4

RELATIONAL ALGEBRA AND CALCULUS

Exercise 4.1 Explain the statement that relational algebra operators can be *composed*. Why is the ability to compose operators important?

Answer 4.1 Every operator in relational algebra accepts one or more relation instances as arguments and the result is always an relation instance. So the argument of one operator could be the result of another operator. This is important because, this makes it easy to write complex queries by simply composing the relational algebra operators.

Exercise 4.2 Given two relations $R1$ and $R2$, where $R1$ contains N1 tuples, $R2$ contains N2 tuples, and N2 > N1 > 0, give the minimum and maximum possible sizes (in tuples) for the resulting relation produced by each of the following relational algebra expressions. In each case, state any assumptions about the schemas for $R1$ and $R2$ needed to make the expression meaningful:

(1) $R1 \cup R2$, (2) $R1 \cap R2$, (3) $R1 - R2$, (4) $R1 \times R2$, (5) $\sigma_{a=5}(R1)$, (6) $\pi_a(R1)$, and (7) $R1/R2$

Answer 4.2 Answer omitted.

Exercise 4.3 Consider the following schema:

Suppliers(<u>*sid:* `integer`</u>, *sname:* `string`, *address:* `string`)
Parts(<u>*pid:* `integer`</u>, *pname:* `string`, *color:* `string`)
Catalog(<u>*sid:* `integer`, *pid:* `integer`</u>, *cost:* `real`)

The key fields are underlined, and the domain of each field is listed after the field name. Therefore *sid* is the key for Suppliers, *pid* is the key for Parts, and *sid* and *pid* together form the key for Catalog. The Catalog relation lists the prices charged for parts by Suppliers. Write the following queries in relational algebra, tuple relational calculus, and domain relational calculus:

1. Find the *name*s of suppliers who supply some red part.
2. Find the *sid*s of suppliers who supply some red or green part.
3. Find the *sid*s of suppliers who supply some red part or are at 221 Packer Street.
4. Find the *sid*s of suppliers who supply some red part and some green part.
5. Find the *sid*s of suppliers who supply every part.
6. Find the *sid*s of suppliers who supply every red part.
7. Find the *sid*s of suppliers who supply every red or green part.
8. Find the *sid*s of suppliers who supply every red part or supply every green part.
9. Find pairs of *sid*s such that the supplier with the first *sid* charges more for some part than the supplier with the second *sid*.
10. Find the *pid*s of parts supplied by at least two different suppliers.
11. Find the *pid*s of the most expensive parts supplied by suppliers named Yosemite Sham.
12. Find the *pid*s of parts supplied by every supplier at less than \$200. (If any supplier either does not supply the part or charges more than \$200 for it, the part is not selected.)

Answer 4.3 In the answers below RA refers to Relational Algebra, TRC refers to Tuple Relational Calculus and DRC refers to Domain Relational Calculus.

1. ■ RA

$$\pi_{sname}(\pi_{sid}((\pi_{pid}\sigma_{color='red'}Parts) \bowtie Catalog) \bowtie Suppliers)$$

■ TRC

$$\{T \mid \exists T1 \in Suppliers(\exists X \in Parts(X.color =' red' \wedge \exists Y \in Catalog$$
$$(Y.pid = X.pid \wedge Y.sid = T1.sid)) \wedge T.sname = T1.sname)\}$$

■ DRC

$$\{\langle Y \rangle \mid \langle X, Y, Z \rangle \in Suppliers \wedge \exists P, Q, R(\langle P, Q, R \rangle \in Parts$$
$$\wedge R =' red' \wedge \exists I, J, K(\langle I, J, K \rangle \in Catalog \wedge J = P \wedge I = X))\}$$

■ SQL

```
SELECT S.sname
FROM   Suppliers S, Parts P, Catalog C
WHERE  P.color='red' AND C.pid=P.pid AND C.sid=S.sid
```

2. ■ RA

$$\pi_{sid}(\pi_{pid}(\sigma_{color='red' \vee color='green'} Parts) \bowtie catalog)$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\exists X \in Parts((X.color = \text{`}red' \vee X.color = \text{`}green')$$
$$\wedge X.pid = T1.pid) \wedge T.sid = T1.sid)\}$$

■ DRC

$$\{\langle X\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge \exists A, B, C(\langle A, B, C\rangle \in Parts$$
$$\wedge(C =' red' \vee C =' green') \wedge A = Y)\}$$

■ SQL

```
SELECT C.sid
FROM   Catalog C, Parts P
WHERE  (P.color = 'red' OR P.color = 'green')
       AND P.pid = C.pid
```

3. ■ RA

$$\rho(R1, \pi_{sid}((\pi_{pid}\sigma_{color='red'} Parts) \bowtie Catalog))$$
$$\rho(R2, \pi_{sid}\sigma_{address='221PackerStreet'} Suppliers)$$
$$R1 \cup R2$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\exists X \in Parts(X.color = \text{`}red' \wedge X.pid = T1.pid)$$
$$\wedge T.sid = T1.sid)$$
$$\vee \exists T2 \in Suppliers(T2.address =' 221PackerStreet' \wedge T.sid = T2.sid)\}$$

■ DRC

$$\{\langle X\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge \exists A, B, C(\langle A, B, C\rangle \in Parts$$
$$\wedge C =' red' \wedge A = Y)$$
$$\vee \exists P, Q(\langle X, P, Q\rangle \in Suppliers \wedge Q =' 221PackerStreet')\}$$

■ SQL

```
SELECT S.sid
FROM   Suppliers S
WHERE  S.address = '221 Packer street'
       OR S.sid IN ( SELECT C.sid
                     FROM   Parts P, Catalog C
                     WHERE  P.color='red' AND P.pid = C.pid )
```

4. ■ RA

$$\rho(R1, \pi_{sid}((\pi_{pid}\sigma_{color='red'}Parts) \bowtie Catalog))$$
$$\rho(R2, \pi_{sid}((\pi_{pid}\sigma_{color='green'}Parts) \bowtie Catalog))$$
$$R1 \cap R2$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\exists X \in Parts(X.color = `red' \wedge X.pid = T1.pid)$$
$$\wedge \exists T2 \in Catalog(\exists Y \in Parts(Y.color =' green' \wedge Y.pid = T2.pid)$$
$$\wedge T2.sid = T1.sid) \wedge T.sid = T1.sid)\}$$

■ DRC

$$\{\langle X\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge \exists A, B, C(\langle A, B, C\rangle \in Parts$$
$$\wedge C =' red' \wedge A = Y)$$
$$\wedge \exists P, Q, R(\langle P, Q, R\rangle \in Catalog \wedge \exists E, F, G(\langle E, F, G\rangle \in Parts$$
$$\wedge G =' green' \wedge E = Q) \wedge P = X)\}$$

■ SQL

```
SELECT C.sid
FROM   Parts P, Catalog C
WHERE  P.color = 'red' AND P.pid = C.pid
       AND EXISTS ( SELECT P2.pid
                    FROM   Parts P2, Catalog C2
                    WHERE  P2.color = 'green' AND C2.sid = C.sid
                           AND P2.pid = C2.pid )
```

5. ■ RA

$$(\pi_{sid,pid}Catalog)/(\pi_{pid}Parts)$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\forall X \in Parts(\exists T2 \in Catalog$$
$$(T2.pid = X.pid \wedge T2.sid = T1.sid)) \wedge T.sid = T1.sid)\}$$

- DRC

$$\{\langle X\rangle \mid \langle X,Y,Z\rangle \in Catalog \wedge \forall\langle A,B,C\rangle \in Parts$$
$$(\exists\langle P,Q,R\rangle \in Catalog(Q = A \wedge P = X))\}$$

- SQL

```
SELECT C.sid
FROM   Catalog C
WHERE  NOT EXISTS (SELECT P.pid
                   FROM   Parts P
                   WHERE  NOT EXISTS (SELECT C1.sid
                                      FROM   Catalog C1
                                      WHERE  C1.sid = C.sid
                                             AND C1.pid = P.pid))
```

