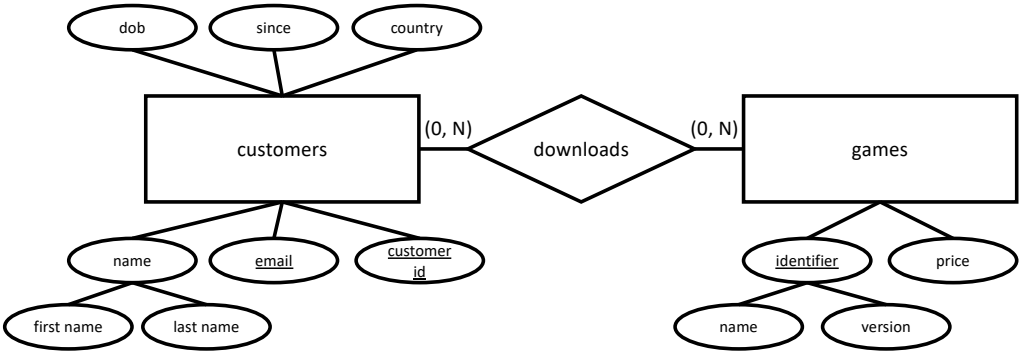
Relational Calculus and Algebra

Stéphane Bressan





We want to develop an application for managing the data of our online app store. We would like to store several items of information about our customers such as their first name, last name, date of birth, e-mail, date and country of registration to our online sales service and the customer identifier that they have chosen . We also want to manage the list of our products, games, their name, their version and their price. The price is fixed for each version of each game. Finally, our customers buy and download games. So we must remember which version of which game each customer has downloaded. It is not important to keep the download date for this application.


dob
since
country
customers
(0, N)
downloads
(0, N)
games
name
email
customer id
first name
last name
identifier
price
name
version

This is the complete schema for our example

```
1  CREATE TABLE IF NOT EXISTS customers (
2   first_name VARCHAR(64) NOT NULL,
3   last_name VARCHAR(64) NOT NULL,
4   email VARCHAR(64) UNIQUE NOT NULL,
5   dob DATE NOT NULL,
6   since DATE NOT NULL,
7   customerid VARCHAR(16) PRIMARY KEY,
8   country VARCHAR(16) NOT NULL);
9
10 CREATE TABLE IF NOT EXISTS games(
11  name VARCHAR(32),
12  version CHAR(3),
13  price NUMERIC NOT NULL,
14  PRIMARY KEY (name, version));
15
16  CREATE TABLE downloads(
17  customerid VARCHAR(16) REFERENCES customers(customerid)
18     ON UPDATE CASCADE ON DELETE CASCADE
19     DEFERRABLE INITIALLY DEFERRED,
20  name VARCHAR(32),
21  version CHAR(3),
22  PRIMARY KEY (customerid, name, version),
23  FOREIGN KEY (name, version) REFERENCES games(name, version)
24     ON UPDATE CASCADE ON DELETE CASCADE
25     DEFERRABLE INITIALLY DEFERRED);
```

The model semantics of propositional logic is defined by the truth tables of connectives, conjunction, $\wedge$ ("and"), disjunction, $\vee$ ("or"), negation, $\neg$ (not), and implication, $\rightarrow$ (imply), for the possible truth values ($TRUE$ and $FALSE$) of the propositions (p and q)

p	q	$p \wedge q$	$p \vee q$	$\neg p$	$p \rightarrow q$
$TRUE$	$TRUE$	$TRUE$	$TRUE$	$FALSE$	$TRUE$
$FALSE$	$TRUE$	$FALSE$	$TRUE$	$TRUE$	$TRUE$
$TRUE$	$FALSE$	$FALSE$	$TRUE$	$FALSE$	$FALSE$
$FALSE$	$FALSE$	$FALSE$	$FALSE$	$TRUE$	$TRUE$

Two propositional formulae are equivalent if and only if they have the same truth table.

$$p \rightarrow q \equiv \neg(p \wedge \neg q)$$

p	q	$\neg q$	$p \wedge \neg q$	$\neg(p \wedge \neg q)$	$p \rightarrow q$
$TRUE$	$TRUE$	$FALSE$	$FALSE$	$TRUE$	$TRUE$
$FALSE$	$TRUE$	$FALSE$	$FALSE$	$TRUE$	$TRUE$
$TRUE$	$FALSE$	$TRUE$	$TRUE$	$FALSE$	$FALSE$
$FALSE$	$FALSE$	$TRUE$	$FALSE$	$TRUE$	$TRUE$

