

Databases You Need in AI Era

SQL is the best program language in the world

Pascal Sun

UWA NLP-TLP Group

The University of Western Australia

November 29, 2024

Table of Contents

1 Introduction

2 PostgreSQL

- Data Model
- Migration Management
- Object Relationship Management
- Index and Query Optimization
- Connection
- Performance Tuning
- DBA

3 GraphDB

- User scenarios
- Data Modelling
- CRUD in GraphDB
- Connection

4 VectorDB

5 Q&A





PostgreSQL variants

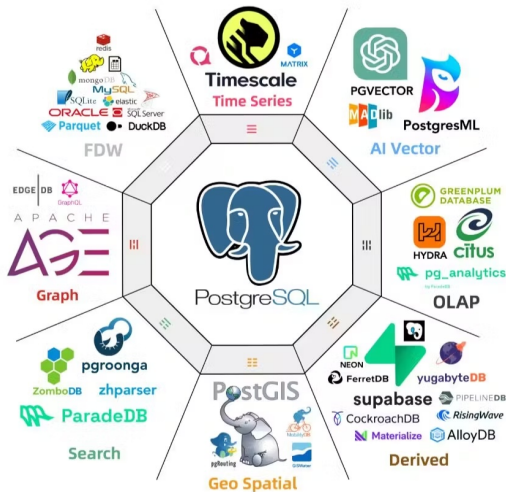


Figure: All in PostgreSQL in future?

Just use Postgres for Everything

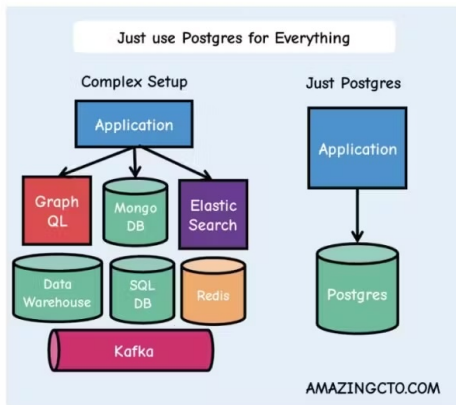


Figure: Timescale Opinion¹

¹<https://www.timescale.com/blog/postgres-for-everything/>

When to Use NoSQL

Ideal Use Cases

- Big data applications
- Real-time streaming
- Dynamic schema needs
- High write loads

PostgreSQL Solution

- JSONB for flexibility
- Partitioning
- Native replication
- Foreign data wrappers

Important Note

If you are building a full stack application, it means you **must** have a relational database! Personal recommendation is PostgreSQL.

Why we need GraphDB?

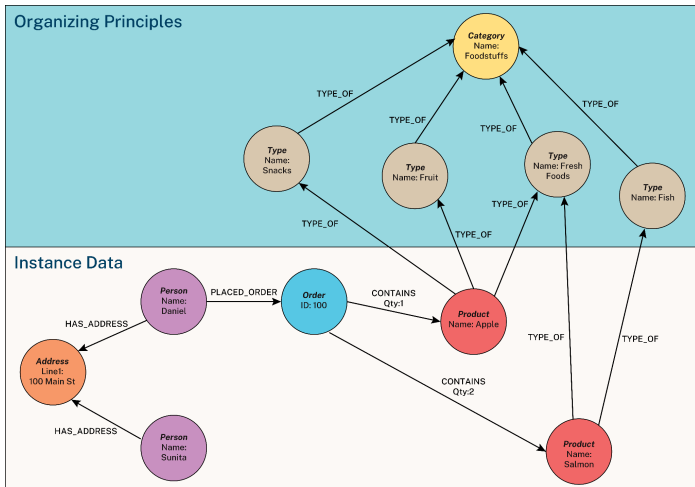


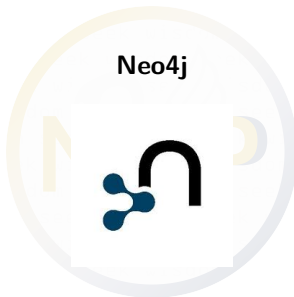
Figure: Ontology and Knowledge Graph

Graph Database Vendors

GraphDB



Neo4j



MemGraph



Vector Databases

Name	Free tier	Self-Host	Cloud	Open Source	License	Hybrid Search
Qdrant	Yes	Yes	Yes	Yes	Apache 2.0	Yes
Weaviate	Yes	Yes	Yes	Yes	Apache 2.0	Yes
Pinecone	Yes	No	Yes	No	Commercial	Yes
Milvus	Yes	Yes	Yes	Yes	Apache 2.0	Partial
ChromaDB	Yes	Yes	Not Yet	Yes	Apache 2.0	Partial

Table: Comparison of Vector Databases, adopted from²

Qdrant³ appears to be a promising contender and a good starting point. For a simpler option, consider **ChromaDB**⁴. It's analogous to SQLite in the relational database world - lightweight and easy to get started with. **Pinecone**⁵ is currently the top-rated option; however, it is only available in a cloud-based commercial version, making it unsuitable for some use cases.

