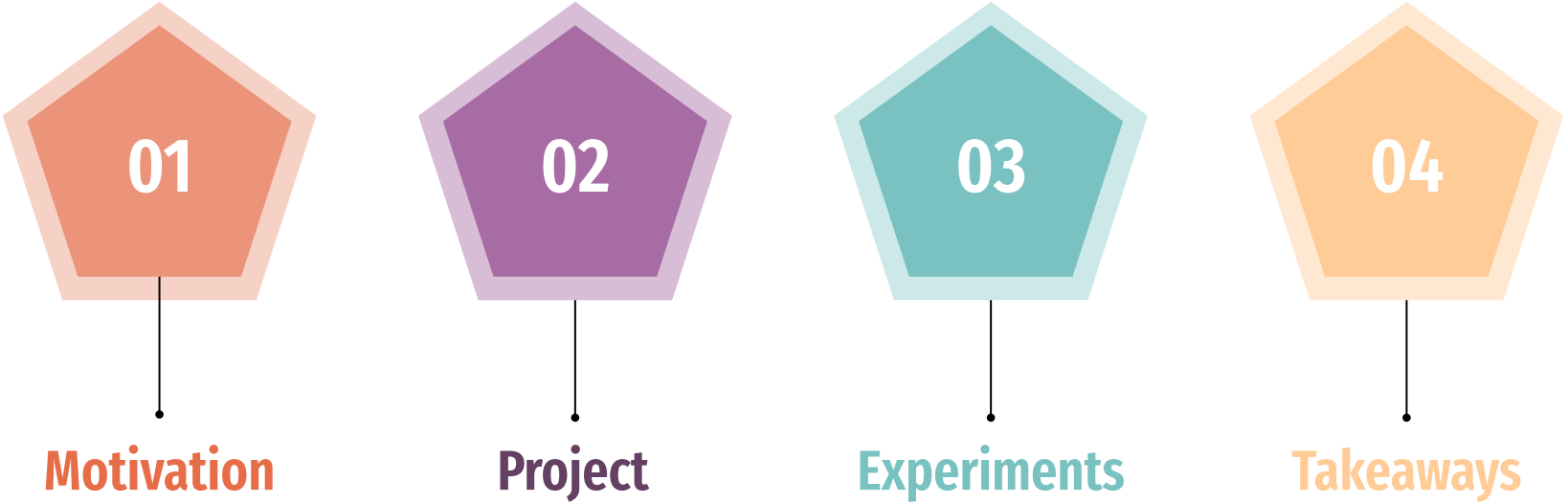
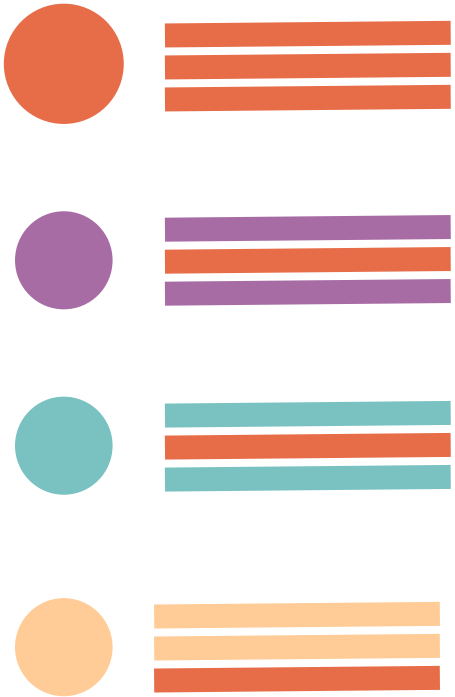
Comparing Data Storage Strategies for Efficient Querying

