests (which slows the applications that the disk is supporting), then there might be two requests for the same half of the disk, in which case one disk is idle and a request waits.

Exercise 3.3

a)

Number of request we can say that will be n; Head travels across 65536 takes $n + 16.38$ ms. The result in the rotational latency of $4.17n$ ms. The time required for transfer is 0.13 ms which in turn gives the total transfer of $0.13n$ ms

$$n + 16.38 + 4.17n + 0.13n = An$$
$$n = \frac{16.38}{A - 5.3}$$

b)

$$\frac{16.38}{A - 5.3}$$

c)

The waiting time for the request will be 0 when the request comes in right. In some cases, the request comes only after the pass has started which is a worst case.

When the request gets its pass then the wait must get serviced. The wait time for service would be 0. The request will wait for the service at 0 ms, in the best case.

$$\frac{16.38}{A - 5.3} - 0.065 \textbf{ ms}$$

Exercise 3.4

Think of the requests as a random sequence of the integers 1 through 4. This sequence can be divided up into segments that do not contain two of the same integer, in a greedy way. For example, the sequence 123142342431 would be divided into 123, 1423, 42, and 31. The disks are serving all the requests from one segment, and each request is generated and starts at about the same time. When a segment is finished, the waiting request begins, along with other requests for other disks that are, by our assumption, generated almost immediately and thus finish at about the same time.

The question is thus: if I choose numbers 1, 2, 3, and 4 at random, how many choices, on the average, can I make without a repeat? The cases are:

- 1/4 of the time we get an immediate repeat on the second choice, and the length is therefore 1.
- $(3/4)(1/2) = 3/8$ of the time we get our first repeat at third number, and the length is 2.
- $(3/4)(1/2)(3/4) = 9/32$ of the time we get our first repeat at the fourth number, and our length is 3.
- The remaining fraction of the time, 3/32, we get four different numbers, and our length is 4.

The average length is therefore:

$$(1/4) \times 1 + (3/8) \times 2 + (9/32) \times 3 + (3/32) \times 4 = 71/32$$

Exercise 3.5

The disk serves all requests from one segment and each of these requests are generated and started at the same time. The waiting request begins when the segment is finished. This request is raised along with other disks and finishes at the same time. We have a request of the random sequence of integers 1 through k. Now we can choose the numbers at random to find the number of choices without repeat.

- When the length is 1, the it takes $\frac{1}{k}$ of the time for the immediate repeat on the second choice.
- When the length is 2, the time will be $\frac{1}{k} \times 2$.
- When the length is k, the time will be $\frac{1}{k} \times k$

Average length is:

$$\frac{1}{k} + 2 \times \frac{1}{k} + \cdots + k \times \frac{1}{k}$$
$$= (1 + 2 + \cdots + k)\frac{1}{k}$$
$$= \frac{k \times (k+1)}{2}\frac{1}{k}$$
$$= \frac{k}{2}$$

13.4 Disk Failures

Exercise 4.1

a)

There is the odd number of 1's so the parity bit is 1.

b)

There is the even number of 1's so the parity bit is 1

c)

There is an odd number of 1's, so the parity bit is 1

Exercise 4.2

a) 1 0

b) 0 0

c) 1 0

Exercise 4.3

$$\frac{1}{1095} \times 0.04 = 3.65 \textbf{ ms}$$

Exercise 4.4

a)

To compute the mean time to failure, we can compute the probability that the system fails in a given year. The MTTF will be the inverse of that probability. Note that there are 8760 hours in a year. The system fails if the second disk fails while the first is being repaired. The probability of a failure of one of the two disks in a year is $2F$. The probability that the second disk will fail during the H hours that the other is being prepared is $FH/8760$. Thus, the probability of a failure in any year is $2F^{2H/8760}$, and the MTTF is $4380/F^{2H}$.

b)

The system fails if any of the other $N-1$ disks fails while the first is being repaired. The probability of a failure of one of the N disks in a year is NF. The probability that a second disk will fail during the H hours that the other is being prepared is $(N-1)FH/8760$. Thus, the probability of a failure in any year is $N(N-1)F^{2H/8760}$.

