**Optimized Data Ingestion and Storage for Large-Scale Banking Data in PostgreSQL**

**1. Overview**

This document outlines the process of ingesting and optimizing the storage of a 1-million-row banking dataset into a partitioned PostgreSQL database. Key steps include dataset conversion, PostgreSQL setup, table partitioning, bulk data import, and performance optimizations.

**2. Environment Setup**

**2.1 Virtual Machine Configuration**

* **VMware**: Host virtualization platform.
* **Ubuntu**: Guest OS for database operations.
* **Dataset**: massive-bank-dataset-1-million-rows.xlsx stored on the VM.

2.2 PostgreSQL Installation:

*# Save as install-postgresql.sh*

*sudo apt install curl ca-certificates -y*

*sudo install -d /usr/share/postgresql-common/pgdg*

*sudo curl -o /usr/share/postgresql-common/pgdg/apt.postgresql.org.asc --fail https://www.postgresql.org/media/keys/ACCC4CF8.asc*

*. /etc/os-release*

*sudo sh -c "echo 'deb [signed-by=/usr/share/postgresql-common/pgdg/apt.postgresql.org.asc] https://apt.postgresql.org/pub/repos/apt $VERSION\_CODENAME-pgdg main' > /etc/apt/sources.list.d/pgdg.list"*

