Solutions Manual

Chapter 10

Section 10.1

Exercise 10.1.1

(a) SELECT on MovieStar, SELECT on MovieExec.

(b) SELECT on MovieExec, SELECT on Movies, SELECT on StarsIn.

(c) SELECT on Movies, SELECT on Studio, INSERT on Studio (or INSERT(name) on Studio).

(d) DELETE on StarsIn.

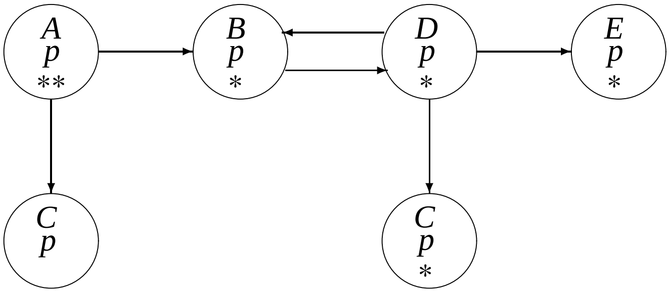
(e) UPDATE on MovieExec (or UPDATE(name) on MovieExec).

(f) REFERENCES on MovieStar (or REFERNCES(gender, name) on MovieStar).

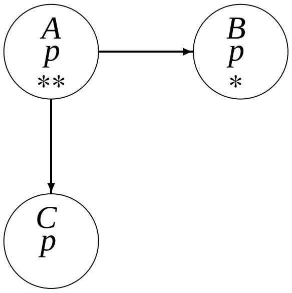
(g) REFERENCES on Studio, REFERENCES on MovieExec (or REFERENCES(name, presC#) on Studio, REFERENCES(cert#, netWorth) on MovieExec).

Exercise 10.1.2

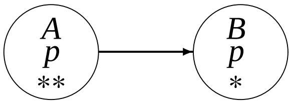
After step (4), the grant diagram is as follows:



After step (5), the grant diagram is as follows:

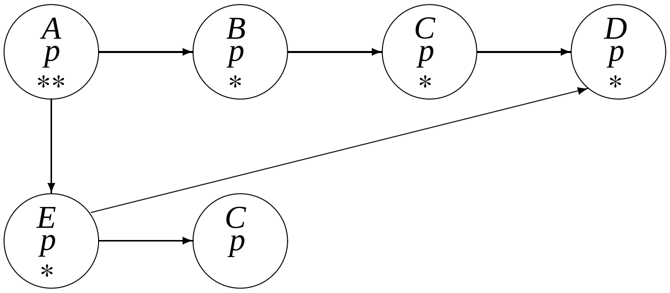


After step (6), the grant diagram is as follows:

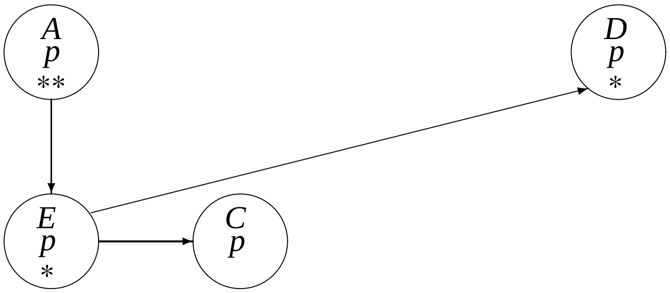


Exercise 10.1.3

After step (5), the grant diagram is as follows:

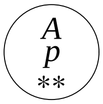


After step (6), the grant diagram is as follows:



Exercise 10.1.4

The grant diagram after the final step is as follows:



Section 10.2

Exercise 10.2.1

(a) The rules for trips that have reasonable connections are:

$$
\begin{aligned}
\mathtt{Trips}(x, y, \mathtt{dep}, \mathtt{arr}) &\leftarrow \mathtt{Flights}(\_, x, y, \mathtt{dep}, \mathtt{arr})\\
\mathtt{Trips}(x, y, \mathtt{dep}, \mathtt{arr}) &\leftarrow \mathtt{Trips}(x, z, \mathtt{dep1}, \mathtt{arr1})\ \mathrm{AND}\\
&\qquad \mathtt{Trips}(z, y, \mathtt{dep2}, \mathtt{arr2})\ \mathrm{AND}\\
&\qquad \mathtt{arr1} \leqslant \mathtt{dep2} - 100
\end{aligned}
$$

