Week 2

Theory 1

Relational algebra operators:

- Set operations:
  - Union: $R \cup S$, elms in R or S
  - Intersection: $R \cap S$, elms in R and S
  - Difference: $R - S$, elms in R but not in S
  - Need schemas with identical sets of attributes and domains. Order should be the same as well.
- Projection:
  - Removes some columns
  - Denoted $\pi_{a,b,c}(R)$
    - Gives columns a, b, c
- Selection:
  - Removes some rows
  - Gives subset of entries
  - Denoted $\sigma_C(R)$
    - Where C is some condition, e.g. $length \geq 100$
- Cartesian product
  - Aka. Cross product, aka product
  - Denoted $R \times S$
  - Set of pairs formed by choosing first in R second in S
  - Schema is the union of the schemas of R and S
    - If attribute names in common, we invent new names $R.A, S.A$
- Joins
  - Natural join:
    - Denoted $R \bowtie S$
    - Pair tuples that agree in whatever attributes common in their schemas
  - Theta join:
    - Denoted $R \bowtie_C S$
      - Where C is some condition
    - A join on some condition e.g. $R \bowtie_{A<D} S$
- Renaming
  - Renames both the relation and the attributes
  - Denoted $\rho_{S(a,b,c)}(R)$
    - Renames relation R to S with attributes a, b, c
- Linear notation:

$$\texttt{R(t,y,l,i,s,p)} := \sigma_{length \geq 100}(\texttt{Movies})$$
$$\texttt{S(t,y,l,i,s,p)} := \sigma_{studioName='\texttt{Fox}'}(\texttt{Movies})$$
$$\texttt{T(t,y,l,i,s,p)} := \texttt{R} \cap \texttt{S}$$
$$\texttt{Answer(title, year)} := \pi_{t,y}(\texttt{T})$$

Bags:

- "multiset": allow element to be present more than once
- How implemented in DBMS
- Some operations more efficient on bags
  - Union: just copy (no duplicate elimination)
  - Projection no duplicate elimination
- R, S are bags with t appearing n, m times respectively:
  - $R \cup S$: t appears $n + m$ times
  - $R \cup S$: t appears $\min(n, m)$ times
  - $R - S$: t appears $\max(0, n - m)$ times

SQL:

- Pattern matching: $s\ LIKE\ p$
  - Where p is some pattern
  - _ means a char
  - % means 0 or more chars
- Values could be $NULL$
  - Using operators means getting truth value $UNKNOWN$

x	y	x AND y	x OR y	NOT x
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

- Order output by using ORDER BY <list of attr>

Exercises for 3rd exercise session
4.1 (page 49-51)
4.2 (page 51). Draw the trees for at least three expressions from exercise 4, but you don't need to draw them all.

4.5, 4.6, 4.7 (page 53-54)
4.8 (page 54). One expression is enough
4.9 (page 54)

1.2 (page 207), 1.2 (page 250-251), 1.3 (page 252), 2.1 (page 261)

