Practice for
SQL and Constraints
(Product-PC-Laptop-Printer)

Exercise – PC/Laptop/Printer

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

a) Find those manufacturers that sell Laptops, but not PC's
b) Find those hard-disk sizes that occur in two or more PC's.
c) Find those manufacturers of at least two different computers (PC or Laptops) with speed of at least 700.
d) Find the manufacturers who sell exactly three different models of PC.
e) Using two INSERT statements, store in the database the fact that PC model 1100 is made by manufacturer C, has speed 1800, RAM 256, hard disk 80, a 20x DVD, and sells for $2499.
f) Insert the facts that for every PC there is a laptop with the same manufacturer, speed, RAM and hard disk, a 15-inch screen, a model number 1000 greater, and a price $500 more.
g) Delete all PC's with less than 20 GB of hard disk.
h) Delete all laptops made a manufacturer that doesn't make printers.
i) Manufacturer A buys manufacturer B. Change all products made by B so they are now made by A.
j) For each PC, double the amount of RAM and add 20 GB to the amount of hard disk.
k) For each laptop made by manufacturer B, add one inch to the screen size and subtract $100 from the price.

Creation of Tables

```
CREATE TABLE Product (
  maker CHAR(10),
  model INT,
  type CHAR(5)
);

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

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

CREATE TABLE Printer (
  model INT,
  color CHAR(1),
  type CHAR(5),
  price INT
);
```

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

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

```
(SELECT maker
 FROM Laptop NATURAL JOIN Product)
   MINUS
(SELECT maker
 FROM PC NATURAL JOIN Product);
```

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

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

```
SELECT hd
FROM PC
GROUP BY hd
HAVING COUNT(model) >= 2;
```

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

c) Find those manufacturers of at least two different computers (PC or Laptops) with speed of at least 700.

```
SELECT maker
FROM (
    (SELECT model, speed
     FROM PC) UNION
    (SELECT model, speed
     FROM Laptop) )
     NATURAL JOIN
     Product
WHERE speed>=700
GROUP BY maker
HAVING COUNT(model) >= 2;
```

Or:

```
SELECT maker
FROM (
     (SELECT model, speed
      FROM PC) UNION
     (SELECT model, speed
      FROM Laptop)
     ) C
     JOIN
     Product
        ON C.model=Product.model
WHERE C.speed>=700
GROUP BY Product.maker
HAVING COUNT(C.model) >= 2;
```

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

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

```
SELECT Product.maker
FROM PC, Product
WHERE PC.model=Product.model
GROUP BY Product.maker
HAVING COUNT(PC.model)=3;

Or:
SELECT maker
FROM PC NATURAL JOIN Product
GROUP BY maker
HAVING COUNT(model)=3;
```

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

e) Using two INSERT statements, store in the database the fact that PC model 1100 is made by manufacturer C, has speed 1800, RAM 256, hard disk 80, a 20x DVD, and sells for $2499.

```
INSERT INTO Product(maker,model,type)
VALUES('C',1100,'PC');

INSERT INTO PC(model,speed,ram,hd,rd,price)
VALUES(1100,1800,256,80,20,2499);
```

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

f) Insert the facts that for every PC there is a laptop with the same manufacturer, speed, RAM and hard disk, a 15-inch screen, a model number 1000 greater, and a price $500 more.

```
INSERT INTO Product(maker,model,type)
(SELECT maker,model+1000,'Laptop'
 FROM Product
 WHERE type='PC'
);

INSERT INTO Laptop(model,speed,ram,hd,screen,price)
(SELECT model+1000, speed, ram, hd, 15, price+500
 FROM PC
);
```

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

g) Delete all PC's with less than 20 GB of hard disk.

```
DELETE FROM PC
WHERE hd<20;
```