Predicate logic, or first order logic, consists of formulae built from predicates (lower case) , operators ($=$, $\leq$, etc.) constants (lower case), variables (upper case, quantified or free), connectives ($\wedge$, $\vee$, $\neg$, and $\rightarrow$), and quantifiers ($\forall$ and $\exists$).

$$\forall Z_1 \ \forall Z_2 \ ((\exists X \ \exists Y \ (games(X, Y, Z_1) \wedge games(X, Y, Z_2))) \rightarrow Z_1 = Z_2)$$

$$\forall T_1 \ \forall T_2$$
$$(T_1 \in games \ \wedge T_2 \in games \wedge T_1.name = T_2.name \wedge T_1.version = T_2.version)$$
$$\rightarrow T_1.price = T_2.price)$$

$$\{name, version\} \rightarrow \{price\}$$

The existential quantifier, $\exists$, is a generalisation of disjunction.

$$\exists X\, F[X] \equiv F[a] \vee F[b] \vee \cdots$$

The universal quantifier, $\forall$, is a generalisation of conjunction.

$$\forall X\, F[X] \equiv F[a] \wedge F[b] \wedge \cdots$$

Accordingly, the De Morgan laws can be generalised to quantifiers:

$$\neg(\exists X\ F[X]) \equiv \forall X\ \neg F[X]$$

$$\neg(\forall X\ F[X]) \equiv \exists X\ \neg F[X]$$

Predicate logic formulae with free variables (variables that are not quantified) can be used to write queries, sets in intension, that denote, results, sets in extension.

Find the natural numbers between 1 and 5:

$$\{X \mid X \in \mathbb{N} \ \wedge X > 1 \ \wedge X < 5\} = \{2, 3, 4\}$$

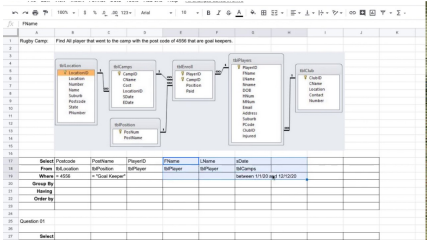
Variables in the head of the query are free and, by convention, they are the only free variable.

By convention variables are quantified at most once and can only appear in the scope of their quantifier.

In Domain Relation Calculus (DRC) variables range over values of fields of tables (they range over domains.) Domain relational calculus is the theoretical basis of *Query by Example*.

$$\{< X, Y, Z > | < X, Y, Z > \in games\}$$

$$\{< X, Y, Z > | \ games(X, Y, Z)\}$$



In Tuple Relation Calculus (TRC) variables values of rows of tables (t-uples.) Tuple relational calculus. is the theoretical basis of SQL.

$$\{T \mid T \in games)\}$$

We only discuss tuple relational calculus.

Find all the games.

$$\{T \mid T \in games\}$$

$$\{T \mid \exists G\ (G \in games \wedge G = T)\}$$

```
1 SELECT *
2 FROM games g;
```

Find the names, versions, and prices of the games.

Conjunction is commutative: $p \wedge q \equiv q \wedge p$.

$$\{T \mid \exists G$$

$$(G \in games \wedge T.name = G.name \wedge T.version = G.version \wedge T.price = G.price)\}$$

$$\{T \mid \exists G$$

$$(T.name = G.name \wedge T.version = G.version \wedge T.price = G.price \wedge G \in games)\}$$

```
1 SELECT g.name, g.version, g.price
2 FROM games g;
```

Find the first and last names of the customers in Singapore.

$$\{T \mid \exists C \ (\\ T.firstname = C.firstname \ \wedge \ T.lastname = C.lastname \wedge \\ C \in customers \wedge \\ C.country = 'Singapore')\}$$

```
1 SELECT c.firstname , c.lastname
2 FROM customers c
3 WHERE c.country = 'Singapore';
```

Find the first and last names of the customers and the prices of the games that they have downloaded.

$$\{T \mid \exists C\ \exists D\ \exists G\ (T.firstname = C.firstname\ \wedge\ T.lastname = C.lastname\ \wedge\ T.price = G.price\ \wedge\ C \in customers\ \wedge\ D \in download\ \wedge\ G \in games\ \wedge\ D.customerid = C.customerid\ \wedge\ D.name = G.name\ \wedge\ D.version = G.version)\}$$

```
1 SELECT c.firstname, c.lastname, g.price
2 FROM customers c, downloads d, games g
3 WHERE d.customerid = c.customerid
4 AND d.name = g.name
5 AND d.version = g.version;
```

