Database roles and access control

DATABASE DESIGN

Lis Sulmont
Curriculum Manager

DataCamp

Granting and revoking access to a view

`GRANT privilege(s)` or `REVOKE privilege(s)`

`ON object`

`TO role` or `FROM role`

- **Privileges:** `SELECT`, `INSERT`, `UPDATE`, `DELETE`, etc.
- **Objects:** table, view, schema, etc.
- **Roles:** a database user or a group of database users

```
GRANT UPDATE ON ratings TO PUBLIC;
REVOKE INSERT ON films FROM db_user;
```

Database roles

- Manage database access permissions
- A database role is an entity that contains information that:
  - Define the role's privileges
    - Can you login?
    - Can you create databases?
    - Can you write to tables?
  - Interact with the client authentication system
    - Password
- Roles can be assigned to one or more users
- Roles are global across a database cluster installation

Create a role

- Empty role

```
CREATE ROLE data_analyst;
```

  - Roles with some attributes set

```
CREATE ROLE intern WITH PASSWORD 'PasswordForIntern' VALID UNTIL '2020-01-01';
```

```
CREATE ROLE admin CREATEDB;
```

```
ALTER ROLE admin CREATEROLE;
```

[1] http://bit.ly/postgresql_attributes

GRANT and REVOKE privileges from roles

```
GRANT UPDATE ON ratings TO data_analyst;
```

```
REVOKE UPDATE ON ratings FROM data_analyst;
```

The available privileges in PostgreSQL are:

- `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, `REFERENCES`, `TRIGGER`, `CREATE`, `CONNECT`, `TEMPORARY`, `EXECUTE`, and `USAGE`

[1] http://bit.ly/postgresql_privileges

Users and groups (are both roles)

- A role is an entity that can function as a user and/or a group
  - User roles
  - Group roles

Users and groups (are both roles)

Group role

```
CREATE ROLE data_analyst;
```

User role

```
CREATE ROLE intern WITH PASSWORD 'PasswordForIntern' VALID UNTIL '2020-01-01';
```

Users and groups (are both roles)

Group role

```
CREATE ROLE data_analyst;
```

User role

```
CREATE ROLE alex WITH PASSWORD 'PasswordForIntern' VALID UNTIL '2020-01-01';
```

```
GRANT data_analyst TO alex;
```

```
REVOKE data_analyst FROM alex;
```

Common PostgreSQL roles

Role	Allowed access
pg_read_all_settings	Read all configuration variables, even those normally visible only to superusers.
pg_read_all_stats	Read all pg_stat_* views and use various statistics related extensions, even those normally visible only to superusers.
pg_signal_backend	Send signals to other backends (eg: cancel query, terminate).
More...	More...

[1] http://bit.ly/default_roles_postgresql

Benefits and pitfalls of roles

Benefits

- Roles live on after users are deleted
- Roles can be created before user accounts
- Save DBAs time

Pitfalls

- Sometimes a role gives a specific user too much access
  - You need to pay attention

Let's practice!

DATABASE DESIGN

DataCamp

Table partitioning

DATABASE DESIGN

Lis Sulmont
Curriculum Manager

DataCamp

Why partition?

Tables grow (100s Gb / Tb)

Problem: queries/updates become slower

Because: e.g., indices don't fit memory

Solution: split table into smaller parts (= **partitioning**)

Data modeling refresher

1. Conceptual data model

2. Logical data model

For partitioning, logical data model is the same

3. Physical data model

Partitioning is part of physical data model

Vertical partitioning

Key	Column 1	Column 2	Column 3	Column 4
1	...	...	...	...
2	...	...	...	...

Key	Column 1	Column 2	Column 3
1	...	...	...
2	...	...	...

Key	Column 4
1	...
2	...

*Split table even when fully **normalized***

Vertical partitioning: an example

id	name	short_description	price	long_descriptio
1	Sunglasses	Cool sunglasses	$25	When you put on
2	Leather wallet	Leather wallet with writing	$127	The wallet says 'B

id	name	short_description	price
1	Sunglasses	Cool sunglasses	$25
2	Leather wallet	Leather wallet with writing	$127

id	long_description
1	When you put on these ..
2	The wallet says 'Bad ...