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

h) Delete all laptops made by a manufacturer that doesn't make printers.

```
DELETE FROM Laptop
WHERE model IN
(SELECT model
 FROM Product
 WHERE maker IN (
   (SELECT maker
    FROM Product NATURAL JOIN Laptop)
      MINUS
   (SELECT maker
    FROM Product NATURAL JOIN Printer)
 )
);
```

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

i) Manufacturer A buys manufacturer B. Change all products made by B so they are now made by A.

```
UPDATE Product
SET maker='B'
WHERE maker='C';
```

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

j) For each PC, double the amount of RAM and add 20 GB to the amount of hard disk.

```
UPDATE PC
SET ram=ram*2, hd=hd+20;
```

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

k) For each laptop made by manufacturer B, add one inch to the screen size and subtract $100 from the price.

```
UPDATE Laptop
SET screen=screen+1, price=price-100
WHERE model IN
(SELECT model
 FROM Product
 WHERE maker='B'
);
```

Constraints – PCs, Laptops, Printers

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

First create keys and foreign key references.

Then create the following constraints.

a) The speed of a laptop must be at least 800.

b) The only types of printers are laser, ink-jet, and bubble.

c) A model of a product must also be the model of a PC, a laptop, or a printer.

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

First create keys and foreign key references.

```
CREATE TABLE Product (
  maker VARCHAR(10),
  model INT PRIMARY KEY,
  type  VARCHAR(10)
);

CREATE TABLE PC (
  model INT PRIMARY KEY,
  speed INT,
  ram INT,
  hd INT,
  rd INT,
  price FLOAT,
  CONSTRAINT fk_pc FOREIGN KEY(model)
     REFERENCES Product(model)
    ON DELETE CASCADE
);
```

```
CREATE TABLE Laptop (
  model INT PRIMARY KEY,
  speed INT,
  ram INT,
  hd INT,
  screen INT,
  price FLOAT,
  CONSTRAINT fk_lap FOREIGN
     KEY(model) REFERENCES
     Product(model)
    ON DELETE CASCADE
);
```

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

a) The speed of a laptop must be at least 800.

```
CREATE TABLE Laptop (
  model INT PRIMARY KEY,
  speed INT CHECK(speed >= 800),
  ram INT,
  hd INT,
  screen INT,
  price FLOAT,
  CONSTRAINT fk_lap FOREIGN
     KEY(model) REFERENCES
     Product(model)
   ON DELETE CASCADE
);
```

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

b) The only types of printers are laser, ink-jet, and bubble.

```
CREATE TABLE Printer (
  model INT PRIMARY KEY,
  color VARCHAR(10),
  type VARCHAR(10)
   CHECK(type IN ('laser', 'ink-jet', 'bubble')),
  price FLOAT,
  CONSTRAINT fk_printer FOREIGN KEY(model) REFERENCES Product(model)
    ON DELETE CASCADE
);
```

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

c) A model of a product must also be the model of a PC, a laptop, or a printer.

```
CREATE VIEW ProductSafe(maker,model,type) AS
SELECT maker, model, type
FROM Product
WHERE model IN (
  (SELECT model FROM PC) UNION
  (SELECT model FROM Laptop) UNION
 (SELECT model FROM Printer)
)
WITH CHECK OPTION;
```

Then, we insert into this view as opposed to directly into Product.

Also, make the FOREIGN KEY constraints in PC, Laptop, and Printer *deferrable initially deferred*.

Practice
(Suppliers, Parts, Catalog)

Exercise – Suppliers and Parts

Suppliers(sid,sname,address)

Parts(pid,pname,color)

Catalog(sid, pid,price)

a) Find the names of suppliers who supply every part.

b) Find the names of suppliers who supply every red part.

c) Find the part names supplied by IBM and no one else.

d) Find the sid's of suppliers who charge more for some part that the average price of that part (averaged over all suppliers who supply that part.)

e) For each part, find the name of the supplier who charges the least for that part.

f) For all suppliers that supply more than three red parts find how many green parts they supply.