(b) Using the book's syntax, the SQL is:

```
WITH RECURSIVE Trips(frm, to, dep, arr) AS
  (SELECT frm, to, dep, arr
   FROM   Flights          )
  UNION
  (SELECT T.frm, F.to, T.dep, F.arr
   FROM   Trips T, Flights F
   WHERE  T.to = F.from
     AND  T.arr <= F.dep - 100     )
SELECT *
FROM   Trips;
```

Exercise 10.2.2

Because `FROM` is one of the SQL reserved words, using it as an identifier is not recommended. Note that most major vendors do not prohibit the use of reserved words when the use is not ambiguous (e.g. SELECT FROM FROM FROM is not ambiguous and will work), but such use is highly discouraged for readability and portability reasons.

Exercise 10.2.3

(a)

$$\begin{aligned}
\texttt{FollowOn}(\texttt{x},\texttt{y}) &\leftarrow \texttt{SequelOf}(\texttt{x},\texttt{y}) \\
\texttt{FollowOn}(\texttt{x},\texttt{y}) &\leftarrow \texttt{FollowOn}(\texttt{x},\texttt{z})\ \texttt{AND} \\
&\ \texttt{SequelOf}(\texttt{z},\texttt{y})
\end{aligned}$$

(b) Using the book's syntax, the SQL is:

```
WITH RECURSIVE FollowOn(movie, followOn) AS
  (SELECT movie, sequel
   FROM   SequelOf     )
  UNION
  (SELECT F.movie, S.sequel
   FROM   FollowOn F, Sequel S
   WHERE  F.followOn = S.movie)
SELECT *
FROM   FollowOn;
```

(c) Using the book's syntax, the SQL is:

```
WITH RECURSIVE FollowOn(movie, followOn) AS
  (SELECT movie, sequel
   FROM   SequelOf     )
  UNION
  (SELECT F.movie, S.sequel
   FROM   FollowOn F, Sequel S
   WHERE  F.followOn = S.movie)
SELECT movie, followOn
FROM   FollowOn
EXCEPT
SELECT movie, sequel
FROM   SequelOf;
```

(Similarly, NOT IN or NOT EXISTS can be used instead of EXCEPT).

(d) One of the ways is to first get all of the recursive tuples as for the original FollowOn in (a), and then subtract the those tuples that represent sequel or sequel of a sequel. Using the book's syntax, the SQL would be:

```
WITH RECURSIVE FollowOn(movie, followOn) AS
  (SELECT movie, sequel
   FROM   SequelOf     )
  UNION
  (SELECT F.movie, S.sequel
   FROM   FollowOn F, Sequel S
   WHERE  F.followOn = S.movie)
SELECT movie, followOn
FROM   FollowOn
EXCEPT
(SELECT movie, sequel
 FROM   SequelOf
 UNION
 SELECT X.movie, Y.sequel
 FROM   Sequel X, Sequel Y
 WHERE  X.sequel = Y.movie);
```

Another way would be to start FollowOn tuples only from the tuples of movies that have more than two sequels (using a join similar to the one above but with three Sequel tables).

