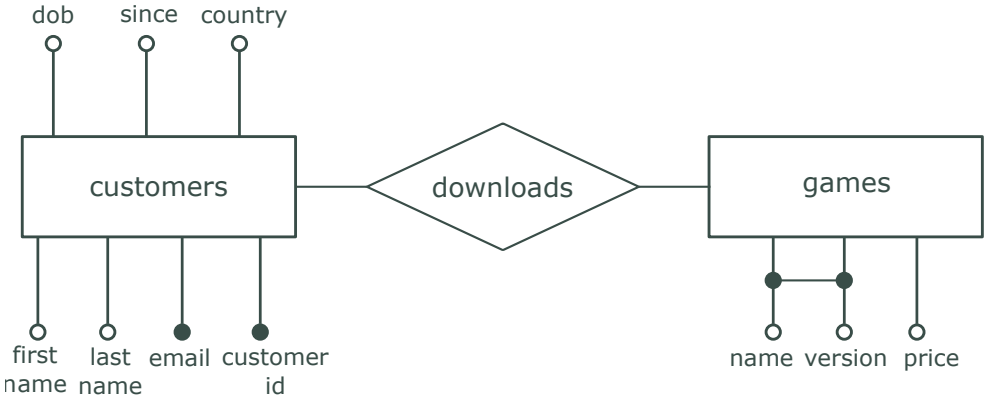
Aggregate Queries in SQL

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

We can use SQLite interactive terminal `sqlite3.exe`.

```
sqlite> .open myfile.db
sqlite> .mode column
sqlite> .headers on
sqlite> PRAGMA foreign_keys = ON;
sqlite> .read AppStoreSchema.sql
sqlite> .read AppStoreCustomers.sql
sqlite> .read AppStoreGames.sql
sqlite> .read AppStoreDownloads.sql
...
sqlite> .quit
```

We can use PostgreSQL interactive terminal `psql`.

```
% psql -h localhost -U postgres
Password for user postgres:
psql (9.6.3, server 9.6.4)
Type "help" for help.
postgres=# CREATE DATABASE dedomenology;
CREATE DATABASE
postgres=# \c dedomenology;
psql (9.6.3, server 9.6.4)
You are now connected to database "dedomenology" as user "postgres".
postgres=# \i AppStoreSchema.sql
CREATE TABLE
CREATE TABLE
CREATE TABLE
postgres=#  \i AppStoreCustomers.sql
INSERT 0 1
...
postgres=# \i AppStoreGames.sql
INSERT 0 1
...
postgres=# \i AppStoreDownloads.sql
INSERT 0 1
...
postgres=# \q
```

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 values of a column can be aggregated aggregation functions such as `COUNT()`, `SUM()`, `MAX()`, `MIN()`, `AVG()`, `STDDEV()` etc.
PostgreSQL also allows User-defined aggregate functions.

```
1 SELECT COUNT(*)
2 FROM customers;
```

count
1000

The above query prints the number of rows in the table `customers`.

```
1 SELECT COUNT(c.customerid)
2 FROM customers c;
```

count
1000

The above query prints the number of rows in the table `customers`.

```
1 SELECT COUNT(c.country)
2 FROM customers c;
```

count
1000

The above query also prints the number of rows in the table `customers`.

```sql
SELECT COUNT(c.country)
FROM customers c;
```

```sql
SELECT COUNT(ALL c.country)
FROM customers c;
```

count
1000

The two queries above are the same.
The keyword `ALL` is generally omitted as it is the default.

```
1 SELECT COUNT(DISTINCT c.country)
2 FROM customers c;
```

count
5

We need to add the keyword `DISTINCT` inside the `COUNT()` aggregate function if we want to count the number of different countries in the column `country` of the table `customers`.
`DISTINCT` can be used in other aggregate functions similarly.

The following query finds the maximum, minimum, average and standard deviation prices of our games.
It uses the arithmetic function `TRUNC()` to display two decimal places for average and standard deviation.

```
1 SELECT MAX(g.price),
2 MIN(g.price),
3 TRUNC(AVG(g.price), 2) AS ave,
4 TRUNC(STDDEV(g.price),2) AS std
5 FROM games g;
```

max	min	avg	std
12	1.99	6.97	3.96

The GROUP BY clause creates groups of records that have the same values for the specified fields before computing the aggregate functions.

first_name	last_name	email	...	country
"Deborah"	"Ruiz"	"druiz0@drupal.org"	...	"Singapore"
"Tammy"	"Lee"	"tlee1@barnesandnoble.com"	...	"Singapore"
...				
"Raymond"	"Tan"	"rtan1z@nature.com "	...	"Thailand"
"Jean"	"Ling"	"jlingpn@walmart.com"	...	"Thailand"
...				
"Russel"	"Hakim"	"rhakim7y@si.edu"	...	"Vietnam"
"Gerald"	"Ford"	"gfordij@zdnet.com"	...	"Vietnam"
...				
"Marie"	"Flores"	"mfloresk2@sogou.com"	...	"Indonesia"
"Cheryl"	"Reyes"	"creyesjl@jalbum.net"	...	"Indonesia"
...				
"Cynthia"	"Pierce"	"cpierce25@prlog.org"	...	"Malaysia"
"Nicole"	"Lee"	"nleef1@whitehouse.gov "	...	"Malaysia"
...				

```
1 GROUP BY c.country;
```

The aggregation functions are calculated for each group.

```
1 SELECT c.country, COUNT(*)
2 FROM customers c
3 GROUP BY c.country;
```

country	count
"Vietnam"	98
"Singapore"	391
"Thailand"	100
"Indonesia"	243
"Malaysia"	168

```
1 SELECT COUNT(*)
2 FROM customers c
```

If no GROUP BY clause is specified only one group is formed as soon as one aggregate function is used.

