1 2005 Paper 5 Question 8

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2005p5q8.pdf

(a) Define the core operators of the relational algebra.

The core operators of the relational algebra are:

- R is the base relation.

  This is the Relational Algebra equivalent of a table.
- $\sigma_p(Q)$ is selection.

  This will select only the records in Q for which the predicate p holds.
- $\pi_X(Q)$ is projection.

  This returns the columns X in the relation Q.
- $Q_1 \times Q_2$ is product.

  This returns all possible combinations from combining records in Q_1 and Q_2.
- $Q_1 - Q_2$ is difference

  This will return all records which are in Q_1 and **not** in Q_2.
- $Q_1 \cup Q_2$ is union.

  This will combine relations Q_1 and Q_2 and return any records which are in **either** relation.
- $Q_1 \cap Q_2$ is intersection.

  This will return all records which are in **both** Q_1 and Q_2.
- $\rho_M(Q)$ is renaming.

  This will rename the column or relation specified in M. It is particularly useful when combining relations, as there may be multiple columns with the same name.

(b) Describe *two* differences and *two* similarities between the relational algebra and SQL.

Two differences between relational algebra and SQL are:

- SQL is based on multisets while relational algebra is based on sets.
- SQL is used in real-world applications, while relational algebra is theoretical.

Two similarities between relational algebra and SQL are:

- Both manipulate "relations".
- They are both capable of the same operations.

(c) Suppose that $S(a, b, ...)$ and $R(a, ...)$ are relations (the notation indicates that attribute a is in the schema of both S and R, while attribute b is only in the schema of S). Suppose that v is a value; is the following equation always valid?

$$\sigma_{(a=v \text{ or } b=v)}(R \bowtie S) = (\sigma_{a=v}(R)) \bowtie (\sigma_{b=v}(S))$$

If yes, provide a short proof. If no, provide a counter-example.

No: they do not always return the same things.

Take the counter-example below:

R

A	C
0	5
1	2
2	3
3	1

S

A	B
0	4
1	4
2	4

If $v = 4$, then $\sigma_{(a=v \text{ or } b=v)}(R \bowtie S)$ will join the relations. In the joined relation: $b = v$ for every record, so the entire joined relation will be returned:

$\sigma_{(a=v \text{ or } b=v)}(R \bowtie S)$

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

However, if $v = 4$ then $(\sigma_{a=v}(R)) \bowtie (\sigma_{b=v}(S))$ will not select any records in R (since no values of A are equal to v), then every record in S. It will then perform a join with S and an empty relation. This will return an empty relation.

So the equation is not always valid.

(d) Various *normal forms* are important in relational schema design.

(i) Define Third Normal Form (3NF).

If there are no attributes in the database which can be determined from any set of attributes which is not a subset of the primary key and the database is in 2NF then the database is in 3NF.

(ii) Define Boyce-Codd Normal Form (BCNF).

If any set of attributes which can be used to determine any other set of attributes is either the primary key or a superset of the primary key and the database is in 2NF then the database is in BCNF.

(iii) For databases with many concurrent update transactions, explain why schemas in normal form are important for good performance.

Schemas in normal form minimise data redundancy. The advantages of this when concurrently updating data are:

- There will not be cases where data is temporarliy inconsistent (and could be read by a different user).
- Records only contain data which strictly needs to be in them. So when locking records, access to data is not prevented unneccessarily.
- Normalisation means fewer memory accesses to update duplicate data. So fewer write requests and less locking records.

2 2002 Paper 6 Question 8

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2002p6q8.pdf

(b) The core relational algebra is often extended with other operators. For the following operators give a definition and an example of their behaviour:

(i) the full outer join operator;

The full outer join operator will join two relations together on a specified attribute. If there are no matches for one record, then all empty attributes will be filled in as *Null*.

Example: a full outer join of the relations A and B is shown below.

