Optimizing for Excellence: Advanced Performance Tuning Techniques for Aurora Postgres

And an interesting application localizing PII with Aurora and Foreign Data Wrappers

Dane Falkner
Postgres Conference 2024

Dane Falkner

https://www.linkedin.com/in/dfalkner/

- RiskRecon (a Mastercard Company) — Administration, tuning, management, design, and refactoring of over 20 highly available Amazon Aurora databases as large as twenty terabytes across regions and countries supporting cyber intelligence functions and services

- Surgeworks, Inc. — Data Warehouse Design and Implementation for multiple regional banks and countless database engineering consultation engagements over four decades

- Founding member of the Agile Alliance, Independent Signatory, first Chairperson for DSDM (Dynamic Systems Development Method) in North America. Signing of the Agile Manifesto…I was there.

Goals

- Provide an overview of the levers to achieve high performance
- Brief overview of Aurora architecture as it relates to performance
- Introduction to key monitoring tools
- Overview of configuration parameters
- An interesting, for some, application of Aurora Postgres for regional separation of PII using global and regional clusters and foreign data wrappers (FDW)

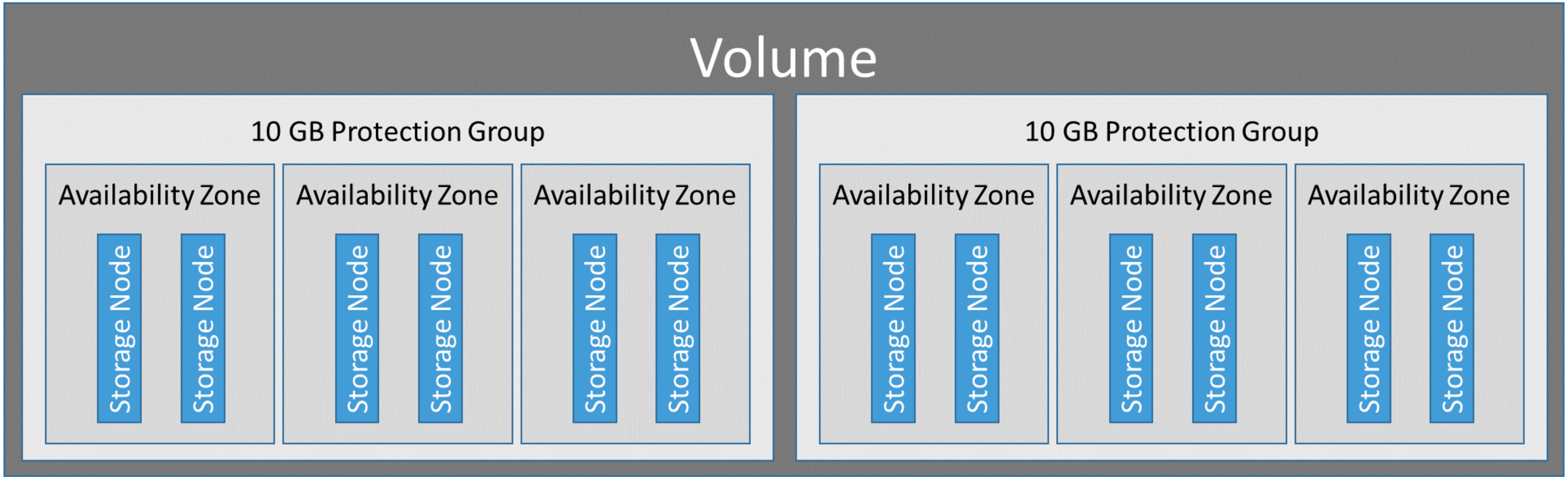
Your levers

(we won’t cover all of this today)

- **Instance size and type** -> compute, memory, network
- **Configuration** -> parameters (cluster level and instance level)
- Aurora **wait states** and performance monitoring/tuning
- **Schema** and SQL Query optimization for Aurora architecture
- ***Indexes****, their types, optimized for their usage mindful of Aurora architecture*
- ***Memory management*** *(sessions, work_mem, buffers, etc.)*

Aurora Storage

- Conceptually, a SAN distributed across three AWS Availability Zones (AZs) ***decoupled*** from compute
- Protection groups — 10 GB logical blocks are replicated between six storage nodes allocated across three AZs.
- Writes are sent to six storage nodes in parallel (complete with 4/6 nodes ack) .
- Reads are satisfied by 3/6 nodes, but more often only 1



Understanding Aurora Postgres Architecture

A few more performance enhancing features

- Buffer pool lives in separate address space from server so more shared buffer space and fast recovery
- Shared buffers are 75% of RAM in Aurora vs 25% in RDS for a given instance
- Cluster cache management (apg_ccm_enabled=on) keeps replica cache hot
- Query Plan Management (QPM)

AWS DB Instances

Aurora Serverless v2

- Aurora adjusts the compute, memory, and network resources dynamically as the workload changes.

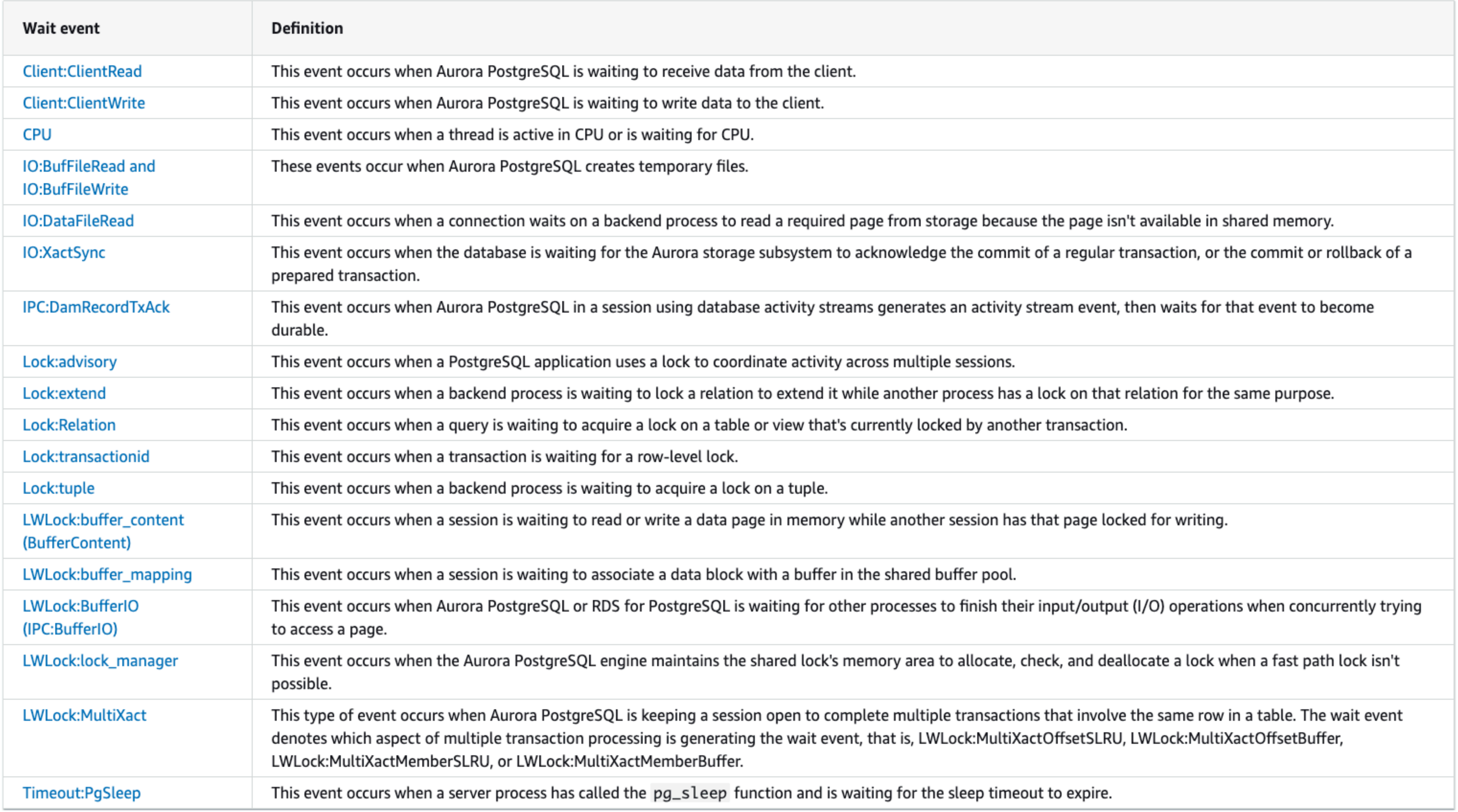
Memory-optimized R family instance class types