6. - RA

$$(\pi_{sid,pid}Catalog)/(\pi_{pid}\sigma_{color='red'}Parts)$$

- TRC

$$\{T \mid \exists T1 \in Catalog(\forall X \in Parts(X.color \neq \text{`}red\text{'}$$
$$\vee \exists T2 \in Catalog(T2.pid = X.pid \wedge T2.sid = T1.sid))$$
$$\wedge T.sid = T1.sid)\}$$

- DRC

$$\{\langle X\rangle \mid \langle X,Y,Z\rangle \in Catalog \wedge \forall\langle A,B,C\rangle \in Parts$$
$$(C \neq \text{`}red\text{'} \vee \exists\langle P,Q,R\rangle \in Catalog(Q = A \wedge P = X))\}$$

- SQL

```
SELECT C.sid
FROM   Catalog C
WHERE  NOT EXISTS (SELECT P.pid
                   FROM   Parts P
                   WHERE  P.color = 'red'
                   AND (NOT EXISTS (SELECT C1.sid
                                    FROM   Catalog C1
                                    WHERE  C1.sid = C.sid AND
                                           C1.pid = P.pid)))
```

7. - RA

$$(\pi_{sid,pid}Catalog)/(\pi_{pid}\sigma_{color='red' \vee color='green'}Parts)$$

- TRC

$$\{T \mid \exists T1 \in Catalog(\forall X \in Parts((X.color \neq \text{`}red' \\ \wedge X.color \neq \text{`}green') \vee \exists T2 \in Catalog \\ (T2.pid = X.pid \wedge T2.sid = T1.sid)) \wedge T.sid = T1.sid)\}$$

- DRC

$$\{\langle X\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge \forall \langle A, B, C\rangle \in Parts \\ ((C \neq \text{`}red' \wedge C \neq \text{`}green') \vee \exists \langle P, Q, R\rangle \in Catalog \\ (Q = A \wedge P = X))\}$$

- SQL

```
SELECT C.sid
FROM   Catalog C
WHERE  NOT EXISTS (SELECT P.pid
                   FROM   Parts P
                   WHERE  (P.color = 'red' OR P.color = 'green')
                   AND (NOT EXISTS (SELECT C1.sid
                                    FROM   Catalog C1
                                    WHERE  C1.sid = C.sid AND
                                           C1.pid = P.pid)))
```

8. - RA

$$\rho(R1, ((\pi_{sid,pid}Catalog)/(\pi_{pid}\sigma_{color='red'}Parts))) \\ \rho(R2, ((\pi_{sid,pid}Catalog)/(\pi_{pid}\sigma_{color='green'}Parts))) \\ R1 \cup R2$$

- TRC

$$\{T \mid \exists T1 \in Catalog((\forall X \in Parts \\ (X.color \neq \text{`}red' \vee \exists Y \in Catalog(Y.pid = X.pid \wedge Y.sid = T1.sid)) \\ \vee \forall Z \in Parts(Z.color \neq \text{`}green' \vee \exists P \in Catalog \\ (P.pid = Z.pid \wedge P.sid = T1.sid))) \wedge T.sid = T1.sid)\}$$

- DRC

$$\{\langle X\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge (\forall \langle A, B, C\rangle \in Parts \\ (C \neq \text{`}red' \vee \exists \langle P, Q, R\rangle \in Catalog(Q = A \wedge P = X)) \\ \vee \forall \langle U, V, W\rangle \in Parts(W \neq \text{`}green' \vee \langle M, N, L\rangle \in Catalog \\ (N = U \wedge M = X)))\}$$

- SQL

```
SELECT C.sid
FROM   Catalog C
WHERE  (NOT EXISTS (SELECT P.pid
                    FROM   Parts P
                    WHERE  P.color = 'red' AND
                    (NOT EXISTS (SELECT C1.sid
                                 FROM   Catalog C1
                                 WHERE  C1.sid = C.sid AND
                                        C1.pid = P.pid))))
       OR ( NOT EXISTS (SELECT P1.pid
                        FROM   Parts P1
                        WHERE  P1.color = 'green' AND
                        (NOT EXISTS (SELECT C2.sid
                                     FROM   Catalog C2
                                     WHERE  C2.sid = C.sid AND
                                            C2.pid = P1.pid))))
```

9. - RA

$$\rho(R1, Catalog)$$
$$\rho(R2, Catalog)$$
$$\pi_{R1.sid,R2.sid}(\sigma_{R1.pid=R2.pid \wedge R1.sid \neq R2.sid \wedge R1.cost > R2.cost}(R1 \times R2))$$

- TRC

$$\begin{aligned}\{T \quad | \quad & \exists T1 \in Catalog(\exists T2 \in Catalog \\ & (T2.pid = T1.pid \wedge T2.sid \neq T1.sid \\ & \wedge T2.cost < T1.cost \wedge T.sid2 = T2.sid) \\ & \wedge T.sid1 = T1.sid)\}\end{aligned}$$

- DRC

$$\begin{aligned}&\{\langle X, P\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge \exists P, Q, R \\ &(\langle P, Q, R\rangle \in Catalog \wedge Q = Y \wedge P \neq X \wedge R < Z)\}\end{aligned}$$

- SQL

```
SELECT C1.sid, C2.sid
FROM   Catalog C1, Catalog C2
WHERE  C1.pid = C2.pid AND C1.sid ≠ C2.sid
       AND C1.cost > C2.cost
```

10. ■ RA

$$\rho(R1, Catalog)$$
$$\rho(R2, Catalog)$$
$$\pi_{R1.pid}\sigma_{R1.pid=R2.pid \wedge R1.sid \neq R2.sid}(R1 \times R2)$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\exists T2 \in Catalog$$
$$(T2.pid = T1.pid \wedge T2.sid \neq T1.sid)$$
$$\wedge T.pid = T1.pid)\}$$

■ DRC

$$\{\langle X \rangle \mid \langle X, Y, Z \rangle \in Catalog \wedge \exists A, B, C$$
$$(\langle A, B, C \rangle \in Catalog \wedge B = Y \wedge A \neq X)\}$$

■ SQL

```
SELECT  C.pid
FROM    Catalog C
WHERE   EXISTS (SELECT  C1.sid
                FROM    Catalog C1
                WHERE   C1.pid = C.pid AND C1.sid ≠ C.sid )
```

11. ■ RA

$$\rho(R1, \pi_{sid}\sigma_{sname='YosemiteSham'}Suppliers)$$
$$\rho(R2, R1 \bowtie Catalog)$$
$$\rho(R3, R2)$$
$$\rho(R4(1 \rightarrow sid, 2 \rightarrow pid, 3 \rightarrow cost), \sigma_{R3.cost<R2.cost}(R3 \times R2))$$
$$\pi_{pid}(R2 - \pi_{sid,pid,cost}R4)$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\exists X \in Suppliers$$
$$(X.sname =' YosemiteSham' \wedge X.sid = T1.sid) \wedge \neg(\exists S \in Suppliers$$
$$(S.sname =' YosemiteSham' \wedge \exists Z \in Catalog$$
$$(Z.sid = S.sid \wedge Z.cost > T1.cost))) \wedge T.pid = T1.pid)$$

■ DRC

$$\{\langle Y \rangle \mid \langle X, Y, Z \rangle \in Catalog \wedge \exists A, B, C$$
$$(\langle A, B, C \rangle \in Suppliers \wedge C =' YosemiteSham' \wedge A = X)$$
$$\wedge \neg(\exists P, Q, R(\langle P, Q, R \rangle \in Suppliers \wedge R =' YosemiteSham'$$
$$\wedge \exists I, J, K(\langle I, J, K \rangle \in Catalog(I = P \wedge K > Z))))\}$$

- SQL

```
SELECT C.pid
FROM   Catalog C, Suppliers S
WHERE  S.sname = 'Yosemite Sham' AND C.sid = S.sid
       AND C.cost ≥ ALL (Select C2.cost
                         FROM  Catalog C2, Suppliers S2
                         WHERE S2.sname = 'Yosemite Sham'
                               AND C2.sid = S2.sid)
```