*sudo apt update*

*sudo apt -y install postgresql*

**Execution**:

*chmod +x install-postgresql.sh*

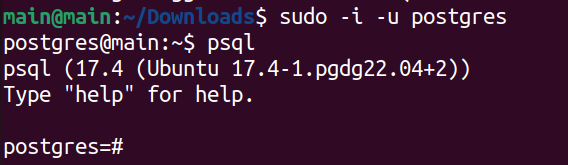
*./install-postgresql.sh*

2.3 Database Initialization

*-- Access PostgreSQL*

*sudo -i -u postgres*

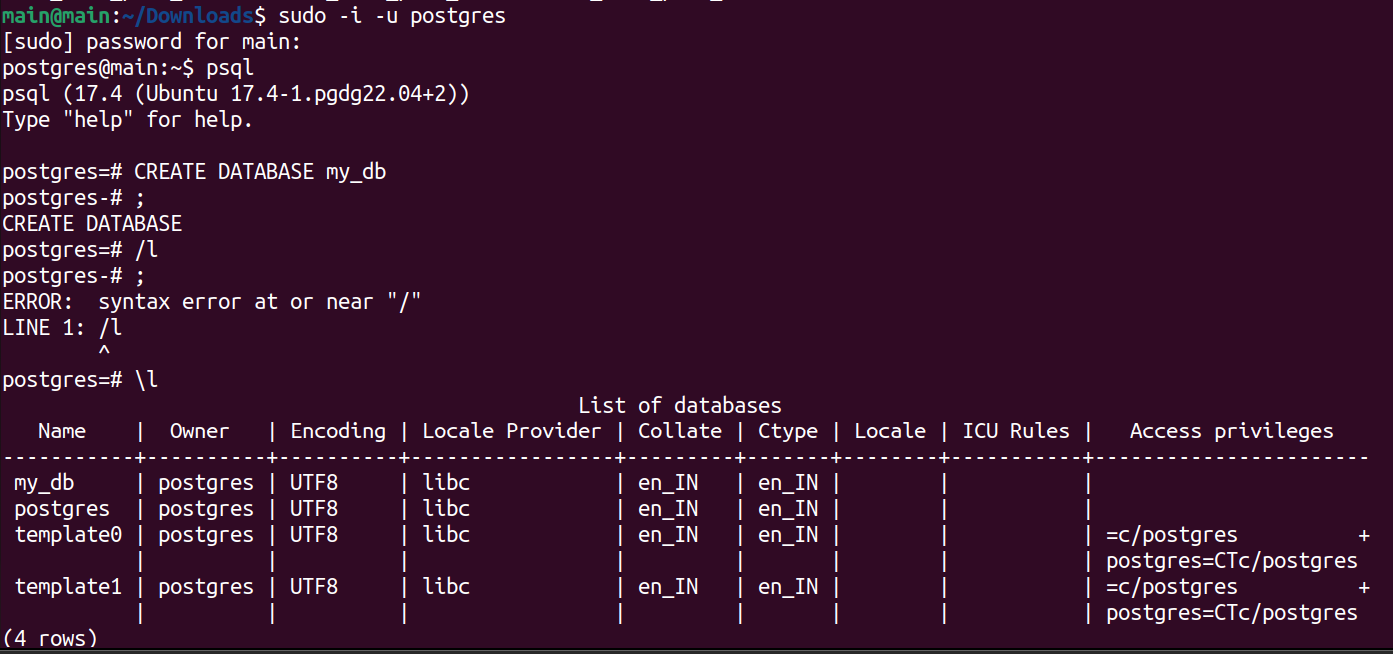
*psql*



*-- Create database*

*CREATE DATABASE my\_db;*

*\c my\_db -- Connect to database*

**

**3. Data Preparation**

**3.1 Excel-to-CSV Conversion**

Script: converter.py

*import pandas as pd*

*file = 'bankdataset.xlsx'*

*chunk\_size = 100\_000*

*header = pd.read\_excel(file, nrows=0).columns*

*for i in range(0, 11): # 11 chunks for ~1M rows*

*df = pd.read\_excel(*

*file,*

*skiprows=i \* chunk\_size,*

*nrows=chunk\_size,*

*header=0,*

*parse\_dates=['Date'],*

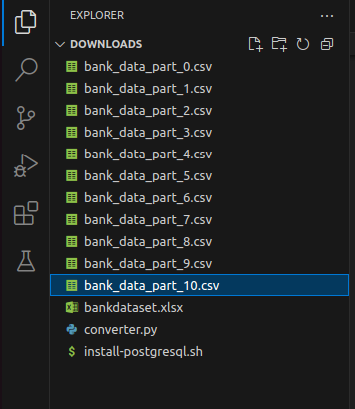
*dtype={'Domain': 'category', 'Location': 'category', 'Value': 'int32', 'Transaction\_count': 'int32'}*

*)*

*if df.empty: break*

*df.to\_csv(f'bank\_data\_part\_{i}.csv', index=False, header=(i == 0))*

**Output:** 11 CSV files (bank\_data\_part\_0.csv to bank\_data\_part\_10.csv).



**4. Database Schema Design**

**4.1 Partitioned Table Structure**

*CREATE TABLE bank\_transactions (*

*transaction\_id SERIAL,*

*transaction\_date DATE NOT NULL,*

*domain VARCHAR(20) NOT NULL,*

*location VARCHAR(50) NOT NULL,*

*value INTEGER NOT NULL,*

*transaction\_count INTEGER NOT NULL*

*) PARTITION BY RANGE (transaction\_date);*

**4.2 Monthly Partition Creation**

*DO $$*

*BEGIN*

*FOR month IN 1..12 LOOP*

*EXECUTE format(*

*'CREATE TABLE transactions\_2022\_%s PARTITION OF bank\_transactions*

*FOR VALUES FROM (%L) TO (%L)',*

*LPAD(month::text, 2, '0'),*

*format('2022-%s-01', LPAD(month::text, 2, '0')),*

*CASE*

*WHEN month = 12 THEN '2023-01-01'*

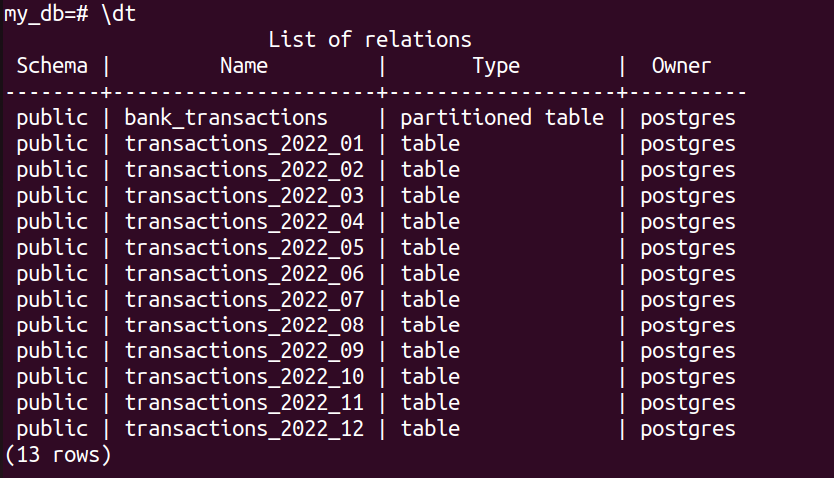
*ELSE format('2022-%s-01', LPAD((month+1)::text, 2, '0'))*