Creation of Tables

```
CREATE TABLE Suppliers (
    sid INT,
    sname VARCHAR(20),
    address VARCHAR(20)
);

CREATE TABLE Parts (
    pid INT,
    pname VARCHAR(10),
    color VARCHAR(10)
);

CREATE TABLE Catalog (
    sid INT,
    pid INT,
    price INT
);
```

Suppliers(sid,sname,address)

Parts(pid,pname,color)

Catalog(sid, pid,price)

a) Find the names of suppliers who supply every part.

```
SELECT sname
FROM Suppliers X
WHERE NOT EXISTS (
    --If a supplier supplies all the parts, then this subq. should return empty result
      (SELECT pid FROM Parts)
       MINUS
      (SELECT pid
       FROM Catalog
       WHERE sid=X.sid)
);
```

Suppliers(sid,sname,address)

Parts(pid,pname,color)

Catalog(sid, pid,price)

b) Find the names of suppliers who supply every red part.

```
SELECT sname
FROM Suppliers X
WHERE NOT EXISTS (
        (SELECT pid FROM Parts WHERE color='red')
          MINUS
         (SELECT pid
          FROM Catalog NATURAL JOIN Parts
          WHERE sid=X.sid AND color='red')
);
```

Suppliers(sid,sname,address)

Parts(pid,pname,color)

Catalog(sid, pid,price)

c) Find the part names supplied by IBM and no one else.

```
CREATE VIEW SupCatPar AS
  SELECT sid, sname, address, pid, pname, color, price
  FROM Suppliers NATURAL JOIN Catalog NATURAL JOIN Parts;

SELECT pname
FROM SupCatPar
WHERE sname='IBM' AND pid NOT IN (
  SELECT pid
  FROM SupCatPar
  WHERE sname<>'IBM'
);
```

Suppliers(sid,sname,address)

Parts(pid,pname,color)

Catalog(sid, pid,price)

d) Find the sid's of suppliers who charge more for some part that the average price of that part (averaged over all suppliers who supply that part.)

```
SELECT sid
FROM Catalog X
WHERE price > (
  SELECT AVG(price)
  FROM Catalog
  WHERE pid=X.pid
);
```

Suppliers(sid,sname,address)

Parts(pid,pname,color)

Catalog(sid, pid,price)

e) For each part, find the name of the supplier who charges the least for that part.

```
SELECT pname, sname
FROM SupCatPar X
WHERE X.price = (
  SELECT MIN(price)
  FROM Catalog
  WHERE pid=X.pid
);
```

Suppliers(sid,sname,address)

Parts(pid,pname,color)

Catalog(sid, pid,price)

f) For all suppliers that supply more than three red parts find how many green parts they supply.

```
CREATE VIEW SIDs_RED AS
  SELECT sid
  FROM SupCatPar
  WHERE color='red'
  GROUP BY sid
  HAVING COUNT(pid)>3;

CREATE VIEW SupCatPar_Green AS
  SELECT *
  FROM SupCatPar
  WHERE color='green';

SELECT sname, COUNT(pid) AS number_green_parts
FROM SIDs_RED NATURAL LEFT OUTER JOIN SupCatPar_Green
GROUP BY sid, sname;

DROP VIEW SIDs_RED;
DROP VIEW SupCatPar_Green;
```

Student number:_____ **Name**:_____

UNIVERSITY OF VICTORIA
Faculty of Engineering
Department of Computer Science

CSC 370 (Database Systems)
Instructor: Daniel M. German

Midterm
Oct 15, 2012

Duration: 75 minutes

This is a closed-book exam.

This examination paper consists of **6** pages and **2** sections. Please bring any discrepancy to the attention of an invigilator. The number in parenthesis at the start of each question is the number of points the question is worth.

Answer all questions.

Please write your answers clearly.

For instructor's use:

	Score
1 (44)	
2 (4)	
Total (48)	

For this exam, consider the following schema and instances of the relations. Feel free to remove this page from the exam.

