

## 14\_Specialized Storage Paradigms

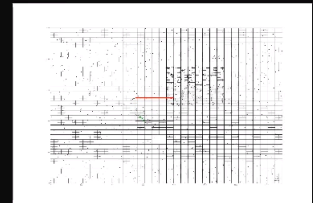
### *Specialized Storage Paradigms*

Blob Store Time Series DB Graph DB Spatial DB  
+ Quadtree

GCS  
S3

InfluxDB  
Prometheus

Neo4j



Blob Store (binary large object: arbitrary unstructure data - video, imge, large file) - GCS/S3

Time Series DB - InfluxDB, Prometheus

Graph DB - Neo4j


```
cypher_query.cql — specializedStorageParadigms
cypher_query.cql x sql_query.sql
cypher_query.cql
1  -- Populate data.
2  CREATE (facebook:Company {name:'Facebook'})
3
4  CREATE (clement:Candidate {name:'Clement'})
5  CREATE (antoine:Candidate {name:'Antoine'})
6  CREATE (simon:Candidate {name:'Simon'})
7
8  CREATE (alex:Interviewer {name:'Alex'})
9  CREATE (meghan:Interviewer {name:'Meghan'})
10 CREATE (marli:Interviewer {name:'Marli'})
11 CREATE (sandeep:Interviewer {name:'Sandeep'})
12 CREATE (molly:Interviewer {name:'Molly'})
13 CREATE (akshay:Interviewer {name:'Akshay'})
14 CREATE (aditya:Interviewer {name:'Aditya'})
15 CREATE (brandon:Interviewer {name:'Brandon'})
16 CREATE (pedro:Interviewer {name:'Pedro'})
17 CREATE (ryan:Interviewer {name:'Ryan'})
18 CREATE (xi:Interviewer {name:'Xi'})
19 CREATE (simran:Interviewer {name:'Simran'})
20 CREATE (amanda:Interviewer {name:'Amanda'})
21
22 CREATE (alex)-[:INTERVIEWED {score: 'passed'}]->(clement)
23 CREATE (meghan)-[:INTERVIEWED {score: 'passed'}]->(clement)
24 CREATE (simran)-[:INTERVIEWED {score: 'passed'}]->(clement)
25 CREATE (molly)-[:INTERVIEWED {score: 'failed'}]->(clement)
26 CREATE (marli)-[:INTERVIEWED {score: 'failed'}]->(antoine)
27 CREATE (akshay)-[:INTERVIEWED {score: 'passed'}]->(antoine)
28 CREATE (aditya)-[:INTERVIEWED {score: 'passed'}]->(antoine)
29 CREATE (meghan)-[:INTERVIEWED {score: 'passed'}]->(antoine)
30 CREATE (marli)-[:INTERVIEWED {score: 'failed'}]->(simon)
31 CREATE (meghan)-[:INTERVIEWED {score: 'failed'}]->(simon)
32 CREATE (brandon)-[:INTERVIEWED {score: 'passed'}]->(simon)
33 CREATE (xi)-[:INTERVIEWED {score: 'failed'}]->(simon)
34
35 CREATE (ryan)-[:APPLIED {status: 'rejected'}]->(facebook)
36 CREATE (simran)-[:APPLIED {status: 'accepted'}]->(facebook)
37 CREATE (xi)-[:APPLIED {status: 'rejected'}]->(facebook)
38 CREATE (molly)-[:APPLIED {status: 'rejected'}]->(facebook)
39 CREATE (alex)-[:APPLIED {status: 'rejected'}]->(facebook);
40
41 -- Find the interviewers who interviewed and failed Clement
42 -- and who also applied to and got rejected by Facebook.
43 MATCH (interviewer:Interviewer)-[:INTERVIEWED {score:'failed'}]->(:Candidate {name:
44 WHERE (interviewer)-[:APPLIED {status:'rejected'}]->(Company {name:'Facebook'})
```

```
sql_query.sql — specializedStorageParadigms

cypher_query.cql  sql_query.sql X

sql_query.sql
1  -- Find the interviewers who interviewed and failed Clement
2  -- and who also applied to and got rejected by Facebook.
3  SELECT interviewers.name
4  FROM (
5      ... candidates
6      ... JOIN interviews ON (candidates.id = interviews.candidate_id AND candidates.name
7      ... JOIN interviewers ON (interviewers.id = interviews.interviewer_id)
8  )
9  WHERE EXISTS (
10     ... SELECT *
11     ... FROM applications
12     ... WHERE company = 'Facebook' AND candidate_id = interviewers.id AND status = 're
13  );
```

# Specialized Storage Paradigms

**Database Information**

☆

🔍

Node Labels

\*(17)

Candidate

Company

Interviewer

Relationship Types

\*(17)

APPLIED

INTERVIEWED

Property Keys

name

score

status

\$

☆

🔍

▶

\$ MATCH (n) RETURN n LIMIT 25

🔍

🔗

↕

⬆

🔄

✕

Graph

\*(17)