(e) We simply need to count the number of followon values per movie. Using the book's syntax, the SQL would be:

```
WITH RECURSIVE FollowOn(movie, followOn) AS
  (SELECT movie, sequel
   FROM   SequelOf     )
  UNION
  (SELECT F.movie, S.sequel
   FROM   FollowOn F, Sequel S
   WHERE  F.followOn = S.movie)
SELECT movie
```

```
FROM   FollowOn
GROUP BY movie
HAVING COUNT(followon) >= 2;
```

(f) This is, in a sense, a reverse of (e) above, because to have at most one followon means that the total count of the tuples grouped by the given movie x must be no greater than 2 (one for the movie and its sequel, and the other for the sequel and its sequel). Using the book's syntax, the SQL would be:

```
WITH RECURSIVE FollowOn(movie, followOn) AS
  (SELECT movie, sequel
   FROM   SequelOf     )
  UNION
  (SELECT F.movie, S.sequel
   FROM   FollowOn F, Sequel S
   WHERE  F.followOn = S.movie)
SELECT movie, followon
FROM   FollowOn
WHERE  movie IN(SELECT movie
                FROM   FollowOn
                GROUP BY movie
                HAVING COUNT(followon) <= 2);
```

Exercise 10.2.4

(a)
```
WITH RECURSIVE Path(class, rclass) AS
  (SELECT class, rclass
   FROM   Rel          )
  UNION
  (SELECT Path.class, Rel.rclass
   FROM   Path, Rel
   WHERE  Path.rclass = Rel.class)
SELECT *
FROM   Path;
```

(b)
```
WITH RECURSIVE Path(class, rclass) AS
  (SELECT class, rclass
   FROM   Rel
   WHERE  mult = ’single’)
  UNION
  (SELECT Path.class, Rel.rclass
   FROM   Path, Rel
   WHERE  Path.rclass = Rel.class
     AND  Rel.mult = ’single’    )
SELECT *
FROM   Path;
```

(c)
```
WITH RECURSIVE Path(class, rclass) AS
 (SELECT class, rclass
  FROM   Rel
  WHERE  mult = ’multi’)
 UNION
 (SELECT Path.class, Rel.rclass
  FROM   Path, Rel
  WHERE  Path.rclass = Rel.class)
 UNION
 (SELECT Rel.class, Path.rclass
  FROM   Path, Rel
  WHERE  Rel.rclass = Path.class)
SELECT *
FROM   Path;
```

(d) This could be viewed as relation from (a) EXCEPT relation from (b).

```
WITH RECURSIVE PathAll(class, rclass) AS
  (SELECT class, rclass
   FROM   Rel          )
  UNION
  (SELECT PathAll.class, Rel.rclass
   FROM   PathAll, Rel
   WHERE  PathAll.rclass = Rel.class),
RECURSIVE PathSingle(class, rclass) AS
  (SELECT class, rclass
   FROM   Rel
   WHERE  mult = 'single')
  UNION
  (SELECT PathSingle.class, Rel.rclass
   FROM   PathSingle, Rel
   WHERE  PathSingle.rclass = Rel.class
     AND  Rel.mult = 'single'          )
SELECT class, rclass
FROM   PathAll
EXCEPT
SELECT class, rclass
FROM   PathSingle
;
```

(e) We include the edge label as part of the recursive relation and then, basically, we build the path as in (a) except we only add edges that have an opposite label.

```
WITH RECURSIVE Path(class, rclass, mult) AS
  (SELECT class, rclass, mult
   FROM   Rel                )
  UNION
  (SELECT Path.class, Rel.rclass, Rel.mult
   FROM   Path, Rel
   WHERE  Path.rclass  = Rel.class
     AND  Path.mult   <> Rel.mult )
SELECT *
FROM   Path;
```

(f)

```
WITH RECURSIVE Path(class, rclass) AS
  (SELECT class, rclass
   FROM   Rel
   WHERE  mult = 'single')
  UNION
  (SELECT Path.class, Rel.rclass
   FROM   Path, Rel
   WHERE  Path.rclass = Rel.class
     AND  Rel.mult = 'single'    )
```

```
SELECT *
FROM   Path X
WHERE  EXISTS(SELECT 1
              FROM   Path Y
              WHERE  Y.class  = X.rclass
                AND  Y.rclass = X.class )
;
```

Section 10.3

Exercise 10.3.1

(a) `Stars(name, address, birthdate)`
`Movies(title, year, length, stars({*Stars}))`

(b) `Stars(name, address, birthdate)`
`Movies(title, year, length, stars({*Stars}))`
`Studios(name, address, movies({*Movies}))`