Chapter 2

4.1

Exercise 2.4.1: This exercise builds upon the products schema of Exercise 2.3.1. Recall that the database schema consists of four relations, whose schemas are:

```
Product(maker, model, type)
PC(model, speed, ram, hd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

Some sample data for the relation `Product` is shown in Fig. 2.20. Sample data for the other three relations is shown in Fig. 2.21. Manufacturers and model numbers have been "sanitized," but the data is typical of products on sale at the beginning of 2007.

Write expressions of relational algebra to answer the following queries. You may use the linear notation of Section 2.4.13 if you wish. For the data of Figs. 2.20 and 2.21, show the result of your query. However, your answer should work for arbitrary data, not just the data of these figures.

a) What PC models have a speed of at least 3.00?

b) Which manufacturers make laptops with a hard disk of at least 100GB?

c) Find the model number and price of all products (of any type) made by manufacturer B.

d) Find the model numbers of all color laser printers.

e) Find those manufacturers that sell Laptops, but not PC's.

! f) Find those hard-disk sizes that occur in two or more PC's.

! g) Find those pairs of PC models that have both the same speed and RAM. A pair should be listed only once; e.g., list (i, j) but not (j, i).

!! h) Find those manufacturers of at least two different computers (PC's or laptops) with speeds of at least 2.80.

!! i) Find the manufacturer(s) of the computer (PC or laptop) with the highest available speed.

!! j) Find the manufacturers of PC's with at least three different speeds.

!! k) Find the manufacturers who sell exactly three different models of PC.

a. $\pi_{model}\left(\sigma_{speed\geq 3.00}(PC)\right)$

b. $\pi_{maker}\left((Product) \bowtie \left(\sigma_{hd\leq 100}(Laptop)\right)\right)$

c. $\pi_{model,price}\big((\sigma_{maker=B}(Product) \bowtie PC) \cup (\sigma_{maker=B}(Product) \bowtie Laptop) \cup (\sigma_{maker=B}(Printer) \bowtie PC)\big)$

d. $\pi_{model}\left(\sigma_{type=laser\ AND\ color=True}(Printer)\right)$

e. $\pi_{maker}\left(\sigma_{type=laptop}(Product)\right) - \pi_{maker}\left(\sigma_{type=pc}(Product)\right)$

f. $\pi_{hd}\big(\rho_{PC2(m,s,r,h,p)}(PC) \bowtie_{pc.hd=pc2.hd\ AND\ pc.model<>pc2.model} PC\big)$

g. $\pi_{pc1.maker,pc2.maker}\big(PC1 \bowtie_{PC1.speed=pc2.speed\ AND\ pc1.RAM=pc2.RAM\ AND\ pc1.model<pc2.model} PC2\big)$

h.

$$R1 \coloneqq \left(\sigma_{speed\geq 2.8}(PC) \cup \sigma_{speed\geq 2.8}(Laptop)\right) \bowtie Product$$

$$R2 \coloneqq \rho(R1)$$

$$\pi_{maker}(R1 \bowtie_{R1.maker=R2.maker\ AND\ R1.model<R2.model} R2)$$

i.

$$R1 \coloneqq \pi_{model,speed}(PC) \cup \pi_{model,speed}(Laptop)$$

$$R2 \coloneqq \rho_{R2(model2,speed2)}(R1)$$

Finding all speeds which is not the largest:

$$R3 \coloneqq \pi_{model,speed}\big(R1 \bowtie_{speed<speed2} R2\big)$$

$$R4 := R1 - R3$$

$$R5 := \pi_{maker}(R4 \bowtie Product)$$

j. …

$$R1 := \pi_{maker,speed}(PC \bowtie Product)$$

$$R2 := \rho_{R2(maker2,speed2)}(R1)$$

$$R3 := \rho_{R3(maker3,speed3)}(R1)$$

$$R4 := R1 \bowtie_{speed<>speed2\ AND\ maker=maker2} R2$$

$$R5 = \pi_{maker}\left(R4 \bowtie_{maker=maker3\ AND\ speed<>speed3\ AND\ speed2<>speed3} R3\right)$$

k. …

$$R1 := \pi_{maker,model}\left(\sigma_{type=pc}(Product)\right)$$

$$R2 := \rho_{R2(maker2,model2)}R1$$

$$R3 := \rho_{R3(maker3,model3)}R1$$

$$R4 := \rho_{R2(maker4,model4)}R1$$

$$R5 := R1 \bowtie_{maker=maker2\ AND\ model<>model2} R2$$

$$R6 := R5 \bowtie_{maker=maker3\ AND\ model<>model3\ AND\ model2<>model3} R3$$

$$R7 := R4 \bowtie_{maker=maker4\ AND\ (model4=model\ OR\ model4=model2\ OR\ model4=model3)} R6$$

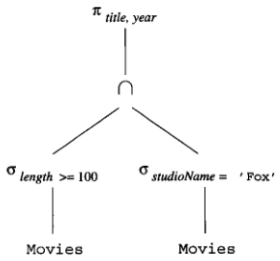
$$R8 := \pi_{maker}(R7)$$

4.2

Exercise 2.4.2: Draw expression trees for each of your expressions of Exercise 2.4.1.

Three trees are enough. Made on blackboard.

Trees like this, basically just splitting up the statement.



4.5