Exercise 4.4 Consider the Supplier-Parts-Catalog schema from the previous question. State what the following queries compute:

1. $\pi_{sname}(\pi_{sid}((\sigma_{color='red'}Parts) \bowtie (\sigma_{cost<100}Catalog)) \bowtie Suppliers)$
2. $\pi_{sname}(\pi_{sid}((\sigma_{color='red'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers))$
3. $(\pi_{sname}((\sigma_{color='red'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers)) \cap$
   $$(\pi_{sname}((\sigma_{color='green'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers))$$
4. $(\pi_{sid}((\sigma_{color='red'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers)) \cap$
   $$(\pi_{sid}((\sigma_{color='green'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers))$$
5. $\pi_{sname}((\pi_{sid,sname}((\sigma_{color='red'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers)) \cap$
   $$(\pi_{sid,sname}((\sigma_{color='green'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers)))$$

Answer 4.4 The statements can be interpreted as:

1. Find the Supplier names of the suppliers who supply a red part that costs less than 100 dollars.
2. This Relational Algebra statement does not return anything because of the sequence of projection operators. Once the sid is projected, it is the only field in the set. Therefore, projecting on sname will not return anything.
3. Find the Supplier names of the suppliers who supply a red part that costs less than 100 dollars and a green part that costs less than 100 dollars.
4. Find the Supplier ids of the suppliers who supply a red part that costs less than 100 dollars and a green part that costs less than 100 dollars.
5. Find the Supplier names of the suppliers who supply a red part that costs less than 100 dollars and a green part that costs less than 100 dollars.

Exercise 4.5 Consider the following relations containing airline flight information:

Flights(*flno:* `integer`, *from:* `string`, *to:* `string`,
distance: `integer`, *departs:* `time`, *arrives:* `time`)
Aircraft(*aid:* `integer`, *aname:* `string`, *cruisingrange:* `integer`)
Certified(*eid:* `integer`, *aid:* `integer`)
Employees(*eid:* `integer`, *ename:* `string`, *salary:* `integer`)

Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft (otherwise, he or she would not qualify as a pilot), and only pilots are certified to fly.

Write the following queries in relational algebra, tuple relational calculus, and domain relational calculus. Note that some of these queries may not be expressible in relational algebra (and, therefore, also not expressible in tuple and domain relational calculus)! For such queries, informally explain why they cannot be expressed. (See the exercises at the end of Chapter 5 for additional queries over the airline schema.)

1. Find the *eid*s of pilots certified for some Boeing aircraft.
2. Find the *name*s of pilots certified for some Boeing aircraft.
3. Find the *aid*s of all aircraft that can be used on non-stop flights from Bonn to Madras.
4. Identify the flights that can be piloted by every pilot whose salary is more than $100,000.
5. Find the names of pilots who can operate planes with a range greater than 3,000 miles but are not certified on any Boeing aircraft.
6. Find the *eid*s of employees who make the highest salary.
7. Find the *eid*s of employees who make the second highest salary.
8. Find the *eid*s of employees who are certified for the largest number of aircraft.
9. Find the *eid*s of employees who are certified for exactly three aircraft.
10. Find the total amount paid to employees as salaries.
11. Is there a sequence of flights from Madison to Timbuktu? Each flight in the sequence is required to depart from the city that is the destination of the previous flight; the first flight must leave Madison, the last flight must reach Timbuktu, and there is no restriction on the number of intermediate flights. Your query must determine whether a sequence of flights from Madison to Timbuktu exists for *any* input Flights relation instance.

Answer 4.5 In the answers below RA refers to Relational Algebra, TRC refers to Tuple Relational Calculus and DRC refers to Domain Relational Calculus.

1. - RA

$$\pi_{eid}(\sigma_{aname=\text{`}Boeing'}(Aircraft \bowtie Certified))$$

   - TRC

$$\{C.eid \mid C \in Certified \wedge \exists A \in Aircraft(A.aid = C.aid \wedge A.aname = \text{`}Boeing'))\}$$

   - DRC

$$\{\langle Ceid\rangle \mid \langle Ceid, Caid\rangle \in Certified \wedge \exists Aid, AN, AR(\langle Aid, AN, AR\rangle \in Aircraft \wedge Aid = Caid \wedge AN = \text{`}Boeing'))\}$$

   - SQL

```
SELECT C.eid
FROM   Aircraft A, Certified C
WHERE  A.aid = C.aid AND A.aname = 'Boeing'
```

2. - RA

$$\pi_{ename}(\sigma_{aname=\text{`}Boeing'}(Aircraft \bowtie Certified \bowtie Employees))$$

   - TRC

$\{E.ename \mid E \in Employees \wedge \exists C \in Certified$
$(\exists A \in Aircraft(A.aid = C.aid \wedge A.aname = \text{`}Boeing' \wedge E.eid = C.eid))\}$

   - DRC

$\{\langle EN\rangle \mid \langle Eid, EN, ES\rangle \in Employees \wedge$
$\exists Ceid, Caid(\langle Ceid, Caid\rangle \in Certified \wedge$
$\exists Aid, AN, AR(\langle Aid, AN, AR\rangle \in Aircraft \wedge$
$Aid = Caid \wedge AN = \text{`}Boeing' \wedge Eid = Ceid)\}$

   - SQL

```
SELECT E.ename
FROM   Aircraft A, Certified C, Employees E
WHERE  A.aid = C.aid AND A.aname = 'Boeing' AND E.eid = C.eid
```

3. ■ RA

   $\rho(BonnToMadrid, \sigma_{from=\text{`}Bonn' \wedge to=\text{`}Madrid'}(Flights))$
   $\pi_{aid}(\sigma_{cruisingrange>distance}(Aircraft \times BonnToMadrid))$

   ■ TRC

   $\{A.aid \mid A \in Aircraft \wedge \exists F \in Flights$
   $(F.from = \text{`}Bonn' \wedge F.to = \text{`}Madrid' \wedge A.cruisingrange > F.distance)\}$

   ■ DRC

   $\{Aid \mid \langle Aid, AN, AR\rangle \in Aircraft \wedge$
   $(\exists FN, FF, FT, FDi, FDe, FA(\langle FN, FF, FT, FDi, FDe, FA\rangle \in Flights \wedge$
   $FF = \text{`}Bonn' \wedge FT = \text{`}Madrid' \wedge FDi < AR))\}$

   ■ SQL

```
SELECT A.aid
FROM   Aircraft A, Flights F
WHERE  F.from = 'Bonn' AND F.to = 'Madrid' AND
       A.cruisingrange > F.distance
```

4. ■ RA

   $\pi_{flno}(\sigma_{distance<cruisingrange \wedge salary>100,000}(Flights \bowtie Aircraft \bowtie$
   $Certified \bowtie Employees)))$

   ■ TRC $\{F.flno \mid F \in Flights \wedge \exists A \in Aircraft \exists C \in Certified$
   $\exists E \in Employees(A.cruisingrange > F.distance \wedge E.salary > 100,000 \wedge$
   $A.aid = C.aid \wedge E.eid = C.eid)\}$

   ■ DRC

   $\{FN \mid \langle FN, FF, FT, FDi, FDe, FA\rangle \in Flights \wedge$
   $\exists Ceid, Caid(\langle Ceid, Caid\rangle \in Certified \wedge$
   $\exists Aid, AN, AR(\langle Aid, AN, AR\rangle \in Aircraft \wedge$
   $\exists Eid, EN, ES(\langle Eid, EN, ES\rangle \in Employees$
   $(AR > FDi \wedge ES > 100,000 \wedge Aid = Caid \wedge Eid = Ceid)\}$

   ■ SQL

```
SELECT E.ename
FROM   Aircraft A, Certified C, Employees E, Flights F
WHERE  A.aid = C.aid AND E.eid = C.eid AND
       distance < cruisingrange AND salary > 100,000
```