(c) `Stars(name, address, birthdate)`
`Movies(title, year, length, studio(name, address), stars({*Stars}))`

Exercise 10.3.2

```
Customers(name, address, phone, ssNo, accts({*Accounts}))
Accounts(number, type, balance, owners({*Customers}))
```

Exercise 10.3.3

```
Customers(name, address, phone, ssNo, accts({*Accounts}))
Accounts(number, type, balance, owner(*Customers))
```

Exercise 10.3.4

```
Players(name)
Teams(name, players({*Players}), captain(*Players), colors)
Fans(name, fav_teams({*Teams}), fav_players({*Players}), fav_color)
```

Exercise 10.3.5

```
People(name, mother(*People), father(*People), children({*People}))
```

Section 10.4

Exercise 10.4.1

```
Movies(
 title        TitleType,
 year         YearType,
 length       DurationType,
 genre        GenreType,
 studioName   BusinessNameType,
 producerC#   CertificateType
)

MovieStar(
 name         PersonNameType,
 address      AddressType,
 gender       GenderType,
 birthdate    DateType
)

StarsIn(
 movieTitle    TitleType,
 movieYear     YearType,
 starName      PersonNameType
)

MovieExec(
 name          PersonNameType,
 address       AddressType,
 cert#         CertificateType,
 netWorth      CurrencyType
)

Studio(
 name          BusinessNameType,
 address       AddressType,
 presC#        CertificateType
)
```

Exercise 10.4.2

(a)
```
CREATE TYPE NameType AS(
    first   VARCHAR(30),
    middle  VARCHAR(50),
    last    VARCHAR(30),
    title   VARCHAR(10)
   );
```

(b)
```
CREATE TYPE PersonType AS(
    name    NameType,
    mother  REF(PersonType),
    father  REF(PersonType)
  );
```

(c)
```
CREATE TYPE MarriageType AS(
    date     DATE,
    husband  REF(PersonType),
    wife     REF(PersonType)
  );
```

Exercise 10.4.3

```
CREATE TYPE ProductType AS(
  maker       CHAR(5),
  model       INTEGER,
  type        CHAR(8)
);

CREATE TABLE Product OF ProductType(
  REF IS ProductId SYSTEM GENERATED
);
```

```
CREATE TABLE PC(
  model       REF(ProductType) SCOPE Product,
  speed       DECIMAL(5,2),
  ram         INTEGER,
  hd          INTEGER
  price       DECIMAL(10,2)
);

CREATE TABLE Laptop(
  model       REF(ProductType) SCOPE Product,
  speed       DECIMAL(5,2),
  ram         INTEGER,
  hd          INTEGER
  screen      DECIMAL(5,2),
  price       DECIMAL(10,2)
);

CREATE TABLE Printer(
  model       REF(ProductType) SCOPE Product,
  color       CHAR(1),
  type        VARCHAR(10),
  price       DECIMAL(10,2)
);
```

Exercise 10.4.4

Model attribute in Products cannot be a reference to the tuple in the relation for that type of product because that would create a circular reference situation where the model is a reference to the relation itself which has a model attribute but is a reference, etc. There would not be a column that stores the actual model values.

Exercise 10.4.5

```
CREATE TYPE ClassType AS (
  class       VARCHAR(30),
  type        CHAR(2),
  country     VACHAR(30),
  numGuns     INTEGER,
  bore        INTEGER,
  disp        INTEGER
);

CREATE TYPE ShipType AS (
  name        VARCHAR(30),
  class       REF(ClassType),
  launched    INTEGER
);

CREATE TYPE BattleType AS (
  name        VARCHAR(30),
  date        DATE
);

CREATE TYPE OutcomeType AS (
  ship        REF(ShipType),
  battle      REF(BattleType),
  result      VARCHAR(10)
);

CREATE TABLE Classes OF ClassType (
  REF IS classID SYSTEM GENERATED
);

CREATE TABLE Ships OF ShipType(
   REF IS shipID SYSTEM GENERATED
);

CREATE TABLE Battles OF TYPE BattleType(
   REF IS battleID SYSTEM GENERATED
);

CREATE TABLE Outcomes OF TYPE OutcomeType(
   REF IS outcomeID SYSTEM GENERATED
);
```