Our database is very simple. It is composed of three relations: *Parts*, *Suppliers* and *Catalog*. The *Catalog* table contains the parts that are being offered by a given supplier at a given price (a part is missing a price if this field is NULL). Every *pid* in *Catalog* exists in *Parts*, and every *sid* in *Catalog* exists in *Suppliers*.

```
Parts(pid: integer, pname character(40), color character(20));
```

Primary key: *pid*.

pid	*pname*	*color*
6	Anti-Gravity Turbine Generator	Cyan
7	Anti-Gravity Turbine Generator	Magenta
8	Fire Hydrant Cap	Red
9	7 Segment Display	Green
10	SQL queries	Green

```
Suppliers(sid: character(10), sname: character(40), address: char(50));
```

Primary key: *sid*.

sid	*sname*	*address*
amazon	Amazon Canada	1 Grub St., Potemkin Village, IL 61801
walmart	Walmart Inc	4 My Way, Bermuda Shorts, OR 90305
rim	Research in Motion	99999 Short Pier, Terra Del Fuego, TX 41299
google	Google Inc.	2 Groom Lake, Rachel, NV 51902

```
Catalog(sid: character(10), pid: integer, price: real);
```

Primary key: *(sid,pid)*.

sid	*pid*	*cost*
amazon	8	11.7
walmart	8	7.95
rim	8	12.5
rim	9	1
amazon	10	10.5
amazon	9	

1. Writing queries in Relational Algebra and SQL

Give both relational algebra and SQL queries to answer the following questions. **Your relational algebra should match your SQL queries.**

1.1) [4] For every supplier, lists its *sname* and the *pid* of each of the parts they offer. Result should contain two attributes.

$$\Pi_{sname,\ pid}(C \bowtie S)$$

```
SELECT sname, pid FROM
  Catalog NATURAL JOIN Suppliers;
```

1.2) [4] List the *pname* of parts that are being offered at $10 or more. Result should contain only one attribute.

$$\Pi_{pname}\sigma_{price>=10}(C \bowtie P)$$

```
SELECT pname FROM
   Catalog NATURAL JOIN Parts
   WHERE PRICE >= 10;
```

1.3) [4] For every *pid* in relation *Parts*, list the number of suppliers that offer it, and the minimal price at which it is offered. Result should contain three attributes.

$$\Pi_{pid,count,m}\gamma_{pid,count(sid)\rightarrow count,min(price)\rightarrow m}(P \bowtie_L C)$$

```
SELECT pid, count(sid) as count, min(price) as m FROM
   Parts NATURAL LEFT JOIN Catalog
   GROUP BY pid;
```

1.4) [4] How many parts in table *Parts* are not being offered by any supplier? Result should contain only one attribute.

$$\gamma_{count(pid)}(\Pi_{pid}P - \Pi_{pid}C)$$

```
SELECT count(pid) FROM
  (SELECT pid FROM Parts EXCEPT SELECT pid FROM Catalog);
```

1.5) [4] List the *pid* and *sid* of parts that offered by such supplier and are missing a *price*. Result should contain two attributes.

$$\Pi_{pid,sid}\sigma_{price\ is\ NULL}C$$

```
SELECT pid, sid FROM
   Catalog
   WHERE price is NULL;
```

1.6) [4] For every supplier, list its *sid* and the average *price* of the parts they offer. Result should contain two attributes.

$$\Pi_{sid,avg}\gamma_{sid,avg(price)}C$$

```
SELECT sid, avg(price) AS avg FROM
   Catalog
   GROUP BY sid
```

1.7) [4] List the *pid* and the *pname* of parts that are offered by exactly 3 suppliers. Result should contain two attributes.