A

A_1	A_2
0	1
3	4
4	9

B

A_1	A_3
1	0
3	1
4	4

A *outer join* B

A_1	A_2	A_3
0	1	Null
1	Null	0
3	4	1
4	9	4

(ii) the aggregate and grouping operator;

The aggregate and grouping operator puts records in groups based on the value of an attribute. This limits the things which can be returned from the table.

IE

Given a relation People(*Person_ID*:integer, *name*:string, *age*:integer)

```
select age, count(*)
from
group by age
order by age
;
```

would return the age and the number of people of that age.

(c) X, Y and Z are all relations with a single attribute A. A naïve user wishes to compute the set-theoretic expression $X \cap (Y \cup Z)$ and writes the following SQL query.
SELECT $X.A$
FROM X,Y,Z
WHERE $X.A = Y.A$ OR $X.A = Z.A$

(i) Give the relational algebra term that this query would be compiled to.

$\pi_{X.A}(\sigma_{X.A=Y.A \text{ or } X.A=Z.A}(X, Y, Z))$

(ii) Does the SQL query satisfy the user's expectation? Justify your answer.

In this specific case - since X, Y and Z are all sets and have no repeated values - it would satisfy the users expectation. However, if X, Y and Z were multisets (and could have multiple instances of the same value), then this sql query would produce duplicates.

In this specific case:

There is only one instance of a value - which may or may not be present in a table. If it is not present in X, then it is not displayed. If it is present in X and in (Y or Z) then it will be displayed. So this will do what the user wants.

3 2006 Paper 6 Question 8

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2006p6q8.pdf

Suppose we have the following relational schema
Person(*pid*:integer, *name*:string, *street*:string, *postcode*:string)
Car(*cid*:integer, *year*:integer, *model*:string)
OwnedBy(*pid*:integer, *cid*:integer)

AccidentReport(*rid*:integer, *damage*:integer, *details*:string)
ParticipatedIn(*pid*:integer, *rid*:integer, *cid*:integer)
where the underlined attributes represent the primary keys of the associated relation. The table *OwnedBy* implements a relationship between persons and cars using foreign keys. The table *ParticipatedIn* implements a relationship between persons, accident reports, and cars, where tuple (p, r, c) indicates that the person p was the driver of the car c associated with the accident report r.

(a) Write an SQL query to return those *pid*'s of persons driving in at least one accident, with no duplicates.

```
select distinct pid
from ParticipatedIn
;
```

(b) Write an SQL query to return all tuples (pid, c), where c is the number of cars owned by person *pid* (records where $c = 0$ do not have to be generated).

```
select pid, count(*) as c
from OwnedBy
group by pid
;
```

(c) Write an SQL query to return all tuples (cid, c), where c is the number of persons owning car *cid* (records where $c = 0$ do not have to be generated).

```
select cid, count(*) as c
from OwnedBy
group by cid
;
```

(d) Write a (nested) SQL query to return all tuples (pid, rid) where *pid* was driving in the accident reported in *rid*, but the car driven by *pid* is not owned by *pid*.

```
select ar.pid, ar.rid
from AccidentReport as ar
join OwnedBy as ob on ob.cid = ar.cid
where ob.pid != ar.pid
;
```

(e) Write an SQL query to return all tuples (rid, c), where c is the number of drivers involved in the accident reported in by *rid* (records where c = 0 do not have to be generated).

```
select rid, count(distinct pid) as c
from ParticipatedIn
group by rid
;
```

(f) Write an SQL query to return all tuples (rid, c), where c is the number of cars involved in the accident reported in by *rid* (records where $c = 0$ do not have to be generated).

```
select rid, count(distinct cid) as c
from ParticipatedIn
group by rid
;
```

(g) Do the functional dependencies implied by the schema imply that the results of queries (e) and (f) will always be the same? Explain.

No. The functional dependencies allow the number of cars and the number of drivers to be different. Since the primary key of the ParticipatedIn table is *rid*, *pid*, *cid*, it

is possible for there to be one driver who drove two cars in the accident – or a car which was driven by two people. This allows for the number of cars and the number of drivers in an accident to be different. So (e) and (f) can be different.