Exercise 2.4.5: What is the difference between the natural join $R \bowtie S$ and the theta-join $R \bowtie_C S$ where the condition C is that $R.A = S.A$ for each attribute A appearing in the schemas of both R and S?

Difference is: theta join columns would appear twice, not in natural. Merging vs not merging columns.

4.6

! **Exercise 2.4.6:** An operator on relations is said to be *monotone* if whenever we add a tuple to one of its arguments, the result contains all the tuples that it contained before adding the tuple, plus perhaps more tuples. Which of the operators described in this section are monotone? For each, either explain why it is monotone or give an example showing it is not.

$\pi, \sigma, \bowtie, \cup, \cap$

This is not:

_

4.7

! **Exercise 2.4.7:** Suppose relations R and S have n tuples and m tuples, respectively. Give the minimum and maximum numbers of tuples that the results of the following expressions can have.

a) $R \cup S$.

b) $R \bowtie S$.

c) $\sigma_C(R) \times S$, for some condition C.

d) $\pi_L(R) - S$, for some list of attributes L.

a. $min = \max(m, n), max = n + m$

Min if all tuples in one appears in other

Max if sets disjoint.

b. $min = 0, max = m \cdot n$

Max if all matches all.

c. $min = 0, max = m \cdot n$
d. $min = \max(n - m, 0), max = n$

4.8

! **Exercise 2.4.8:** The *semijoin* of relations R and S, written $R \ltimes S$, is the set of tuples t in R such that there is at least one tuple in S that agrees with t in all attributes that R and S have in common. Give three different expressions of relational algebra that are equivalent to $R \ltimes S$.

One expression is enough

$$\pi_R(R \bowtie S)$$

4.9

! **Exercise 2.4.9:** The *antisemijoin* $R \overline{\ltimes} S$ is the set of tuples t in R that do *not* agree with any tuple of S in the attributes common to R and S. Give an expression of relational algebra equivalent to $R \overline{\ltimes} S$.

$$R - \left(\pi_R(R \bowtie S)\right)$$

Chapter 5

1.2

!! **Exercise 5.1.5:** The following algebraic laws hold for sets but not for bags. Explain why they hold for sets and give counterexamples to show that they do not hold for bags.

a) $(R \cap S) - T = R \cap (S - T)$.

b) The distributive law of intersection over union:

$$R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$$

c) $\sigma_{C \text{ OR } D}(R) = \sigma_C(R) \cup \sigma_D(R)$. Here, C and D are arbitrary conditions about the tuples of R.

a. …

For sets:

- Taking intersection subtracting T, same as subtracting from S and taking the intersection
- Does not matter if we take intersection and subtract or other order.

For bags:

- Lhs: something may be apparent more than once in the intersection and subtracted once, meaning seen once totally.

- Rhs: will not have this elm at all due to subtracting elm from S, meaning will not be a part of the intersection with R.

Numbers represent # times some tuple appears in the different relations. $R = 2, S = 3, T = 1$

$$(R \cap S) - T = R \cap (S - T)$$

$$(2 \cap 3) - 1 = 2 \cap (3 - 1)$$

$$2 - 1 = 2 \cap 2$$

$$1 = 2$$

b. …

For sets:

- Intersection with the union same as union of the intersections

For bags:

- If something in all relations, due to allowing duplicates it would appear once on lhs, and twice on rhs.

Numbers represent # times some tuple appears in the different relations. $R = 5, S = 4, T = 4$

$$R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$$

$$5 \cap (4 \cup 4) = (5 \cap 4) \cup (5 \cap 4)$$

$$5 \cap 8 = 4 \cup 4$$

$$5 = 8$$

c. …

For sets:

- Just one of the conditions should hold, then in the result set. This is the same as the union due to duplicates being eliminated

For bags:

- Duplicates not being eliminated, meaning that if both conditions hold, some tuple would be once on lhs and twice on rhs.

R(age, height):

- (35, 170)
- (30, 180)

$$\sigma_{age<40} R \; OR \; \sigma_{height<190} \; R = \sigma_{age<40} R \cup \sigma_{height<190} \; R$$

$$|2| = |4|$$

Chapter 6

1.2

Exercise 6.1.2: Write the following queries, based on our running movie database example