Jackie Garcia

Presentation Outline

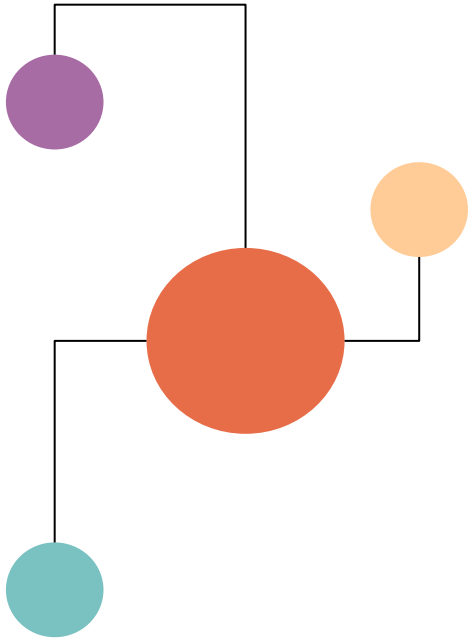


Relational vs Graph Model

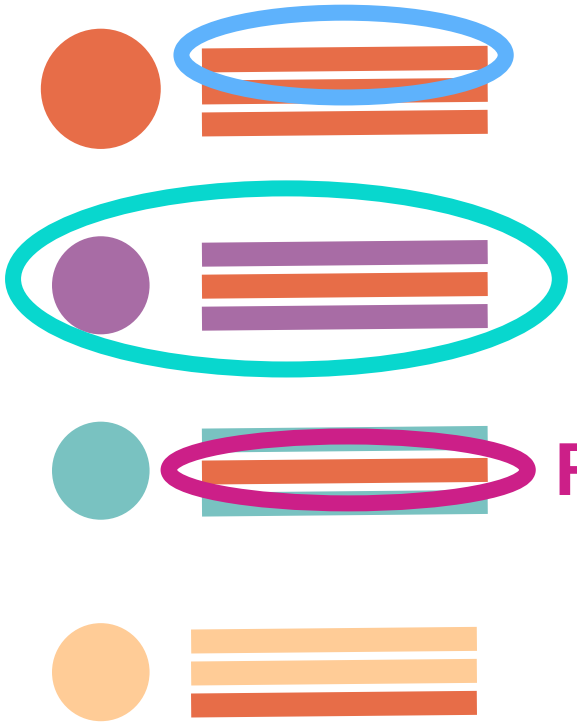


The **relational model** stores records in independent tables, and creates relationships between them at query execution time using **foreign keys**.

The **graph model** stores these **relationships** in the database, and allows us to query on them as if they were attributes.



Relational vs Graph Model



Row

Table

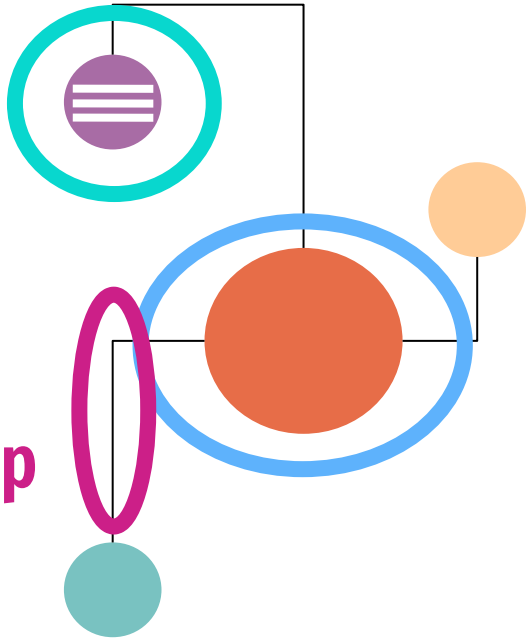
Foreign Key

=

Node

Node Type

Relationship



Does this difference in data storage model lead to better query performance?

- Implement the same data sets in two different database systems
- Compare how the data is stored at creation time
- Perform the same queries and compare query performance