²<https://medium.com/the-ai-forum/which-vector-database-should-you-use-choosing-the-best-one-for-your-needs-5108ec7ba133>

³<https://qdrant.tech/documentation/quickstart/>

⁴<https://docs.trychroma.com/getting-started>

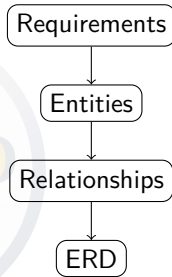
⁵<https://www.pinecone.io/>



How to model business logic in databases?

Steps

- 1 Gather Requirements
- 2 Identify Entities
- 3 Define Relationships
- 4 Normalize Data
- 5 Create ERD



Best Practice

Start with business logic, not technical constraints

How to create and change database structures?

1 Initial Design

- You design User table with email only
- Create table via SQL or ORM

2 Business Change

- Product team: "We need user's phone number!"
- How to update existing table?

3 Challenge

- Multiple environments (Dev/Test/Prod)
- Multiple developers
- How to track changes?

Solution

Database Migration Tools: Track and manage DB changes like code!

Migration Example

Initial Migration (001_create_users.sql)

```
1 CREATE TABLE users (  
2     id SERIAL PRIMARY KEY,  
3     email VARCHAR(255) UNIQUE,  
4     created_at TIMESTAMP DEFAULT NOW()  
5 );  
6
```

Add Column (002_add_user_name.sql)

```
1 ALTER TABLE users  
2 ADD COLUMN name VARCHAR(100);  
3
```

Warning

Never modify existing migrations - always create new ones!

Migration Management Tools

Manual Migration Risks

- Human errors
- Destructive commands
- No version control
- Inconsistent states

Django Migration Benefits

- Version controlled
- Automated checks
- Rollback support
- Change validation

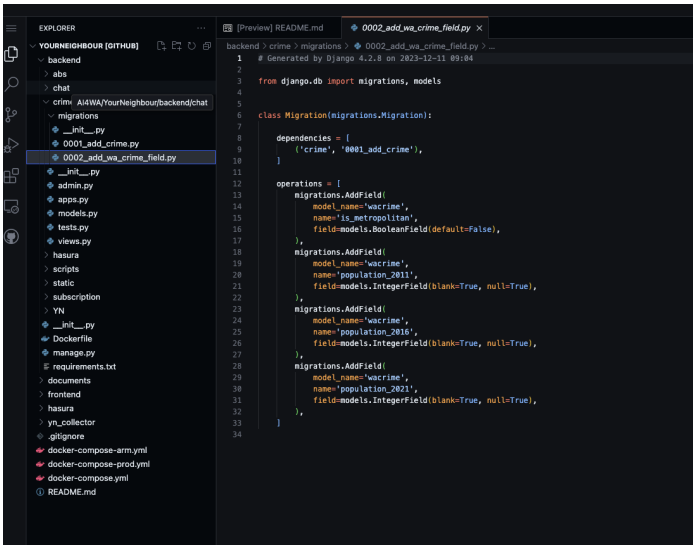
Django Migration Example

```
1 python manage.py makemigrations
2 python manage.py migrate
3
```

Warning

Never write raw SQL migrations unless absolutely necessary!

Example: Django Migration Files



```
1 # Generated by Django 4.2.8 on 2023-12-11 09:18:4
2
3 from django.db import migrations, models
4
5
6 class Migration(migrations.Migration):
7
8     dependencies = [
9         ('crime', '0001_add_crime'),
10     ]
11
12     operations = [
13         migrations.AddField(
14             model_name='wacrine',
15             name='is_metropolitan',
16             field=models.BooleanField(default=False),
17         ),
18         migrations.AddField(
19             model_name='wacrine',
20             name='population_2011',
21             field=models.IntegerField(blank=True, null=True),
22         ),
23         migrations.AddField(
24             model_name='wacrine',
25             name='population_2016',
26             field=models.IntegerField(blank=True, null=True),
27         ),
28         migrations.AddField(
29             model_name='wacrine',
30             name='population_2021',
31             field=models.IntegerField(blank=True, null=True),
32         ),
33     ]
34
```

