Advanced SQL II

Lætitia Avrot

Lætitia Avrot

- Field CTO – EDB
- PostgreSQL Europe Treasurer
- Postgres Women Founder
- Recognized contributor to the PostgreSQL project

Summary of previous episodes

SQL is...

- A declarative language
- Turing-complete
- Unknown
- Based originally on relational algebra

NULL...

- Marks there is no value
- Exists for all datatypes

NULL is not...

- An empty string
- A string with spaces
- The string 'NULL'

Quizz!

How many kind of joins do exist in SQL?

- 2
- 4
- 7
- 12

There are 7 kinds of joins:

- Inner join
- Left or right outer join
- Full outer join
- Cartesian product
- Natural join
- Lateral join
- Anti join

Do we need to add a group by to this query?

```
select
  name,
  avg(salary)
from employeeHistory
```

```
select
  name,
  avg(salary)
from employeeHistory
group by name
```

`Group by` is not implicit in SQL.
You can always group on more columns than the one listed in the select.
You can't omit one of the non aggregated columns from the select in the `group by`.

How can we filter an aggregate result?

- In the where clause
- In the select clause
- In the having clause
- All of the above

```
select
  name,
  avg(salary)
from employeeHistory
group by name
having avg(salary) < 10000
```

`Having` was invented for that purpose. It is the simplest way of filtering an aggregation result.

```
With avgSalary(name, avgSalary) as (
  select
    name,
    avg(salary)
  from employeeHistory
  group by name
)
select *
from avgSalary
where avgSalary < 10000
```

A little overcomplicated, but doable.

```
with avgSalary(name, avgSalary) as (
  select
    name,
    avg(salary)
  from employeeHistory
  group by name
)
select name filter (where avgSalary < 10000),
  avgSalary (where avgSalary < 10000)
from avgSalary
```

Way too complicated.

How can you create an auto incremented column in PostgreSQL?

- Manually with a sequence
- Automatically with a sequence and a default value
- Automatically with the `serial` datatype
- Automatically with a generated column
- All of the above

```
laetitia=# create table test(id integer primary
key, value text);
CREATE TABLE
laetitia=# create sequence my_seq;
CREATE SEQUENCE
laetitia=# insert into test (select
nextval('my_seq'), 'blabla');
INSERT 0 1
```

Manually with a sequence

```
laetitia=# create sequence my_seq;
CREATE SEQUENCE
laetitia=# create table test (id integer
default nextval('my_seq') primary key,
  value text);
CREATE TABLE
laetitia=# insert into test(value) values
('blabla');
INSERT 0 1
```

Automatically with a sequence and a default value

```
laetitia=# create table test (id serial primary
key, value text);
CREATE TABLE
laetitia=# insert into test (value) values
('blabla');
INSERT 0 1
```

Automatically with the `serial` datatype

```
laetitia=# create table test (id integer generated by default as identity primary
key, value text);
CREATE TABLE
laetitia=# \d test
                         Table "public.test"
 Column |  Type   | Collation | Nullable |             Default
--------+---------+-----------+----------+----------------------------------
 id     | integer |           | not null | generated by default as identity
 value  | text    |           |          |
Indexes:
    "test_pkey" PRIMARY KEY, btree (id)

laetitia=# insert into test (value) values ('blabla');
INSERT 0 1
```

Automatically with a generated column

```
laetitia=# insert into test (id, value) values
(2,'blabla');

ERROR:  cannot insert a non-DEFAULT value into
column "id"
DETAIL:  Column "id" is an identity column
defined as GENERATED ALWAYS.
HINT:  Use OVERRIDING SYSTEM VALUE to override.
```

A bonus from generated columns

	Sequence	Serial	Identity column
Nextval automatically as default value	No	Yes	Yes
Not null constraint	No	Yes	Yes
Prevent manual inserts	No	No	With `always`

Why should you use CTEs (Common Table Expressions)?

- To show off in front of developers
- To make your code more readable
- To confuse the optimiser
- All of the above

```
with CTEName1 (list of CTE columns) as (
...
),
CTEName2 (list of CTE columns) as (
...
)
Select columnsName
From CTEName2
```

`Not in` is often faster than `Not exist`

- True
- False

```
select surname,
  firstname
from members
where memid not in
  (
   select memid
   from bookings
  )
```

Not in example

```sql
select surname,
  firstname
from members
where memid not exist
  (
   select 1
   from bookings
   where members.memid = bookings.memid
)
```

Not exist example

What is the difference between cube and rollup?

- Rollup is hierarchic while Cube takes a combination of all columns
- Cube, contrary to Rollup, needs a `grouping set`
- Cube has not real use case, contrary to Rollup
- All of the above

`Rollup` allows hierarchic aggregations, so that not all combinations of columns will be displayed.

`Cube` will calculate the aggregation for all possible combinations of the columns.

```sql
select
  coalesce (department, 'All Departments') as Department,
  coalesce (gender,'All Genders') as Gender,
  sum(salary) as Salary_Sum
from employee
Group by rollup (department, gender)
```

Department	**Gender**	**Salary_Sum**
Finance	Female	11800
Finance	Male	5000
Finance	**All Genders**	16800
HR	Female	6000
HR	Male	14200
HR	**All Genders**	20200
All Departments	**All Genders**	37000

```
select
  coalesce (department, 'All Departments') as Department,
  coalesce (gender,'All Genders') as Gender,
  sum(salary) as Salary_Sum
from employee
Group by cube (department, gender)
```

Row No	Department	Gender	Salary_Sum
1	Finance	Female	11800
2	HR	Female	6000
5	All Departments	Female	17800
6	Finance	Male	5000
7	HR	Male	14200
10	All Departments	Male	19200
11	All Departments	All Genders	37000

Some resources

(To get better in SQL)

- https://mystery.knightlab.com/
- https://pgexercises.com/
- https://modern-sql.com/
- https://theartofpostgresql.com/

EDB™

Thank you!