Section 10.5

Exercise 10.5.1

(a)
```
SELECT star->name
FROM   StarsIn
WHERE  movie->title = 'Dogma';
```

(b)
```
SELECT DISTINCT movie->title, movie->year
FROM   StarsIn
WHERE  star->address.city() = 'Malibu';
```

(c)
```
SELECT movie
FROM   StarsIn
WHERE  star->name = 'Melanie Griffith';
```

(d)
```
SELECT   movie->title, movie->year
FROM     StarsIn
GROUP BY movie->title, movie->year
HAVING   COUNT(*) >= 5;
```

Exercise 10.5.2

(a)
```
SELECT model->maker
FROM   PC
WHERE  hd > 60;
```

(b)
```
SELECT DISTINCT model->maker
FORM   Printers
WHERE  type = 'laser';
```

(c)
```
WITH MaxSpeedsPerMaker(maker, maxSpeed) AS(
  SELECT   model->maker, MAX(speed)
  FROM     Laptops
  GROUP BY model->maker                  ),
MakerTopModel(maker,topModel) AS(
  SELECT M.maker, L.model->model
  FROM   Laptops L, MaxSpeedsPerMaker M
  WHERE  L.model->maker = M.maker
    AND  L.speed        = maxSpeed     )
```

```
SELECT model->model, topModel
FROM   Laptops L, MakerTopModel M
WHERE  L.model->maker = M.maker
;
```

Exercise 10.5.3

(a)
```
SELECT x.name
FROM   Ships x
WHERE  x.class->disp > 35000;
```

(b)
```
SELECT DISTINCT x.battle->name
FROM   Outcomes x
WHERE  x.result = 'sunk';
```

(c)
```
SELECT DISTINCT x.class->class
FROM   Ships x
WHERE  x.launched > 1930;
```

(d)
```
SELECT DISTINCT x.battle->name
FROM   Outcomes x
WHERE  x.result = 'damaged'
  AND  x.ship->class->country = 'USA';
```

Exercise 10.5.4

```
CREATE FUNCTION StarLEG(p1 StarType,
                        p2 StarType )
RETURNS INTEGER
  IF     p1.name < p2.name THEN RETURN(-1)
  ELSEIF p1.name > p2.name THEN RETURN( 1)
  ELSE   RETURN(AddrLEG(p1.address,p2.addres))
  ENDIF
;
CREATE ORDERING FOR StarType
  ORDERING FULL BY RELATIVE WITH StarLEG;
```

Exercise 10.5.5

```
CREATE PROCEDURE DeleteStar(IN pName VARCHAR(50))
BEGIN
  DELETE FROM StarsIn
  WHERE  star->name = pName;

  DELETE FROM MovieStar x
  WHERE  x.name = pName;
END;
```

Section 10.6

Exercise 10.6.1

(a) Dimension attributes are: cust, date, proc, memory, hd, od.
Dependent attributes are: quant, price.

(b)
```
Cust(custID, name, address, phone, creditCard)
Proc(procID, manufacturer, name, model, speed)
HD(hdID, manufacturer, name, model, capacity,
   cylinders, surfaces, speed)
OD(odID, manufacturer, type, capacity, speed)
```

Exercise 10.6.2

First we could select the number of orders that had DVD disks and the number of orders that had CD disks. This would show just the totals over all orders.

```
SELECT   D1.type, COUNT(*)
FROM     Orders F, OD D1
WHERE    F.od = D1.odID
GROUP BY D1.type
HAVING   D1.type IN('DVD','CD')
;
```

Then we could drill-down to see what the totals are per month, hopefully seeing that the numbers for DVDs increase and the numbers for CDs decrease.

```
SELECT   MONTH(F.date) MONTHS, D1.type, COUNT(*)
FROM     Orders F, OD D1
WHERE    F.od = D1.odID
GROUP BY MONTHS, D1.type
HAVING   D1.type IN('DVD','CD')
;
```

Next we could drill-up to show the totals per year.

```
SELECT   YEAR(F.date) YEARS, D1.type, COUNT(*)
FROM     Orders F, OD D1
WHERE    F.od = D1.odID
GROUP BY YEARS, D1.type
HAVING   D1.type IN('DVD','CD')
;
```

Section 10.7

Exercise 10.7.1

(a) The ratio is $\left(\frac{11}{10}\right)^{10}$, or about 2.59.