- **db.r7g** — AWS Graviton3 processors
- **db.r6g** — AWS Graviton2 processors
- **db.r6i** — 3rd Generation Intel Xeon Scalable processors
- **db.r5** — Intel Xeon Platinum

vCPUs, ***Memory***, and ***Network*** performance are based on size ranging from **.large** to **.32xlarge**

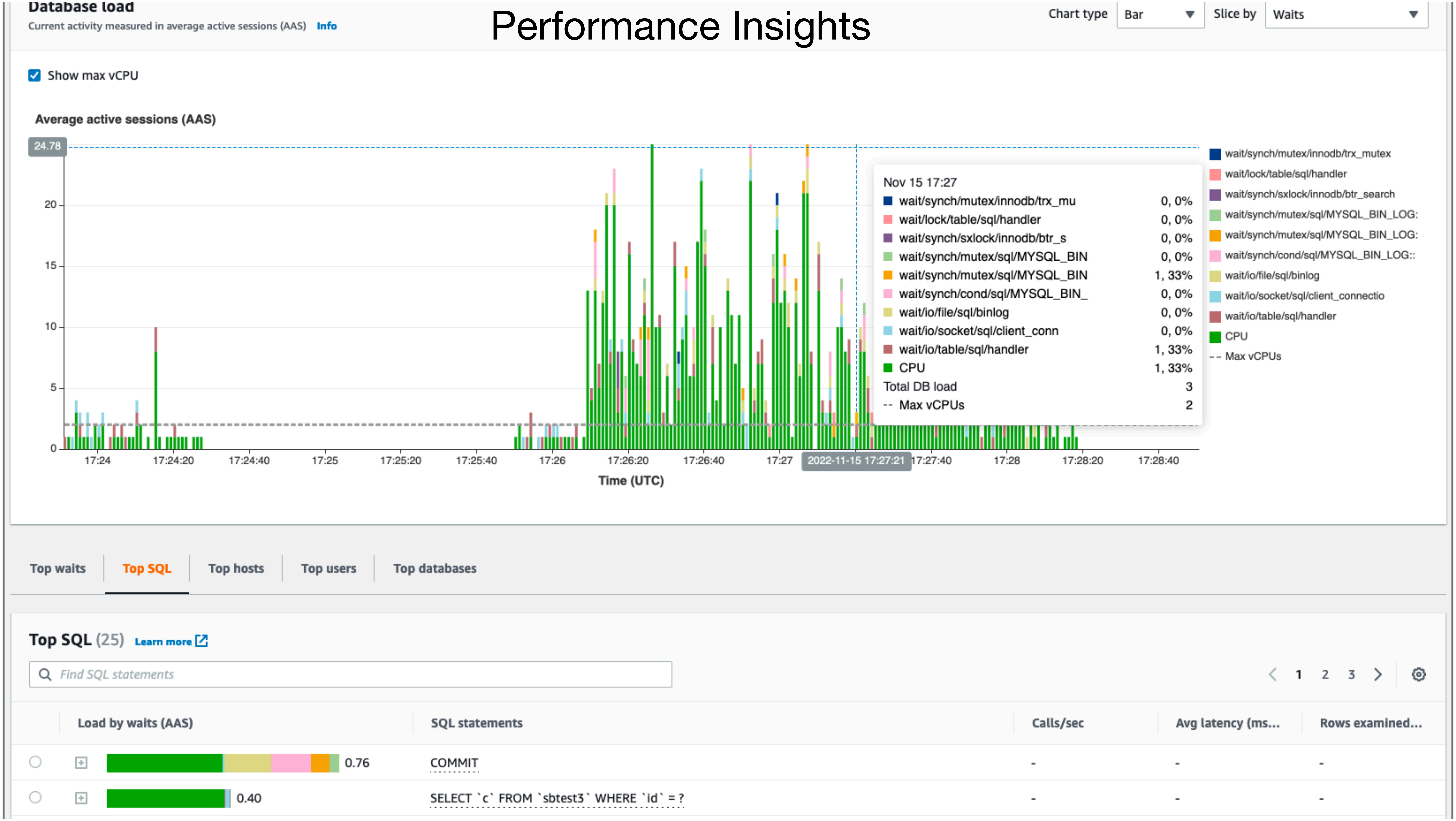
See https://aws.amazon.com/rds/instance-types/

Aurora Postgres wait events



Wait event	Definition
Client:ClientRead	This event occurs when Aurora PostgreSQL is waiting to receive data from the client.
Client:ClientWrite	This event occurs when Aurora PostgreSQL is waiting to write data to the client.
CPU	This event occurs when a thread is active in CPU or is waiting for CPU.
IO:BufFileRead and IO:BufFileWrite	These events occur when Aurora PostgreSQL creates temporary files.
IO:DataFileRead	This event occurs when a connection waits on a backend process to read a required page from storage because the page isn't available in shared memory.
IO:XactSync	This event occurs when the database is waiting for the Aurora storage subsystem to acknowledge the commit of a regular transaction, or the commit or rollback of a prepared transaction.
IPC:DamRecordTxAck	This event occurs when Aurora PostgreSQL in a session using database activity streams generates an activity stream event, then waits for that event to become durable.
Lock:advisory	This event occurs when a PostgreSQL application uses a lock to coordinate activity across multiple sessions.
Lock:extend	This event occurs when a backend process is waiting to lock a relation to extend it while another process has a lock on that relation for the same purpose.
Lock:Relation	This event occurs when a query is waiting to acquire a lock on a table or view that's currently locked by another transaction.
Lock:transactionid	This event occurs when a transaction is waiting for a row-level lock.
Lock:tuple	This event occurs when a backend process is waiting to acquire a lock on a tuple.
LWLock:buffer_content (BufferContent)	This event occurs when a session is waiting to read or write a data page in memory while another session has that page locked for writing.
LWLock:buffer_mapping	This event occurs when a session is waiting to associate a data block with a buffer in the shared buffer pool.
LWLock:BufferIO (IPC:BufferIO)	This event occurs when Aurora PostgreSQL or RDS for PostgreSQL is waiting for other processes to finish their input/output (I/O) operations when concurrently trying to access a page.
LWLock:lock_manager	This event occurs when the Aurora PostgreSQL engine maintains the shared lock's memory area to allocate, check, and deallocate a lock when a fast path lock isn't possible.
LWLock:MultiXact	This type of event occurs when Aurora PostgreSQL is keeping a session open to complete multiple transactions that involve the same row in a table. The wait event denotes which aspect of multiple transaction processing is generating the wait event, that is, LWLock:MultiXactOffsetSLRU, LWLock:MultiXactOffsetBuffer, LWLock:MultiXactMemberSLRU, or LWLock:MultiXactMemberBuffer.
Timeout:PgSleep	This event occurs when a server process has called the `pg_sleep` function and is waiting for the sleep timeout to expire.

Performance Insights



Database load
Current activity measured in average active sessions (AAS) Info
Chart type Bar ▼
Slice by Waits ▼
☑ Show max vCPU
Average active sessions (AAS)
24.78
20
15
10
5
0
17:24
17:24:20
17:24:40
17:25
17:25:20
17:25:40
17:26
17:26:20
17:26:40
17:27
2022-11-15 17:27:21
17:27:40
17:28
17:28:20
17:28:40
Time (UTC)
wait/synch/mutex/innodb/trx_mutex
wait/lock/table/sql/handler
wait/synch/sxlock/innodb/btr_search
wait/synch/mutex/sql/MYSQL_BIN_LOG:
wait/synch/mutex/sql/MYSQL_BIN_LOG:
wait/synch/cond/sql/MYSQL_BIN_LOG::
wait/io/file/sql/binlog
wait/io/socket/sql/client_connectio
wait/io/table/sql/handler
CPU
-- Max vCPUs
Top waits
Top SQL
Top hosts
Top users
Top databases
Learn more
Find SQL statements
1 2 3