What does this query find?

```
1  SELECT c.first_name, c.last_name
2  FROM customers c
3  WHERE NOT EXISTS(
4      SELECT *
5      FROM games g
6      WHERE g.name ='Aerified'
7      AND NOT EXISTS (
8          SELECT *
9          FROM downloads d
10         WHERE d.customerid = c.customerid
11         AND d.name = g.name
12         AND d.version = g.version));
```

Find the first and last names of the customers who downloaded all versions of Aerified.

Find the first and last names of the customers who downloaded all versions of Aerified.

$$\{T \mid \exists C\ (T.firstname = C.firstname\ \wedge\ T.lastname = C.lastname \wedge C \in customers \wedge (\forall G\ ((G \in games\ \wedge\ G.name = 'Aerified') \rightarrow (\exists D\ (D \in downloads \wedge D.customerid = C.customerid\ \wedge\ D.name = G.name\ \wedge\ D.version = G.version)))))\}$$

"Find the first and last names of customers such that
if there is a game is called 'Aerified',
then the customer has downloaded that game."

Remember that $p \rightarrow q \equiv \neg(p \wedge \neg q)$.

$$\{T \mid \exists C \ (\\ T.firstname = C.firstname \ \wedge \ T.lastname = C.lastname \wedge \\ C \in customers \wedge \\ (\forall G \ \neg((G \in games \ \wedge \ G.name = 'Aerified') \wedge \\ \neg(\exists D \ (D \in downloads \wedge \\ D.customerid = C.customerid \ \wedge \ D.name = G.name \ \wedge \ D.version = \\ G.version)))))\}$$

Remember that $\neg(\exists X\ F[X]) \equiv \forall X\ \neg F[X]$ and that $\neg(\forall X\ F[X]) \equiv \exists X\ \neg F[X]$.

$$\{T \mid \exists C\ \forall G\ \exists D\ (T.firstname = C.firstname\ \wedge\ T.lastname = C.lastname \wedge C \in customers \wedge \neg(G \in games\ \wedge\ G.name = 'Aerified' \wedge \neg(D \in downloads \wedge D.customerid = C.customerid \wedge D.name = G.name \wedge D.version = G.version)))\}$$

This form has all the quantifiers at the beginning but SQL does not offer a general direct support for the universal quantifier.

Remember that $\neg(\exists X\ F[X]) \equiv \forall X\ \neg F[X]$ as well as $\neg(\forall X\ F[X]) \equiv \exists X\ \neg F[X]$.

$$\{T \mid \exists C\ (T.firstname = C.firstname\ \wedge\ T.lastname = C.lastname \wedge C \in customers \wedge \neg(\exists G\ (G \in games \wedge G.name = \text{'}Aerified\text{'} \wedge \neg(\exists D\ (D \in downloads \wedge D.customerid = C.customerid\ \wedge\ D.name = G.name\ \wedge\ D.version = G.version))))\}$$

This form can easily be translated into SQL.

$$\{T \mid \exists C\ (T.firstname = C.firstname\ \wedge\ T.lastname = C.lastname \wedge C \in customers \wedge \neg(\exists G\ (G \in games \wedge G.name = 'Aerified' \wedge \neg(\exists D\ (D \in downloads \wedge D.customerid = C.customerid\ \wedge\ D.name = G.name\ \wedge\ D.version = G.version)))))\}$$

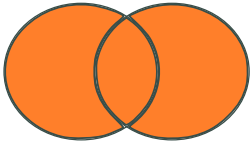
```
1  SELECT c.firstname , c.lastname
2  FROM customers c
3  WHERE NOT EXISTS (
4      SELECT *
5      FROM games g
6      WHERE g.name = 'Aerified'
7      AND NOT EXISTS (
8          SELECT *
9          FROM downloads d
10         WHERE d.customerid = c.customerid AND d.name = g.name AND d.version = g.version));
```

Relations are sets of tuples.

The main algebraïc operators are $\cup$, union, $\cap$, intersection, $\setminus$, non-symmetric difference, ρ, renaming, σ, selection, π, projection, $\times$, cross or Cartesian product, and $\bowtie$, join (θ-join or inner join), There are many more.