Interviewer(13)

Candidate(3)

Company(1)

\*(17)


APPLIED(5)

INTERVIEWED(12)

Table

Text

Code



```
graph LR; Ryan((Ryan)) -- APPLIED --> Facebook((Facebook)); Xi((Xi)) -- APPLIED --> Facebook; Xi -- INTERVIEWED --> Simon((Simon)); Facebook -- APPLIED --> Simran((Simran)); Facebook -- APPLIED --> Molly((Molly)); Facebook -- APPLIED --> Alex((Alex)); Simran -- INTERVIEWED --> Clement((Clement)); Molly -- INTERVIEWED --> Clement; Alex -- INTERVIEWED --> Clement; Simon -- INTERVIEWED --> Clement; Simon -- INTERVIEWED --> Meghan((Meghan)); Meghan -- INTERVIEWED --> Clement; Meghan -- INTERVIEWED --> Marli((Marli));
```

Candidate

<id>: 24

name: Simon

\$ CREATE (facebook:Company {name:'Fa...

🔗

↕

⬆

🔄


✕

Table

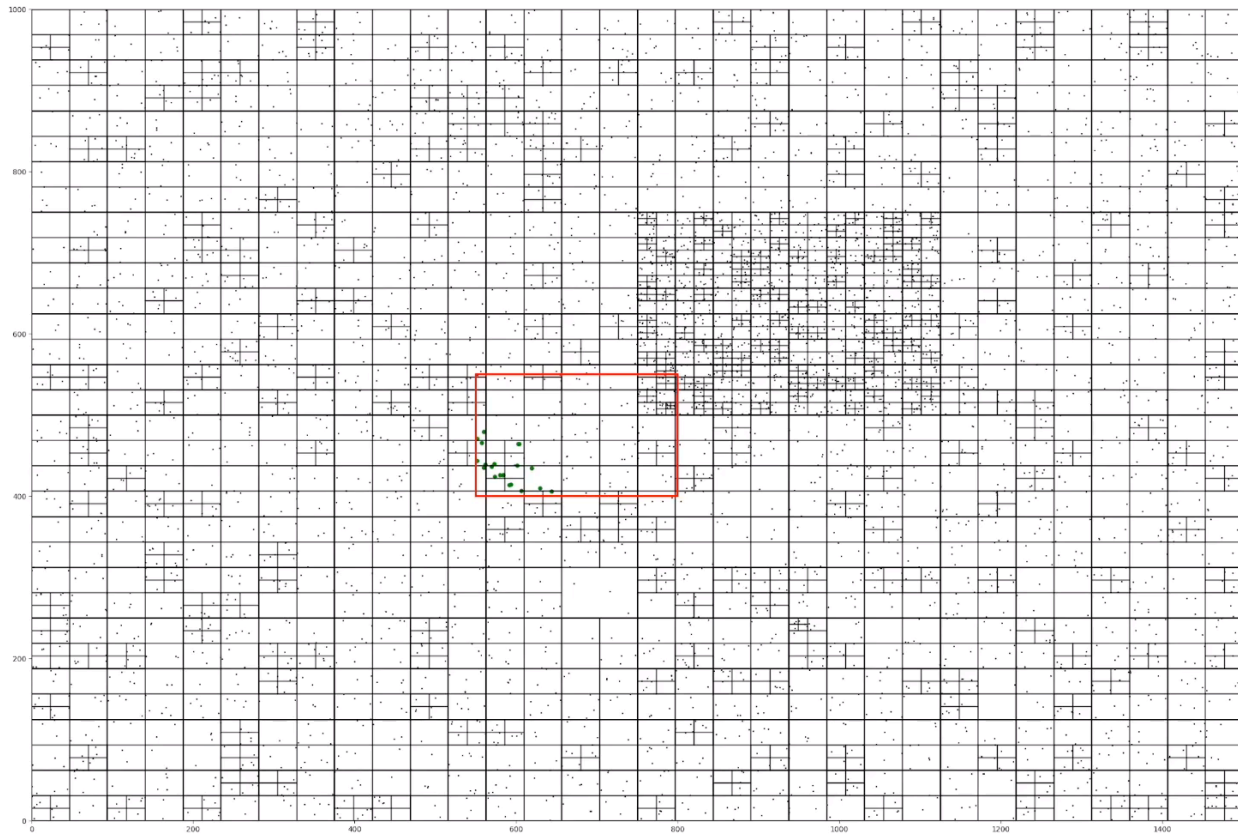
Code

Added 17 labels, created 17 nodes, set 34 properties, created 17 relationships, com...

Added 17 labels, created 17 nodes, set 34 properties, created 17 relationships, comp...



Sptial DB - Quadree



## 8 Prerequisites

### Relational Database

A type of structured database in which data is stored following a tabular format; often supports powerful querying using SQL.

### Non-Relational Database

In contrast with relational database (SQL databases), a type of database that is free of imposed, tabular-like structure. Non-relational databases are often referred to as NoSQL databases.