Nov 15 17:27

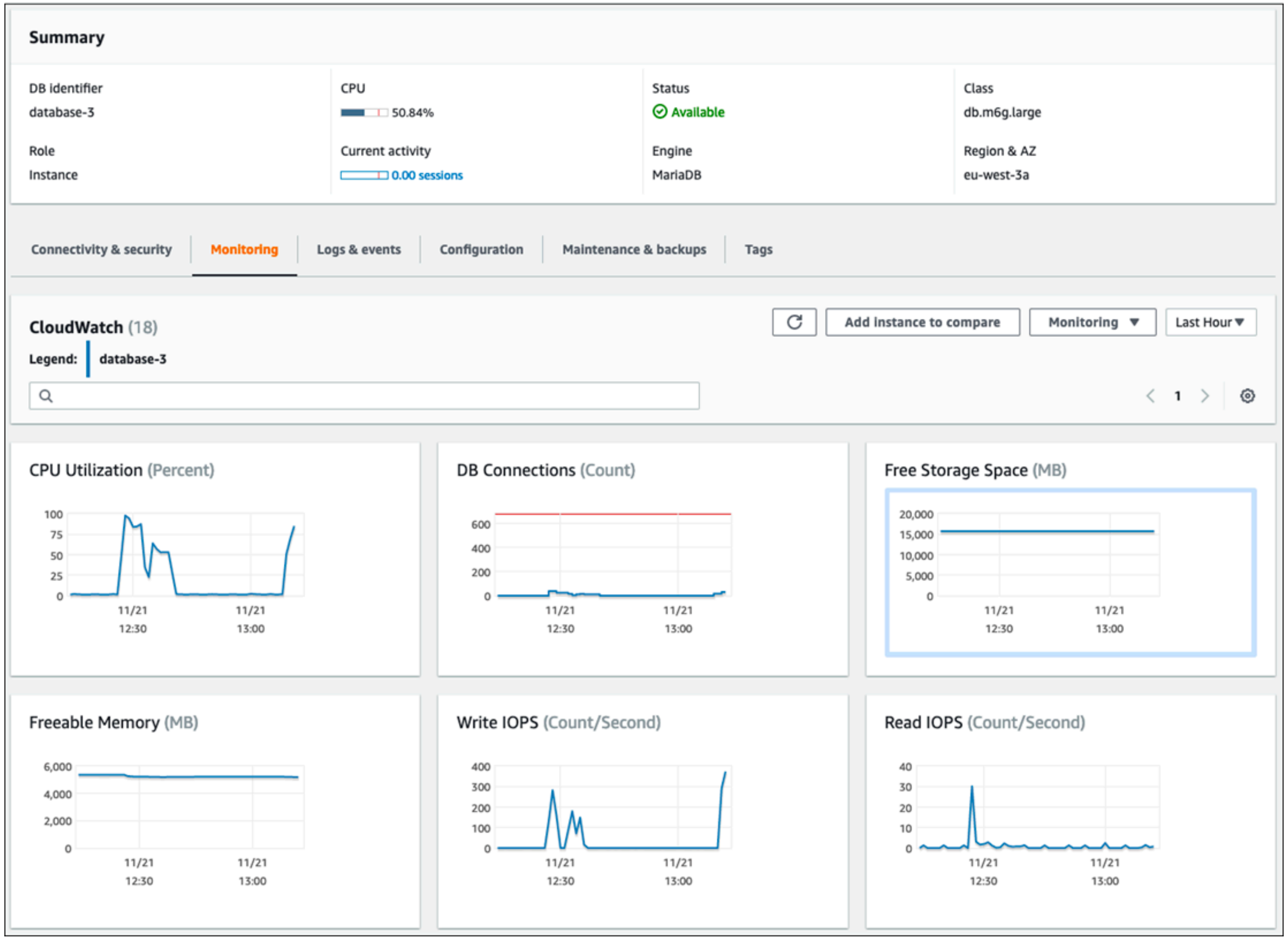
wait/synch/mutex/innodb/trx_mu	0, 0%
wait/lock/table/sql/handler	0, 0%
wait/synch/sxlock/innodb/btr_s	0, 0%
wait/synch/mutex/sql/MYSQL_BIN	0, 0%
wait/synch/mutex/sql/MYSQL_BIN	1, 33%
wait/synch/cond/sql/MYSQL_BIN_	0, 0%
wait/io/file/sql/binlog	0, 0%
wait/io/socket/sql/client_conn	0, 0%
wait/io/table/sql/handler	1, 33%
CPU	1, 33%
Total DB load	3
-- Max vCPUs	2

Top SQL (25)

	Load by waits (AAS)	SQL statements	Calls/sec	Avg latency (ms...	Rows examined...
○ ⊞	0.76	COMMIT	-	-	-
○ ⊞	0.40	SELECT `c` FROM `sbtest3` WHERE `id` = ?	-	-	-

Source: https://docs.aws.amazon.com/prescriptive-guidance/latest/amazon-rds-monitoring-alerting/db-instance-performance-insights.html

Enhanced Monitoring and CloudWatch



Summary

DB identifier database-3	CPU 50.84%	Status ✓ Available	Class db.m6g.large
Role Instance	Current activity 0.00 sessions	Engine MariaDB	Region & AZ eu-west-3a

Source: https://docs.aws.amazon.com/prescriptive-guidance/latest/amazon-rds-monitoring-alerting/os-monitoring.html

Aurora Postgres Parameters

Aurora uses a two-level system for configuration settings

DB cluster parameter group
- Applies to *every* DB instance within the cluster
- 413 parameters, 373 are modifiable

DB parameter group
- Applies to a *single* DB instance within the cluster
- Where the parameters overlap with DB cluster parameters they supersede
- 300 parameters, 268 are modifiable

RDS Postgres (not Aurora)
- 395 parameters, 348 are modifiable

Aurora can assign default parameter groups at creation, but specify custom groups
- default groups do not allow changes and require reboot to apply custom groups
- parameters are your levers for tuning, troubleshooting, and logging
- many parameters can be changed without restart
- use Performance Insights, Enhanced Monitoring, and CloudWatch to inform your parameter changes
- notably, parameters for checkpoints, bgwriter_lru_maxpages, and others are missing

Schema

Database physical design for Aurora

- Partition large tables
- Index according to access patterns and experiment with different index types
- Storing data
  - third-normal form or dimensional — ORM's?
  - proper data types
  - alignment with fixed length columns before variable columns for efficient storage in pages
  - add defaults after loading tables so defaults are not stored during load

Strategies for Aurora

But, not enough time in this session

- Indexing strategies and maintenance
- Query optimization that leverage Aurora's capabilities
- Vertical and horizontal scaling strategies with Aurora
- Query Plan Management (QPM)

An interesting application of Aurora

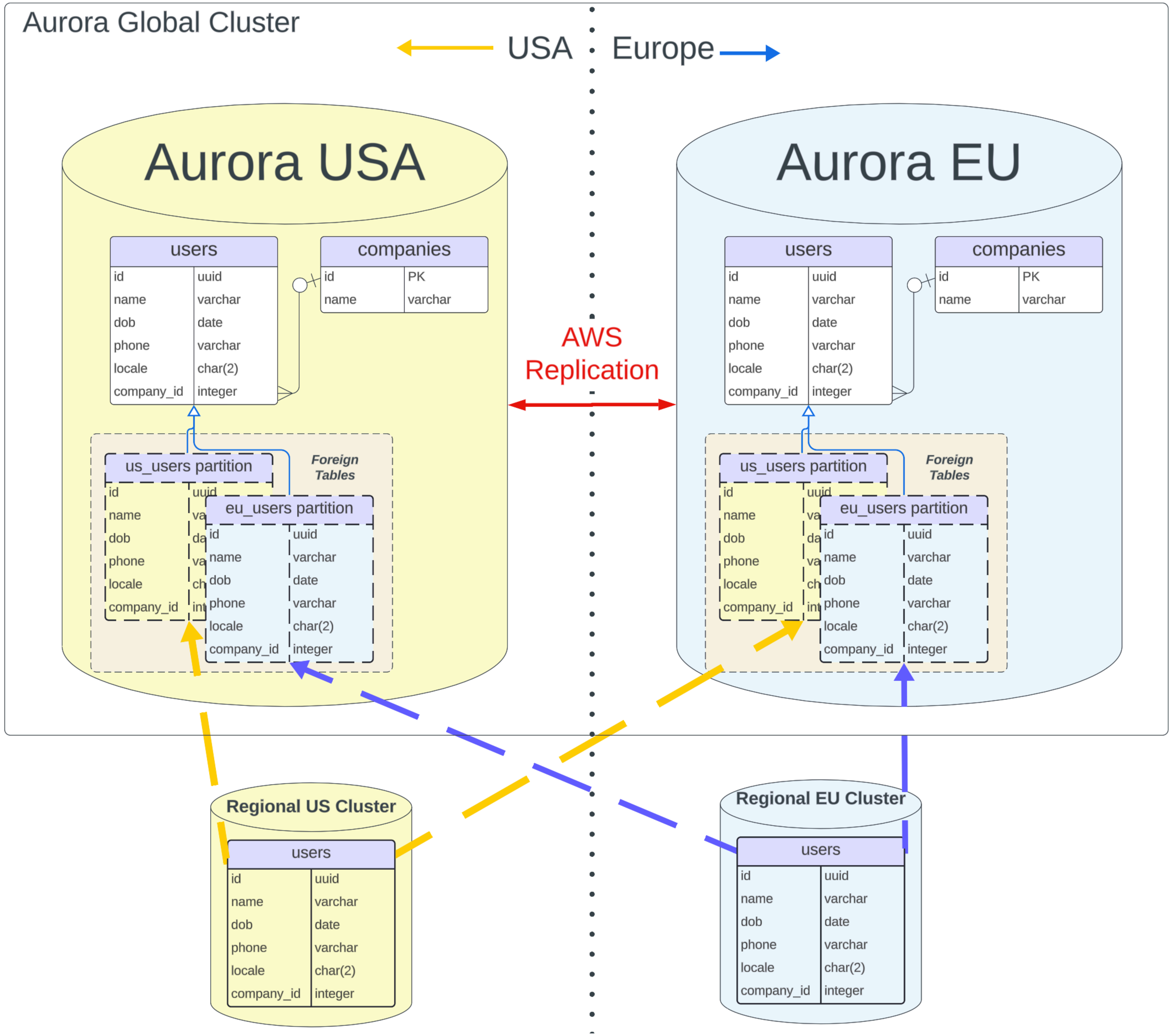
Global data with PII data kept local

How does a global company running applications in many countries keep customers and users data localized to a country or region while continuing to share data on global level?