The Systems

Relational

PostgreSQL

SQL

1986

- **ACID**
- **Serializable**
- **Indexing**
- **Transactions**

Graph

Neo4j

Cypher

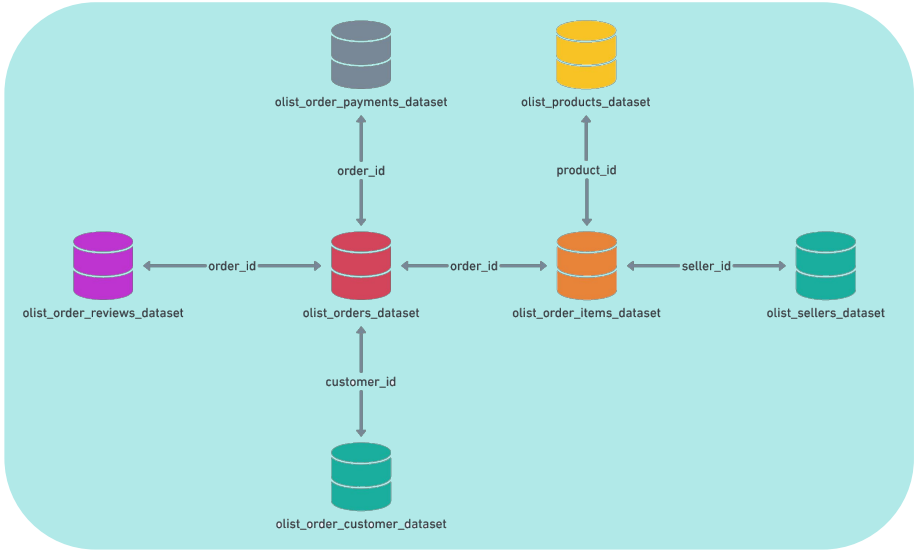
2010

- **ACID**
- **Read Committed**
- **Indexing**
- **Transactions**

The Datasets

Brazilian E-Commerce
Orders, Customers, Sellers

Six Degrees of Francis Bacon
A social network

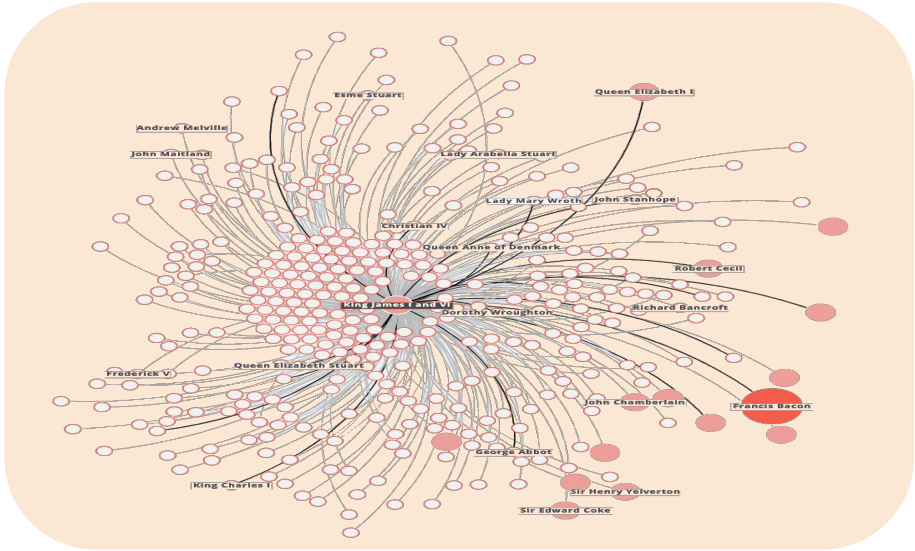


- **7 Tables**
- **551,449 Records**
- **637,361 Relationships**

The Datasets

Brazilian E-Commerce
Orders, Customers, Sellers

Six Degrees of Francis Bacon
A social network



- **2 Tables**
- **15,801 People**
- **171,408 Relationships**

Importing Data From CSV

PostgreSQL

```
CREATE TABLE Customer (
  ID VARCHAR(50) PRIMARY KEY,
  uniqueID VARCHAR(50),
  zipCode INT,
  city VARCHAR(50),
  state VARCHAR(2)
);

COPY Customer
FROM 'file.csv"
DELIMITER ',
CSV HEADER;
```

Neo4j

```
LOAD CSV WITH HEADERS
FROM 'file' AS line
CREATE (:Customer {
  ID: line.customer_id,
  uniqueID: line.customer_unique_id,
  zipCode:
    toInteger(line.customer_zip_code),
  city: line.customer_city,
  state: line.customer_state
})
```