5. ■ RA $\rho(R1, \pi_{eid}(\sigma_{cruisingrange>3000}(Aircraft \bowtie Certified)))$
$\pi_{ename}(Employees \bowtie (R1 - \pi_{eid}(\sigma_{aname=\text{`}Boeing'}(Aircraft \bowtie Certified))))$

■ TRC
$\{E.ename \mid E \in Employees \wedge \exists C \in Certified(\exists A \in Aircraft$
$(A.aid = C.aid \wedge E.eid = C.eid \wedge A.cruisingrange > 3000))\wedge$
$\neg(\exists C2 \in Certified(\exists A2 \in Aircraft(A2.aname = \text{`}Boeing' \wedge C2.aid =$
$A2.aid \wedge C2.eid = E.eid)))\}$

■ DRC
$\{\langle EN \rangle \mid \langle Eid, EN, ES \rangle \in Employees \wedge$
$\exists Ceid, Caid(\langle Ceid, Caid \rangle \in Certified \wedge$
$\exists Aid, AN, AR(\langle Aid, AN, AR \rangle \in Aircraft \wedge$
$Aid = Caid \wedge Eid = Ceid \wedge AR > 3000))\wedge$
$\neg(\exists Aid2, AN2, AR2(\langle Aid2, AN2, AR2 \rangle \in Aircraft \wedge$
$\exists Ceid2, Caid2(\langle Ceid2, Caid2 \rangle \in Certified$
$\wedge Aid2 = Caid2 \wedge Eid = Ceid2 \wedge AN2 = \text{`}Boeing')))\}$

■ SQL

```
SELECT E.ename
FROM   Certified C, Employees E, Aircraft A
WHERE  A.aid = C.aid AND E.eid = C.eid AND A.cruisingrange > 3000
AND E.eid NOT IN ( SELECT C2.eid
FROM Certified C2, Aircraft A2
WHERE C2.aid = A2.aid AND A2.aname = 'Boeing' )
```

6. ■ RA

The approach to take is first find all the employees who do not have the highest salary. Subtract these from the original list of employees and what is left is the highest paid employees.

$\rho(E1, Employees)$
$\rho(E2, Employees)$
$\rho(E3, \pi_{E2.eid}(E1 \bowtie_{E1.salary>E2.salary} E2)$
$(\pi_{eid}E1) - E3$

■ TRC

$\{E1.eid \mid E1 \in Employees \wedge \neg(\exists E2 \in Employees(E2.salary > E1.salary))\}$

■ DRC

$\{\langle Eid1\rangle \mid \langle Eid1, EN1, ES1\rangle \in Employees \wedge$
$\neg(\exists Eid2, EN2, ES2(\langle Eid2, EN2, ES2\rangle \in Employees \wedge ES2 > ES1))\}$

- SQL

```
SELECT E.eid
FROM   Employees E
WHERE  E.salary = ( Select MAX (E2.salary)
                    FROM  Employees E2 )
```

7. - RA

The approach taken is similar to the solution for the previous exercise. First find all the employees who do not have the highest salary. Remove these from the original list of employees and what is left is the highest paid employees. Remove the highest paid employees from the original list. What is left is the second highest paid employees together with the rest of the employees. Then find the highest paid employees of this new list. This is the list of the second highest paid employees.
$\rho(E1, Employees)$
$\rho(E2, Employees)$
$\rho(E3, \pi_{E2.eid}(E1 \bowtie_{E1.salary>E2.salary} E2)$
$\rho(E4, E2 \bowtie E3)$
$\rho(E5, E2 \bowtie E3)$
$\rho(E6, \pi_{E5.eid}(E4 \bowtie_{E1.salary>E5.salary} E5)$
$(\pi_{eid}E3) - E6$

- TRC

$\{E1.eid \mid E1 \in Employees \wedge \exists E2 \in Employees(E2.salary > E1.salary$
$\wedge\neg(\exists E3 \in Employees(E3.salary > E2.salary)))\}$

- DRC

$\{\langle Eid1\rangle \mid \langle Eid1, EN1, ES1\rangle \in Employees \wedge$
$\exists Eid2, EN2, ES2(\langle Eid2, EN2, ES2\rangle \in Employees(ES2 > ES1)$
$\wedge\neg(\exists Eid3, EN3, ES3(\langle Eid3, EN3, ES3\rangle \in Employees(ES3 > ES2))))\}$

- SQL

```
SELECT E.eid
FROM   Employees E
WHERE  E.salary = (SELECT MAX (E2.salary)
                   FROM   Employees E2
                   WHERE  E2.salary ≠ (SELECT MAX (E3.salary)
                                       FROM   Employees E3 ))
```

8. This cannot be expressed in relational algebra (or calculus) because there is no operator to count, and this query requires the ability to count up to a number that depends on the data. The query can however be expressed in SQL as follows:

```
SELECT Temp.eid
FROM   ( SELECT   C.eid AS eid, COUNT (C.aid) AS cnt,
         FROM     Certified C
         GROUP BY C.eid) AS Temp
WHERE  Temp.cnt = ( SELECT   MAX (Temp.cnt)
                    FROM     Temp)
```

9. ■ RA

   The approach behind this query is to first find the employees who are certified for at least three aircraft (they appear at least three times in the Certified relation). Then find the employees who are certified for at least four aircraft. Subtract the second from the first and what is left is the employees who are certified for exactly three aircraft.

   $\rho(R1, Certified)$
   $\rho(R2, Certified)$
   $\rho(R3, Certified)$
   $\rho(R4, Certified)$
   $\rho(R5, \pi_{eid}(\sigma_{(R1.eid=R2.eid=R3.eid)\wedge(R1.aid\neq R2.aid\neq R3.aid)}(R1 \times R2 \times R3)))$
   $\rho(R6, \pi_{eid}(\sigma_{(R1.eid=R2.eid=R3.eid=R4.eid)\wedge(R1.aid\neq R2.aid\neq R3.aid\neq R4.aid)}$
   $(R1 \times R2 \times R3 \times R4)))$
   $R5 - R6$

   ■ TRC

   $\{C1.eid \mid C1 \in Certified \wedge \exists C2 \in Certified(\exists C3 \in Certified$
   $(C1.eid = C2.eid \wedge C2.eid = C3.eid \wedge$
   $C1.aid \neq C2.aid \wedge C2.aid \neq C3.aid \wedge C3.aid \neq C1.aid \wedge$
   $\neg(\exists C4 \in Certified$
   $(C3.eid = C4.eid \wedge C1.aid \neq C4.aid \wedge$
   $C2.aid \neq C4.aid \wedge C3.aid \neq C4.aid))))\}$

   ■ DRC

   $\{\langle CE1\rangle \mid \langle CE1, CA1\rangle \in Certified \wedge$
   $\exists CE2, CA2(\langle CE2, CA2\rangle \in Certified \wedge$
   $\exists CE3, CA3(\langle CE3, CA3\rangle \in Certified \wedge$
   $(CE1 = CE2 \wedge CE2 = CE3 \wedge$
   $CA1 \neq CA2 \wedge CA2 \neq CA3 \wedge CA3 \neq CA1 \wedge$
   $\neg(\exists CE4, CA4(\langle CE4, CA4\rangle \in Certified \wedge$

$(CE3 = CE4 \wedge CA1 \neq CA4 \wedge$
$CA2 \neq CA4 \wedge CA3 \neq CA4))))\}$

- SQL

```
SELECT C1.eid
FROM   Certified C1, Certified C2, Certified C3
WHERE  (C1.eid = C2.eid AND C2.eid = C3.eid AND
       C1.aid ≠ C2.aid AND C2.aid ≠ C3.aid AND C3.aid ≠ C1.aid)
EXCEPT
SELECT C4.eid
FROM   Certified C4, Certified C5, Certified C6, Certified C7,
WHERE  (C4.eid = C5.eid AND C5.eid = C6.eid AND C6.eid = C7.eid AND
       C4.aid ≠ C5.aid AND C4.aid ≠ C6.aid AND C4.aid ≠ C7.aid AND
       C5.aid ≠ C6.aid AND C5.aid ≠ C7.aid AND C6.aid ≠ C7.aid )
```

This could also be done in SQL using COUNT.