Exercise 4.5

$$\frac{8760^2}{3F^3H^2}$$

Exercise 4.6

a) 01010110

b) 00110110

Exercise 4.7

a) 01010110

b) 00110110

Exercise 4.8

a) 10101010

b) 01101100

Exercise 4.9

a)

- 00111100
- 11000111
- 01010101
- 10000100
- 10101110
- 01111111
- 11101101

b)

- 00111100
- 00001111

- 01010101
- 10000100
- 01100110
- 10110111
- 11101101

Exercise 4.10

a)
(Row 1) Using disks 2, 3, and 5, recover disk 1 (Row 3) Using disks 1, 3, and 4, recover disk 7
b)
(Row 1) Using disks 2, 3, and 5, recover disk 1 (Row 2) Using disks 1, 2, and 6, recover disk 4
c)
(Row 1) Using disks 1, 2, and 5, recover disk 3 (Row 2) Using disks 1, 2, and 4, recover disk 6

13.5 Arranging Data on Disk

Exercise 5.1

We first display all the bytes we need to store:

- A character string: 15 bytes
- An integer: 2 bytes
- A SQL date: 10 bytes
- A SQL time: 8 bytes

a)

$$15 + 2 + 10 + 8 = 35 \textbf{ bytes}$$

b)

$$(15 + 1) + (2 + 2) + (10 + 2) + 8 = 40 \textbf{ bytes}$$

c)

$$(15 + 1) + (2 + 6) + (10 + 6) + 8 = 48\textbf{bytes}$$

Exercise 5.2

We first display all the bytes we need to store:

- The real value: 8 bytes
- A character string: 17 bytes
- A single byte: 1 bytes
- A SQL date: 10 bytes

a)

$$8 + 17 + 1 + 10 = 36 \textbf{ bytes}$$

b)

$$8 + (17 + 7) + (1 + 3) + (10 + 2) = 48 \textbf{ bytes}$$

c)

$$8 + (17 + 7) + (1 + 7) + (10 + 6) = 56\textbf{bytes}$$

Exercise 5.3

We should use the answer from Exercise 5.1.

a)

$$4 + 4 + 1 + 35 = 44 \textbf{ bytes}$$

b)

$$4 + 4 + (1 + 3) + 40 = 52 \textbf{ bytes}$$

c)

$$(4 + 4) + (4 + 4) + (1 + 7) + 48 = 72\textbf{bytes}$$

Exercise 5.4

We should use the answer from Exercise 5.2.

a)

$$8 + 20 + 36 = 64 \textbf{ bytes}$$

b)

$$8 + 20 + 48 = 76 \textbf{ bytes}$$

c)

$$8 + (20 + 4) + 56 = 88\textbf{bytes}$$

13.6 Record Modification

This section needs community help.

Exercise 6.1

Exercise 6.2

Exercise 6.3

Exercise 6.4

Exercise 6.5

Exercise 6.6

Exercise 6.7

Exercise 6.8

Exercise 6.9

13.7 Variable-Length Data and Records

Exercise 7.1

$$4 + (3 - 1) \times 4 + 3 \times 10 = 42 \textbf{ bytes}$$

Here, I briefly talk about the meaning:

- 4: Record length.
- (3-1): The number of pointer to variable-length fields.
- 3: The number of fix-length fields.