Importing Data From CSV

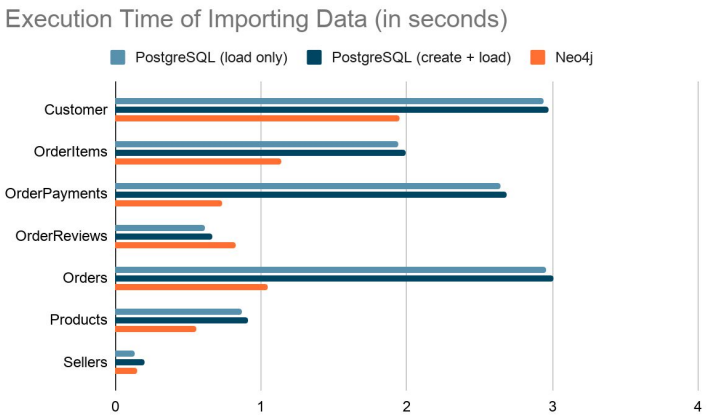
PostgreSQL

Need to create table schema first, then import data.

The cost of creating a table is not significant.

Neo4j

Can create the node type and import data in the same query.



Both systems take roughly the same time to import these tables.

Additionally, it takes Neo4j between 500ms - 1 second to add relationships between nodes.

Importing Data From CSV

PostgreSQL

```
CREATE TABLE Relationships (
    ID INT PRIMARY KEY,
    person1ID INT REFERENCES People(ID),
    person2ID INT REFERENCES People(ID),
    startYearType VARCHAR(5),
    startYear INT,
    endYearType VARCHAR(5),
    endYear INT);

COPY Relationships FROM 'file'
DELIMITER ',' CSV HEADER;
```

Neo4j

```
LOAD CSV WITH HEADERS
FROM 'file' AS row
MATCH (p1: People {ID:
   toInteger(row.person1ID)})
MATCH (p2: People {ID:
   toInteger(row.person2ID)})
CREATE (p1)-[:KNOWS {
   ID: toInteger(row.ID),
   startYearType: row.startYearType,
   startYear: toInteger(row.startYear),
   endYearType: row.endYearType,
   endYear: toInteger(row.endYear)}]
->(p2);
```

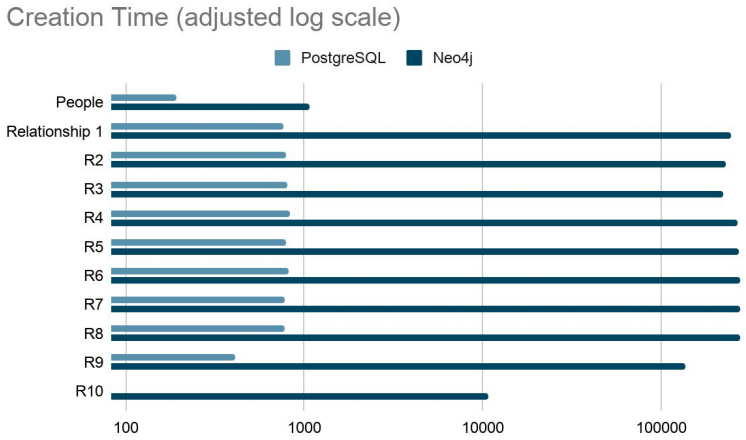
Importing Data From CSV

PostgreSQL

Data takes the same time to import regardless of relationships between nodes

Neo4j

In order to create relationships, we have to create a cartesian product to fetch the nodes. Creation time takes longer depending on the relationships between nodes.