E.g., store `long_description` on slower medium

Horizontal partitioning

Key	Column 1	timestamp
1	...	2018-01-01
2	...	...
3	...	2018-31-01
4	...	2019-01-01
5	...	...
6	...	2019-31-01

Key	Column 1	timestamp
1	...	2018-01-01
2	...	...
3	...	2018-31-01

Key	Column 1	timestamp
4	...	2019-01-01
5	...	...
6	...	2019-31-01

Horizontal partitioning: an example

id	product_id	amount	total_price	timestan
1	123	1	$102	2019-04-(
2	101	7	$21	2019-23-(
3	18202	1	$499	2019-30-(
4	1762	15	$1500	2019-21-(
5	10	1	$5	2019-30-(
6	123	1	$102	2019-29-(

Horizontal partitioning: an example

id	product_id	amount	total_price	timestan
1	123	1	$102	2019-04-
2	101	7	$21	2019-23-
3	18202	1	$499	2019-30-
4	1762	15	$1500	2019-21-
5	10	1	$5	2019-30-
6	123	1	$102	2019-29-

Q1 partition

Q2 partition

Q3 partition

Q4

```
CREATE TABLE sales (
    ...
    timestamp DATE NOT NULL
)
PARTITION BY RANGE (timestamp);

CREATE TABLE sales_2019_q1 PARTITION OF sales
    FOR VALUES FROM ('2019-01-01') TO ('2019-03-31');
...
CREATE TABLE sales_2019_q4 PARTITION OF sales
    FOR VALUES FROM ('2019-09-01') TO ('2019-12-31');
CREATE INDEX ON sales ('timestamp');
```

Pros/cons of horizontal partitioning

Pros

- Indices of **heavily-used partitions** fit in memory
- Move to **specific medium**: slower vs. faster
- Used for both OLAP as OLTP

Cons

- Partitioning **existing table** can be a hassle
- Some **constraints** can not be set

Relation to sharding

Let's practice!

DATABASE DESIGN

DataCamp

Data integration

DATABASE DESIGN

Lis Sulmont
Curriculum Manager

DataCamp

What is data integration

Data Integration combines data from different sources, formats, technologies to provide users with a translated and unified view of that data.

Business case examples

- 360-degree customer view
- Acquisition
- Legacy systems

Unified data model

Data sources

Data sources format

Unified data model format

Example: DataCamp

Update cadence - sales

Update cadence - air traffic

Different update cadences

So simple?

DATA SOURCE 1
(PostgreSQL)
HOURLY
DATA SOURCE 2
(MongoDB)
WEEKLY
DATA SOURCE 3
(CSV)
REAL
TIME
UNIFIED
DATA MODEL
(Redshift)

Not really

DATA SOURCE 1
(PostgreSQL)
HOURLY
DATA SOURCE 2
(MongoDB)
WEEKLY
DATA SOURCE 3
(CSV)
REAL
TIME
UNIFIED
DATA MODEL
(Redshift)

Transformations

Transformation - tools

Choosing a data integration tool

- Flexible
- Reliable
- Scalable

Automated testing and proactive alerts

Security

DATA SOURCE 1
(PostgreSQL)
H
DATA SOURCE 2
(MongoDB)
W
DATA SOURCE 3
(CSV)
RT
ETL
UNIFIED
DATA MODEL
(Redshift)

Security - credit card anonymization

Data governance - lineage

Let's practice!

DATABASE DESIGN

DataCamp

Picking a Database Management System (DBMS)

DATABASE DESIGN

Lis Sulmont
Curriculum Manager

DataCamp

DBMS

- **DBMS**: DataBase Management System
- Create and maintain databases
  - Data
  - Database schema
  - Database engine
- Interface between database and end users

DBMS types

- Choice of DBMS depends on database type
- Two types:
  - **SQL DBMS**
  - **NoSQL DBMS**

SQL DBMS

- Relational DataBase Management System (**RDBMS**)
- Based on the relational model of data
- Query language: SQL
- Best option when:
  - Data is structured and unchanging
  - Data must be consistent

NoSQL DBMS

- Less structured
- Document-centered rather than table-centered
- Data doesn't have to fit into well-defined rows and columns
- Best option when:
  - Rapid growth
  - No clear schema definitions
  - Large quantities of data
- Types: key-value store, document store, columnar database, graph database

NoSQL DBMS - key-value store

- Combinations of keys and values
  - Key: unique identifier
  - Value: anything
- **Use case**: managing the shopping cart for an on-line buyer
- Example:

redis

NoSQL DBMS - document store

- Similar to key-value
- Values (= documents) are structured
- **Use case**: content management
- Example:

mongoDB®

NoSQL DBMS - columnar database

- Store data in columns
- Scalable
- **Use case**: big data analytics where speed is important
- Example:

cassandra

NoSQL DBMS - graph database

- Data is interconnected and best represented as a graph
- **Use case**: social media data, recommendations
- Example:

neo4j

Choosing a DBMS

Let's practice!

DATABASE DESIGN

DataCamp