Union

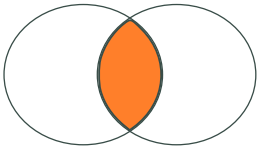
$$R_1 \cup R_2 = \{T \mid T \in R_1 \vee T \in R_2\}$$



The two relation must be union-compatible, i.e. they must have the same columns.

Intersection

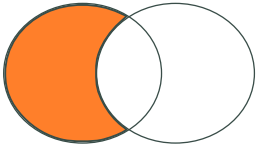
$$R_1 \cap R_2 = \{T \mid T \in R_1 \wedge T \in R_2\}$$



The two relation must be union-compatible, i.e. they must have the same columns.

Symmetric difference

$$R_1 \setminus R_2 = \{T \mid T \in R_1 \wedge \neg(T \in R_2)\}$$



The two relation must be union-compatible, i.e. they must have the same columns.

Renaming can be used to change the name of the relation, of some of its attributes, or both.

$$\rho(R_1, R_2) = R_2 = \{T \mid \exists T_1 \ (T_1 \in R_1\}$$

$$\rho(R_1, R_1(a_1 \to b_1, a_2 \to b_2)) = \cdots$$

Projection

$\pi_L(R) = \{T \mid \exists T_1 \ (T_1 \in R \ \bigwedge_{l \in L} T.l = T_1.l)\}$, where L is a set (list) of columns.

Find the names, versions, and prices of the games.

$$\pi_{g.name,g.version,g.price}(\rho(games, g))$$

```
1 SELECT g.name, g.version, g.price
2 FROM games g;
```

Selection

$\sigma_C(R) = \{T \mid T \in R \wedge C\}$, where C is a condition.

Find the customers from Singapore.

$$\sigma_{c.country='Singapore'}(\rho(customers, c))$$

```
1 SELECT *
2 FROM customers c
3 WHERE c.country = 'Singapore';
```

Find the first and last names of the customers in Singapore.

$$\pi_{c.firstname,c.lastname}(\sigma_{c.country='Singapore'}(\rho(customers, c)))$$

```
1 SELECT first_name, last_name
2 FROM customers c
3 WHERE c.country = 'Singapore';
```

Cross product, Cartesian product.

$$R_1 \times R_2 = \{T \mid \exists T_1\ \exists T_2\ (T_1 \in R_1 \wedge T_2 \in R_2 \wedge T.a_1 \cdots = T_1.a_1 \cdots \wedge T.a_n \cdots = T_1.a_n \wedge T.b_1 \cdots = T_2.b_1 \cdots \wedge T.b_m \cdots = T_2.b_m)\}.$$

Combine each row of R_1 with each row of R_2 and keep the n columns of R_1 and the m columns of R_2.

```
1 SELECT *
2 FROM R1, R2;
```

Join

$$R_1 \bowtie_C R_2 = \sigma_C(R_1 \times R_2).$$

Find who downloaded what.

$\rho(customers, c) \bowtie_{d.customerid=c.customerid}$
$\rho(downloads, d) \bowtie_{d.name=g.name \wedge d.version=g.version} \rho(games, g).$

```
1 SELECT *
2 FROM customers c, downloads  d, games g
3 WHERE d.customerid = c.customerid
4 AND d.name = g.name
5 AND d.version = g.version;
```

```
1 SELECT *
2 FROM customers c INNER JOIN downloads  d ON   d.customerid = c.customerid  INNER JOIN  games g ON d.name
      = g.name AND d.version = g.version;
```

We practice writing queries in tuple relational calculus to improve our mastery of SQL.

$$\{T \mid \exists C\ (T.firstname = C.firstname\ \wedge\ T.lastname = C.lastname \wedge C \in customers \wedge \neg(\exists G\ (G \in games\ \wedge G.name = 'Aerified' \wedge \neg(\exists D\ (D \in downloads\ \wedge D.customerid = C.customerid\ \wedge\ D.name = G.name\ \wedge\ D.version = G.version)))))\}$$

```
1  SELECT c.firstname , c.lastname
2  FROM customers c
3  WHERE NOT EXISTS (
4      SELECT *
5      FROM games g
6      WHERE g.name = 'Aerified'
7      AND NOT EXISTS (
8          SELECT *
9          FROM downloads d
10         WHERE d.customerid = c.customerid AND d.name = g.name AND d.version = g.version));
```

Relational algebra is the basis of the implementation of relational database management systems: SQL queries are translated into execution plans that orchestrate physical implementations of relational algebra operators.