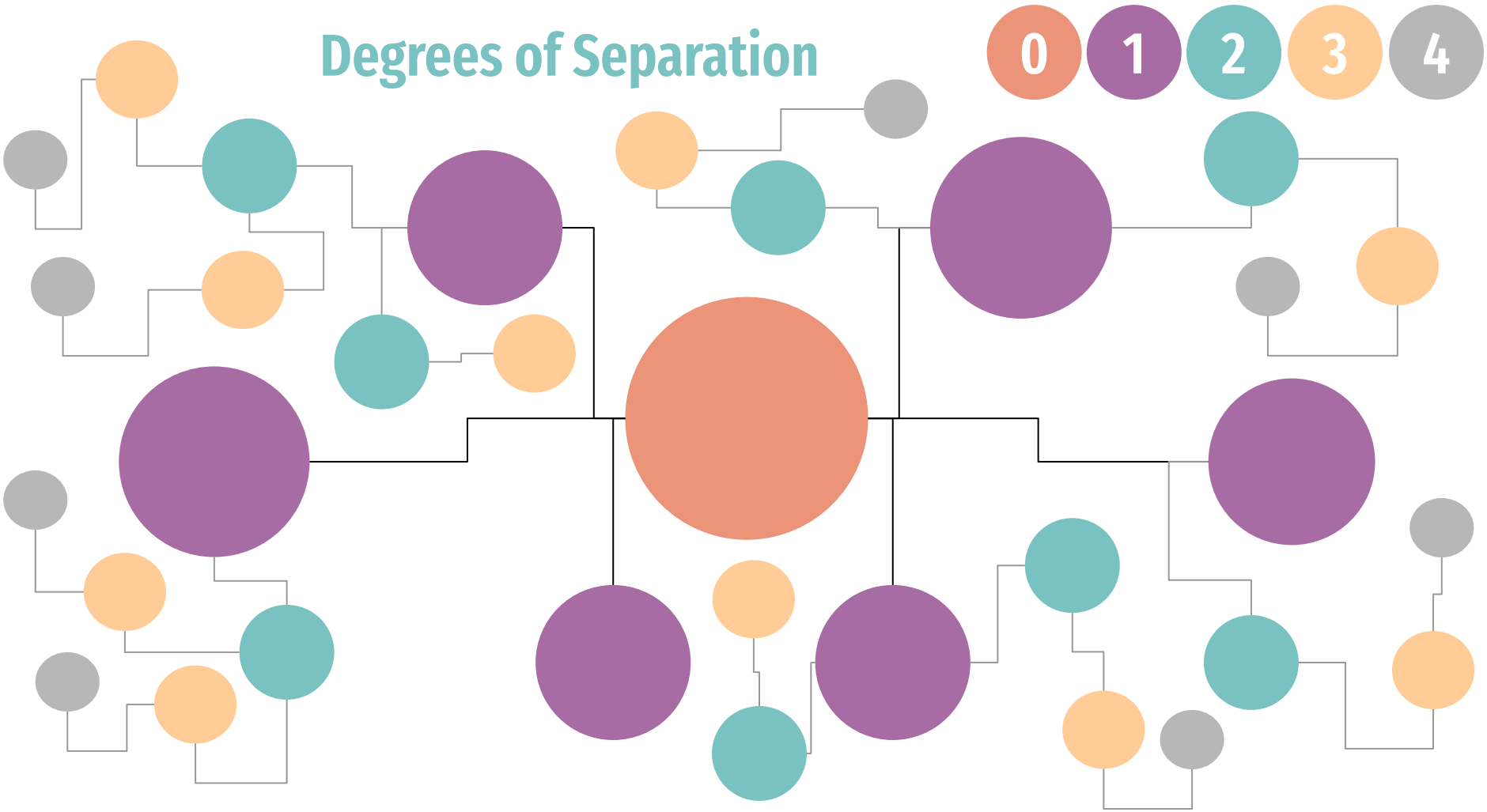


Neo4j takes roughly **300 times longer** to import Relationships than Postgres

800 ms → ~4 minutes

Degrees of Separation

0 1 2 3 4



Degrees of Separation

In a graph model, we can find these by just requesting all nodes within X relationships of our starting node.

In the relational model, there is no intuitive way to do this. The simplest is to create a view for every level, and perform joins with the Relationship table on those views.

The graph database is not heavily impacted as the degrees of separation increase, since the relationships can be queried on directly.

	Postgres	Neo4j
0	0.052s	0.007s
1	0.077s	0.011s
2	6.47s	0.014s
3	2m 50s	0.017s
4	22m	0.11s

Degrees of Separation: Larger Result Size

0 1 2 3 4

In order to test the systems' ability to handle larger intermediate sizes, I ran the degrees of separation again, but this time on all males instead of one person's network

	Postgres	Neo4j
0	0.084s	0.008s
1	2.379s	0.221s
2	38m 39s	17.518s
3	2hr 38m	312.119s