### SQL

Structured Query Language. Relational databases can be used using a derivative of SQL such as PostgreSQL in the case of Postgres.

### SQL Database

Any database that supports SQL. This term is often used synonymously with "Relational Database", though in practice, not every relational database supports SQL.

### NoSQL Database

Any database that is not SQL-compatible is called NoSQL.

### Key-Value Store

A Key-Value Store is a flexible NoSQL database that's often used for caching and dynamic configuration. Popular options include DynamoDB, Etcd, Redis, and ZooKeeper.

### Database Index

A special auxiliary data structure that allows your database to perform certain queries much faster. Indexes can typically only exist to reference structured data, like data stored in relational databases. In practice, you create an index on one or multiple columns in your database to greatly speed up **read** queries that you run very often, with the downside of slightly longer **writes** to your database, since writes have to also take place in the relevant index.

### Postgres ⚡

A relational database that uses a dialect of SQL called PostgreSQL. Provides ACID transactions.

## 11 Key Terms



### Blob Storage

Widely used kind of storage, in small and large scale systems. They don't really count as databases per se, partially because they only allow the user to store and retrieve data based on the name of the blob. This is sort of like a key-value store but usually blob stores have different guarantees. They might be slower than KV stores but values can be megabytes large (or sometimes gigabytes large). Usually people use this to store things like **large binaries, database snapshots, or images** and other static assets that a website might have.

Blob storage is rather complicated to have on premise, and only giant companies like Google and Amazon have infrastructure that supports it. So usually in the context of System Design interviews you can assume that you will be able to use **GCS** or **S3**. These are blob storage services hosted by Google and Amazon respectively, that cost money depending on how much storage you use and how often you store and retrieve blobs from that storage.

### Time Series Database

A **TSDB** is a special kind of database optimized for storing and analyzing time-indexed data: data points that specifically occur at a given moment in time. Examples of TSDBs are InfluxDB, Prometheus, and Graphite.

### Graph Database

A type of database that stores data following the graph data model. Data entries in a graph database can have explicitly defined relationships, much like nodes in a graph can have edges.

Graph databases take advantage of their underlying graph structure to perform complex queries on deeply connected data very fast.

Graph databases are thus often preferred to relational databases when dealing with systems where data points naturally form a graph and have multiple levels of relationships—for example, social networks.

### Cypher

A **graph query language** that was originally developed for the Neo4j graph database, but that has since been standardized to be used with other graph databases in an effort to make it the "SQL for graphs."

Cypher queries are often much simpler than their SQL counterparts. Example Cypher query to find data in **Neo4j**, a popular graph database:

```
MATCH (some_node:SomeLabel)-[:SOME_RELATIONSHIP]->(some_other_node:SomeLabel {some_property:'value'})
```

### Spatial Database

A type of database optimized for storing and querying spatial data like locations on a map. Spatial databases rely on spatial indexes like **quadtrees** to quickly perform spatial queries like finding all locations in the vicinity of a region.

## Quadtree

A tree data structure most commonly used to index two-dimensional spatial data. Each node in a quadtree has either zero children nodes (and is therefore a leaf node) or exactly four children nodes.

Typically, quadtree nodes contain some form of spatial data—for example, locations on a map—with a maximum capacity of some specified number **n**. So long as nodes aren't at capacity, they remain leaf nodes; once they reach capacity, they're given four children nodes, and their data entries are split across the four children nodes.

A quadtree lends itself well to storing spatial data because it can be represented as a grid filled with rectangles that are recursively subdivided into four sub-rectangles, where each quadtree node is represented by a rectangle and each rectangle represents a spatial region. Assuming we're storing locations in the world, we can imagine a quadtree with a maximum node-capacity **n** as follows:

- The root node, which represents the entire world, is the outermost rectangle.
- If the entire world has more than **n** locations, the outermost rectangle is divided into four quadrants, each representing a region of the world.
- So long as a region has more than **n** locations, its corresponding rectangle is subdivided into four quadrants (the corresponding node in the quadtree is given four children nodes).
- Regions that have fewer than **n** locations are undivided rectangles (leaf nodes).
- The parts of the grid that have many subdivided rectangles represent densely populated areas (like cities), while the parts of the grid that have few subdivided rectangles represent sparsely populated areas (like rural areas).

Finding a given location in a perfect quadtree is an extremely fast operation that runs in  $\log_4(x)$  time (where **x** is the total number of locations), since quadtree nodes have four children nodes.

## Google Cloud Storage ⚡

GCS is a blob storage service provided by Google.

## S3 ⚡

S3 is a blob storage service provided by Amazon through **Amazon Web Services (AWS)**.

## InfluxDB ⚡

A popular open-source time series database.

## Prometheus ⚡

A popular open-source time series database, typically used for monitoring purposes.

## Neo4j ⚡

A popular graph database that consists of **nodes**, **relationships**, **properties**, and **labels**.