Figure: Generated Django Migration Files

Example: Django Migration Table

id	app	name	applied
1	abs	0001_abs_init	2023-10-10 14:27:15.398848 +00:00
2	contenttypes	0001_initial	2023-10-10 14:27:15.425765 +00:00
3	auth	0001_initial	2023-10-10 14:27:15.541973 +00:00
4	admin	0001_initial	2023-10-10 14:27:15.582418 +00:00
5	admin	0002_logentry_remove_auto_add	2023-10-10 14:27:15.630353 +00:00
6	admin	0003_logentry_add_action_flag_choices	2023-10-10 14:27:15.628330 +00:00
7	contenttypes	0002_remove_content_type_name	2023-10-10 14:27:15.650443 +00:00
8	auth	0002_alter_permission_name_max_length	2023-10-10 14:27:15.666793 +00:00
9	auth	0003_alter_user_email_max_length	2023-10-10 14:27:15.677043 +00:00
10	auth	0004_alter_user_username_opts	2023-10-10 14:27:15.686423 +00:00
11	auth	0005_alter_user_last_login_null	2023-10-10 14:27:15.693884 +00:00
12	auth	0006_require_contenttypes_0002	2023-10-10 14:27:15.696543 +00:00
13	auth	0007_alter_validators_add_error_messages	2023-10-10 14:27:15.704099 +00:00
14	auth	0008_alter_user_username_max_length	2023-10-10 14:27:15.715332 +00:00
15	auth	0009_alter_user_last_name_max_length	2023-10-10 14:27:15.731022 +00:00
16	auth	0010_alter_group_name_max_length	2023-10-10 14:27:15.742632 +00:00
17	auth	0011_update_proxy_permissions	2023-10-10 14:27:15.754380 +00:00
18	auth	0012_alter_user_first_name_max_length	2023-10-10 14:27:15.762226 +00:00
19	crime	0001_add_crime	2023-10-10 14:27:15.782368 +00:00
20	hasura	0001_init_hasura	2023-10-10 14:27:15.836076 +00:00
21	sessions	0001_initial	2023-10-10 14:27:15.868322 +00:00
22	abs	0002_add_hydrate	2023-10-12 15:36:06.678934 +00:00
23	hasura	0002_add_comments	2023-11-01 16:17:03.936864 +00:00
24	hasura	0003_add_original	2023-11-01 16:17:03.932664 +00:00
25	subscription	0001_add_user	2023-12-06 16:56:14.489889 +00:00
26	chat	0001_add_chat_record	2023-12-13 06:49:38.338979 +00:00
27	chat	0002_relax_chat_record	2023-12-13 06:49:38.363458 +00:00
28	crime	0002_add_aa_crime_field	2023-12-13 06:49:38.379255 +00:00

Figure: Django Migration Table

How to CRUD the data in databases?

Raw SQL

```
1 SELECT * FROM users WHERE age > 18;  
2
```

ORM (Django)

```
1 User.objects.filter(age__gt=18)  
2
```

Why ORM?
Write database operations in your programming language!

ORM Benefits

Developer Benefits

- Type safety
- Code completion
- Cross-DB support
- Less boilerplate

Business Benefits

- Faster development
- Fewer errors
- Easy maintenance
- DB independence



CRUD with ORM

```
1 # Create
2 user = User.objects.create(
3     name="John", email="john@example.com"
4 )
5
6 # Read
7 users = User.objects.filter(age__gt=18)
8
9 # Update
10 User.objects.filter(id=1).update(
11     name="New Name"
12 )
13
14 # Delete
15 User.objects.filter(inactive=True).delete()
16
```

Best Practice

Use ORM for simple queries, raw SQL for complex ones

Popular ORM Tools

Python

- Django ORM
- SQLAlchemy

Other Languages

- Hibernate (Java)
- Entity Framework (.NET)
- Prisma (Node.js)

Django Model Example

```
1 class User(models.Model):  
2     name = models.CharField(max_length=100)  
3     email = models.EmailField(unique=True)  
4
```

Why Raw SQL for Complex Queries?