Groups are formed after the rows have been filtered by the WHERE clause.

```
1 SELECT c.country, COUNT(*)
2 FROM customers c
3 WHERE c.dob >= '2000-01-01'
4 GROUP BY c.country;
```

country	count
"Vietnam"	4
"Singapore"	25
"Thailand"	5
"Indonesia"	15
"Malaysia"	12

The following query finds the total spending for each customer.

```
1 SELECT c.customerid, c.first_name, c.last_name, SUM(g.price)
2 FROM customers c, downloads d, games g
3 WHERE c.customerid = d.customerid
4 AND d.name = g.name AND d.version = g.version
5 GROUP BY c.customerid, c.first_name, c.last_name;
```

fist_name	last_name	sum
"Adam"	"Howell"	28.98
"Jimmy"	"Gibson"	20.99
"Margaret"	"Mitchell"	13.99
...		

Note that we include the columns `first_name` and `last_name` in the GROUP BY clause because we want to print them.

It is recommended (and required according to the SQL standard) to include the attributes projected in the SELECT clause in the GROUP BY clause.

```
1 SELECT c.first_name, c.last_name, SUM(g.price)
2 FROM customers c, downloads d, games g
3 WHERE c.customerid = d.customerid
4 AND d.name = g.name AND d.version = g.version
5 GROUP BY c.customerid;
```

The above query works only because the first and last name are guaranteed to be unique for a given customer identifier (which is the primary key of the table `customers`).

Do not write such queries for the sake of readability and portability.

We should write the query as follows, making sure that every column mentioned in the SELECT clause is mentioned in the GROUP BY unless it is used in an aggregate function.

```
1 SELECT c.first_name, c.last_name, SUM(g.price)
2 FROM customers c, downloads d, games g
3 WHERE c.customerid = d.customerid
4 AND d.name = g.name AND d.version = g.version
5 GROUP BY c.customerid, c.first_name, c.last_name;
```

The query below does not work in PostgreSQL (it does work in SQLite but such queries could give wrong results).

```
1 SELECT c.customerid, c.first_name, c.last_name, SUM(g.price)
2 FROM customers c, downloads d, games g
3 WHERE c.customerid = d.customerid
4 AND d.name = g.name AND d.version = g.version
5 GROUP BY c.first_name, c.last_name;
```

```
1 ERROR:  column "c.customerid" must appear in the GROUP BY clause or be used in an aggregate function
2 LINE 1: SELECT c.customerid, c.first_name, c.last_name, SUM(g.price)
3                ^
4 SQL state: 42803
5 Character: 8
```

The following query displays the number of downloads by country of registration and year of birth of the customers. `EXTRACT()` is a PostgreSQL function. `STRFTIME()` is a SQLite function.

```
SELECT c.country, EXTRACT(YEAR FROM c.since) AS regyear, COUNT(*) AS total
FROM customers c, downloads d
WHERE c.customerid = d.customerid
GROUP BY c.country, regyear
ORDER BY regyear, c.country;
```

```
SELECT c.country, STRFTIME('%Y', c.since) AS regyear, COUNT(*) AS total
FROM customers c, downloads d
WHERE c.customerid = d.customerid
GROUP BY c.country, regyear
ORDER BY regyear, c.country;
```

country	regyear	count
...		
"Thailand"	"2015"	6
"Vietnam"	"2015"	3
"Indonesia"	"2016"	998
"Malaysia"	"2016"	713
...		

The order of columns in the GROUP BY clause does not change the meaning of the query.

```
1 SELECT c.country, EXTRACT(YEAR FROM c.since) AS regyear, COUNT(*) AS total
2 FROM customers c, downloads d
3 WHERE c.customerid = d.customerid
4 GROUP BY regyear, c.country
5 ORDER BY regyear, c.country;
```

```
1 SELECT c.country, STRFTIME('%Y', c.since)  AS regyear, COUNT(*) AS total
2 FROM customers c, downloads d
3 WHERE c.customerid = d.customerid
4 GROUP BY regyear, c.country
5 ORDER BY regyear, c.country;
```

country	regyear	count
...		
"Thailand"	"2015"	6
"Vietnam"	"2015"	3
"Indonesia"	"2016"	998
"Malaysia"	"2016"	713
...		

Aggregate functions can be used in conditions.

```
1 SELECT c.country
2 FROM customers c
3 WHERE COUNT(*) >= 100
4 GROUP BY c.country;
```

```
1 ERROR:  aggregate functions are not allowed in WHERE
2 LINE 4: WHERE COUNT(*) >= 100
3               ^
4 SQL state: 42803
5 Character: 42
```

However aggregate functions are not allowed in the WHERE clause. This is because they can only be evaluated after the groups are formed. The groups are formed after rows are filtered by the WHERE clause.

Instead, we use a new clause: the HAVING clause to add conditions to be checked after the evaluation of the GROUP BY clause.

The HAVING clause can only involve aggregate functions, columns listed in the GROUP BY clause and subqueries.

The following query finds the countries in which there are more than 100 customers.

```
1 SELECT c.country
2 FROM customers c
3 GROUP BY c.country
4 HAVING COUNT(*) >= 100;
```

country
"Singapore"
"Thailand"
"Indonesia"
"Malaysia"