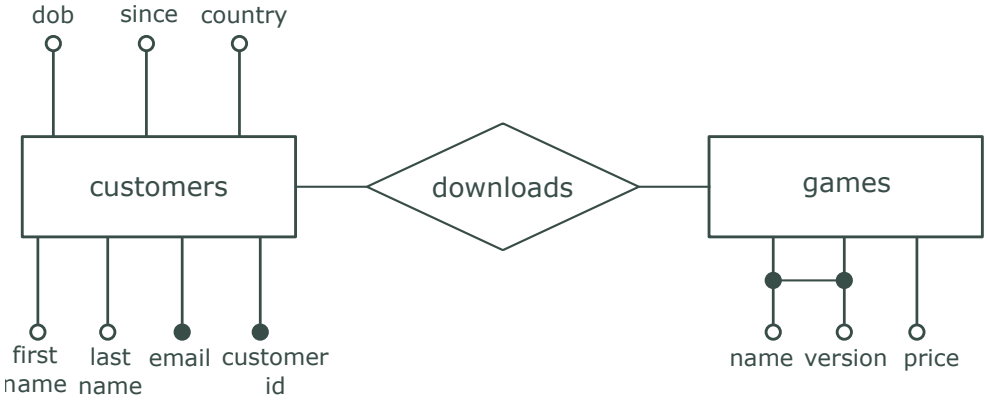
Algebraïc Queries in SQL

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
downloads
games
first name
last name
email
customer id
name
version
price

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);
```

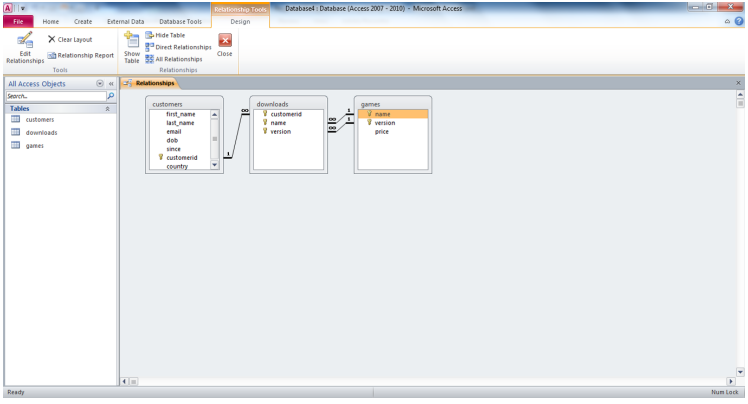
```
1 SELECT *
2 FROM customers c CROSS JOIN downloads d CROSS JOIN games g;
```

The comma in the FROM clause is a convenient shorthand for the keyword `CROSS JOIN`. Consequently, nobody uses the key word `CROSS JOIN` and uses the comma instead.

```
1 SELECT *
2 FROM customers c, downloads d, games g;
```

The result has thirteen columns and one billion eight hundred twelve million twenty thousand rows.

"," is a synonym of `CROSS JOIN`. It is the preferred join and the preferred notation.



We generally (it could be otherwise, it depends on the query) should join the tables according to their primary keys and foreign keys to make sense of the data,

We add the condition that foreign key columns are equal to the corresponding primary key columns.

```
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;
```

The result has thirteen columns and four thousand two hundred and fourteen rows, i.e. one row per download.

The INNER JOIN is a popular construct among SQL programmers.

```
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;
```

There is no added expressive power with INNER JOIN. In modern database management systems, there is no performance difference either. The query with INNER JOIN is equivalent to the query with CROSS JOIN below.

```
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 JOIN downloads d ON d.customerid = c.customerid JOIN games g ON d.name = g.name AND d.
      version = g.version;
```

`JOIN` is a synonym of `INNER JOIN`. It is not recommended to use either of them.

If we manage to give the same name to columns that are the same across tables, we can use a natural join. It joins the rows that have the same values for their columns that have the same names. It also prints one of the two equated columns.

```
1 SELECT *
2 FROM customers c NATURAL JOIN downloads d NATURAL JOIN games g;
```

The result has 10 columns (there are 10 different column names) and 4214 rows (the meaningful combinations of a row from each of the three tables corresponding to a customer downloading a game.)

```
1 SELECT  g.name, g.version, c.customerid, c.first_name, c.last_name, c.email, c.dob, c.since, c.country,
      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;
```

The outer join keeps the columns of the rows in the left (left outer join), right (right outer join) or in both (full outer join) tables that do not match anything in the other table according to the join condition and pad the remaining columns with null values.

Clearly, it is better to avoid outer joins whenever possible as they introduce null values. They can sometimes be justified by efficiency reasons.

```
1 SELECT c.customerid, c.email, d.customerid, d.name, d.version
2 FROM customers c LEFT OUTER JOIN downloads d ON c.customerid = d.customerid;
```

c.customerid	c.email	d.customerid	d.name	d.version
...				
"Willie90"	"wlongjj@moonfruit.com"	"Willie90"	"Ronstring"	"1.1"
"Willie90"	"wlongjj@moonfruit.com"	"Willie90"	"Veribet"	"2.1"
"Al8"	"ahansenp3@webnode.com"	null	null	null
"Johnny1997"	"jstevensb0@un.org"	null	null	null
...				

In the example above the customers, from the table on the left of the join, who never downloaded a game are combined with null values to replace the missing values for the columns of the `download` table. Columns from the table on the right of the join are padded with null values.

In the example below the games, from the table on the right of the join, that have never been downloaded are combined with null values to replace the missing values for the columns of the `download` table. Columns from the table on the left of the join are padded with null values.