Exercise 7.2

$$2 + 40 + \frac{10 + 50}{2} + \frac{20 + 80}{2} + \frac{0 + 1000}{2} = 622 \textbf{ bytes}$$

Exercise 7.3

Need community help

Exercise 7.4

For this question, we assume that there are n patients.

a)

$$4 + (40 \times n \times p)$$

b)

$$4 + ((40 + 4) \times n \times p)$$

c)

$$40k + 4$$

d)

$$k = n \times p$$

Exercise 7.5

Every record and record fragment requires a fragment header to support spanned records. Since $500 < r \leq 1000$, we can store only a single record when we don't support spanned records. Therefore, we don't use $1000 - r$ bytes in this case. If we support them, we can store one record and one record fragment and we need to use 32 bytes for fragment headers. Therefore, if $r < 1000 - 32 = 968$, then we can improve space usage.

Exercise 7.6

$$\frac{8 \times 256 \times 4096}{298.262 \times 67.140125} = 418$$

If we organize 418 chunks from different movies fit into 8 tracks and all movies occupy consecutive cylinders, we can play as many as 418 movies with an initial delay of 100 ms.

13.8 Record Modification

Exercise 8.1

- Performances. There is no need for calculating the size of every tuple.

- Better space utilization. The variable-length tuples need allocation of space large enough for handling the maximum possible size of tuples. Many tuples are much smaller and thus storage space could be wasted. Thus, the fixed-length tuple is preferred over variable-length tuple.
- Simple processing. It is easy to update, insert and delete fixed-length tuple.

Chapter 14 Index Structures

14.1 Index-Structure Basics

Exercise 1.1

a)

The number of blocks to store the record is:

$$\frac{n}{3}$$

The number of blocks to store the pointer is:

$$\frac{n}{10}$$

Thus, the answer is:

$$\frac{n}{3} + \frac{n}{10} = \frac{13n}{30}$$

b)

The number of blocks to store the record is:

$$\frac{n}{3}$$

The number of blocks to store the pointer is:

$$\frac{n/3}{10}$$

Thus, the answer is:

$$\frac{n}{3} + \frac{n/3}{10} = \frac{11n}{30}$$

Exercise 1.2

a)

The number of blocks to store the record is:

$$\frac{n}{200 \times 80\%}$$

The number of blocks to store the pointer is:

$$\frac{n}{30 \times 80\%}$$

Thus, the answer is:

$$\frac{n}{200 \times 80\%} + \frac{n}{30 \times 80\%} = \frac{23n}{480}$$

b)

The number of blocks to store the record is:

$$\frac{n}{30 \times 80\%}$$

The number of blocks to store the pointer is:

$$\frac{n}{30 \times 80\% \times 200 \times 80\%}$$

Thus, the answer is:

$$\frac{n}{30 \times 80\%} + \frac{n}{30 \times 80\% \times 200 \times 80\%} = \frac{161n}{3840}$$

14.2 B-Trees

Exercise 2.1

a)

The number to store the data is

$$1000000/10 = 100000 \textbf{ blocks}$$

The number to store the pointers is

$$1000000/70 \approx 14286 \textbf{ blocks}$$

The number to store the key to points is:

$$14286/79 \approx 70 \textbf{ blocks}$$

And we can defer recursively:

$$205/70 \approx 3 \textbf{ blocks}$$
$$3/70 \approx 1 \textbf{ blocks}$$

Total blocks:

$$100000 + 14286 + 205 + 3 + 1 = 114495 textbf blocks$$

And the average number of disk operations should be 5.

b)

Same as (a) because the ordering does not matter.

c)

$$1000000/10 = 100000 \textbf{ blocks}$$
$$100000/70 \approx 1429 \textbf{ blocks}$$
$$1429/70 \approx 21 \textbf{ blocks}$$
$$21/70 \approx 1\textbf{block}$$
$$100000 + 1429 + 21 + 1 = 101451 \textbf{ blocks}$$

And the average number of disk operations should be 4.

d)

$$1000000/7 \approx 142858 \textbf{ blocks}$$
$$142858/70 \approx 2041 \textbf{ blocks}$$
$$2041/70 \approx 30 \textbf{ blocks}$$
$$30/70 \approx 1\textbf{block}$$
$$142858 + 2041 + 30 + 1 = 144930 \textbf{ blocks}$$

And the average number of disk operations should be 4.

e)

The data file is a sequential file, and the B-tree is a sparse index, but each primary block of the data file has one overflow block. On average, the primary block is full, and the overflow block is half full. However, records are in no particular order within a primary block and its overflow

Total blocks: 134302.