Solution involves:

- Aurora Global Cluster
- RDS Postgres database or Aurora regional clusters
- Foreign Data Wrappers

Global Aurora Cluster (USA - EU) and Local DBs



Aurora Global Cluster

USA

Europe

Aurora USA

users	
id	uuid
name	varchar
dob	date
phone	varchar
locale	char(2)
company_id	integer

companies	
id	PK
name	varchar

Foreign Tables

us_users partition	
id	uuid
name	va
dob	da
phone	va
locale	ch
company_id	int

eu_users partition	
id	uuid
name	varchar
dob	date
phone	varchar
locale	char(2)
company_id	integer

AWS Replication

Aurora EU

users	
id	uuid
name	varchar
dob	date
phone	varchar
locale	char(2)
company_id	integer

companies	
id	PK
name	varchar

Foreign Tables

us_users partition	
id	uuid
name	va
dob	da
phone	va
locale	ch
company_id	int

eu_users partition	
id	uuid
name	varchar
dob	date
phone	varchar
locale	char(2)
company_id	integer

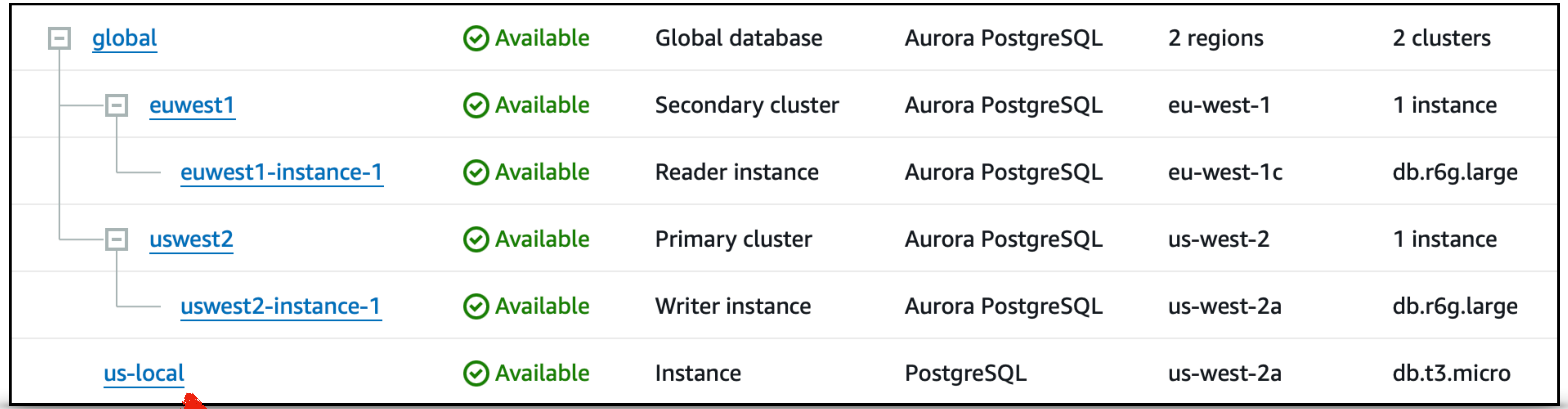
Regional US Cluster

users	
id	uuid
name	varchar
dob	date
phone	varchar
locale	char(2)
company_id	integer

Regional EU Cluster

users	
id	uuid
name	varchar
dob	date
phone	varchar
locale	char(2)
company_id	integer

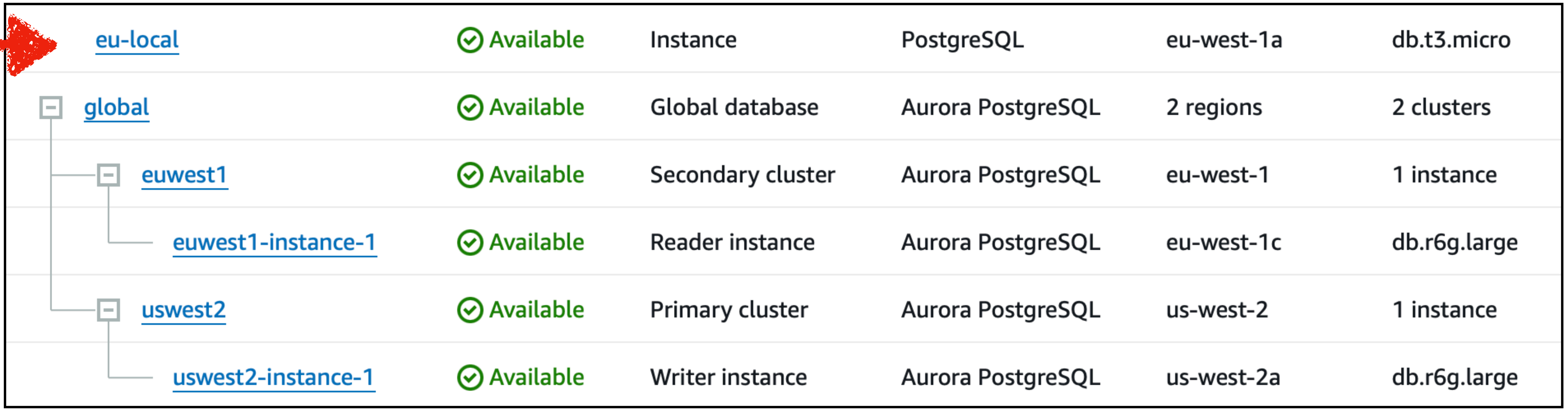
AWS Console in Oregon (us-west-2)



global	Available	Global database	Aurora PostgreSQL	2 regions	2 clusters
euwest1	Available	Secondary cluster	Aurora PostgreSQL	eu-west-1	1 instance
euwest1-instance-1	Available	Reader instance	Aurora PostgreSQL	eu-west-1c	db.r6g.large
uswest2	Available	Primary cluster	Aurora PostgreSQL	us-west-2	1 instance
uswest2-instance-1	Available	Writer instance	Aurora PostgreSQL	us-west-2a	db.r6g.large
us-local	Available	Instance	PostgreSQL	us-west-2a	db.t3.micro