ORM Limitations

- Complex joins
- Custom functions
- Performance tuning
- Advanced features

Raw SQL Benefits

- Full DB control
- Better performance
- Query optimization
- Advanced indexing

When to Use Raw SQL

Complex queries requiring performance optimization

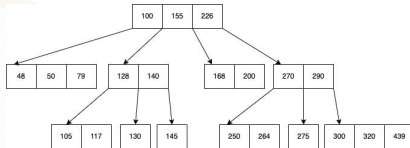
Understanding Indexes

Without Index

- Full table scan
- Slow for large tables
- High I/O cost

With Index

- Quick lookup
- Sorted access
- Reduced I/O



Creating Indexes

Basic Index

```
1 CREATE INDEX idx_user_email
2 ON users(email);
3
```

Composite Index

```
1 CREATE INDEX idx_user_name_email
2 ON users(name, email);
3
```

Unique Index

```
1 CREATE UNIQUE INDEX idx_unique_email
2 ON users(email);
3
```

Query Optimization with EXPLAIN ANALYZE

Example Query

```
1 EXPLAIN ANALYZE
2 SELECT * FROM abs_abssuburbgeo
3 WHERE ste_name21 = 'Western Australia';
4
```

The screenshot shows a database query tool interface. At the top, there are tabs for different databases: 'auth_grou...ermissions [ChatGEO-Prod]', 'authentic...ser_groups [ChatGEO-Prod]', 'abs_g35_n...y_resident [YN-PROD]', and 'abs_abssub'. Below the tabs, there is a toolbar with icons for running, refreshing, and other query actions. The main area displays the query: 'EXPLAIN ANALYZE SELECT * FROM abs_abssuburbgeo WHERE ste_name21 = 'Western Australia';'. Below the query, there is a section for the query plan, labeled 'Session' and 'Result 3'. The query plan shows the following steps:

- 1 Seq Scan on abs_abssuburbgeo (cost=0.00..5887.91 rows=1701 width=9394) (actual time=0.023..12.377 rows=1701 loops=1)
- 2 Filter: (ste_name21 = 'Western Australia'::text)
- 3 Rows Removed by Filter: 13652
- 4 Planning Time: 0.143 ms
- 5 Execution Time: 12.600 ms

Query Optimization with EXPLAIN ANALYZE

Index Creation

```
1 CREATE INDEX ste2_name
2 ON abs_abssuburbgeo(ste_name21);
3
```



The screenshot shows a database query plan for a query on the table 'abs_abssuburbgeo'. The plan is displayed in a dark-themed interface with a 'Session' tab and a 'Result 5' tab. The query plan is shown in a table format with 7 rows. The first row is the main query plan, and the subsequent rows show the execution details of the subplans. The plan shows a Bitmap Heap Scan on the table 'abs_abssuburbgeo' with a cost of 21.47..3701.21, 1701 rows, and a width of 9394. The actual time taken was 0.865..8.426 seconds. The subplan is a Bitmap Index Scan on the index 'ste2_name' with a cost of 0.00..21.04, 1701 rows, and a width of 0. The actual time taken was 0.439..0.440 seconds. The plan also shows the index condition, the number of heap blocks, and the planning and execution times.

1	Bitmap Heap Scan on abs_abssuburbgeo	(cost=21.47..3701.21 rows=1701 width=9394) (actual time=0.865..8.426 rows=1701 loops=1)
2	Recheck Cond: (ste_name21 = 'Western Australia'::text)	
3	Heap Blocks: exact=633	
4	-> Bitmap Index Scan on ste2_name	(cost=0.00..21.04 rows=1701 width=0) (actual time=0.439..0.440 rows=1701 loops=1)
5	Index Cond: (ste_name21 = 'Western Australia'::text)	
6	Planning Time: 0.523 ms	
7	Execution Time: 8.772 ms	

How to connect to my databases?

Direct Connection

- Connection URI
- Client-Server Model
- Database Drivers

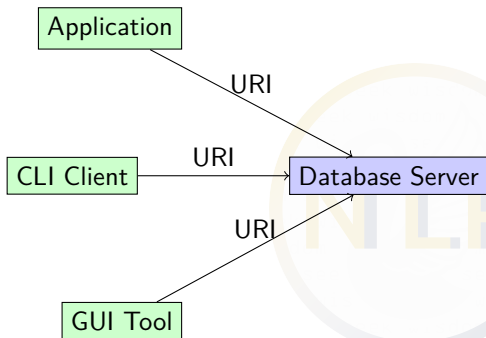
API Connection

- RESTful APIs
- GraphQL

Connection URI Example

```
1 postgresql://user:pass@localhost:5432/db
2 mongodb://user:pass@host:27017/db
3
```

Direct Connection Components



Security Note

Always use secure connections and never expose database ports directly!