10. This cannot be expressed in relational algebra (or calculus) because there is no operator to sum values. The query can however be expressed in SQL as follows:

```
SELECT SUM (E.salaries)
FROM   Employees E
```

11. This cannot be expressed in relational algebra or relational calculus or SQL. The problem is that there is no restriction on the number of intermediate flights. All of the query methods could find if there was a flight directly from Madison to Timbuktu and if there was a sequence of two flights that started in Madison and ended in Timbuktu. They could even find a sequence of n flights that started in Madison and ended in Timbuktu as long as there is a static (i.e., data-independent) upper bound on the number of intermediate flights. (For large n, this would of course be long and impractical, but at least possible.) In this query, however, the upper bound is not static but dynamic (based upon the set of tuples in the Flights relation).

    In summary, if we had a static upper bound (say k), we could write an algebra or SQL query that repeatedly computes (upto k) joins on the Flights relation. If the upper bound is dynamic, then we cannot write such a query because k is not known when writing the query.

Exercise 4.6 What is *relational completeness*? If a query language is relationally complete, can you write any desired query in that language?

Answer 4.6 Answer omitted.

Exercise 4.7 What is an *unsafe* query? Give an example and explain why it is important to disallow such queries.

Answer 4.7 An $unsafe$ query is a query in relational calculus that has an infinite number of results. An example of such a query is:

$$\{S \mid \neg(S \in Sailors)\}$$

The query is for all things that are not sailors which of course is everything else. Clearly there is an infinite number of answers, and this query is $unsafe$. It is important to disallow $unsafe$ queries because we want to be able to get back to users with a list of all the answers to a query after a finite amount of time.

5

SQL: QUERIES, CONSTRAINTS, TRIGGERS

Online material is available for all exercises in this chapter on the book's webpage at

`http://www.cs.wisc.edu/~dbbook`

This includes scripts to create tables for each exercise for use with Oracle, IBM DB2, Microsoft SQL Server, Microsoft Access and MySQL.

Exercise 5.1 Consider the following relations:

> Student(*snum:* `integer`, *sname:* `string`, *major:* `string`, *level:* `string`, *age:* `integer`)
> Class(*name:* `string`, *meets_at:* `string`, *room:* `string`, *fid:* `integer`)
> Enrolled(*snum:* `integer`, *cname:* `string`)
> Faculty(*fid*: `integer`, *fname:* `string`, *deptid:* `integer`)

The meaning of these relations is straightforward; for example, Enrolled has one record per student-class pair such that the student is enrolled in the class.

Write the following queries in SQL. No duplicates should be printed in any of the answers.

1. Find the names of all Juniors (level = JR) who are enrolled in a class taught by I. Teach.
2. Find the age of the oldest student who is either a History major or enrolled in a course taught by I. Teach.
3. Find the names of all classes that either meet in room R128 or have five or more students enrolled.
4. Find the names of all students who are enrolled in two classes that meet at the same time.

5. Find the names of faculty members who teach in every room in which some class is taught.

6. Find the names of faculty members for whom the combined enrollment of the courses that they teach is less than five.

7. For each level, print the level and the average age of students for that level.

8. For all levels except JR, print the level and the average age of students for that level.

9. For each faculty member that has taught classes only in room R128, print the faculty member's name and the total number of classes she or he has taught.

10. Find the names of students enrolled in the maximum number of classes.

11. Find the names of students not enrolled in any class.

12. For each age value that appears in Students, find the level value that appears most often. For example, if there are more FR level students aged 18 than SR, JR, or SO students aged 18, you should print the pair (18, FR).

Answer 5.1 The answers are given below:

1.
```
SELECT DISTINCT S.Sname
FROM   Student S, Class C, Enrolled E, Faculty F
WHERE  S.snum = E.snum AND E.cname = C.name AND C.fid = F.fid AND
       F.fname = 'I.Teach' AND S.level = 'JR'
```

2.
```
SELECT MAX(S.age)
FROM   Student S
WHERE  (S.major = 'History')
       OR S.snum IN (SELECT E.snum
                     FROM   Class C, Enrolled E, Faculty F
                     WHERE  E.cname = C.name AND C.fid = F.fid
                            AND F.fname = 'I.Teach' )
```

3.
```
SELECT   C.name
FROM     Class C
WHERE    C.room = 'R128'
         OR C.name IN (SELECT   E.cname
                       FROM     Enrolled E
                       GROUP BY E.cname
                       HAVING   COUNT (*) >= 5)
```

4.
```
SELECT DISTINCT S.sname
FROM   Student S
WHERE  S.snum IN (SELECT E1.snum
                  FROM   Enrolled E1, Enrolled E2, Class C1, Class C2
                  WHERE  E1.snum = E2.snum AND E1.cname <> E2.cname
                  AND E1.cname = C1.name
                  AND E2.cname = C2.name AND C1.meets_at = C2.meets_at)
```

5.
```
SELECT DISTINCT F.fname
FROM   Faculty F
WHERE  NOT EXISTS (( SELECT *
                     FROM   Class C )
                     EXCEPT
                     (SELECT C1.room
                     FROM   Class C1
                     WHERE  C1.fid = F.fid ))
```

6.
```
SELECT   DISTINCT F.fname
FROM     Faculty F
WHERE    5 > (SELECT COUNT (E.snum)
              FROM   Class C, Enrolled E
              WHERE  C.name = E.cname
              AND    C.fid = F.fid)
```

7.
```
SELECT   S.level, AVG(S.age)
FROM     Student S
GROUP BY S.level
```

8.
```
SELECT   S.level, AVG(S.age)
FROM     Student S
WHERE    S.level <> 'JR'
GROUP BY S.level
```

9.
```
SELECT   F.fname, COUNT(*) AS CourseCount
FROM     Faculty F, Class C
WHERE    F.fid = C.fid
GROUP BY F.fid, F.fname
HAVING   EVERY ( C.room = 'R128' )
```

10.
```
SELECT   DISTINCT S.sname
FROM     Student S
WHERE    S.snum IN (SELECT   E.snum
                    FROM     Enrolled E
                    GROUP BY E.snum
```

```
                              HAVING   COUNT (*) >= ALL (SELECT   COUNT (*)
                                                         FROM     Enrolled E2
                                                         GROUP BY E2.snum ))
```

11.
```
SELECT DISTINCT S.sname
FROM   Student S
WHERE  S.snum NOT IN (SELECT E.snum
                      FROM   Enrolled E )
```

12.
```
SELECT   S.age, S.level
FROM     Student S
GROUP BY S.age, S.level,
HAVING   S.level IN (SELECT   S1.level
                     FROM     Student S1
                     WHERE    S1.age = S.age
                     GROUP BY S1.level, S1.age
                     HAVING   COUNT (*) >= ALL (SELECT   COUNT (*)
                                                FROM     Student S2
                                                WHERE s1.age = S2.age
                                                GROUP BY S2.level, S2.age))
```

Exercise 5.2 Consider the following schema:

Suppliers(*sid:* integer, *sname:* string, *address:* string)
Parts(*pid:* integer, *pname:* string, *color:* string)
Catalog(*sid:* integer, *pid:* integer, *cost:* real)

The Catalog relation lists the prices charged for parts by Suppliers. Write the following queries in SQL:

1. Find the *pname*s of parts for which there is some supplier.
2. Find the *sname*s of suppliers who supply every part.
3. Find the *sname*s of suppliers who supply every red part.
4. Find the *pname*s of parts supplied by Acme Widget Suppliers and no one else.
5. Find the *sid*s of suppliers who charge more for some part than the average cost of that part (averaged over all the suppliers who supply that part).
6. For each part, find the *sname* of the supplier who charges the most for that part.
7. Find the *sid*s of suppliers who supply only red parts.
8. Find the *sid*s of suppliers who supply a red part and a green part.

9. Find the *sid*s of suppliers who supply a red part or a green part.

10. For every supplier that only supplies green parts, print the name of the supplier and the total number of parts that she supplies.