$$\Pi_{pid,pname,count}(\sigma_{count=3}\gamma_{pid,count(sid)\rightarrow count}C) \bowtie P$$

```
SELECT pid, pname FROM
   (SELECT pid, count(sid) AS count FROM
    Catalog
    GROUP BY pid
    HAVING count = 3)
   NATURAL JOIN Parts
```

1.8) [4] List the *pid* of the parts that are being offered by both suppliers: `Amazon` and `Walmart` (these are their *sid*). Result should contain one attribute.

$$(\Pi_{pid}\sigma_{sid='Amazon'}C) \bigcap (\Pi_{pid}\sigma_{sid='Walmart'}C)$$

```
SELECT pid FROM Catalog WHERE sid = 'Amazon'
INTERSECT
SELECT pid FROM Catalog WHERE sid = 'Walmart'
```

1.9) [4] Compute the difference between the average price of parts with *pid* `12` and `32`. In other words, compute (the average price of partid `12`) minus (the average price of pid `32`). The result should contain one tuple with one attribute.

$$\Pi_{x-y}[(\gamma_{avg(price)\rightarrow x}\sigma_{pid=12}C) \times (\gamma_{avg(price)\rightarrow y}\sigma_{pid=32}C)]$$

```
SELECT x - y FROM
   (SELECT avg(price) AS x FROM
      Catalog
      WHERE pid = 12
      GROUP by pid) as A,
   (SELECT avg(price) AS y FROM
      Catalog
      WHERE pid = 32
      GROUP by pid) as B
```

1.10) [4] For every *pid* in the relation *Catalog*, list the *sname* of the supplier who offers it a the lowest *price*, and such *price*. Result should contain three attributes.

This requires a bit of explanation. First get the minimum price of each part. Call it M. Then use this to find the tuples in C that have this price (you can do this with a join or an IN, then join this to Suppliers to find their sname.

$$M = \gamma_{pid,min(price)\rightarrow price}C$$

$$\Pi_{pid,sname,price}(M \bowtie C \bowtie S)$$

```
SELECT pid, sname, price FROM
   (SELECT pid, min(price) as price) FROM CATALOG) as M
   NATURAL JOIN Parts NATURAL JOIN Catalog
```

1.11) [4] List the *pid* of parts that are being offered by at least two suppliers at exactly the same price. Your result should contain two columns: the *pid* of the two parts, and their *price*.

$$\Pi_{pid,price}\sigma_{c>=2}\gamma_{pid,price,count(sid)\rightarrow c}C$$

```
SELECT pid, price FROM
Catalog
GROUP BY pid, price
HAVING count(sid) >= 2
```

2. Relational Model

2.1) [4] Given the relation $R(A, B, C, D)$ and the set of functional dependencies $A \rightarrow BC$, $BC \rightarrow A$, and $B \rightarrow D$. Find all the candidate keys of this relation. Show all your work.

For this you have to compute the closure of each combination of attributes ABCD, ABC, ABD, ... A, B, C, D (15 in total). The candidate keys are only A and BC.

End of examination
Total pages: 6
Total marks: 48

UNIVERSITY OF VICTORIA
Faculty of Engineering
Department of Computer Science

CSC 370 (Database Systems)
Instructor: Daniel M. German

Mid-Midterm Exam
Feb 28, 2014

Duration: 50 minutes

This is a closed-book exam. You are allowed one sheet of paper, letter size, handwritten

This examination paper consists of **5** pages and **3** sections. Please bring any discrepancy to the attention of an invigilator. The number in parenthesis at the start of each question is the number of points the question is worth.

Answer all questions.

Please write your answers clearly.

For instructor's use:

	Score
1 (8)	
2 (12)	
3 (10)	
Total (30)	

For this exam, consider the following schema of a simple university database. It includes information about instructors, students, and the courses offered. Feel free to remove this page from the exam.