Local to
each region

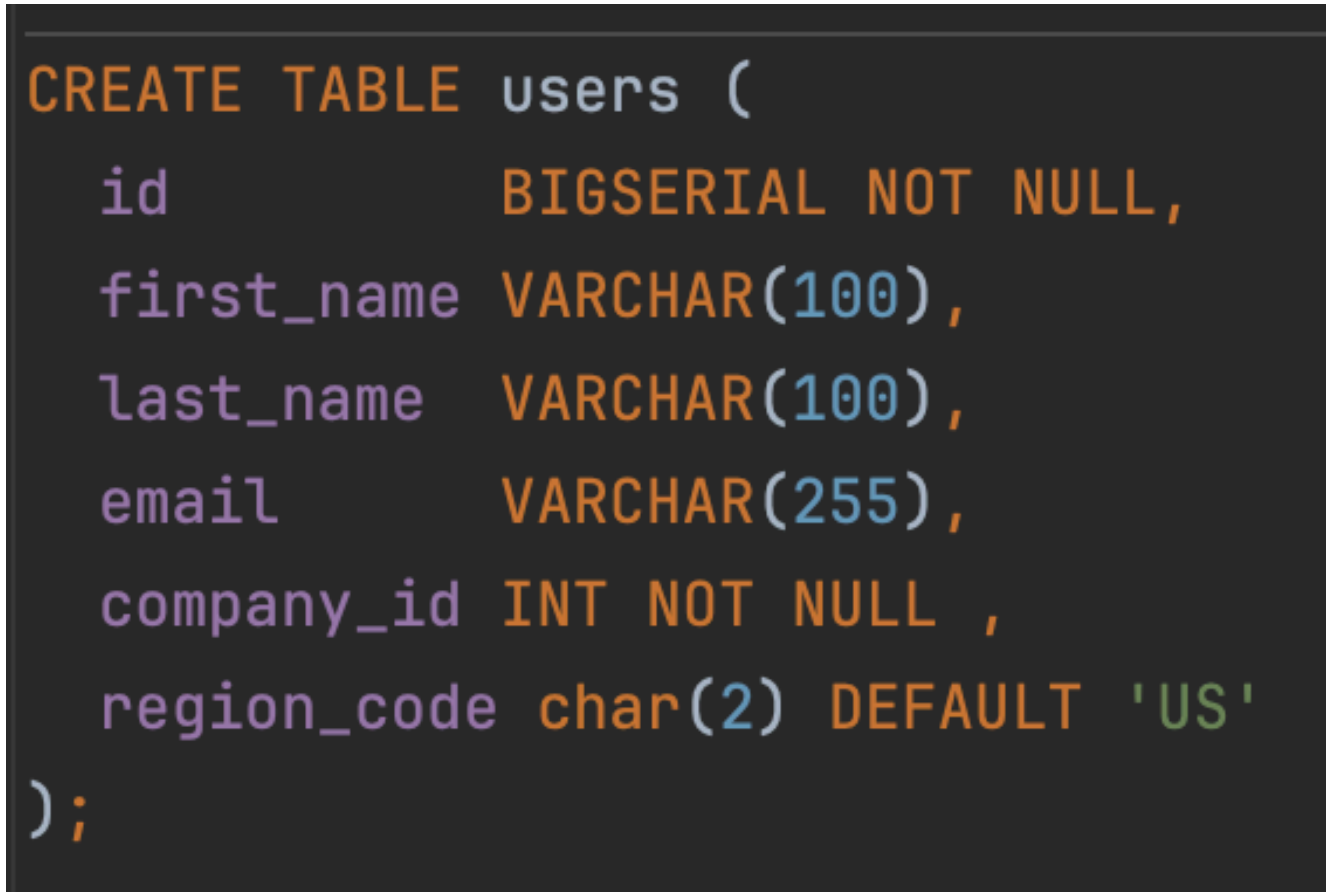
AWS Console in Ireland (eu-west-1)



eu-local	Available	Instance	PostgreSQL	eu-west-1a	db.t3.micro
global	Available	Global database	Aurora PostgreSQL	2 regions	2 clusters
euwest1	Available	Secondary cluster	Aurora PostgreSQL	eu-west-1	1 instance
euwest1-instance-1	Available	Reader instance	Aurora PostgreSQL	eu-west-1c	db.r6g.large
uswest2	Available	Primary cluster	Aurora PostgreSQL	us-west-2	1 instance
uswest2-instance-1	Available	Writer instance	Aurora PostgreSQL	us-west-2a	db.r6g.large

Step 1: Create users table in each (local) region db

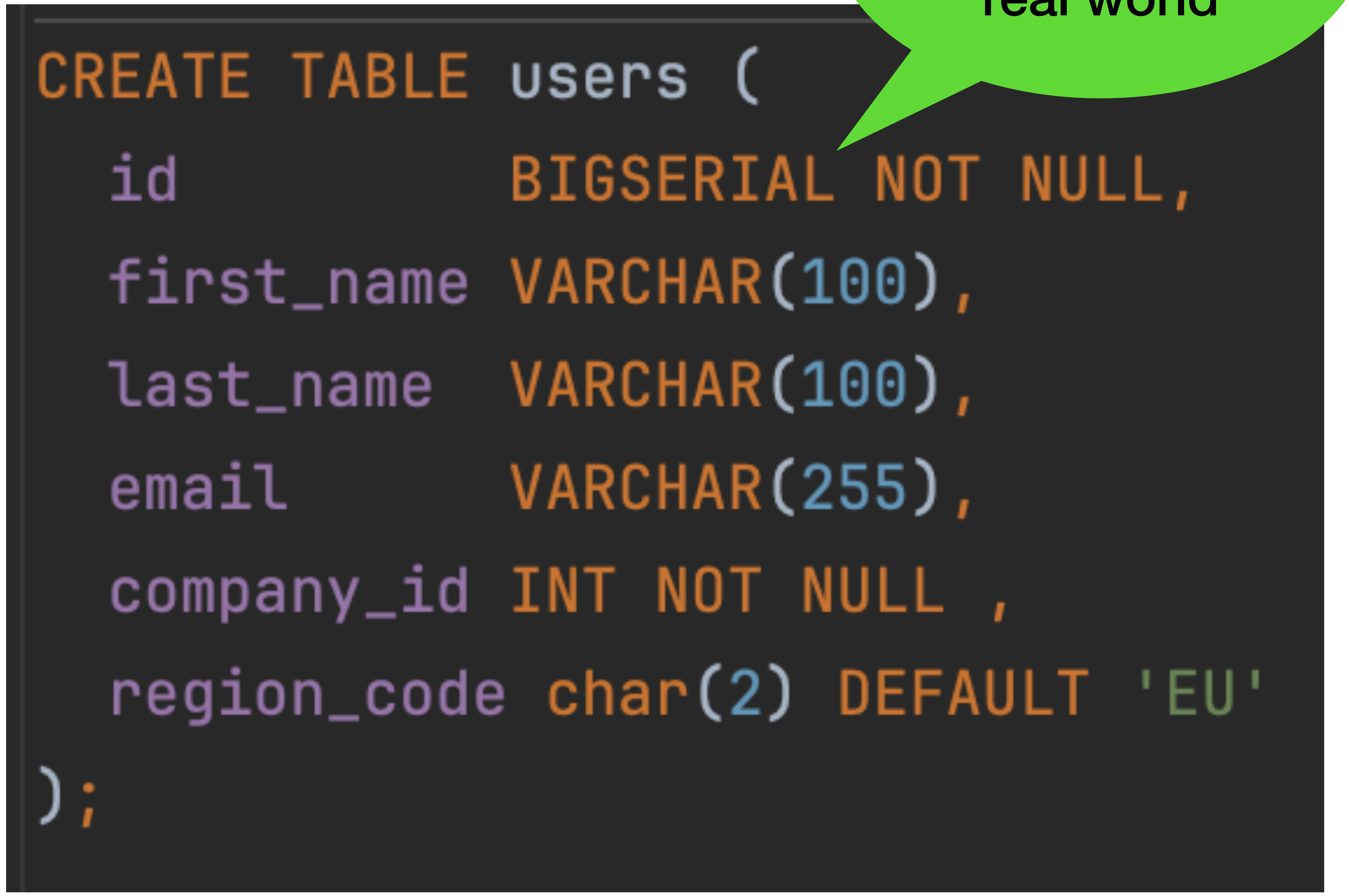
Oregon (us-west-2)



```
CREATE TABLE users (
  id         BIGSERIAL NOT NULL,
  first_name VARCHAR(100),
  last_name  VARCHAR(100),
  email      VARCHAR(255),
  company_id INT NOT NULL ,
  region_code char(2) DEFAULT 'US'
);
```

Ireland (eu-west-1)

