Explaining the 5 types of database and how to choose between them

Tibs (they / he)

18th July 2025, EuroPython 2025

Slides available at https://github.com/Aiven-Labs/the-5-types-of-database

https://aiven.io/tibs

I think there are 5 database shapes

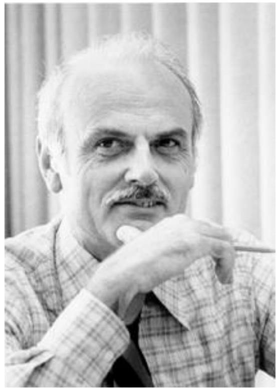
- Relational
- Columnar
- Document
- Key Value
- Graph

1. Relational

A table called **books**

id	title	author
1	This Book	Tibs
2	That Book	Tibs
3	John's Book	John Smith

Edgar F. Codd and relational theory



Picture of Edgar “Ted” Codd from wikipedia

1970 “A Relational Model of Data for Large Shared Data Banks”

- Just simple enough
- Just abstract enough
- Represent just about anything

- *Relation* ≡ table
- Took until the mid-1980s to “win”

Relational tables

books

id	title	author_id
1	This Book	273
2	That Book	273
3	John’s Book	301

authors

id	name
273	Tibs
301	John Smith
308	John Smith

Concept: SQL

“a domain-specific language used to manage data, especially in a relational database management system” – en.wikipedia.org/wiki/SQL

- Originates in the 1970s
- Originally called “SEQUEL” (Structured English Query Language)
- Standardised in the 1980s
- Latest version 2023

How to create those tables

```sql
CREATE TABLE authors (
  id INT NOT NULL PRIMARY KEY,
  name TEXT NOT NULL,
);

CREATE TABLE books (
  id INT NOT NULL PRIMARY KEY,
  title TEXT NOT NULL,
  author_id INT REFERENCES authors(id)
);
```

Finding my books…

```
SELECT books.title FROM books
  JOIN authors ON authors.id=books.author_id
  WHERE authors.name="Tibs";
```

gives the results

```
This book
That book
```

Concept: Transactions

- If data is split between multiple tables
  - then we'll need to change multiple tables "at the same time"
- Transactions let us do this

1. `START` a transaction
2. Do all the edits
3. Either `COMMIT` or `ROLLBACK`

Transaction example

```
START TRANSACTION;
UPDATE authors SET name = "Eric Smith" WHERE id = 301;
UPDATE books SET name = "Eric's book" WHERE id = 3;
COMMIT;
```

gives

books		
id	title	author_id
1	This Book	273
2	That Book	273
3	Eric's Book	301

authors	
id	name
273	Tibs
301	Eric Smith
308	John Smith

Characteristics of relational databases

- Tables and rows and columns
- Schema design up front
- Transactions (pretty much always)
  - OLTP (online transaction processing)

Relational example 1: PostgreSQL®

“PostgreSQL is a powerful, open source object-relational database system with over 35 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance.”

– www.postgresql.org/about

More on PostgreSQL

- Rich datatypes
- Stored functions
- Extensibility
- Excellent documentation
- Always a good place to start

Relational example 2: SQLite

"SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SQLite is the most used database engine in the world." – www.sqlite.org

"Small. Fast. Reliable. Choose any three." – www.sqlite.org

More on SQLite

- A library
- Built into the Python standard library
  - It's everywhere
- Single user
- Slightly odd in some ways (schema is optional)
- Use it instead of JSON/YAML/TOML for local storage!

When to use a relational database

- Almost always a good place to start
- If your data fits
    - It probably does…
- Whatever you need to do, some RDB can probably do it
    - and likely fast enough

… but please still stay for the rest of this talk!

2. Columnar

book sales

dt	id	title	price	quantity	customer_id
20250101 12:01	1	This Book	5.20	1	1005
20250101 12:14	1	This Book	4.50	2	923
20250101 19:27	1	This Book	5.20	1	85
20250101 20:14	3	Eric’s Book	4.00	1	1002
20250101 20:14	2	That Book	5.20	1	1002

Characteristics of columnar databases

- Essentially an optimisation of the relational idea
- Store data as columns, not rows
- We know the column datatype, so we can **compress** column data
  - Giving more efficient data storage
  - Good for data that doesn't change a lot

Compressed columns

book sales

dt	id	title	price	quantity	customer_id
20250101 12:01	1	This Book	5.20	1	1005
20250101 12:14			4.50	2	923
20250101 19:27			5.20	1	85
20250101 20:14	3	Eric’s Book	4.00		1002
	2	That Book	5.20		

What's fast, what's slow

Fast:

- Adding new rows
- Adding new columns
- Querying a few columns out of many

Slow:

- Changing or deleting rows

What data does that suit?