11. For every supplier that supplies a green part and a red part, print the name and price of the most expensive part that she supplies.

Answer 5.2 Answer omitted.

Exercise 5.3 The following relations keep track of airline flight information:

> Flights(*flno:* `integer`, *from:* `string`, *to:* `string`, *distance:* `integer`,
> *departs:* `time`, *arrives:* `time`, *price:* `real`)
> Aircraft(*aid:* `integer`, *aname:* `string`, *cruisingrange:* `integer`)
> Certified(*eid:* `integer`, *aid:* `integer`)
> Employees(*eid:* `integer`, *ename:* `string`, *salary:* `integer`)

Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft, and only pilots are certified to fly. Write each of the following queries in SQL. (*Additional queries using the same schema are listed in the exercises for Chapter 4.*)

1. Find the names of aircraft such that all pilots certified to operate them have salaries more than $80,000.

2. For each pilot who is certified for more than three aircraft, find the *eid* and the maximum *cruisingrange* of the aircraft for which she or he is certified.

3. Find the names of pilots whose *salary* is less than the price of the cheapest route from Los Angeles to Honolulu.

4. For all aircraft with *cruisingrange* over 1000 miles, find the name of the aircraft and the average salary of all pilots certified for this aircraft.

5. Find the names of pilots certified for some Boeing aircraft.

6. Find the *aid*s of all aircraft that can be used on routes from Los Angeles to Chicago.

7. Identify the routes that can be piloted by every pilot who makes more than $100,000.

8. Print the *ename*s of pilots who can operate planes with *cruisingrange* greater than 3000 miles but are not certified on any Boeing aircraft.

9. A customer wants to travel from Madison to New York with no more than two changes of flight. List the choice of departure times from Madison if the customer wants to arrive in New York by 6 p.m.

10. Compute the difference between the average salary of a pilot and the average salary of all employees (including pilots).

11. Print the name and salary of every nonpilot whose salary is more than the average salary for pilots.

12. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles.

13. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles, but on at least two such aircrafts.

14. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles and who are certified on some Boeing aircraft.

Answer 5.3 The answers are given below:

1.
```
SELECT DISTINCT A.aname
FROM   Aircraft A
WHERE  A.Aid IN (SELECT C.aid
                 FROM   Certified C, Employees E
                 WHERE  C.eid = E.eid AND
                 NOT EXISTS ( SELECT *
                              FROM   Employees E1
                              WHERE  E1.eid = E.eid AND E1.salary < 80000 ))
```

2.
```
SELECT   C.eid, MAX (A.cruisingrange)
FROM     Certified C, Aircraft A
WHERE    C.aid = A.aid
GROUP BY C.eid
HAVING   COUNT (*) > 3
```

3.
```
SELECT DISTINCT E.ename
FROM   Employees E
WHERE  E.salary < ( SELECT MIN (F.price)
                    FROM   Flights F
                    WHERE  F.from = 'Los Angeles' AND F.to = 'Honolulu' )
```

4. Observe that *aid* is the key for Aircraft, but the question asks for aircraft names; we deal with this complication by using an intermediate relation Temp:

```
SELECT Temp.name, Temp.AvgSalary
FROM   ( SELECT   A.aid, A.aname AS name,
                  AVG (E.salary) AS AvgSalary
         FROM     Aircraft A, Certified C, Employees E
         WHERE    A.aid = C.aid AND
                  C.eid = E.eid AND A.cruisingrange > 1000
         GROUP BY A.aid, A.aname ) AS Temp
```

5.

```
SELECT DISTINCT E.ename
FROM   Employees E, Certified C, Aircraft A
WHERE  E.eid = C.eid AND
       C.aid = A.aid AND
       A.aname LIKE 'Boeing%'
```

6.

```
SELECT A.aid
FROM   Aircraft A
WHERE  A.cruisingrange > ( SELECT MIN (F.distance)
                           FROM   Flights F
                           WHERE  F.from = 'Los Angeles' AND F.to = 'Chicago' )
```

7.

```
SELECT DISTINCT F.from, F.to
FROM   Flights F
WHERE  NOT EXISTS ( SELECT *
                    FROM   Employees E
                    WHERE  E.salary > 100000
                    AND
                    NOT EXISTS (SELECT *
                                FROM   Aircraft A, Certified C
                                WHERE  A.cruisingrange > F.distance
                                AND E.eid = C.eid
                                AND A.aid = C.aid) )
```

8.

```
SELECT DISTINCT E.ename
FROM   Employees E
WHERE  E.eid IN ( ( SELECT C.eid
                   FROM   Certified C
                   WHERE  EXISTS ( SELECT A.aid
                                   FROM   Aircraft A
                                   WHERE  A.aid = C.aid
                                   AND    A.cruisingrange > 3000 )
                   AND
                   NOT EXISTS ( SELECT A1.aid
```

```
                                    FROM    Aircraft A1
                                    WHERE   A1.aid = C.aid
                                    AND     A1.aname LIKE 'Boeing%' ))
```

9.

```
SELECT F.departs
FROM   Flights F
WHERE  F.flno IN ( ( SELECT F0.flno
                     FROM   Flights F0
                     WHERE  F0.from = 'Madison' AND F0.to = 'New York'
                            AND F0.arrives < '18:00' )
                   UNION
                   ( SELECT F0.flno
                     FROM   Flights F0, Flights F1
                     WHERE  F0.from = 'Madison' AND F0.to <> 'New York'
                            AND F0.to = F1.from AND F1.to = 'New York'
                            AND F1.departs > F0.arrives
                            AND F1.arrives < '18:00' )
                   UNION
                   ( SELECT F0.flno
                     FROM   Flights F0, Flights F1, Flights F2
                     WHERE  F0.from = 'Madison'
                            AND F0.to = F1.from
                            AND F1.to = F2.from
                            AND F2.to = 'New York'
                            AND F0.to <> 'New York'
                            AND F1.to <> 'New York'
                            AND F1.departs > F0.arrives
                            AND F2.departs > F1.arrives
                            AND F2.arrives < '18:00' ))
```

10.

```
SELECT Temp1.avg - Temp2.avg
FROM   (SELECT AVG (E.salary) AS avg
        FROM   Employees E
        WHERE  E.eid IN (SELECT DISTINCT C.eid
                         FROM Certified C )) AS Temp1,
       (SELECT AVG (E1.salary) AS avg
        FROM   Employees E1 ) AS Temp2
```

11.

```
SELECT E.ename, E.salary
FROM   Employees E
WHERE  E.eid NOT IN ( SELECT DISTINCT C.eid
                      FROM   Certified C )
```

```
AND E.salary > ( SELECT AVG (E1.salary)
                 FROM   Employees E1
                 WHERE  E1.eid IN
                        ( SELECT DISTINCT C1.eid
                          FROM   Certified C1 ) )
```

12.

```
SELECT   E.ename
FROM     Employees E, Certified C, Aircraft A
WHERE    C.aid = A.aid AND E.eid = C.eid
GROUP BY E.eid, E.ename
HAVING   EVERY (A.cruisingrange > 1000)
```

13.

```
SELECT   E.ename
FROM     Employees E, Certified C, Aircraft A
WHERE    C.aid = A.aid AND E.eid = C.eid
GROUP BY E.eid, E.ename
HAVING   EVERY (A.cruisingrange > 1000) AND COUNT (*) > 1
```

14.

```
SELECT   E.ename
FROM     Employees E, Certified C, Aircraft A
WHERE    C.aid = A.aid AND E.eid = C.eid
GROUP BY E.eid, E.ename
HAVING   EVERY (A.cruisingrange > 1000) AND ANY (A.aname = 'Boeing')
```

Exercise 5.4 Consider the following relational schema. An employee can work in more than one department; the *pct_time* field of the Works relation shows the percentage of time that a given employee works in a given department.