Connection Methods Comparison

Direct Connection

- Lower latency
- Full DB features
- Security risks
- Direct exposure

API Connection

- Better security
- Controlled access
- Added latency
- Limited features

We will use both in our Full Stack Architecture.

What if my table is over 1 billion record?

Challenges

- Slow queries
- High memory usage
- Backup difficulties
- Index inefficiency

Solutions

- Table Partitioning
- Materialized Views
- Efficient indexing
- Query optimization



Table Partitioning

Example: Partition by Date Range

```
1 CREATE TABLE orders (  
2     id SERIAL,  
3     order_date DATE,  
4     amount DECIMAL  
5 ) PARTITION BY RANGE (order_date);  
6  
7 CREATE TABLE orders_2023  
8 PARTITION OF orders  
9 FOR VALUES FROM ('2023-01-01')  
10 TO ('2024-01-01');  
11
```

Materialized Views

Create Materialized View

```
1 CREATE MATERIALIZED VIEW monthly_sales AS
2 SELECT
3     date_trunc('month', order_date) as month,
4     SUM(amount) as total_sales
5 FROM orders
6 GROUP BY 1
7 WITH DATA;
8
```

Refresh View

```
1 REFRESH MATERIALIZED VIEW monthly_sales;
2
```


What I need to do to make databases up and running properly

Key Responsibilities

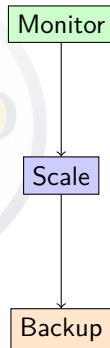
1 Monitoring

- CPU usage
- Memory utilization
- Disk I/O
- Query performance

2 Scaling

- Vertical (Scale Up)
- Horizontal (Scale Out)

3 Backup/Restore



Database Monitoring

Key Metrics

- Query Response Time
- Connection Count
- Cache Hit Ratio
- Buffer Usage
- I/O Operations

Monitoring Tools

- pgAdmin
- Grafana
- Prometheus
- CloudWatch

Critical Thresholds

Monitor and set alerts for:

- CPU > 80%
- Memory > 90%
- Disk Usage > 85%

Database Scaling

Vertical Scaling

- Add more CPU
- Increase RAM
- Faster Storage
- Single instance

Small DB

Medium DB

Large DB

Horizontal Scaling

- Read replicas
- Sharding
- Load balancing
- Multiple instances

DB

DB

DB

Backup and Restore

Backup Commands

```
1 # Full backup
2 pg_dump dbname > backup.sql
3
4 # Compressed backup
5 pg_dump dbname | gzip > backup.gz
```




When to use it?



Figure: Intuitive user scenario

When to use it?

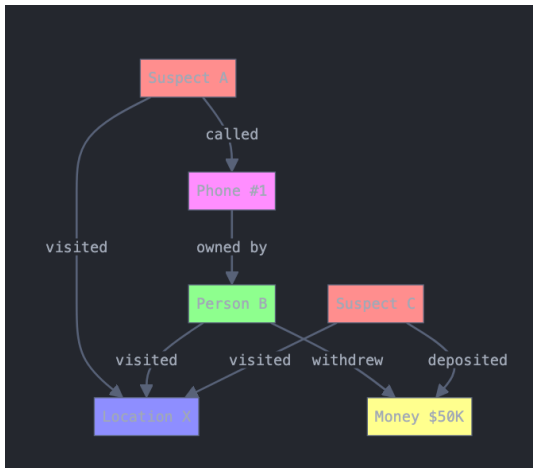


Figure: Example KG

Why now?

A tool helps human

- Exact matches only
- Limited pattern matching
- A lot of manual effort

Work together with human

- Vector embeddings
- Fuzzy search
- Semantic matching
- LLM integration



Example "KG_RAG" Workflow

Vector search → Subgraph extraction → LLM analysis

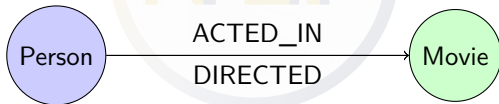
How to do GraphDB modeling?