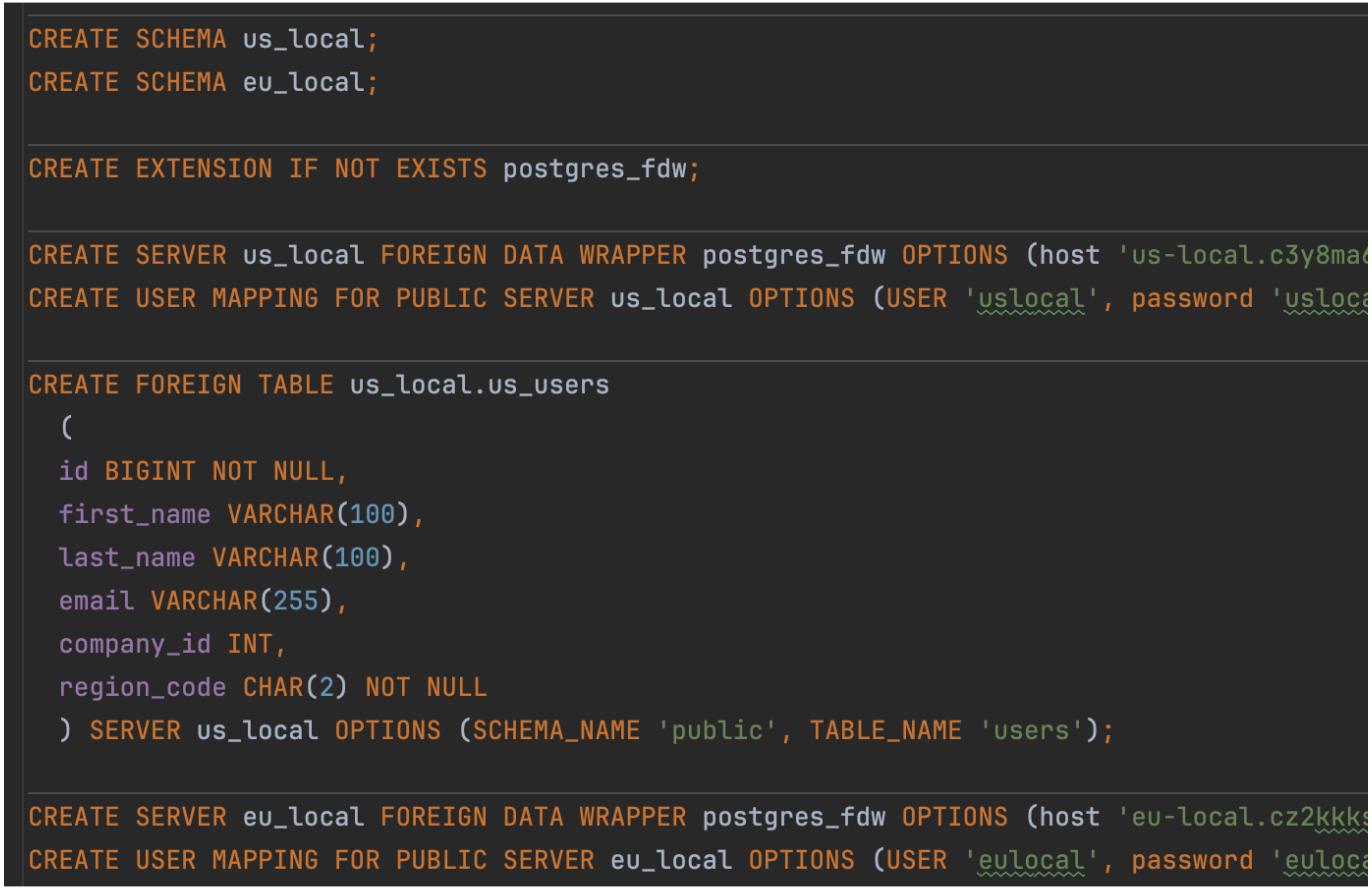
uuids would be much better in the real world



```
CREATE TABLE users (
  id         BIGSERIAL NOT NULL,
  first_name VARCHAR(100),
  last_name  VARCHAR(100),
  email      VARCHAR(255),
  company_id INT NOT NULL ,
  region_code char(2) DEFAULT 'EU'
);
```

Step 2: Create foreign tables in global cluster

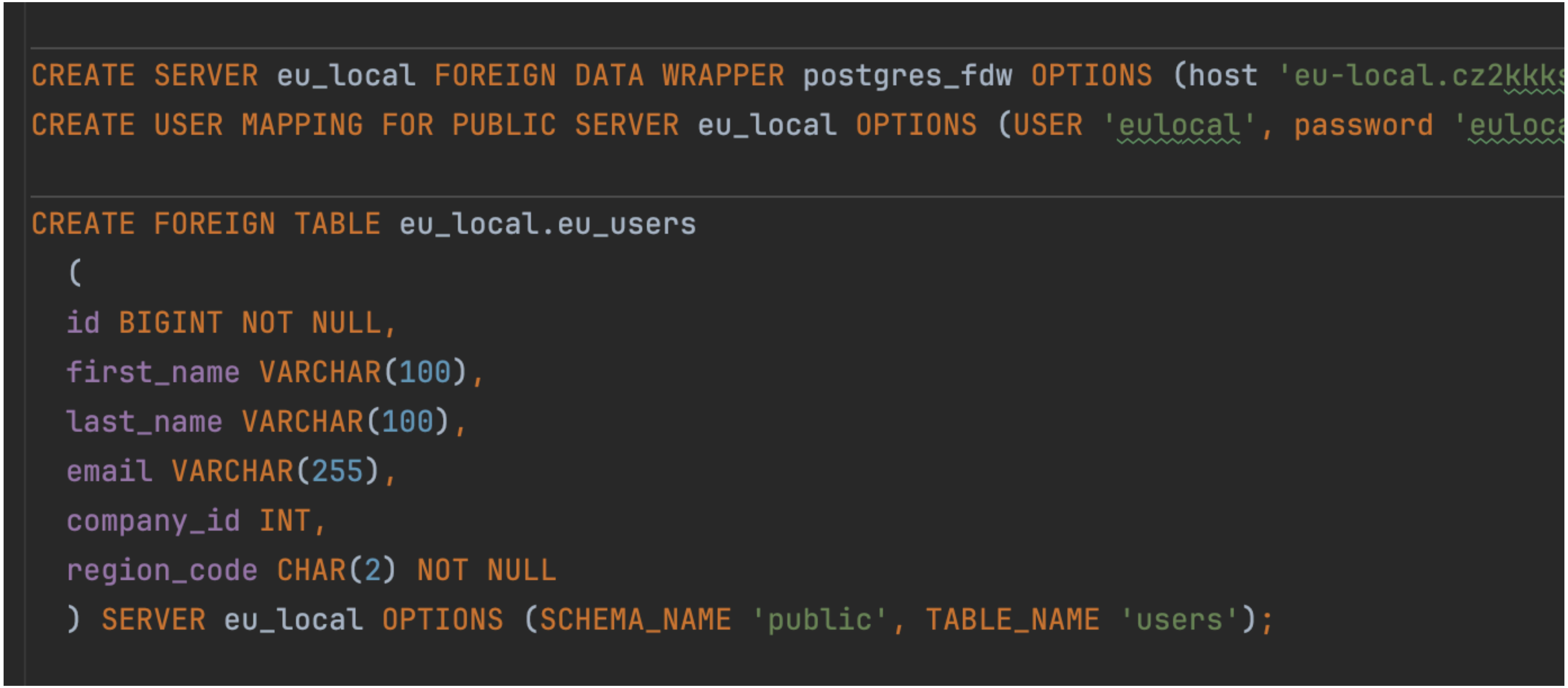
Oregon users table



```sql
CREATE SCHEMA us_local;
CREATE SCHEMA eu_local;

CREATE EXTENSION IF NOT EXISTS postgres_fdw;

CREATE SERVER us_local FOREIGN DATA WRAPPER postgres_fdw OPTIONS (host 'us-local.c3y8ma
CREATE USER MAPPING FOR PUBLIC SERVER us_local OPTIONS (USER 'uslocal', password 'usloca

CREATE FOREIGN TABLE us_local.us_users
  (
  id BIGINT NOT NULL,
  first_name VARCHAR(100),
  last_name VARCHAR(100),
  email VARCHAR(255),
  company_id INT,
  region_code CHAR(2) NOT NULL
  ) SERVER us_local OPTIONS (SCHEMA_NAME 'public', TABLE_NAME 'users');

CREATE SERVER eu_local FOREIGN DATA WRAPPER postgres_fdw OPTIONS (host 'eu-local.cz2kkk
CREATE USER MAPPING FOR PUBLIC SERVER eu_local OPTIONS (USER 'eulocal', password 'euloca
```