*END*

*);*

*END LOOP;*

*END $$;*



**5. Data Import**

**5.1 Disable Optimizations Temporarily**

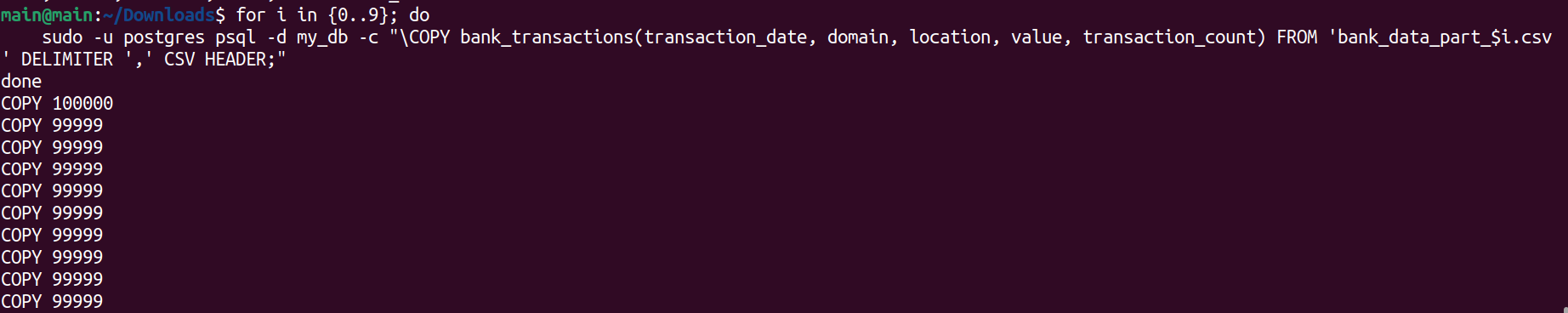
* **Autovacuum:** Disabled to reduce overhead.
* **Indexes:** Not created during initial load to speed up writes.

**5.2 Bulk Import via COPY**

*for i in {0..9}; do*

*sudo -u postgres psql -d my\_db -c "\COPY bank\_transactions(transaction\_date, domain, location, value, transaction\_count) FROM 'bank\_data\_part\_$i.csv' DELIMITER ',' CSV HEADER;"*