(b) The ratio is $\left(\frac{3}{2}\right)^{10}$, or about 57.66.

Exercise 10.7.2

(a) Assuming the column name for SUM(val) in SalesCube is val:

```
SELECT   dealer, val
FROM     SalesCube
WHERE    model IS NULL
  AND    color = 'blue'
  AND    date IS NULL
  AND    dealer IS NOT NULL
;
```

(b) Assuming the column name for SUM(cnt) in SalesCube is cnt:

```
SELECT   cnt
FROM     SalesCube
WHERE    model = 'Gobi'
  AND    color = 'green'
  AND    date IS NULL
  AND    dealer = 'Smilin'' Sally'
;
```

(c) Assuming the column names for SUM(cnt) and SUM(val) in SalesCube are cnt and val:

```
SELECT   val/cnt
FROM     SalesCube
WHERE    model = 'Gobi'
  AND    color IS NULL
  AND    YEAR(date) = 2007
  AND    MONTH(date) = 3
  AND    dealer IS NOT NULL
;
```

Exercise 10.7.3

The rollup would not help and would make it more difficult to ensure that we do not double count the rows and only consider the rows that are in CUBE(Sales) but not in Sales.

Exercise 10.7.4

```
CREATE MATERIALIZED VIEW OrdersCube(
  cust, date, proc, memory, hd, od, tquant, tprice)
 AS(
  SELECT cust, date, proc, memory, hd, od, SUM(quant), SUM(price)
  FROM   Orders
  GROUP BY cust, date, proc, memory, hd, od)
WITH CUBE;
```

Exercise 10.7.5

(a)

```
SELECT D1.speed, MONTH(F.date), SUM(F.tquant)
FROM   OrdersCube F, Proc D1
WHERE  F.proc = D1.procID
  AND  F.cust IS NULL
  AND  YEAR(F.date) = 2007
  AND  F.memory IS NULL
  AND  F.hd IS NULL,
  AND  F.od IS NULL
GROUP BY D1.speed, MONTH(F.date)
;
```

(b)

```
SELECT D1.type, D2.type, SUM(F.tquant)
FROM   OrdersCube F, Proc D1, HD D2
WHERE  F.proc = D1.procID
  AND  F.hd   = D2.hdID
  AND  F.cust IS NULL
  AND  F.date IS NULL
  AND  F.memory IS NULL
  AND  F.od IS NULL
GROUP BY D1.type, D2.type
;
```

(c)

```
SELECT MONTH(F.date), SUM(tprice)/SUM(F.tquant)
FROM   OrdersCube F, Proc D1
WHERE  F.proc = D1.procID
  AND  D1.speed = 3.0
  AND  F.cust IS NULL
  AND  F.date >= '01/01/2005'
  AND  F.memory IS NULL
  AND  F.hd IS NULL,
  AND  F.od IS NULL
GROUP BY MONTH(F.date)
;
```

Exercise 10.7.6

Yes, other rollups could contain these tuples. Those rollups can be formed by rearranging the group by list so that columns we need to be aggregated are at the tail of the list. For instance, to include tuple

```
('Gobi', NULL, '2001-05-21', 'Friendly Fred', 152000, 7)
```

The group by list would be:

```
GROUP BY model, date, dealer, color WITH ROLLUP
```

Exercise 10.7.7

In the worst case, the fact table could have only one row, the CUBE(F) would add an additional 2^n tuples, and so the ratio would be 2^n.