Step 2: Create foreign tables in global cluster

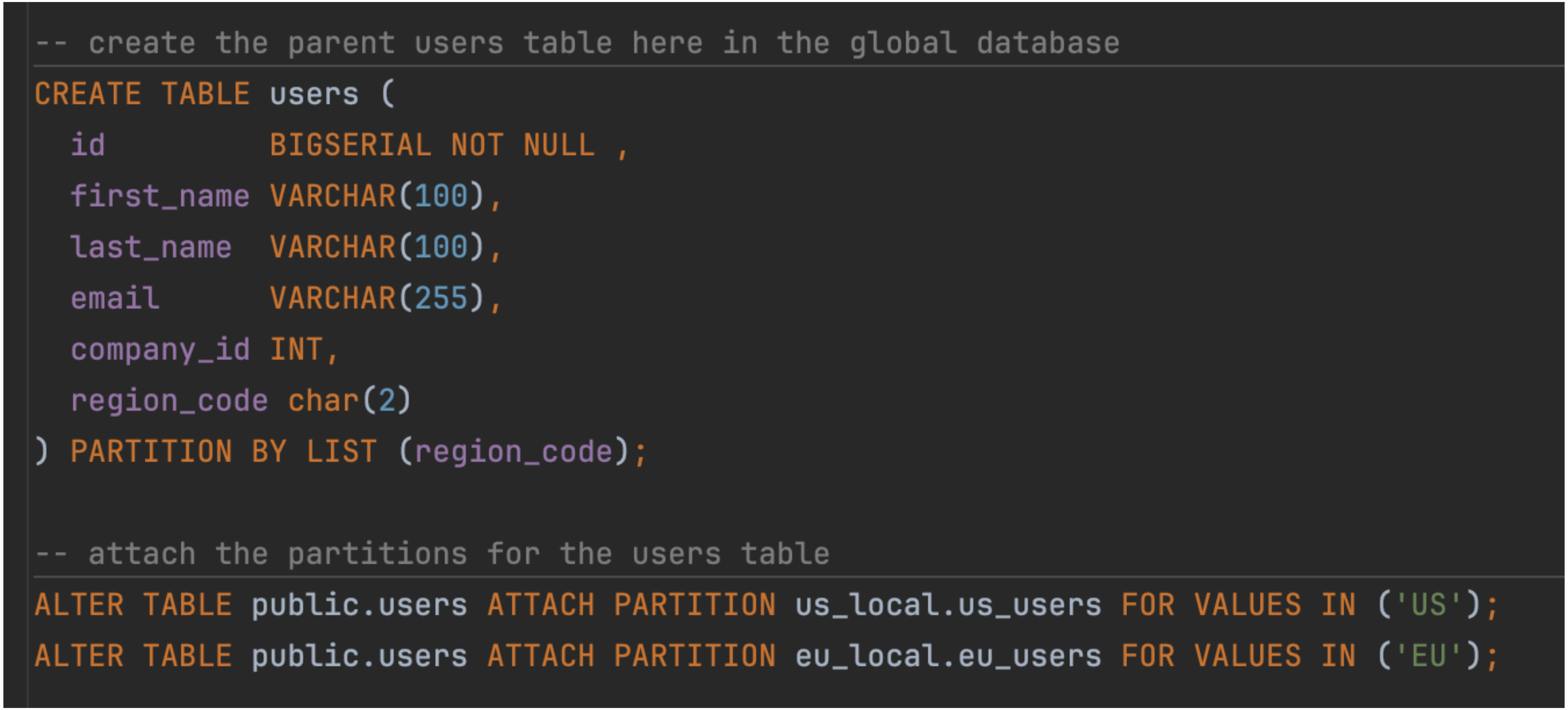
Ireland users table



```
CREATE SERVER eu_local FOREIGN DATA WRAPPER postgres_fdw OPTIONS (host 'eu-local.cz2kkks
CREATE USER MAPPING FOR PUBLIC SERVER eu_local OPTIONS (USER 'eulocal', password 'euloca

CREATE FOREIGN TABLE eu_local.eu_users
  (
  id BIGINT NOT NULL,
  first_name VARCHAR(100),
  last_name VARCHAR(100),
  email VARCHAR(255),
  company_id INT,
  region_code CHAR(2) NOT NULL
  ) SERVER eu_local OPTIONS (SCHEMA_NAME 'public', TABLE_NAME 'users');
```

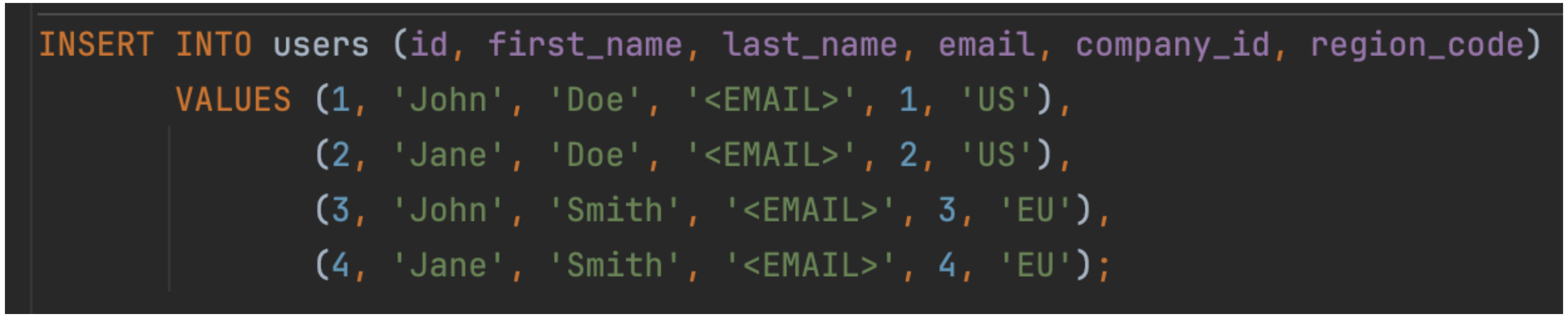
Step 3: Create parent users table and attach partitions

Within global cluster



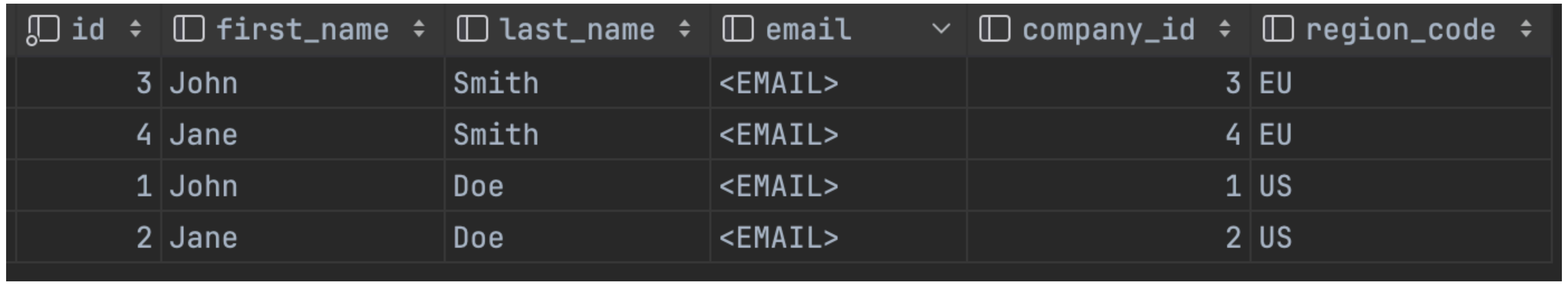
```sql
-- create the parent users table here in the global database
CREATE TABLE users (
  id         BIGSERIAL NOT NULL ,
  first_name VARCHAR(100),
  last_name  VARCHAR(100),
  email      VARCHAR(255),
  company_id INT,
  region_code char(2)
) PARTITION BY LIST (region_code);


-- attach the partitions for the users table
ALTER TABLE public.users ATTACH PARTITION us_local.us_users FOR VALUES IN ('US');
ALTER TABLE public.users ATTACH PARTITION eu_local.eu_users FOR VALUES IN ('EU');
```

Insert users in the global db



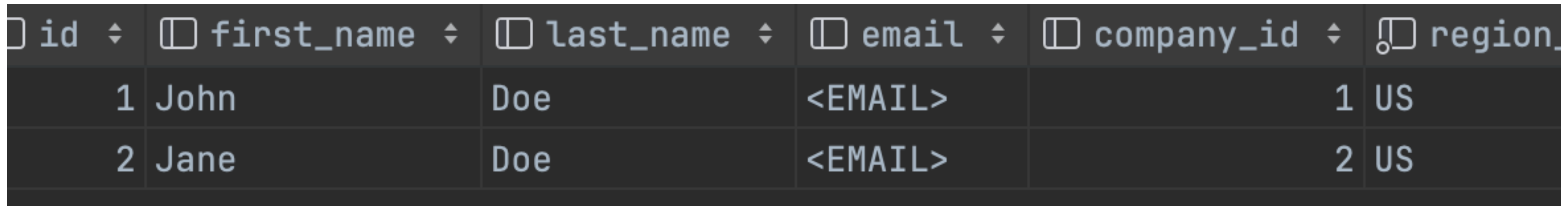
```sql
INSERT INTO users (id, first_name, last_name, email, company_id, region_code)
       VALUES (1, 'John', 'Doe', '<EMAIL>', 1, 'US'),
              (2, 'Jane', 'Doe', '<EMAIL>', 2, 'US'),
              (3, 'John', 'Smith', '<EMAIL>', 3, 'EU'),
              (4, 'Jane', 'Smith', '<EMAIL>', 4, 'EU');
```

SELECT *
FROM USERS;