```
1 SELECT *
2 FROM downloads d RIGHT OUTER JOIN games a ON a.name = d.name AND a.version = d.version;
```

A full outer join pads missing values with null for both the tables on the left and on the right.

The query below finds the 22 customers who never downloaded a game.

```
SELECT c.customerid
FROM customers c LEFT OUTER JOIN downloads d ON c.customerid = d.customerid
WHERE d.customerid ISNULL;
```

The query below finds the 10 customers in Singapore who never downloaded a game.

```
SELECT c.email, c.country
FROM customers c LEFT OUTER JOIN downloads d ON c.customerid = d.customerid
WHERE d.customerid ISNULL
AND c.country = 'Singapore';
```

Notice the different results between the two queries below. The second query has 619 results!

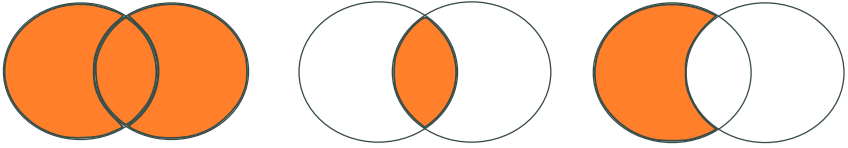
```
1 SELECT c.email, c.country
2 FROM customers c LEFT OUTER JOIN downloads d ON c.customerid = d.customerid
3 WHERE d.customerid ISNULL
4 AND c.country = 'Singapore';
```

```
1 SELECT c.email, c.country
2 FROM customers c LEFT OUTER JOIN downloads d ON c.customerid = d.customerid
3 AND c.country = 'Singapore'
4 WHERE d.customerid ISNULL;
```

Outer joins are not easy to write as conditions in the ON clause are not equivalent to conditions in the WHERE clause as it is the case with inner joins.

`RIGHT JOIN` is a synonym of `RIGHT OUTER JOIN`.
`LEFT JOIN` is a synonym of `LEFT OUTER JOIN`.
`FULL JOIN` is a synonym of `FULL OUTER JOIN`.

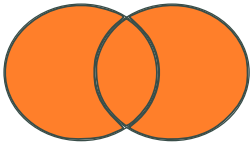
The set operators `UNION`, `INTERSECT` and `EXCEPT` (also called `MINUS` in Oracle) return the union, intersection and non-symmetric difference of the results of two queries, respectively.



The two queries must be union-compatible. They must return the same number of columns with the same domains in the same order.

Union, intersection and symmetric difference eliminate duplicates unless annotated with the keyword `ALL`.

The query below prints the identifiers of the customers who downloaded versions 1.0 or 2.0 of the game Aerified.

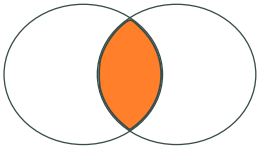


```
1 SELECT d.customerid
2 FROM downloads d
3 WHERE d.name = 'Aerified' AND  d.version = '1.0'
4 UNION
5 SELECT d.customerid
6 FROM downloads d
7 WHERE d.name = 'Aerified' AND d.version = '2.0';
```

The query below prints the names and versions of the games after GST is applied. The first query in the union applies a GST of 7% if it is more than 30 cents. The second query in the union does not apply the GST if it would be less than 30 cents.

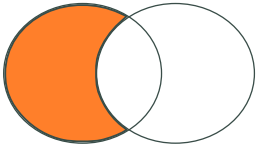
```
1 SELECT name || ' ' || version  AS game, price * 1.07
2 FROM games
3 WHERE price * 0.07 >= 0.3
4 UNION
5 SELECT name || ' ' || version  AS game, price
6 FROM games
7 WHERE price * 0.07 < 0.3;
```

The query below prints the identifiers of the customers who downloaded both versions 1.0 and 2.0 of the game Aerified.



```sql
SELECT d.customerid
FROM downloads d
WHERE d.name = 'Aerified' AND  d.version = '1.0'
INTERSECT
SELECT d.customerid
FROM downloads d
WHERE d.name = 'Aerified' AND d.version = '2.0';
```

The query below prints the identifiers of the customers who downloaded version 1.0 but not version 2.0 of the game Aerified.



```
1 SELECT d.customerid
2 FROM downloads d
3 WHERE d.name = 'Aerified' AND  d.version = '1.0'
4 EXCEPT
5 SELECT d.customerid
6 FROM downloads d
7 WHERE d.name = 'Aerified' AND d.version = '2.0';
```

What does this query find? (in English).

```
1  SELECT c.email, SUM(g.price)
2  FROM customers c, downloads d, games g
3  WHERE c.customerid = d.customerid AND g.name = d.name AND g.version = d.version
4  AND c.country = 'Indonesia' AND g.name = 'Fixflex'
5  GROUP BY c.email
6  UNION
7  SELECT c.email, 0
8  FROM customers c LEFT OUTER JOIN
9  (downloads d INNER JOIN games g
10 ON  g.name = d.name AND g.version = d.version AND g.name = 'Fixflex')
11 ON c.customerid = d.customerid
12 WHERE c.country = 'Indonesia' AND d.name ISNULL;
```

email	sum
jmurrayhg@printfriendly.com	2.99
jstevensb0@un.org	0
rhendersonh2@gmpg.org	0
...