- Log data
- Sensor data
- Time series data in general
- Data stored for historical purposes

Concept: OLAP – Online Analytical Processing.

> At the highest level, you can just read these words backward:
>
> - **Processing**: some source data is processed …
> - **Analytical**: to produce some analytical reports and insights …
> - **Online**: in real-time.
>
> — clickhouse.com/docs/concepts/olap

In contrast to OLTP (Online Transaction Processing)

Columnar example: ClickHouse®

ClickHouse

"ClickHouse® is an open-source column-oriented database management system that allows generating analytical data reports in real-time." – github.com/ClickHouse/ClickHouse

"ClickHouse is the fastest and most resource efficient real-time data warehouse and open-source database." – clickhouse.com

More about ClickHouse

- Queries are still SQL 🙂
  - With some extras and useful utility functions
- Records don’t have to have a unique primary key
  - Although having one can help
- “Full fledged” transactions aren’t supported
  - Do we really need them for OLTP?

Create book sales table

```sql
CREATE TABLE book_sales (
  dt DateTime,
  id BIGINT,
  title String,
  price Decimal(8,2),
  quantity Int,
  customer_id BIGINT,
) ENGINE = MergeTree()
PARTITION BY toYYMM(dt)
ORDER BY (title, dt)
```

Find the 10 top sellers

```sql
SELECT
  id, title, sum(quantity)
AS
     total_quantity
FROM book_sales
GROUP BY id
ORDER BY total_quantity DESC
LIMIT 10
```

When to use a columnar database

- When you want to query on columns not rows
- When you have lots of columns
- When you have a lot of data
  - Which you don’t want to alter

3. Document

```
{
  "title": "This Book",
  "author": "Tibs",
  "isbn": null,
  "publisher": "self-published",
  "tags": ["nonFiction, humour"]
  "summary": "It's just very good",
  "chapterContent": [<chapter 1>, <chapter 2>, ...]
}
```

Document database concepts

- *Documents* are essentially JSON
- An *index* is a collection of documents
- When you search
  - you get back all data that matched
  - with a *relevance score* for how well it matched

Characteristics of document databases

- Relatively unstructured data
- But want indexing
- And rich querying
- OLTP - Store and query rather than update
- No transactions

Document example: OpenSearch®

> OpenSearch is an open-source, enterprise-grade search and observability suite that brings order to unstructured data at scale
>
> — opensearch.org

More about OpenSearch

- Technology origins in document processing, indexing and searching for large bodies of text
- Backed by Apache Lucene
- Queries are written in JSON
- Schema design up front is optional
  - but sometimes advised
- Data visualisation tools built in

Queries: Query DSL

```
query_body = {
  "query": {
    "bool": {
       must": {"match": {"author": "Tibs"}},
       must_not": {"match": {"title": "That Book"}},
    }
  }
}
resp = client.search(index=INDEX_NAME, body=query_body)
```