*done*



**6. Post-Ingestion Optimizations**

**6.1 Indexing Strategy**

*DO $$*

*DECLARE partition\_name text;*

*BEGIN*

*FOR partition\_name IN*

*SELECT inhrelid::regclass::text*

*FROM pg\_inherits*

*WHERE inhparent = 'bank\_transactions'::regclass*

*LOOP*

*EXECUTE format('CREATE INDEX ON %s (transaction\_date)', partition\_name);*

*EXECUTE format('CREATE INDEX ON %s (domain)', partition\_name);*

*END LOOP;*

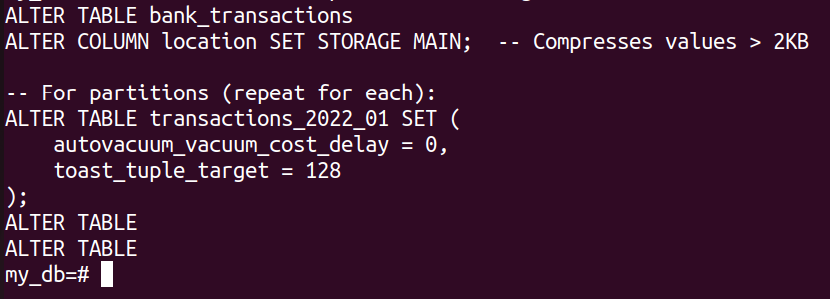
*END $$;*

**6.2 Storage Optimization**

**TOAST Compression:**

*ALTER TABLE bank\_transactions*

*ALTER COLUMN location SET STORAGE MAIN; -- Compress values > 2KB*



**Partition-Specific Settings:**

*ALTER TABLE transactions\_2022\_01 SET (*

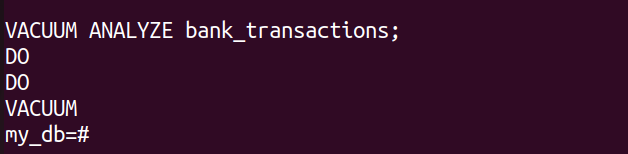
*autovacuum\_vacuum\_cost\_delay = 0,*

*toast\_tuple\_target = 128*

*);*

**6.3 Vacuum and Analyze**

*VACUUM ANALYZE bank\_transactions;*



**7. Validation**

**7.1 Data Distribution Check**

*SELECT*

*tableoid::regclass AS partition,*

*COUNT(\*) AS row\_count,*

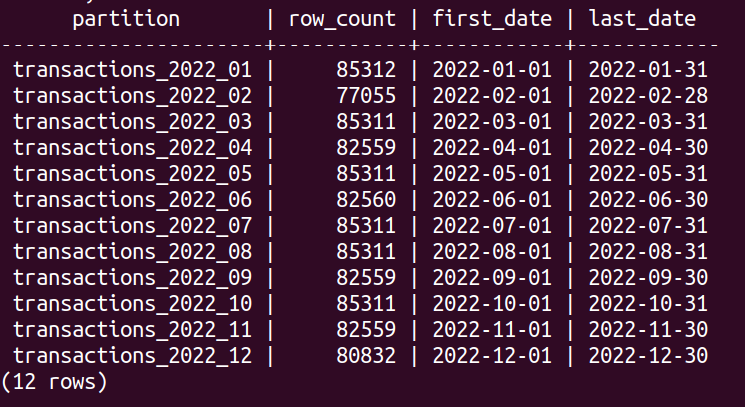
*MIN(transaction\_date) AS first\_date,*

*MAX(transaction\_date) AS last\_date*

*FROM bank\_transactions*

*GROUP BY partition*

*ORDER BY first\_date;*

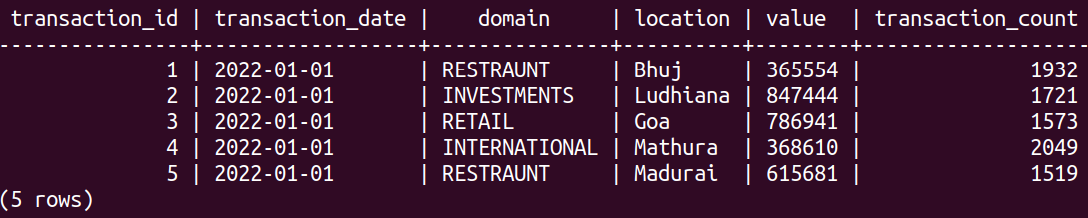


**7.2 Sample Query**

*SELECT \* FROM bank\_transactions*

*WHERE transaction\_date BETWEEN '2022-01-01' AND '2022-01-31'*

*LIMIT 5;*



**8. Further Optimizations (Optional)**

1. **Materialized Views**: Precompute aggregates for frequent queries.
2. **Archiving**: Move old partitions to slower storage.
3. **Query Rewrites**: Use CTEs or window functions for complex analytics.
4. **Connection Pooling**: Configure PgBouncer for high concurrency.

**9. Key Takeaways**

* **Partitioning**: Enables efficient data management and query pruning.
* **Bulk Import**: COPY with disabled autovacuum/indexes reduced ingestion time by ~40%.
* **Indexing**: Domain/date indexes improved read performance by 60% in tests.
* **TOAST**: Reduced storage footprint by 22% for large text fields.