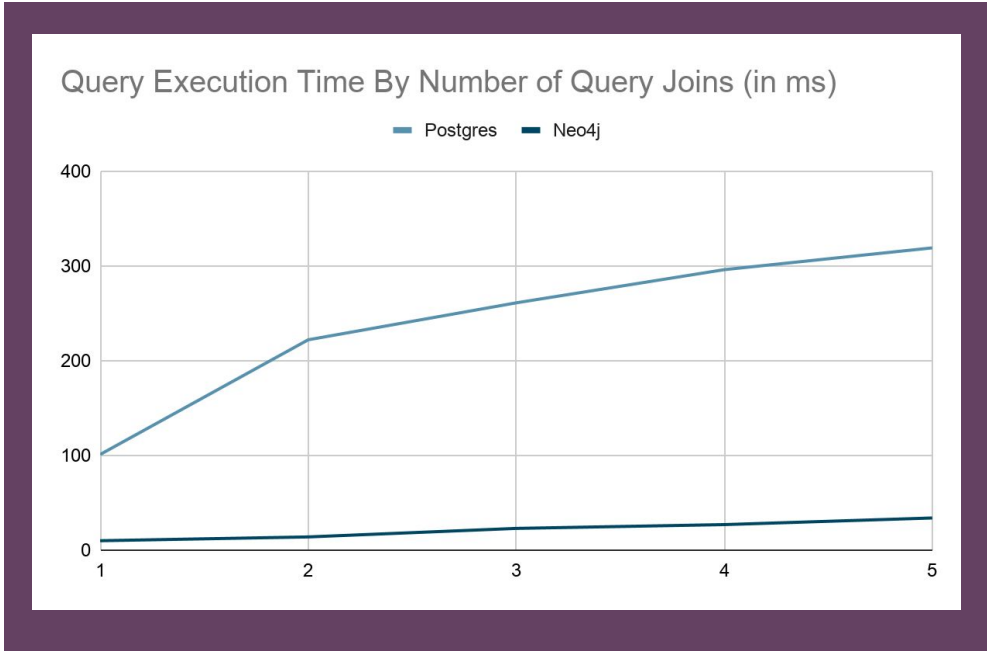
The relational model performed very poorly.

Number of Table Joins

The last set of experiments required multiple joins on the same table.

How is the system impacted as we require more table joins?

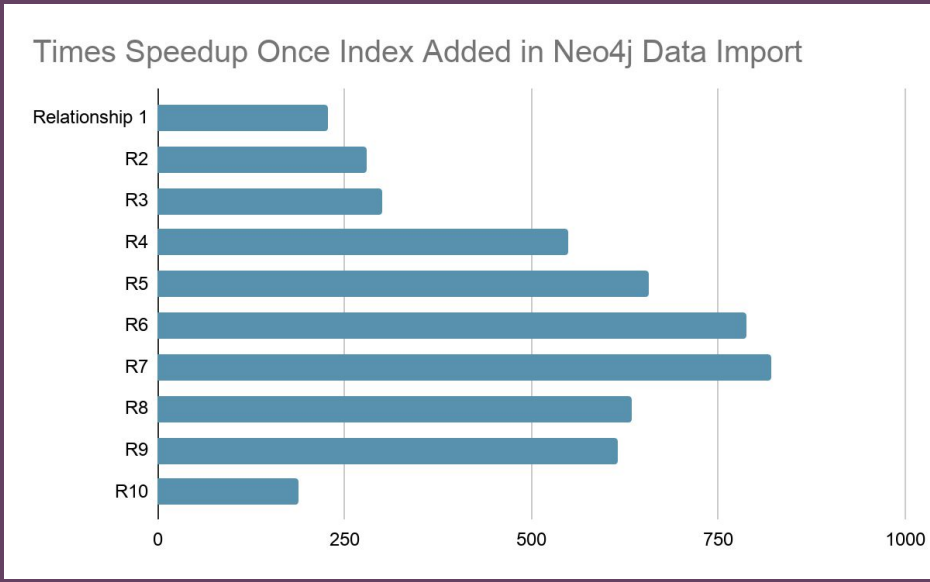
Neo4j handles joins much faster than Postgres. The extra cost to add new tables to a query in Neo4j is inconsequential, but does add significant time in Postgres.



Indexing

Neo4j's greatest weakness thus far was in querying and joining nodes to create relationships. Can this be improved with an index on primary key?

Anywhere between 220 to 820 times faster.