(h) Perhaps there is something wrong with this schema. How would you fix the schema to ensure that results of queries (e) and (f) would always be the same?

To ensure that each driver can only drive one car and that each car can only be driven by one driver, you would change the schema to either: ParticipatedIn(*pid*:integer, *rid*:integer, *cid*:integer) or ParticipatedIn(*cid*:integer, *rid*:integer, *pid*:integer). Both ensure that each car can only have one driver and each person only drive one car in each accident. Hence ensuring that the results from (e) and (f) are the same.

4 2004 Paper 5 Question 8

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2004p5q8.pdf

Assume a simple movie database with the following schema. (You may assume that producers have a unique certification number, *cert*, that is also recorded in the Movie relation as attribute *prodC#*; and no two movies are produced with the same title.)
Movie(*title*, *year*, *length*, *prodC#*)
StarsIn(*movieTitle*, *movieYear*, *starName*)
Producer(*name*, *address*, *cert*)
MovieStar(*name*, *gender*, *birthdate*)

(a) Write the following queries in SQL:

(i) Who were the male stars in the film *The Red Squirrel*?

```
select si.starName
from StarsIn as si
join MovieStar as ms on ms.name = si.starName
where si.movieTitle = "The Red Squirrel" and ms.gender = `male'
;
```

(ii) Which movies are longer than *Titanic*?

```
select title
from Movie
where Movie.length > (select length from Movies where Movie.title =
"Titanic")
;
```

(b) SQL has a boolean-valued operator `IN` such that the expression *s* `IN R` is true when *s* is contained in the relation `R` (assume for simplicity that `R` is a single attribute relation and hence *s* is a simple atomic value).
Consider the following nested SQL query that uses the `IN` operator:

```
SELECT name
FROM Producer
WHERE cert IN (SELECT prod#C
FROM Movie
WHERE title IN (SELECT movieTitle
FROM StarsIn
WHERE starName = "Nancho Novo"))
```

(i) State concisely what this query is intended to mean.

Select all producers of movies which 'Nancho Novo' has starred in.

(ii) Express this nested query as a single `SELECT-FROM-WHERE` query

```
select p.name
from Producer as p
```

```
join Movie as m on m.prodC# = p.cert
join StarsIn as si on si.movieTitle = Movie.title
where si.name = "Nancho Novo"
;
```

(iii) Is your query from part (b)(ii) always equivalent to the original query? If yes, then justify your answer; if not, then explain the difference and show how they could be made equivalent.

No. Our query may not always be equivalent to the query in (b)(i). Take the case where there are two films with the same name and 'Nancho Novo' stars in one of them but not the other.

The first query will select all movies which 'Nancho Novo' starred in, and then use their *u*nique ID's to select the producers of that movie.

While the query in (b)(ii) uses the Movie title – which in this case would not be unique – to select the producer. So the query would select all the directors of the movies which have the same name as a movie which 'Nancho Novo' starred in. In the case above this would mean a producer of a film which 'Nancho Novo' did not star in would be selected. Which is not equivalent to the original statement.

(c) SQL has a boolean-valued operator `EXISTS` such that `EXISTS R` is true if and only if `R` is not empty.
Show how `EXISTS` is, in fact, redundant by giving a simple SQL expression that is equivalent to `EXISTS R` but does not involve `EXISTS` or any cardinality operators, e.g. `COUNT`. [Hint: You may use the `IN` operator.]

"`(select top 1 * from R) like '%'`" is equivalent to "`EXISTS R`".

This selects the top record from R and returns true if it pattern matches it to anything. Which it will do if there is a first record.

This is the same as checking if there is one or more records in R – the same as `EXISTS`.

PS I am aware that this does not use the `IN` operator as hinted that we should. But this works (tested in HyperSQL) and I couldn't get a query using `IN` that worked.