Queries: Lucene syntax

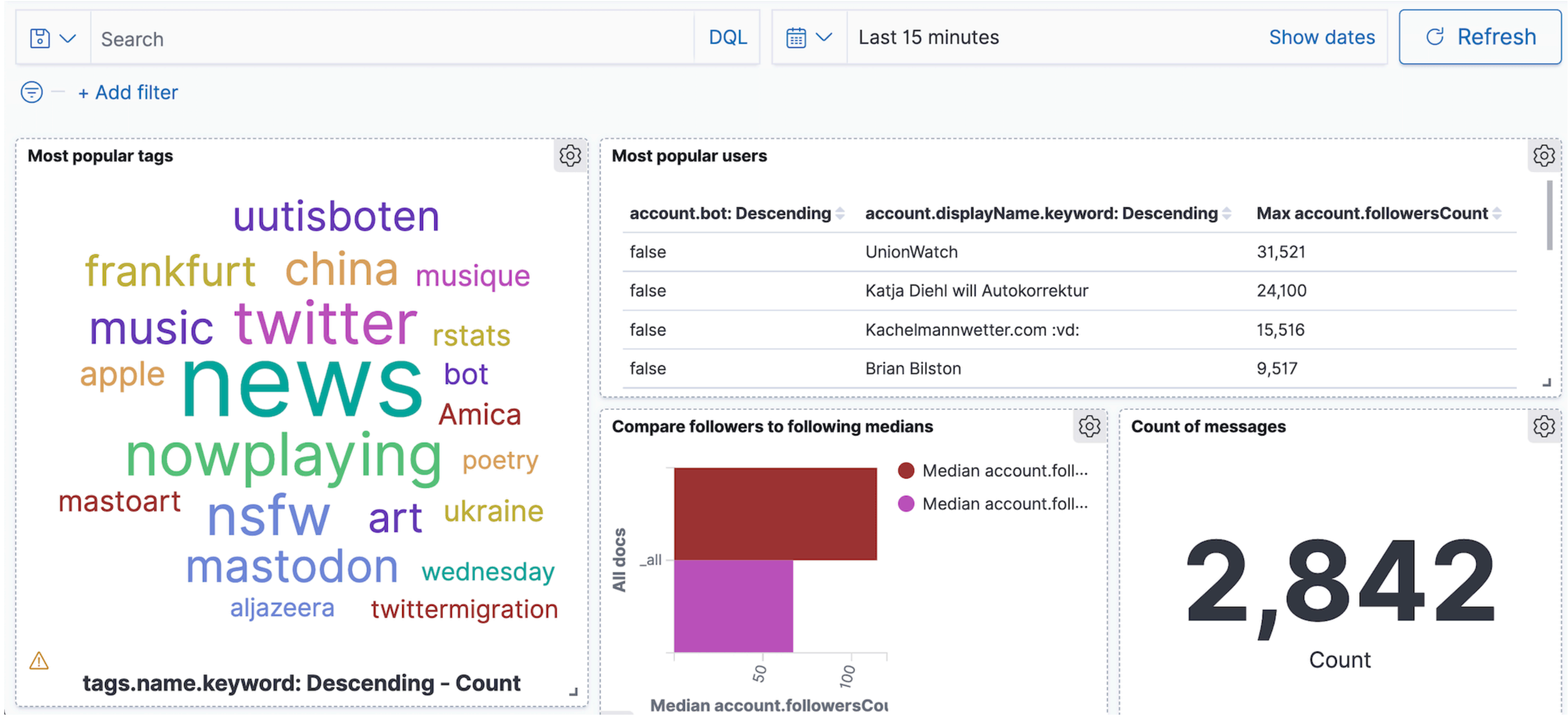
```
client = OpenSearch(SERVICE_URI, use_ssl=True)

client.search({
    index: 'recipes',
    q: 'author:Tibs AND title: (-That Book)'
})
```

Queries: SQL

```
curl -XPOST https://localhost:9200/_plugins/_sql \
    -u 'admin:<custom-admin-password>'           \
    --insecure                                   \
    -H 'Content-Type: application/json'          \
    -d '{"query": "SELECT * FROM my-index*'      \
       '  WHERE title <> "That Book"}'
```

untested code!

A dashboard about mastodon messages



When to use a document database

- Fast, scalable full text search
- Storage of indexable JSON documents
- OpenSearch: sophisticated analytics visualisation

4. Key Value

author:Tibs $\Longrightarrow$ ‘id’:273, ‘name’:‘Tibs’

book:This Book $\Longrightarrow$ ‘id’: 1, ‘title’: ‘This Book’, ‘author’: ‘Tibs’

book: That Book $\Longrightarrow$ ‘id’: 2, ‘title’: ‘That Book’, ‘author’: ‘Tibs’

A picture of a dictionary 🙂

Characteristics of key value databases

- Fast
- Simple
- Sophisticated value data types
- Think like a Python dictionary!

Key Value example: Valkey™

Valkey

"Valkey is an open source (BSD) high-performance key/value datastore that supports a variety of workloads such as caching, message queues, and can act as a primary database." – https://valkey.io

Datatypes

Key: a binary sequence

Value:

- Strings
- Lists
- Sets and Sorted sets
- Hashes
- Streams
- Geospatial indexes
- Bitmaps
- Bitfields
- Hyperloglog
- Bloom filter
- ...plus extensions

Queries

Its own protocol, with its own CLI

It’s actually rather lovely…

```
SET current:greeting "Hello" EX 60
```

```
LSET booklist 0 "This Book"
```

```
HGET "book:This Book" author
```