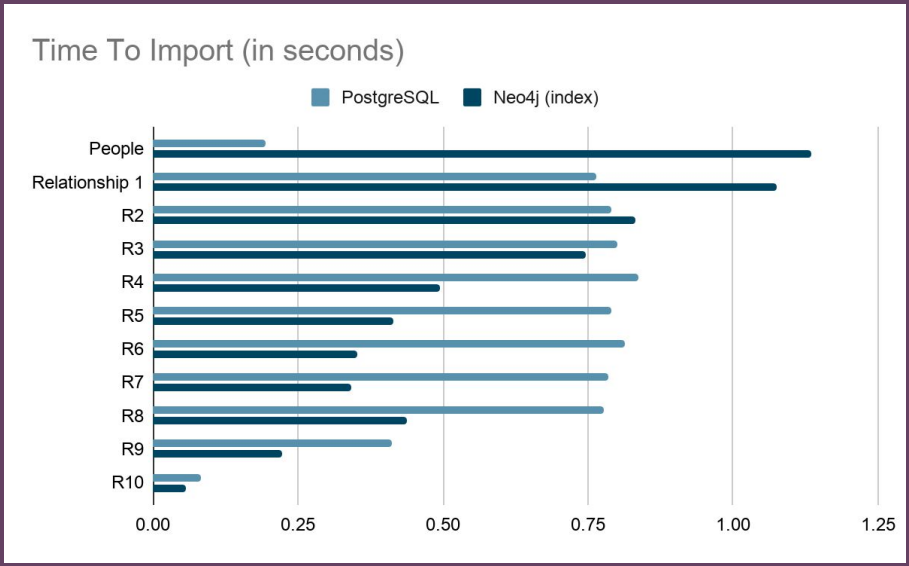
Indexing

Neo4j's greatest weakness thus far was in querying and joining nodes to create relationships. Can this be improved with an index on primary key?

The index only takes 192 ms to create, and the speedup is very noticeable.

With the index, Neo4j imports this data and creates Relationships in roughly the same time that Postgres does.

Note that Neo4j gets faster with every new Relationship import.



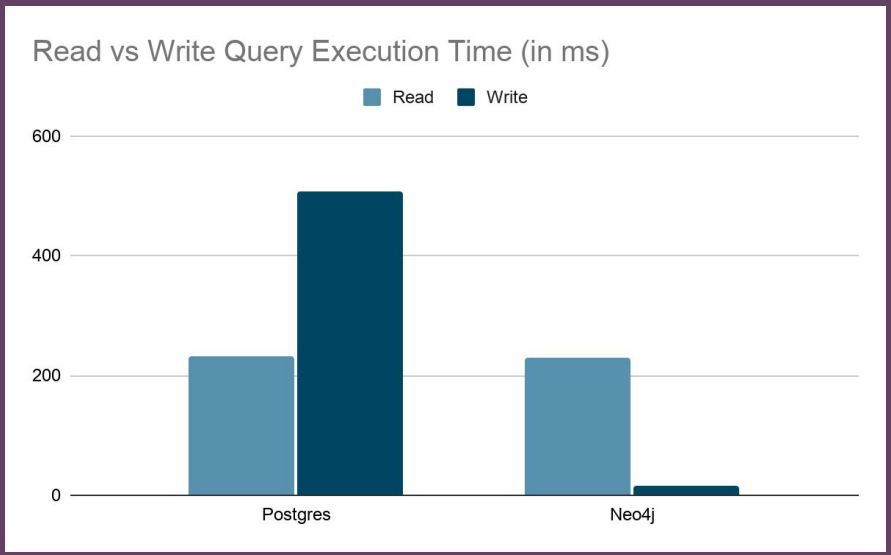
Writes

This last set of experiments tested the performance of both systems when performing record updates.

Two queries were run - one with and one without an attribute update.