And the average number of disk operations should be 13 / 3.

Exercise 2.2

The idea is just like Exercise 2.1

a)

Total blocks: 117.

Average number of disk operations should be 5.

b)

Total blocks: 117.

Average number of disk operations should be 5.

c)

Total blocks: 116.

Average number of disk operations should be 4.

d)

Total blocks: 146857.

Average number of disk operations should be 4.

e)

Total blocks: 136334.

Average number of disk operations should be 13 / 3.

Exercise 2.3

$$\frac{16384}{4} = 4096$$
$$\frac{16384}{12} = 1365$$

Exercise 2.4

a)

Interior nodes: at least 5 keys and 6 pointers. Leaves: at least 5 keys and pointers

b)

Interior nodes: at least 5 keys and 6 pointers. Leaves: at least 6 keys and pointers

Chapter 15 Query Execution

15.1 Introduction to Physical-Query-Plan Operators

There are no exercises in this section.

15.2 One-Pass Algorithms

Exercise 2.1

Need community help.

Exercise 2.2

a)

This is not a blocking operator. The projection operation does not need entire relation to produce the output.

b)

This is not a blocking operator. The distinct tuple is always in the memory.

c)

This is a blocking operator. To group we need to examine every tuple.

d)

This is not a blocking operator. Because we could read each block into the buffer once at a time.

e)

This is not a blocking operator. The operation does not require the entire tuple structure to produce the output.

f)

This can be or can not be a blocking operator. If the computation starts from R to S then each block of R is read and the operation is not blocking. If R is read into $M - 1$ blocks then each block of S is read and the operation is blocking.

g)

This is not a blocking operator. This operation does not need all the tuples to produce the output.

h)

This can be or can not be a blocking operator. This operation is the similar to the operation in the set difference.

i)

This is not a blocking operator. This operation does not need all the tuples to produce the output.

j)

This is not a blocking operator. It just need the matching tuple between the relations to produce the output.

Exercise 2.3

Both arguments are not clustered which means we need to read every tuple.

Operators	Memory required	Disk I/O required
σ, Π	1	T
γ, δ	1	T
$\cap$, $\cup$, $-$,$\times$	$T(S)/T(B)$	T(S) + T(R)

R is clustered and S is not clustered

Operators	Memory required	Disk I/O required
σ, Π	1	T
γ, δ	T	T
$\cap$, $\cup$, $-$,$\times$	$T(S)/T(B)$	B(R) + T(S)

R is not clustered and S is clustered

Operators	Memory required	Disk I/O required
σ, Π	1	T
γ, δ	T	T
$\cap$, $\cup$, $-$,$\times$	$B(S)$	T(R) + B(S)

Exercise 2.4

Need community help.

15.3 Nested-Loop Joins

Exercise 3.1

Need community help.

Exercise 3.2

According to the formula:

$$B(S) + ((B(S)B(R)))/(M-1)$$

We have

$$10000 + (10000^2)/999 = 110101$$

Exercise 3.3

According to the formula:

$$B(S) + ((B(S)B(R)))/(M-1) = IO$$

We can have:

$$M = (IO - B(S))/((B(S)B(R))) + 1$$

(a)

$$M = 1112$$

(b)

$$M = 6.667$$

(c)

$$M = 20.001$$

Exercise 3.4

a)

A block based nested loop join, used to join two relation R and S in a relational database, but nested loop join, used to two relations and the outer and inner join respectively.

b)

case 1: R is not clustered and smaller. Cost of reading all tuples of S (clustered), $T(S) + B(S)$
Cost of reading R tuples and the cost of join with S in the main memory is: $T(R)+B(S)B(R)/M$
The total cost is $T(R) + B(S)B(R)/M + T(S) + T(R)$

case 2: S is not clustered and smaller. Cost of reading all tuples of S (clustered), $T(R) + B(R)$
Cost of reading S tuples and the cost of join with R in the main memory is: $T(S)+B(S)B(R)/M$
The total cost is $T(S) + B(S)B(R)/M + T(R) + B(R)$

Exercise 3.5

Need community help.