```
Movies(title, year, length, genre, studioName, producerC#)
StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)
```

in SQL.

a) Find the address of MGM studios.

b) Find Sandra Bullock's birthdate.

c) Find all the stars that appeared either in a movie made in 1980 or a movie with "Love" in the title.

d) Find all executives worth at least $10,000,000.

e) Find all the stars who either are male or live in Malibu (have string `Malibu` as a part of their address).

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

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

SELECT starName
FROM Movies, StarsIn
WHERE year = 1980 OR title LIKE '%Love%';

SELECT name
FROM MovieExec
WHERE netWorth >= 10000000;

SELECT name
FROM MovieStar
WHERE gender = 'male' OR address = 'Malibu';
```

1.3

Exercise 6.1.5: Let a and b be integer-valued attributes that may be `NULL` in some tuples. For each of the following conditions (as may appear in a `WHERE` clause), describe exactly the set of (a, b) tuples that satisfy the condition, including the case where a and/or b is `NULL`.

a) `a = 10 OR b = 20`

b) `a = 10 AND b = 20`

c) `a < 10 OR a >= 10`

! d) `a = b`

! e) `a <= b`

a. …

$a = 10$	OR	$b = 20$
$True$		$True$
$True$		$False$
$False$		$True$
$True$		$Null$

b. …

$a = 10$	AND	$b = 20$
$True$		$True$

c. …

$a < 10$	OR	$a \geq 10$
$True$		$False$
$False$		$True$

d. …

a	$=$	b
$NOT\ NULL$	$when\ a = b$	$NOT\ NULL$

e. …

a	$\leq$	b
$NOT\ NULL$	$when\ a \leq b$	$NOT\ NULL$

2.1

Exercise 6.2.1: Using the database schema of our running movie example

```
Movies(title, year, length, genre, studioName, producerC#)
StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)
```

write the following queries in SQL.

a) Who were the male stars in *Titanic*?

b) Which stars appeared in movies produced by MGM in 1995?

c) Who is the president of MGM studios?

! d) Which movies are longer than *Gone With the Wind*?

! e) Which executives are worth more than Merv Griffin?

```sql
SELECT starname
FROM starsin, moviestar
WHERE movietitle = 'titanic' AND gender = 'male' AND starname = name;
```

```sql
SELECT starName
FROM StarsIn, Movies
WHERE year = 1995 AND studioName = 'MGM' AND title = movieTitle;

SELECT MovieExec.name
FROM Studio, MovieExec
WHERE presidentCert = cert AND Studio.name = 'MGM';

SELECT title
FROM Movies
WHERE length > (SELECT length
                FROM Movies
                WHERE title = 'Gone With the Wind');

SELECT name
FROM MovieExec
WHERE netWorth > (SELECT netWorth
                  FROM MovieExec
                  WHERE name = 'Merv Griffin');
```

Exercises for 4th exercise session
2.1, 2.2, 2.3 (page 216)
3.1, 3.3, 3.4, 3.5, 3.6, 3.7 (page 273-274)
4.1, 4.2, 4.3 (page 283-284)

Theory 2

Extended operators of relational alg:

- Duplicate-elimination:
  - Turns bag to set
  - Denoted δ
- Aggregation:
  - Possibly used by grouping operator
  - Applied to attributes (columns)
  - Sum, AVG, MIN, MAX, COUNT
- Grouping:
  - Groups tuples according to value in one or more attributes
  - Aggregation can be applied to columns with group
  - Denoted γ
- Extended projection:
  - π with extra features
  - perform computations involving columns possibly producing new columns
  - $\rightarrow$ is a renaming
- Sorting:
  - Denoted τ
  - Turns relation into list of tuples sorted according to one or more attributes
- Outer join:
  - Variant of join that avoids losing dangling tuples
    - Dangling: tuple not matching other relation in common attributes
  - Dangling tuples padded with null
  - Denoted $\bowtie$ with a circle above
  - Variants with subscript L or R, only adding dangling tuples from left or right side.

SQL Joins:

- CROSS JOIN: cartesian product
- JOIN … ON: cartesian product followed by selection on condition
- Natural Join: exactly as $\bowtie$, joining on attributes in common
- Outer Join: padding dangling tuples with null, either full, left or right.

Chapter 5

2.1

Exercise 5.2.1: Here are two relations:

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

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

Compute the following: a) $\pi_{A+B,A^2,B^2}(R)$; b) $\pi_{B+1,C-1}(S)$; c) $\tau_{B,A}(R)$; d) $\tau_{B,C}(S)$; e) $\delta(R)$; f) $\delta(S)$; g) $\gamma_{A,\,\mathtt{SUM}(B)}(R)$; h) $\gamma_{B,\mathtt{AVG}(C)}(S)$; ! i) $\gamma_A(R)$; ! j) $\gamma_{A,\mathtt{MAX}(C)}(R \bowtie S)$; k) $R \overset{\circ}{\bowtie}_L S$; l) $R \overset{\circ}{\bowtie}_R S$; m) $R \overset{\circ}{\bowtie} S$; n) $R \overset{\circ}{\bowtie}_{R.B<S.B} S$.

a. …

A	B	$A+B$	A^2	B^2
0	1	1	0	1
2	3	5	2	9
0	1	1	0	1
0	1	1	0	1
2	4	6	4	16
3	4	7	9	16

b. …

B	C	$B+1$	$C-1$
0	1	1	0
2	4	3	3
2	5	3	4
3	4	4	3
0	2	1	1
3	4	4	3

c. … sorting

B	A
1	0
1	0
1	0
3	2
4	2
4	3

d. … sorting

B	***C***
0	1
0	2
2	4
2	5
3	4
3	4

e. … duplicate elimination

A	***B***
0	1
2	3
2	4
3	4

f. …

B	***C***
0	1
2	4
2	5
3	4
0	2
3	4

g. … group A sum B

A	***B***
0	3
2	7
3	4

h. …group B AVG C

B	***C***
0	1.5
2	4.5
3	4

i. … Will return relation with one column, meaning only the groups.

