Databases: Supervision 1

Daniel Chatfield

February 15, 2015

2003 Paper 5 Question 8

1. (a) i. Define the operators in the core relational algebra. [5]

> **Union** $R \cup S$ is the union of R and S.
>
> **Intersection** $R \cap S$ is the intersection of R and S.
>
> **Difference** $R - S$ is the difference between R and S.
>
> **Product** $R \times S$ is the product of R and S.
>
> **Selection** $\sigma_p(R)$ is a selection over R. P is a propositional formula and the operator selects all tuples in R for which p holds.
>
> **Projection** $\pi_x(R)$ is a projection over $R.x$ is a set of attribute names.

ii. Define the domain relational calculus. [4]

> $Q = \{(A_1 = v_1,\ A_2 = v_2, \ldots,\ A_k = v_k) | P(v_1,\ v_2, \ldots,\ v_k)\}$

iii. Show how the relational algebra can be encoded in the domain relational calculus. [3]

> - Selection
>
> **Relational Algebra** $Q = \sigma_{A>12}(R)$
>
> **Domain Relational Calculus** Encoded as:
> $Q = \{\{(A,a),(B,b),(C,c)\} | \{\{(A,a),(B,b),(C,c)\} \in R \wedge a > 12\}$

- Projection

  Relational Algebra $Q = \pi_{B,C}(R)$

  Domain Relational Calculus Encoded as:
  $Q = \{\{(B,b),(C,c)\}|\{\{(A,a),(B,b),(C,c)\} \in R\}$

(b) A *constraint* can be expressed using relational algebra. For example, $R = \emptyset$ specifies the constraint that relation R must be empty, and $(R \cup S) \subseteq T$ specifies that every tuple in the union of R and S must be in T.

Consider the following schema.

```
RockStar(name, address, gender, birthday)
RockManager(managername, starname)
```

i. Give a constraint to express that rock stars must be either male or female. [1]

$\sigma_{gender\ !='male'\ AND\ gender\ !='female'}(RockManager) = \emptyset$

ii. Give a constraint to express the referential integrity constraint between the `RockStar` and `RockManager` relations. (Note: `starname` is intended to be a foreign key.) [3]

$\pi_{starname}(RockManager) \subseteq \pi_{name}(RockStar)$

iii. Give a constraint to express the functional dependency `name`→`address` for the RockStar relation. [4]

Not covered this yet and I don't understand it from slides alone

2004 Paper 5 Question 8

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)
```

2. (a) Write the following queries in SQL:

   i. Who were the male stars in the film *The Red Squirrel*? [1]

```
(SELECT name
 FROM MovieStar
 WHERE gender='male')
INTERSECT
(SELECT starName
 FROM StarsIn
 WHERE movieTitle='The Red Squirrel')
```

   ii. Which movies are longer than *Titanic*? [2]

```
SELECT *
FROM Movie
WHERE length >
    (SELECT length
     from Movie
     WHERE 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 prodC#
                FROM   Movie
                WHERE  title IN (SELECT movieTitle
                                 FROM   StarsIn
                                 WHERE  starName='Nancho Novo'));
```

   i. State concisely what this query is intended to mean. [1]

> This query returns the names of the producers of the movies that 'Nancho Novo' starred in.

   ii. Express this nested query as a single `SELECT-FROM-WHERE` query. [2]

```
SELECT DISTINCT Producer.name
FROM   StarsIn, Movie, Producer
WHERE  StarsIn.starName = 'nancho Novo',
       Movie.title = StarsIn.movieTitle,
       Producer.cert = Movie.prodC#
```

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. [6]

> The query is always equiavalent, the DISTINCT ensures that a producer is only returned once, just like the original query.
>
> The outer-most part of the original query was over the Producer table and therefore a single producer could only be returned once, even if they produced multiple films with Nancho Novo since the predicate IN is simply *true* or $false$.
>
> The new query joins the tables together which duplicates the Producer field and therefore the `DISTINCT` is required to ensure no duplicates.

(c) `SQL` has a boolean-valued operator `EXISTS` such that `EXISTS R` is true if and only if R is not empty. [8]

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.]*

> *I think this is a valid solution but the number of marks available for this question is making me doubt that.*
>
> ```
> R IN R
> ```
>
> An example:
>
> ```
> SELECT *
> FROM Colleges
> WHERE EXISTS (
>     SELECT *
>     FROM Students
>     WHERE Colleges.id = Students.college
> )
> ```
>
> can be written as:
>
> ```
> SELECT *
> FROM Colleges
> WHERE id IN (
>     SELECT college
>     FROM Students
> )
> ```