15.4 Two-Pass Algorithms Based on Sorting

Exercise 4.1

Need community help.

Exercise 4.2

(a)

$$3 * (B(R) + B(S)) = 3 * (10000 + 10000) = 60000$$

(b)

$$5 * (B(R) + B(S)) = 5 * (10000 + 10000) = 100000$$

(c)

$$3 * (B(R) + B(S)) = 3 * (10000 + 10000) = 60000$$

Exercise 4.3

By using the extra buffers for saving disk I/O, we can also increase the overall performance of I/O. Sometimes I/O is the bottleneck for the is against improvement of runtime performance. That's why, it has a big role in this system. But there are also some types of buffering of disk I/O. They are single buffering, double buffering, circular buffer, an so on. A system transmits data of I/O to one buffer while operating system empties the other in double buffering.

Exercise 4.4

Total number of disk I/O: sort relation R and S + (y-values $*$ number of I/O)

a)

$$4(B(R) + B(S)) + (250 + (3 * 500)) = 9500$$

b)

$$4(B(R) + B(S)) + (5 * 300) = 7500$$

c)

$$4(B(R) + B(S)) + (10 * 150) = 7500$$

Exercise 4.5

a)

$$2(B(R) + B(S)) + (2 * 1750) = 6500$$

b)

$$2(B(R) + B(S)) + (5 * 500) = 5500$$

c)

$$4(B(R) + B(S)) + (10 * 150) = 7500$$

Exercise 4.6

(a)

$$M = \sqrt{B} = 100$$

(b)

$$M = \sqrt{B} = 100$$

(c)

$$M = \sqrt{B(S)B(R)} = 10000$$

Exercise 4.7

Need community help.

Exercise 4.8

Need community help.

Exercise 4.9

Since the merging operation is used for sorting the sequence hence the total time complexity is $O(M \log(M))$ and the space complexity is $\log(M)$. The concerned term is the space complexity here as the I/O operations take place on the disk memory and lesser the consumption of space lesser will be the I/O operations. That is why the amount by which the disk I/O operations will be reduced is equal to $\log(M)$. So if there were M I/O operations were to take place, now only $M - \log(M)$ I/O operations would take place.

15.5 Two-Pass Algorithms Based on Hashing

Exercise 5.1

Need community help.

Exercise 5.2

$$3 - \frac{2M}{B(S)} * (B(R) + B(S))(3 - 0.2) * 20000 = 56000$$

Exercise 5.3

Need community help.

Exercise 5.4

No matter how many members have, each group, requires only one tuple in main memory. Thus, we do not need modifications, in fact, memory use will be pleasantly small.

Exercise 5.5

Need community help.

15.6 Index-Based Algorithms

Exercise 6.1

a)

Initially read and output all the tuples in relation R. Then, for every tuple t in S use the index for attribute $R.a$ to match the tuples on the relation of R with $t.a$. If t is not present on those tuples, then output the tuple t.

This operation works efficiently if the relation S contains small values and R contains large values. So that the main memory does not exceed the size.

b)

Initially read and output all the tuples in relation S. Then, for every tuple t in S use the index for attribute $R.a$ to match the tuples on the relation of R with $t.a$. If t is not present on those tuples, then output the tuple t.