- The **Students** table contains the id of the student (**sid**), his/her name (**sname**), age (in years), and gpa.

```
Students(sid: integer, sname: string,
         age: integer, gpa: real)
```

  - Key: **sid**

- The **Instructors** table contains information about instructors of the courses: their id (**iid**), name (**iname**) and department they belong to (**dept**). An instruct can teach many different courses.

```
Instructors(iid: string, iname: string, dept: string)
```

  - Key: **iid**

- The **Courses** table contains information about courses: their id (**cid**), their name (**cname**), the department that offers it (**dept**), the id of its instructor (**iid**), and the maximum number of students who can take it (**maxenrol**). Every **iid** in this table is also found in the table **Instructors**.

```
Courses(cid: string, cname: string,
        dept: string, iid: string,
        maxenrol:integer
)
```

  - Key: **cid**

- The table **Enrolled** contains what students are registered to which courses, and the grade they receive (NULL if they have not received one yet). A student can only register once to any given course, but he/she can register to as many courses as necessary. Neither **sid** nor **cid** can be NULL. Every **sid** in this table is also found in the table **Students**, and every **cid** in this table is also found in the table **Courses**.

```
Enrolled(sid: integer, cid: string,
         grade: integer)
```

  - Key: (**sid,cid**)

1. Functional Dependencies

(a) [2] Assume a relation $R(T, C, M)$.

- T corresponds to the name of the Theater.
- C corresponds to the name of the City
- M corresponds to the name of the Movie.
- The name of the Theater is unique across all Cities. $T \to C$
- There are several Theaters per City. Explicitly $C \not\to T$
- We only show a given Movie in one Theater per City. $MC \to T$

What are the functional dependencies that apply to this relation?

So fds are $T \to C$, $MC \to T$

(b) [2] Given the functional dependencies: $A \not\to B$, $CH \to A$, $B \not\to E$, $BD \not\to C$, $EG \to H$, $DE \not\to F$, is it possible to generate $ADE \to CH$? Why?

We can generate ADE iff $\{ADE\}^+$ contains CH.

$\{ADE\}^+ = \{ADEBCF\}$ so no, it can not be generated

(c) [4] Consider relation R(A,B,C,D) with functional dependencies: $D \to C$,$CB \to A$,$D \to A$,$AB \to D$. Compute all its candidate keys.

B is never in right hand side, so any key will contain B

Superkeys:

✓	B	A	C	D	BACD
✓	B	A	C		BACD
✓	B	A		D	BADC
✓	B	A			BADC
✓	B		C	D	BCDA
✓	B		C		BCAD
✓	B			D	BDCA
	B				B

BA, BC and BD are candidate keys

2. Normalization

(a) [2] Given the relation $R(ABC)$ with functional dependencies $A \to C$ and $C \to B$. Is the decomposition into relations AC and BC lossless join? Explain.

To be lossless join $\exists$ $AC \cap BC \to A$ or $AC \cap BC \to B$

Since $C \to B$ is given, this Decomp. is lossless join.

(b) [2] Is the previous decomposition FD preserving? Why?

AC has FDs. $A \to C$ and BC has FDs. $C \to B$.

So yes. It is FD preserving.

(c) [2] Assume R is a relation with two or more attributes, and that it has one non-trivial functional dependency. Is R **always** BCNF? Explain.