Emp(<u>*eid:* `integer`</u>, *ename:* `string`, *age:* `integer`, *salary:* `real`)
Works(<u>*eid:* `integer`, *did:* `integer`</u>, *pct_time:* `integer`)
Dept(<u>*did:* `integer`</u>, *dname:* `string`, *budget:* `real`, *managerid:* `integer`)

Write the following queries in SQL:

1. Print the names and ages of each employee who works in both the Hardware department and the Software department.
2. For each department with more than 20 full-time-equivalent employees (i.e., where the part-time and full-time employees add up to at least that many full-time employees), print the *did* together with the number of employees that work in that department.
3. Print the name of each employee whose salary exceeds the budget of all of the departments that he or she works in.

sid	*sname*	*rating*	*age*
18	jones	3	30.0
41	jonah	6	56.0
22	ahab	7	44.0
63	moby	*null*	15.0

Figure 5.1 An Instance of Sailors

4. Find the *managerid*s of managers who manage only departments with budgets greater than $1 million.
5. Find the *ename*s of managers who manage the departments with the largest budgets.
6. If a manager manages more than one department, he or she *controls* the sum of all the budgets for those departments. Find the *managerid*s of managers who control more than $5 million.
7. Find the *managerid*s of managers who control the largest amounts.
8. Find the *ename*s of managers who manage only departments with budgets larger than $1 million, but at least one department with budget less than $5 million.

Answer 5.4 Answer omitted.

Exercise 5.5 Consider the instance of the Sailors relation shown in Figure 5.1.

1. Write SQL queries to compute the average rating, using `AVG`; the sum of the ratings, using `SUM`; and the number of ratings, using `COUNT`.
2. If you divide the sum just computed by the count, would the result be the same as the average? How would your answer change if these steps were carried out with respect to the *age* field instead of *rating*?
3. Consider the following query: *Find the names of sailors with a higher rating than all sailors with age* < 21. The following two SQL queries attempt to obtain the answer to this question. Do they both compute the result? If not, explain why. Under what conditions would they compute the same result?

```
SELECT S.sname
FROM   Sailors S
WHERE  NOT EXISTS ( SELECT *
                    FROM   Sailors S2
                    WHERE  S2.age < 21
                           AND S.rating <= S2.rating )
```

```
SELECT *
FROM   Sailors S
WHERE  S.rating > ANY ( SELECT S2.rating
                        FROM   Sailors S2
                        WHERE  S2.age < 21 )
```

4. Consider the instance of Sailors shown in Figure 5.1. Let us define instance S1 of Sailors to consist of the first two tuples, instance S2 to be the last two tuples, and S to be the given instance.

   (a) Show the left outer join of S with itself, with the join condition being *sid=sid*.

   (b) Show the right outer join of S with itself, with the join condition being *sid=sid*.

   (c) Show the full outer join of S with itself, with the join condition being *sid=sid*.

   (d) Show the left outer join of S1 with S2, with the join condition being *sid=sid*.

   (e) Show the right outer join of S1 with S2, with the join condition being *sid=sid*.

   (f) Show the full outer join of S1 with S2, with the join condition being *sid=sid*.

Answer 5.5 The answers are shown below:

1.

```
SELECT AVG (S.rating) AS AVERAGE
FROM   Sailors S

SELECT SUM (S.rating)
FROM   Sailors S

SELECT COUNT (S.rating)
FROM   Sailors S
```

2. The result using SUM and COUNT would be smaller than the result using AVERAGE if there are tuples with rating = NULL. This is because all the aggregate operators, except for COUNT, ignore NULL values. So the first approach would compute the average over all tuples while the second approach would compute the average over all tuples with non-NULL rating values. However, if the aggregation is done on the age field, the answers using both approaches would be the same since the age field does not take NULL values.
3. Only the first query is correct. The second query returns the names of sailors with a higher rating than *at least one* sailor with age < 21. Note that the answer to the second query does not necessarily contain the answer to the first query. In particular, if all the sailors are at least 21 years old, the second query will return an empty set while the first query will return all the sailors. This is because the NOT EXISTS predicate in the first query will evaluate to *true* if its subquery evaluates

4. (a)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
18	jones	3	30.0	18	jones	3	30.0
41	jonah	6	56.0	41	jonah	6	56.0
22	ahab	7	44.0	22	ahab	7	44.0
63	moby	*null*	15.0	63	moby	*null*	15.0

(b)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
18	jones	3	30.0	18	jones	3	30.0
41	jonah	6	56.0	41	jonah	6	56.0
22	ahab	7	44.0	22	ahab	7	44.0
63	moby	*null*	15.0	63	moby	*null*	15.0

(c)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
18	jones	3	30.0	18	jones	3	30.0
41	jonah	6	56.0	41	jonah	6	56.0
22	ahab	7	44.0	22	ahab	7	44.0
63	moby	*null*	15.0	63	moby	*null*	15.0

to an empty set, while the ANY predicate in the second query will evaluate to *false* if its subquery evaluates to an empty set. The two queries give the same results if and only if one of the following two conditions hold:

- The *Sailors* relation is empty, or
- There is at least one sailor with age > 21 in the *Sailors* relation, and for every sailor s, either s has a higher rating than all sailors under 21 or s has a rating no higher than all sailors under 21.

Exercise 5.6 Answer the following questions:

1. Explain the term *impedance mismatch* in the context of embedding SQL commands in a host language such as C.

(d)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
18	jones	3	30.0	*null*	*null*	*null*	*null*
41	jonah	6	56.0	*null*	*null*	*null*	*null*

(e)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
null	*null*	*null*	*null*	22	ahab	7	44.0
null	*null*	*null*	*null*	63	moby	*null*	15.0

(f)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
18	jones	3	30.0	*null*	*null*	*null*	*null*
41	jonah	6	56.0	*null*	*null*	*null*	*null*
null	*null*	*null*	*null*	22	ahab	7	44.0
null	*null*	*null*	*null*	63	moby	*null*	15.0

2. How can the value of a host language variable be passed to an embedded SQL command?
3. Explain the `WHENEVER` command's use in error and exception handling.
4. Explain the need for cursors.
5. Give an example of a situation that calls for the use of embedded SQL; that is, interactive use of SQL commands is not enough, and some host language capabilities are needed.
6. Write a C program with embedded SQL commands to address your example in the previous answer.
7. Write a C program with embedded SQL commands to find the standard deviation of sailors' ages.
8. Extend the previous program to find all sailors whose age is within one standard deviation of the average age of all sailors.
9. Explain how you would write a C program to compute the transitive closure of a graph, represented as an SQL relation Edges(*from, to*), using embedded SQL commands. (You need not write the program, just explain the main points to be dealt with.)
10. Explain the following terms with respect to cursors: *updatability, sensitivity*, and *scrollability.*
11. Define a cursor on the Sailors relation that is updatable, scrollable, and returns answers sorted by *age.* Which fields of Sailors can such a cursor *not* update? Why?
12. Give an example of a situation that calls for dynamic SQL; that is, even embedded SQL is not sufficient.

Answer 5.6 Answer omitted.

Exercise 5.7 Consider the following relational schema and briefly answer the questions that follow:

Emp(<u>*eid:* `integer`</u>, *ename:* `string`, *age:* `integer`, *salary:* `real`)
Works(<u>*eid:* `integer`, *did:* `integer`</u>, *pct_time:* `integer`)
Dept(<u>*did:* `integer`</u>, *budget:* `real`, *managerid:* `integer`)

1. Define a table constraint on Emp that will ensure that every employee makes at least $10,000.
2. Define a table constraint on Dept that will ensure that all managers have $age > 30$.
3. Define an assertion on Dept that will ensure that all managers have $age > 30$. Compare this assertion with the equivalent table constraint. Explain which is better.
4. Write SQL statements to delete all information about employees whose salaries exceed that of the manager of one or more departments that they work in. Be sure to ensure that all the relevant integrity constraints are satisfied after your updates.