Node (Entity)

- Properties
- Labels
- Unique identifiers

Edge (Relationship)

- Directional
- Types
- Properties



Graph vs Relational Modeling

Relational

- Tables
- Foreign Keys
- Join Tables
- Fixed Schema

Graph

- Nodes
- Edges
- Direct Relations
- Flexible Schema

Key Difference

Graph DB relationships are first-class citizens, not foreign keys

Cypher Query Language⁶

Create Node

```
1 CREATE (p:Person {  
2   name: 'John',  
3   age: 30  
4 })  
5
```

Find Patterns

```
1 MATCH (p:Person)-[:FRIENDS_WITH]->  
2   (f)-[:WATCHED]->(m:Movie)  
3 WHERE p.name = 'John'  
4 RETURN m.title  
5
```

Create Relationship

```
1 MATCH (p:Person), (m:Movie)  
2 WHERE p.name = 'John'  
3   AND m.title = 'Matrix'  
4 CREATE (p)-[:WATCHED]->(m)  
5
```

⁶<https://neo4j.com/docs/cypher-manual/current/introduction/>

How to connect to Neo4j via Python? ⁷

Connection Setup

```
1 from neo4j import GraphDatabase
2
3 uri = "neo4j://localhost:7687"
4 driver = GraphDatabase.driver(
5     uri,
6     auth=("neo4j", "password")
7 )
8
```

Query Execution

```
1 def get_friends(tx, name):
2     query = """
3     MATCH (p:Person {name: $name})
4         -[:FRIENDS_WITH]->(friend)
5     RETURN friend.name
6     """
7     result = tx.run(query, name=name)
8     return [record["friend.name"]
9             for record in result]
10
```

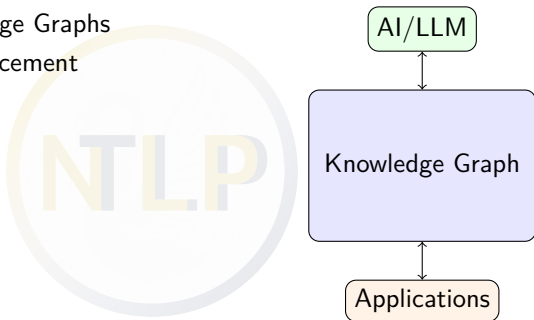
Usage Example

```
1 with driver.session() as session:
2     friends = session.read_transaction(
3         get_friends, "John"
4     )
5
```

Graph Database: Cutting Edge Technology

Emerging Applications

- AI-powered Knowledge Graphs
- LLM Context Enhancement
- Semantic Search
- RAG applications



Building AI Applications with Knowledge Graphs?

If you plan your project to this direction, we can have a discussion first.



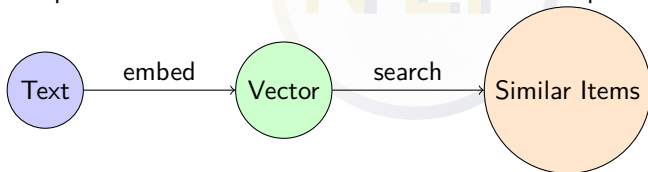
Vector Databases: Core Concepts

What is a Vector DB?

- Stores embeddings
- Similarity search
- High-dimensional data
- Optimized for ANN

Key Features

- Fast search
- Metadata filtering
- Scalability
- CRUD operations



Understanding ANN

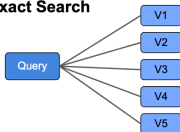
Exact vs Approximate

- Exact: Check every vector
- ANN: Smart shortcuts
- Trade accuracy for speed
- "Good enough" results

Why ANN?

- Billions of vectors
- High dimensions
- Real-time needs
- Resource efficient

Exact Search



ANN Search



Figure: ANN vs Exact Search

Qdrant Architecture

Storage Layer

- Collections
- Vectors
- Payloads
- Indexes

Search Layer

- HNSW algorithm
- Quantization
- Filter contexts
- Scoring

Key Feature

Combines fast vector search with rich filtering capabilities

Using Qdrant with Python

Connection

```
1 from qdrant_client import QdrantClient
2
3 client = QdrantClient(
4     host="localhost",
5     port=6333
6 )
7
```

Insert Vectors

```
1 client.upsert(
2     collection_name="products",
3     points=[
4         PointStruct(
5             id=1,
6             vector=[0.1, 0.2, 0.3],
7             payload={"name": "Product1"})
8 ]
9
```

Searching in Qdrant

Basic Search

```
1 results = client.search(  
2     collection_name="products",  
3     query_vector=[0.2, 0.3, 0.4],  
4     limit=5  
5 )  
6
```

Search with Filters

```
1 results = client.search(  
2     collection_name="products",  
3     query_vector=[0.2, 0.3, 0.4],  
4     query_filter=Filter(  
5         must=[  
6             FieldCondition(  
7                 key="category",  
8                 match=MatchValue(value="electronics")  
9             )]),  
10     limit=5)  
11
```


Q&A

Questions?

Feel free to ask anything