A
0
2
3

j. … group A max C

After join:

A	***B***	***C***
2	3	4
2	3	4

Grouping:

A	***B***	***C***
2	3	4

k. … left outer

A	***B***	**C**
0	1	⊥
2	3	4
2	3	4
0	1	⊥
0	1	⊥
2	4	⊥
3	4	⊥

B	***C***
0	1
2	4
2	5
3	4
0	2
3	4

l. … right outer

A	***B***	**C**
2	3	4
2	3	4
⊥	0	1
⊥	2	4
⊥	2	5
⊥	0	2

B	***C***
0	1
2	4
2	5
3	4
0	2
3	4

m. … outer

A	***B***	**C**
2	3	4
2	3	4
⊥	0	1
⊥	2	4
⊥	2	5
⊥	0	2
0	1	⊥
0	1	⊥
0	1	⊥
2	4	⊥
3	4	⊥

n. …outer with cond R.B < S.B

A	***B***
0	1
0	1
0	1
2	3
2	4
3	4

B	***C***
0	1
0	2
2	4
2	5
3	4
3	4

A	***R.B***	***S.B***	**C**
0	1	2	4
0	1	2	5
0	1	2	4

0	1	2	5
0	1	3	4
0	1	3	4
2	3	⊥	⊥
2	4	⊥	⊥
3	4	⊥	⊥
⊥	⊥	0	1
⊥	⊥	0	2

2.2

! **Exercise 5.2.2:** A unary operator f is said to be *idempotent* if for all relations R, $f(f(R)) = f(R)$. That is, applying f more than once is the same as applying it once. Which of the following operators are idempotent? Either explain why or give a counterexample.

a) δ; b) π_L; c) σ_C; d) γ_L; e) τ.

a. Yep, if duplicates are eliminated nothing happens
b. Projection on some condition, condition still holds so nothing happens
c. Selection on some condition, condition still holds so nothing happens
d. Grouping on some condition, nothing happens
e. We can sort twice, but nothing will happen.

2.3

! **Exercise 5.2.3:** One thing that can be done with an extended projection, but not with the original version of projection that we defined in Section 2.4.5, is to duplicate columns. For example, if $R(A, B)$ is a relation, then $\pi_{A,A}(R)$ produces the tuple (a, a) for every tuple (a, b) in R. Can this operation be done using only the classical operations of relation algebra from Section 2.4? Explain your reasoning.

We can join (theta join on equality) with itself, but if duplicates, multiple will be generated.

E.g. Do example A(1,2)

Chapter 6

3.1

! **Exercise 6.3.3:** Write the query of Fig. 6.10 without any subqueries.

```
1)  SELECT title
2)  FROM Movies Old
3)  WHERE year < ANY
4)       (SELECT year
5)        FROM Movies
6)        WHERE title = Old.title
          );
```

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

3.3

! Exercise 6.3.5: Write the following queries without using the intersection or difference operators:

a) The intersection query of Fig. 6.5.

b) The difference query of Example 6.17.

```
1)  (SELECT name, address
2)   FROM MovieStar
3)   WHERE gender = 'F')
4)        INTERSECT
5)  (SELECT name, address
6)   FROM MovieExec
7)   WHERE netWorth > 10000000);
```

a. …

```
SELECT
  name,
  address
FROM MovieStar
WHERE gender = 'F' AND (name, address) IN (SELECT
                                             name,
                                             address
                                           FROM MovieExec
                                           WHERE netWorth > 1000000);
```

b. …

```
(SELECT name, address FROM MovieStar)
    EXCEPT
(SELECT name, address FROM MovieExec);
```

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

3.4

!! Exercise 6.3.6: We have noticed that certain operators of SQL are redundant, in the sense that they always can be replaced by other operators. For example, we saw that *s* `IN` *R* can be replaced by *s* `= ANY` *R*. Show that `EXISTS` and `NOT EXISTS` are redundant by explaining how to replace any expression of the form `EXISTS` *R* or `NOT EXISTS` *R* by an expression that does not involve `EXISTS` (except perhaps in the expression *R* itself). *Hint*: Remember that it is permissible to have a constant in the `SELECT` clause.