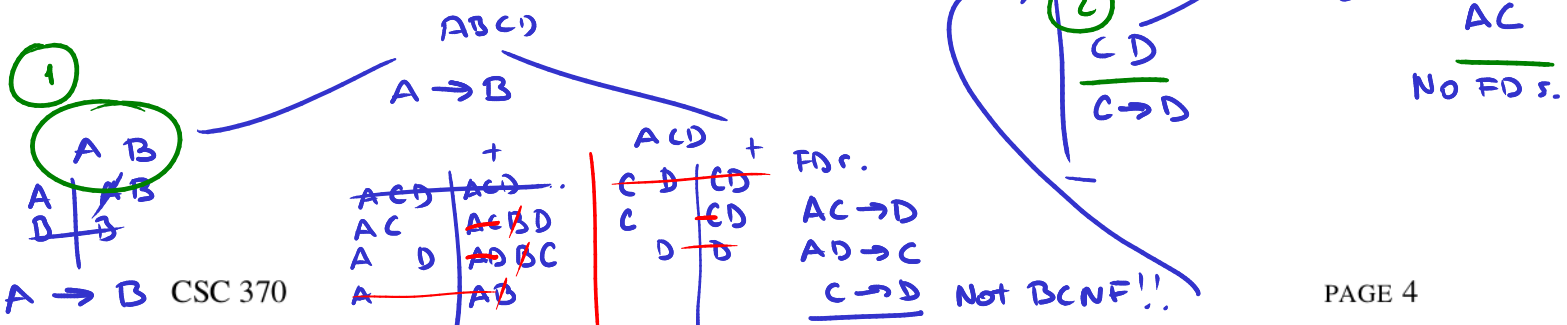
if the FD $A_1 \ldots A_n \to B_1 \ldots B_k$ and it is not trivial than none of $B_1 \ldots B_k$ are in $A_1 \ldots A_n$

It would be BCNF iff $A_1 \ldots A_n \cup B_1 \ldots B_n = R$.

otherwise it is NOT. So no, it is not always BCNF

(d) [6] Consider the relation R(A,B,C,D) with functional dependencies: $A \to B$, $C \to D$, $AD \to C$, $BC \to A$. This table is not BCNF. Decompose this relation into a set of BCNF relations that are functional dependency preserving.

$\{A\}^+ = \{AB\}$ $A \to B$ NOT BCNF



Decomposition into AB, CD, AC loses $BC \to A$ and $AD \to C$

$A \to B$ $C \to D$ NO FDs add ABC and ADC

Optional : we can combine AB & ABC and ADC and CD

⇒ Final Decomp: ABC , ADC and AC with NO FDs.
BC → A, A → B; AD → C, C → D

3. Relational Algebra and SQL

For each of the following questions, provide a relational algebra expression to answer them, and its equivalent SQL query:

(a) [2] What is the average **age** of the students who are taking at least one course? Result should have only one column (and one tuple). Hint. Make sure you average each student's age only once.

$\gamma_{avg(age)}(\sigma_{sid \text{ IN } (\pi_{sid} E)} S)$

```
select avg(age) from students
where sid IN (select sid from Enrolled)
```

(b) [4] For every instructor that is teaching exactly two courses, list the **iid** of the instructor, their name **iname** and the course **cid** they are teaching. There are going to be two tuples for each instructor, one for each course they teach. For instance, your result should look something like this (three columns).

iid	iname	cid
342	M. Zastre	Seng 365
342	M. Zastre	CSC 360
123	D. German	CSC 370
~~456~~ 123	D. German	CSC 225

$A = \pi_{iid} \sigma_{c=2} \gamma_{iid, count(*) \to c} C$

$\pi_{iid, iname, cid}$ C ⋈ A ⋈ I

```
WITH A AS ( select iid from C group by iid
            having count(*) = 2)
select iid, iname, cid from C NATURAL
JOIN A NATURAL JOIN I;
```

(c) [4] List the **sId** and **sname** of the students who are enrolled in the fewest courses. Your result should include three columns: **sid**, **sname** and total number of courses. Make sure you consider students who might not be taking any course (in that case they are enrolled to zero courses).

$T = \gamma_{sid, sname, count(cid) \to c}$ E ⟕$_R$ S

sid → sname, so ok.

$M = \gamma_{min(c) \to c} T$

$\pi_{sid, sname}$ T ⋈ M.

```
WITH T AS (SELECT sid, sname,
           count(cid) as c
           from E NATURAL RIGHT JOIN S)
WITH M AS( select min(c) as c
           from T)
select sid, sname from
T NATURAL JOIN M;
```

End of examination
Total pages: 5
Total marks: 30