This operation works efficiently if the relation S contains small values and R contains large values. So that the main memory does not exceed the size.

c)

This operation uses the index for $R.a$ to process the blocks of 'R' relation. With given index key, organize each block and if there is only one tuple for the given index key the output that tuple t.

Exercise 6.2

a)

$$\frac{10000}{k}$$

b)

$$\frac{500000}{k}$$

c)

We need to retrieve every block of R, which in this case is 10000 disk I/O's.

Exercise 6.3

a)

$$\frac{B(R)}{V(R,a)} * \text{Key} = 1000$$

b)

$$\frac{T(R)}{V(R,a)} * \text{Key} = 50000$$

c)

We need to retrieve every block of R, which in this case is 10000 disk I/O's.

Exercise 6.4

Need community help.

Exercise 6.5

With the index, we have only to retrieve the blocks containing `MovieStar` records for the stars of `King Kong`. We don't know how many stars there are, but it is safe to assume that their records will occupy many more blocks than there are stars in the two `King Kong` movies. Thus, using the index allows us to take the join while accessing only a small part of the `MovieStar` relation and is therefore to be preferred to a sort- or hash-join, each of which will access the entire `MovieStar` relation at least once.

Exercise 6.6

Due to skipping pf sorting phase, it reads one block of S and one block read R and this method works as long as the number of tuples present in the relation R and S which contains the same Y-value that fit into M blocks.

15.7 Buffer Management

Exercise 7.1

a)

The terms $B(R)$ and $B(S)$ represents binary relations between the relations R and S. In one-pass, there is an approximate requirement for the binary operation between the relations R and S as:

$$\min(B(R), B(S)) \leq M$$

This approximation means that the one buffer is used to read the blocks of larger relation and for smaller relations, it requires M buffers additionally with the main memory structure. According to the one-pass algorithm and given terms, the relation either R or S must fit into the memory. So, the approximation rule would be:

$$min(B(R), B(S)) \geq \frac{M}{2}$$

b)

This algorithm might work properly only if the one-pass union, intersection and difference between the relations R_i and S_i whose sizes are found to be $\frac{B(R)}{M-1}$ and $\frac{B(S)}{M-1}$ respectively as it is known, the one-pass algorithm requires operand and it occupies at most $M-1$ blocks.

Therefore the two-pass hash based algorithms requires at least $\min(B(R), B(S)) \leq M^2$ approximately.

Since the worst case of two-pass algorithm contains $\frac{M}{2}$ blocks. The memory between M and $\frac{M}{2}$ requires approximately

$$\min(B(R), B(S)) \leq \frac{M^2}{4}$$

c)

$$\max(B(R), B(S)) \leq \frac{M^2}{4}$$

Exercise 7.2

a)

Following the concept of FIFO the new blocks occupies the buffer by emptying the current longest block in buffer. When doing the nested loop join operation, the FIFO pointer pointing the longest block of the buffer and it need not points out the first tuple in the buffer.

So it will not improve the number of disk I/O's on nested loop join operations.

b)

The clock algorithm contains the handle which places the disk on the available buffer by rotating the handle in clockwise direction. While doing the nested loop join, approach also doesn't

searches for the first tuple as the handle of the clock is present on the available space of the buffer.

It will not improve the number of disk I/O's on nested loop join operations.

Exercise 7.3

Need community help.

Chapter 16 The Query Compiler

16.1 Parsing and Preprocessing

Need community help.

16.2 Algebraic Laws for Improving Query Plans

Exercise 2.1

We could rewrite the $\sigma_C(R \cap S)$ to $\sigma_C(R) \cap \sigma_C(S)$. whenever there exists indexes, it would make the query faster.