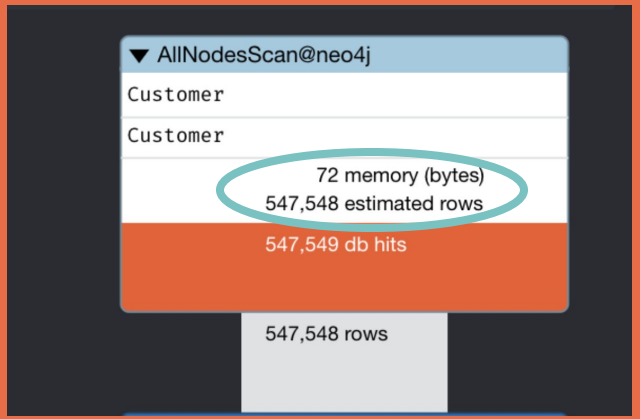
Postgres takes roughly twice as long to perform an update than a simple read.

Neo4j performs writes faster, why could this be?

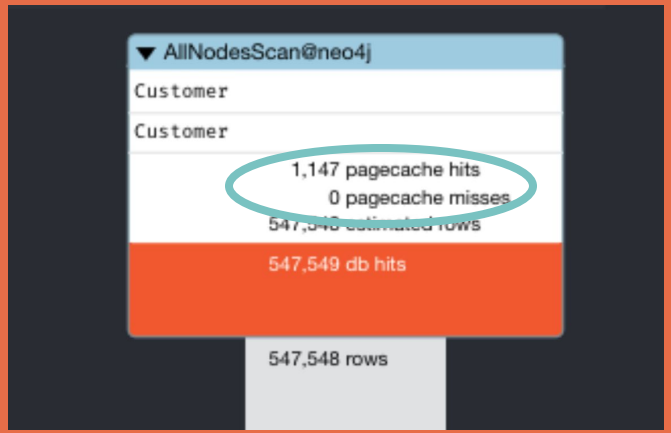


Writes

Using EXPLAIN to see the Query Plans...



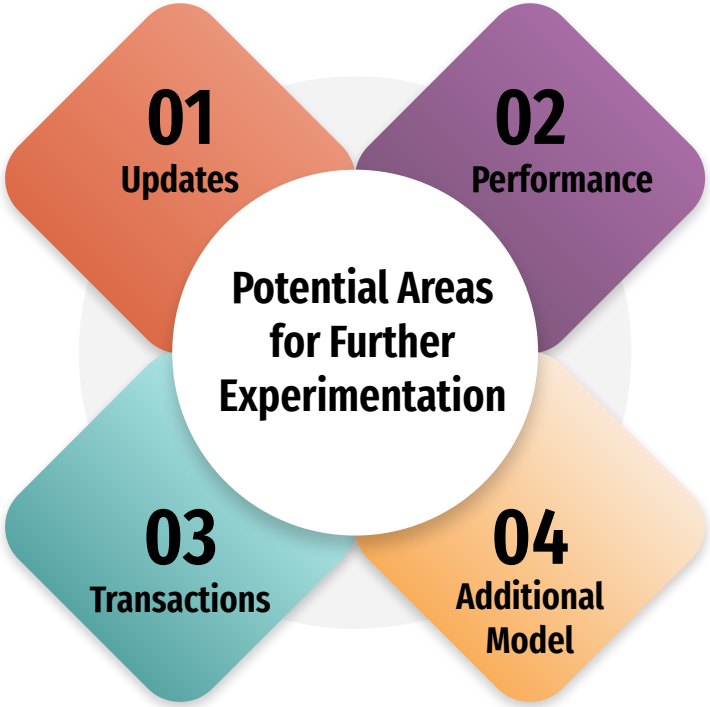
Read Plan



Write Plan

… we can see that Neo4j stores write query results in a page cache, which leads to faster access.

Further Work



Update Testing

How do the systems differ in performance on updating relationships?

Measuring Performance Differently

CPU usage, space amplification, ...

Transaction Processing

READ COMMITTED

Adding Another System

EG A key value store

Key Takeaways

Joins

Neo4j performs much faster joins since it is able to query on relationships.

Consistent

Postgres is more consistent creation time since it is not dependent on relationships between data.

Support

Since relational systems have been around longer, they tend to have support for more functionality, eg isolation levels

References

Olist and André Sionek, “Brazilian E-Commerce Public Dataset by Olist.” Kaggle, 2018, doi: 10.34740/KAGGLE/DSV/195341.

SDFB Team, Six Degrees of Francis Bacon: Reassembling the Early Modern Social Network. www.sixdegreesoffrancisbacon.com (August 29, 2017).

https://github.com/jgarc243/CSE215