Answer 5.7 The answers are given below:

1. Define a table constraint on Emp that will ensure that every employee makes at least $10,000

```
CREATE TABLE Emp ( eid     INTEGER,
                   ename   CHAR(10),
                   age     INTEGER ,
                   salary  REAL,
                   PRIMARY KEY (eid),
                   CHECK ( salary >= 10000 ))
```

2. Define a table constraint on Dept that will ensure that all managers have $age > 30$

```
CREATE TABLE Dept ( did        INTEGER,
                    buget      REAL,
                    managerid  INTEGER ,
                    PRIMARY KEY (did),
                    FOREIGN KEY (managerid) REFERENCES Emp,
                    CHECK (    ( SELECT E.age FROM Emp E, Dept D)
                               WHERE E.eid = D.managerid ) > 30 )
```

3. Define an assertion on Dept that will ensure that all managers have age > 30

```
CREATE TABLE Dept ( did        INTEGER,
                    budget     REAL,
                    managerid  INTEGER ,
                    PRIMARY KEY (did) )
```

```
CREATE ASSERTION managerAge
CHECK ((SELECT E.age
        FROM   Emp E, Dept D
        WHERE  E.eid = D.managerid ) > 30 )
```

Since the constraint involves two relations, it is better to define it as an assertion, independent of any one relation, rather than as a check condition on the Dept relation. The limitation of the latter approach is that the condition is checked only when the Dept relation is being updated. However, since age is an attribute of the Emp relation, it is possible to update the age of a manager which violates the constraint. So the former approach is better since it checks for potential violation of the assertion whenever one of the relations is updated.

4. To write such statements, it is necessary to consider the constraints defined over the tables. We will assume the following:

```
CREATE TABLE Emp (   eid        INTEGER,
                     ename      CHAR(10),
                     age        INTEGER,
                     salary     REAL,
                     PRIMARY KEY (eid) )

CREATE TABLE Works ( eid        INTEGER,
                     did        INTEGER,
                     pcttime    INTEGER,
                     PRIMARY KEY (eid, did),
                     FOREIGN KEY (did) REFERENCES Dept,
                     FOREIGN KEY (eid) REFERENCES Emp,
                     ON DELETE CASCADE)

CREATE TABLE Dept (  did        INTEGER,
                     buget      REAL,
                     managerid  INTEGER ,
                     PRIMARY KEY (did),
                     FOREIGN KEY (managerid) REFERENCES Emp,
                     ON DELETE SET NULL)
```

Now, we can define statements to delete employees who make more than one of their managers:

```
DELETE
FROM    Emp E
WHERE   E.eid IN ( SELECT W.eid
                   FROM   Work W, Emp E2, Dept D
                   WHERE  W.did = D.did
```

```
AND    D.managerid = E2.eid
AND    E.salary > E2.salary )
```

Exercise 5.8 Consider the following relations:

> Student(*snum:* `integer`, *sname:* `string`, *major:* `string`,
> *level:* `string`, *age:* `integer`)
> Class(*name:* `string`, *meets_at:* `time`, *room:* `string`, *fid:* `integer`)
> Enrolled(*snum:* `integer`, *cname:* `string`)
> Faculty(*fid:* `integer`, *fname:* `string`, *deptid:* `integer`)

The meaning of these relations is straightforward; for example, Enrolled has one record per student-class pair such that the student is enrolled in the class.

1. Write the SQL statements required to create these relations, including appropriate versions of all primary and foreign key integrity constraints.
2. Express each of the following integrity constraints in SQL unless it is implied by the primary and foreign key constraint; if so, explain how it is implied. If the constraint cannot be expressed in SQL, say so. For each constraint, state what operations (inserts, deletes, and updates on specific relations) must be monitored to enforce the constraint.
   (a) Every class has a minimum enrollment of 5 students and a maximum enrollment of 30 students.
   (b) At least one class meets in each room.
   (c) Every faculty member must teach at least two courses.
   (d) Only faculty in the department with *deptid=33* teach more than three courses.
   (e) Every student must be enrolled in the course called Math101.
   (f) The room in which the earliest scheduled class (i.e., the class with the smallest *meets_at* value) meets should not be the same as the room in which the latest scheduled class meets.
   (g) Two classes cannot meet in the same room at the same time.
   (h) The department with the most faculty members must have fewer than twice the number of faculty members in the department with the fewest faculty members.
   (i) No department can have more than 10 faculty members.
   (j) A student cannot add more than two courses at a time (i.e., in a single update).
   (k) The number of CS majors must be more than the number of Math majors.

(l) The number of distinct courses in which CS majors are enrolled is greater than the number of distinct courses in which Math majors are enrolled.

(m) The total enrollment in courses taught by faculty in the department with *deptid=33* is greater than the number of Math majors.

(n) There must be at least one CS major if there are any students whatsoever.

(o) Faculty members from different departments cannot teach in the same room.

Answer 5.8 Answer omitted.

Exercise 5.9 Discuss the strengths and weaknesses of the trigger mechanism. Contrast triggers with other integrity constraints supported by SQL.

Answer 5.9 A trigger is a procedure that is automatically invoked in response to a specified change to the database. The advantages of the trigger mechanism include the ability to perform an action based on the result of a query condition. The set of actions that can be taken is a superset of the actions that integrity constraints can take (i.e. report an error). Actions can include invoking new update, delete, or insert queries, perform data definition statements to create new tables or views, or alter security policies. Triggers can also be executed before or after a change is made to the database (that is, use old or new data).

There are also disadvantages to triggers. These include the added complexity when trying to match database modifications to trigger events. Also, integrity constraints are incorporated into database performance optimization; it is more difficult for a database to perform automatic optimization with triggers. If database consistency is the primary goal, then integrity constraints offer the same power as triggers. Integrity constraints are often easier to understand than triggers.

Exercise 5.10 Consider the following relational schema. An employee can work in more than one department; the *pct_time* field of the Works relation shows the percentage of time that a given employee works in a given department.

Emp(*eid:* `integer`, *ename:* `string`, *age:* `integer`, *salary:* `real`)
Works(*eid:* `integer`, *did:* `integer`, *pct_time:* `integer`)
Dept(*did:* `integer`, *budget:* `real`, *managerid:* `integer`)

Write SQL-92 integrity constraints (domain, key, foreign key, or `CHECK` constraints; or assertions) or SQL:1999 triggers to ensure each of the following requirements, considered independently.

1. Employees must make a minimum salary of $1000.

2. Every manager must be also be an employee.
3. The total percentage of all appointments for an employee must be under 100%.
4. A manager must always have a higher salary than any employee that he or she manages.
5. Whenever an employee is given a raise, the manager's salary must be increased to be at least as much.
6. Whenever an employee is given a raise, the manager's salary must be increased to be at least as much. Further, whenever an employee is given a raise, the department's budget must be increased to be greater than the sum of salaries of all employees in the department.

Answer 5.10 Answer omitted.

6

DATABASE APPLICATION DEVELOPMENT

Exercise 6.1 Briefly answer the following questions.

1. Explain the following terms: Cursor, Embedded SQL, JDBC, SQLJ, stored procedure.
2. What are the differences between JDBC and SQLJ? Why do they both exist?
3. Explain the term *stored procedure*, and give examples why stored procedures are useful.

Answer 6.1 The answers are given below:

1. A **cursor** enables individual row access of a relation by positioning itself at a row and reading its contents. **Embedded SQL** refers to the usage of SQL commands within a host program. **JDBC** stands for Java DataBase Connectivity and is an interface that allows a Java program to easily connect to any database system. **SQLJ** is a tool that allows SQL to be embedded directly into a Java program. A **stored procedure** is program that runs on the database server and can be called with a single SQL statement.
2. SQLJ provides embedded SQL statements. These SQL statements are static in nature and thus are preprocessed and precompiled. For instance, syntax checking and schema checking are done at compile time. JDBC allows dynamic queries that are checked at runtime. SQLJ is easier to use than JDBC and is often a better option for static queries. For dynamic queries, JDBC must still be used.
3. **Stored procedures** are programs that run on the database server and can be called with a single SQL statement. They are useful in situations where the processing should be done on the server side rather than the client side. Also, since the procedures are centralized to the server, code writing and maintenance is simplified, because the client programs do not have to duplicate the application logic. Stored procedures can also be used to reduce network communication; the results of a stored procedure can be analyzed and kept on the database server.