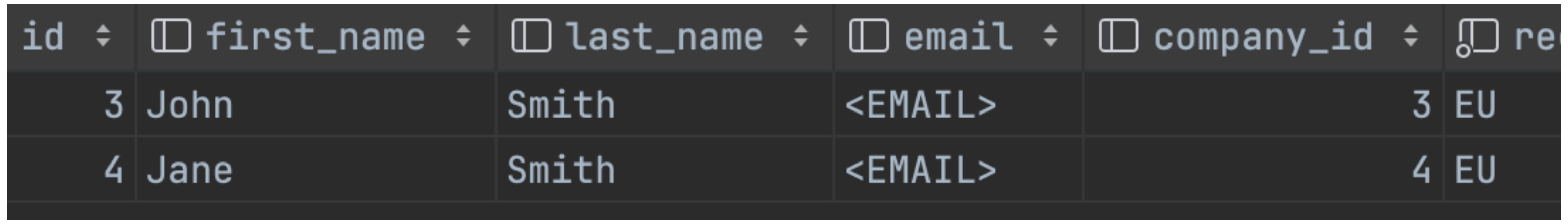
id	first_name	last_name	email	company_id	region_code
3	John	Smith	<EMAIL>	3	EU
4	Jane	Smith	<EMAIL>	4	EU
1	John	Doe	<EMAIL>	1	US
2	Jane	Doe	<EMAIL>	2	US

SELECT *
FROM us_local.us_users;



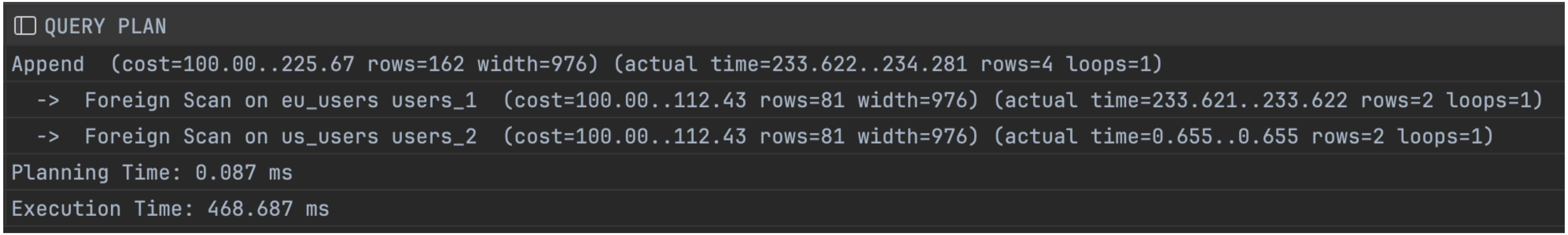
id	first_name	last_name	email	company_id	region_
1	John	Doe	<EMAIL>	1	US
2	Jane	Doe	<EMAIL>	2	US

SELECT *
FROM eu_local.eu_users;



id	first_name	last_name	email	company_id	re
3	John	Smith	<EMAIL>	3	EU
4	Jane	Smith	<EMAIL>	4	EU

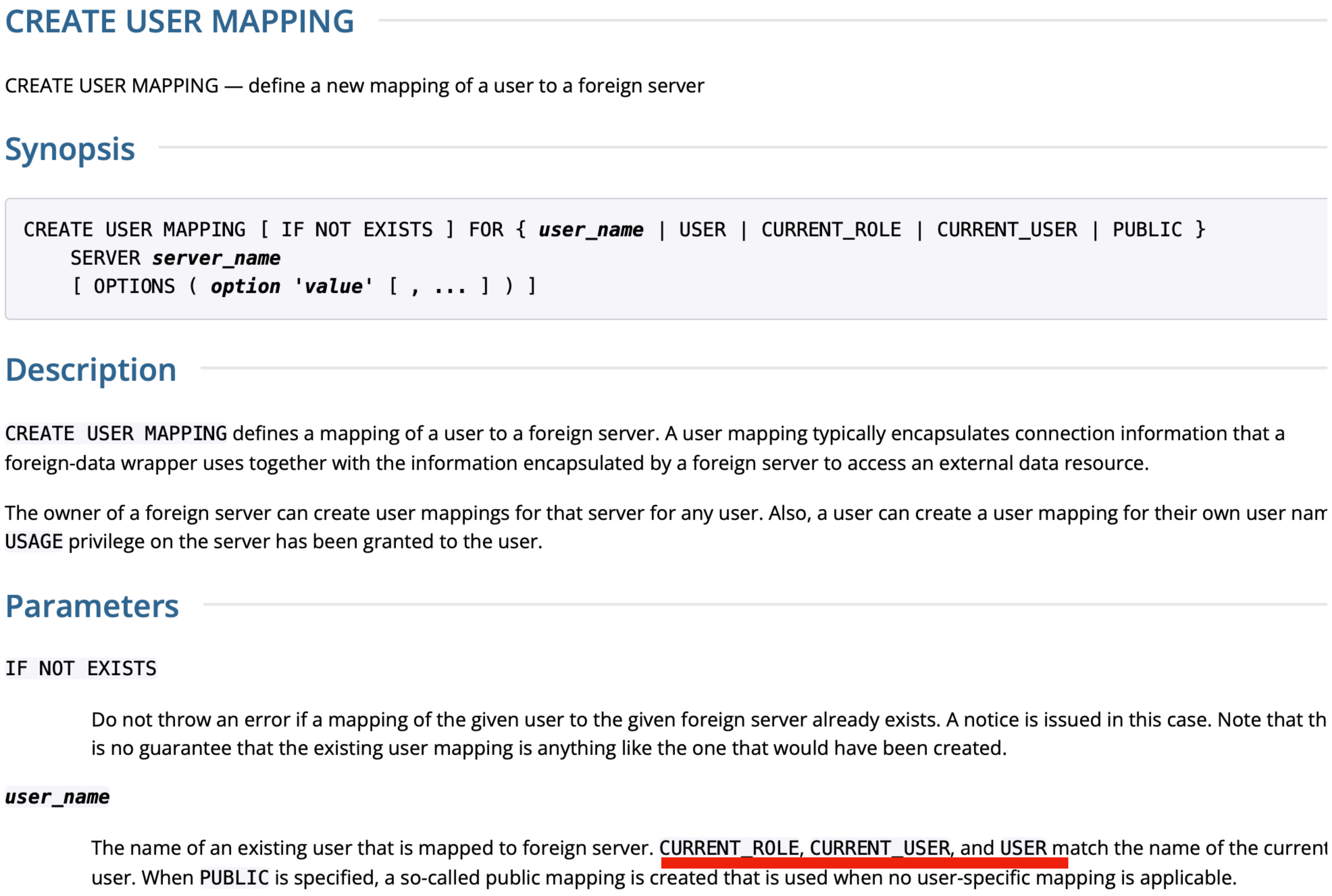
EXPLAIN ANALYSE
SELECT * FROM users;



```
QUERY PLAN
Append  (cost=100.00..225.67 rows=162 width=976) (actual time=233.622..234.281 rows=4 loops=1)
  ->  Foreign Scan on eu_users users_1  (cost=100.00..112.43 rows=81 width=976) (actual time=233.621..233.622 rows=2 loops=1)
  ->  Foreign Scan on us_users users_2  (cost=100.00..112.43 rows=81 width=976) (actual time=0.655..0.655 rows=2 loops=1)
Planning Time: 0.087 ms
Execution Time: 468.687 ms
```

Final Step: Map limited permission roles to FDW

USA roles have no permission to access EU users and vice versa



CREATE USER MAPPING

CREATE USER MAPPING — define a new mapping of a user to a foreign server

Synopsis

```
CREATE USER MAPPING [ IF NOT EXISTS ] FOR { user_name | USER | CURRENT_ROLE | CURRENT_USER | PUBLIC }
    SERVER server_name
    [ OPTIONS ( option 'value' [ , ... ] ) ]
```

Description

`CREATE USER MAPPING` defines a mapping of a user to a foreign server. A user mapping typically encapsulates connection information that a foreign-data wrapper uses together with the information encapsulated by a foreign server to access an external data resource.

The owner of a foreign server can create user mappings for that server for any user. Also, a user can create a user mapping for their own user nam
`USAGE` privilege on the server has been granted to the user.

Parameters

`IF NOT EXISTS`

Do not throw an error if a mapping of the given user to the given foreign server already exists. A notice is issued in this case. Note that th
is no guarantee that the existing user mapping is anything like the one that would have been created.

`user_name`

The name of an existing user that is mapped to foreign server. `CURRENT_ROLE`, `CURRENT_USER`, and `USER` match the name of the current
user. When `PUBLIC` is specified, a so-called public mapping is created that is used when no user-specific mapping is applicable.