Exercise 2.2

a)

Let $R(A, B) = \{(2, 4)\}$ and $S(A, B) = \{2, 7\}$. Then

$$\pi_A(R \cup S) = \{(2), (2)\}$$
$$\pi_A(R) \cup \pi_a(S) = \{(2)\}$$

b)

Let $R(A, B) = \{(2, 4)\}$ and $S(A, B) = \{2, 7\}$. Then

$$\pi_A(R - S) = \{(3)\}$$
$$\pi_A(R) - \pi_A(S) = \emptyset$$

$$\pi_A(R -_B S) = \{(3)\}$$
$$\pi_A(R) -_B \pi_A(S) = \emptyset$$

c)

Let $R(A, B) = \{(2, 4), (2, 5)\}$

$$\delta(\pi_A(R)) = \{(2)\}$$
$$\pi_A(\delta(R)) = \{(2), (2)\}$$

d)

Let $R(A, B) = \{(2, 4), (2, 4)\}$ and $S(A, B) = \{2, 4\}$

$$\delta(R \cup_B S) = \{2, 4\}$$
$$\delta(R) \cup_B \delta(S) = \{(3, 4), (3, 4)\}$$

Exercise 2.3

For every tuple r_i of relation R and every tuple s_j of relation S, we could have the following

$$\begin{aligned}\pi_L(R \cup_B S) &= \pi_L(r_i, s_j) \\ &= \pi_L(r_i), \pi_L(s_j) \\ &= \pi_L(R) \cup_B \pi_L(S)\end{aligned}$$

Exercise 2.4

a)

For every tuple r_i of relation R we could have

$$R \cup R = \{r_i\}$$
$$R \cup_B R = \{r_i, r_i\}$$

b)

For every tuple r_i of relation R we could have

$$R \cap R \in \{r_i\}$$
$$R \cap_B R = \{r_i, r_i\}$$

c)

For every tuple r_i of relation R we could have

$$R - R = \emptyset$$
$$R -_B R = \emptyset$$

d)

Let x be the tuples of relation R, S, and T.

$$\begin{aligned} R \cup_B (S \cap_B T) &= R \cup_B (x, x) \\ &= x, x, x \end{aligned}$$

$$\begin{aligned} (R \cup_B S) \cap_B (R \cup_B T) &= (x, x) \cap_B (x, x) \\ &= x, x, x, x \end{aligned}$$

Exercise 2.5

Need community help.

Exercise 2.6

Need community help.

Exercise 2.7

Need community help.

Exercise 2.8

Need community help.

Exercise 2.9

Need community help.

Exercise 2.10

Need community help.

16.3 From Parse Trees to Logical Query Plans

Exercise 3.1

a)

$$\pi_{a,R.b,S.b,S.c,T.c,d}(R(a,b) \bowtie_{R.b=S.b} S(b,c)) \bowtie_{S.c>T.c} T(c,d)$$

b)

$$\pi_{a,R.b,S.b,S.c,T.c,U.d,e}(R(a,b) \bowtie_{R.b=S.b} S(b,c)) \bowtie_{S.c=T.c} (T(c,d) \bowtie_{T.d=U.d} U(d,e))$$

c)

$$\pi_L(R(a,b) \bowtie_{R.b=S.b} S(b,c)) \bowtie_{R.a=U.a \text{ AND } S.c=T.c} (T(c,d) \bowtie_{T.d=U.d} U(a,d))$$

Exercise 3.2

Need community help.

Exercise 3.3

a)

The rule for the given condition is

$$\pi_L(\sigma_{COUNT>0}(R \times \delta(\pi_\emptyset(S))))$$

- Create an expression for subquery S, the project only the empty list of attributes. Apply δ on the projection to eliminate duplicates.
- Take the product of R and the result from S.
- Check if the product of R and S is empty. If the product is empty, then the projected list π_L is empty. If the product is not empty, the projected list is R.

b)

$$\pi_L(R \bowtie_{a=b} \delta(S))$$

- Create an expression for the subquery S. If S may have duplicates, apply the δ operator to S.
- Get the product of R and S based on the condition that equates a to the corresponding attribute of b of S.
- Project the result of the product to the attributes of L.

c)

$$\pi_L(R \bowtie_{a<>b} \delta(S))$$

- Create an expression for the subquery S. If S may have duplicates, apply the δ operator to S.
- Get the product of R and S based on the condition that do not equates a to the corresponding attribute of b of S.
- Project the result of the product to the attributes of L.