Say we have the expression

“where Exists (R)”

We can replace it with

“where 1=(select 1 from R limit 1)”

Explanation: select 1 from R returns a column with as many entries as in R but with a 1 in each entry (try for yourself in for example elephantsql).

By adding 'limit 1', we say that we only want the top entry. Thus, select 1 from R limit 1 gives us one row containing the number 1 if R is not empty,

and an empty table if R is empty.

3.5

Exercise 3.5 : For these relations from our running movie database schema

```
StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)
```

describe the tuples that would appear in the following SQL expressions:

a) `Studio CROSS JOIN MovieExec;`

b) `StarsIn NATURAL FULL OUTER JOIN MovieStar;`

c) `StarsIn FULL OUTER JOIN MovieStar ON name = starName;`

a. …

We get the attributes:

$studio.name, studio.adress, presC\#, MovieExec.name, MovieExec.adress, cert\#, networth$

Get all combinations of tuples. (Cartesian product)

b. …

We get the attributes:

$$movieTitle, movieYear, starName, name, address, gender, bithdate$$

Dangling tuples from both relations are padded.

c. …

We get the attributes:

$$movieTitle, movieYear, starname, name, adress, birthday, gender$$

We join on a condition, padding all dangling tuples with null.

3.6

! **Exercise 3.6 :** Using the database schema

```
Product(maker, model, type)
PC(model, speed, ram, hd, rd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

write a SQL query that will produce information about all products — PC's, laptops, and printers — including their manufacturer if available, and whatever information about that product is relevant (i.e., found in the relation for that type of product).

```sql
SELECT *
FROM
  (SELECT *
   FROM product
     NATURAL FULL OUTER JOIN pc
   WHERE type = 'pc'
  ) AS p
  NATURAL FULL OUTER JOIN
  (SELECT
     product.maker,
     product.model,
     product.type,
     printer.color,
     printer.type AS printer_type,
     printer.price
   FROM product
     FULL OUTER JOIN printer ON printer.model = product.model
   WHERE product.type = 'printer') AS p2
  NATURAL FULL OUTER JOIN
  (SELECT *
   FROM product
     NATURAL FULL OUTER JOIN laptop
   WHERE type = 'laptop') AS p3;
```

3.7

! **Exercise 3.7:** The join operators (other than outerjoin) we learned in this section are redundant, in the sense that they can always be replaced by select-from-where expressions. Explain how to write expressions of the following forms using select-from-where:

a) `R CROSS JOIN S;`

b) `R NATURAL JOIN S;`

c) `R JOIN S ON` C`;`, where C is a SQL condition.

a. Select .. from R, S.
b. Select .. from R, S where R.A = S.A (for all attributes in common)
c. Select .. from R, S where C

4.1

! **Exercise 6.4.3:** For each of your answers to Exercise 6.3.1, determine whether or not the result of your query can have duplicates. If so, rewrite the query to eliminate duplicates. If not, write a query without subqueries that has the same, duplicate-free answer.

With distinct keyword no problem otherwise yes.

4.2

! **Exercise 6.4.5:** In Example 6.27, we mentioned that different versions of the query "find the producers of Harrison Ford's movies" can have different answers as bags, even though they yield the same set of answers. Consider the version of the query in Example 6.22, where we used a subquery in the `FROM` clause. Does this version produce duplicates, and if so, why?

```
1)  SELECT name
2)  FROM MovieExec, (SELECT producerC#
3)                   FROM Movies, StarsIn
4)                   WHERE title = movieTitle AND
5)                         year = movieYear AND
6)                         starName = 'Harrison Ford'
7)                  ) Prod
8)  WHERE cert# = Prod.producerC#;
```

Could produce duplicates, if same producer has made multiple movies with "Harrison Ford"

4.3

Exercise 6.4.8: In Example 5.10 we gave an example of the query: "find, for each star who has appeared in at least three movies, the earliest year in which they appeared." We wrote this query as a γ operation. Write it in SQL.

$$\gamma_{starName,\ \texttt{MIN}(year)\rightarrow minYear,\ \texttt{COUNT}(title)\rightarrow ctTitle}(\texttt{StarsIn})$$

```
SELECT s.starname
FROM (SELECT
        starname,
        count(*)
      FROM starsIn
      GROUP BY starname) s
WHERE s.count > 3;
```