Exercise 3.4

Need community help.

Exercise 3.5

It's a combinatorics problem.

$$3! \times 3! \times 2 = 72$$

Chapter 18 Concurrency Control

18.1 Serial and Serializable Schedules

Exercise 1.1

- Flight information is retrieved from database element A and B, which means $r_1(A)$ and $r_1(B)$.
- Selects a flight and flight reservation is made for the customer, which means $r_1(B)$ and $w_1(B)$.
- Customer selects a flight seat from database element C, which means $r_1(C)$ and $w_1(C)$.
- Get customer's credit card number and records flight bill in database element D, which means $r_1(D)$ and $w_1(D)$.
- Customer's phone and flight data is inserted to database element E, which means $r_1(E)$ and $w_1(E)$.

Thus we can get the following answer:

$$r_1(A), r_1(B), w_1(B), r_1(C), w_1(C), r_1(D), w_1(D), r_1(E), w_1(E)$$

Exercise 1.2

This is a classical combinatorics question.

$$\frac{10!}{4! \times (10-4)!} = 210$$

18.2 Conflict-Serializability

Exercise 2.1

a)

I don't give an example here, because the t and s is different, thus (T_1, T_2) would be equivalent to (T_2, T_1).

b)

Need community help.

c)

According to the definition of serial schedules. The only serial order that we have is (T_1, T_2) and (T_2, T_1). Thus we can

d)

To identify the number of serializable schedule of the 12 given actions, we need to consider the interleaving of the serial order (T_1, T_2) and (T_2, T_1). And we have the following order:

- T_1 operates on A and B before T_2.
- T_2 operates on A and B before T_1.
- T_1 operates on A first, but T_2 operates on B first.

$$\left(\frac{6!}{3! \times (6-3)!}\right)^2 = 400$$

Exercise 2.2

a)

We cannot swap the adjacent actions without a conflict, only (T_1, T_2) is conflict equivalent to itself. So the answer is 1.

b)

We can swap the adjacent actions without a conflict, so the answer is

c)

$$\left(\frac{4!}{2! \times (4-2)!}\right)^2 = 36$$

d)

The number of actions differs. This implies that we have different answers as number of actions determines the number of possible interleavings for the serializable order.

Exercise 2.3

a)

$$\frac{4!}{2! \times (4-2)!} \times 2 = 12$$

b)

$$\left(\frac{4!}{2! \times (4-2)!}\right)^2 = 36$$

Exercise 2.4

Need community help.

Exercise 2.5

Need community help.

Exercise 2.6

Need community help.

18.3 Enforcing Serializability by Locks

Exercise 3.1

a)

Suppose that we have the schedule derived from the given transactions.

$$r_1(A); r_2(B); w_2(B); r_2(A); w_1(A); r_1(B); w_1(B); w_2(A)$$

Note that $r_1(A)$ must hold a lock on A, since a transaction is granted with lock before its read and write action. Likewise, $r_2(A)$ must also hold a lock on A.

Since the unlock of A for $r_1(A)$ is executed after $w_1(A)$ and $r_2(A)$ appears before $w_1(A)$. Therefore, the schedule is prohibited.

b)

$$\frac{4!}{2! \times (4-2)!} \times \frac{4!}{2! \times (4-2)!} + 2 = 38$$

c)

$$\frac{4!}{2! \times (4-2)!} \times \frac{4!}{2! \times (4-2)!} = 36$$

d)

The number of conflict serializable schedule is 2.

e)

All legal schedules are serializable. It is impossible for a legal schedule to be unserializable.

Exercise 3.2

Need community help.

Exercise 3.3

Need community help.

Exercise 3.4

Need community help.