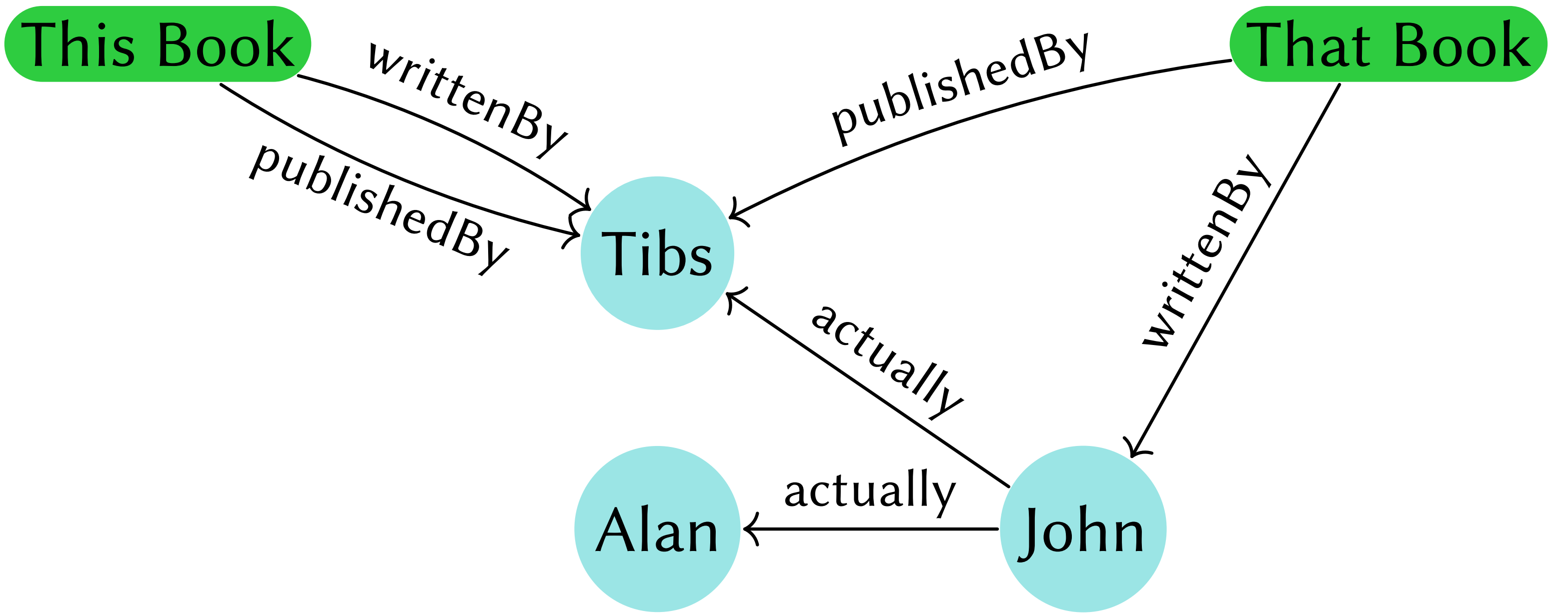
More about Valkey

- In-memory, but persistent to disk
- Use cases include:
  - Data storage and retrieval
  - Caching, leveraging the value expiry support
  - Pub/Sub messaging (`SUBSCRIBE`, `UNSUBSCRIBE`, `PUBLISH`)
  - Streams (append-only log) for message queues (`XADD`, `XREAD`)

When to use a key value database

- When your data fits the “key” -> “value” idea
- Caching (for instance, URL -> page results)
- Valkey:
  - when you want your data to expire
  - pub/sub messaging
  - message queues
  - for its datatypes

5. Graph



not an XY data graph 🙂

Characteristics of graph databases

Nodes, *relationships* and *properties*

- or *objects*, *references* and *attributes*
- or *nodes*, *edges* and *values*

Schemas might be implicit, gradual or designed up-front

Nodes

Nodes have

- a type
- properties
- are linked by relationships

Relationships

Relationships

- are between nodes
- are 1:1 or 1:many or many:1
- depending on design (I have opinions):
  - **may** have properties

Graph example: Neo4J®

“the world’s most-loved graph database” – neo4j.com

“The programmer works with a flexible network structure of nodes and relationships rather than static tables—yet enjoys all the benefits of enterprise-quality database.” – github.com/neo4j/neo4j

More about Neo4J nodes

Nodes

- have *labels*
- have key:value properties
- are indexed

More about Neo4J relationships

Relationships

- have a name
- must have a type, a start node and an end node
- must have a direction
- can have properties

Queries: Neo4J has Cypher

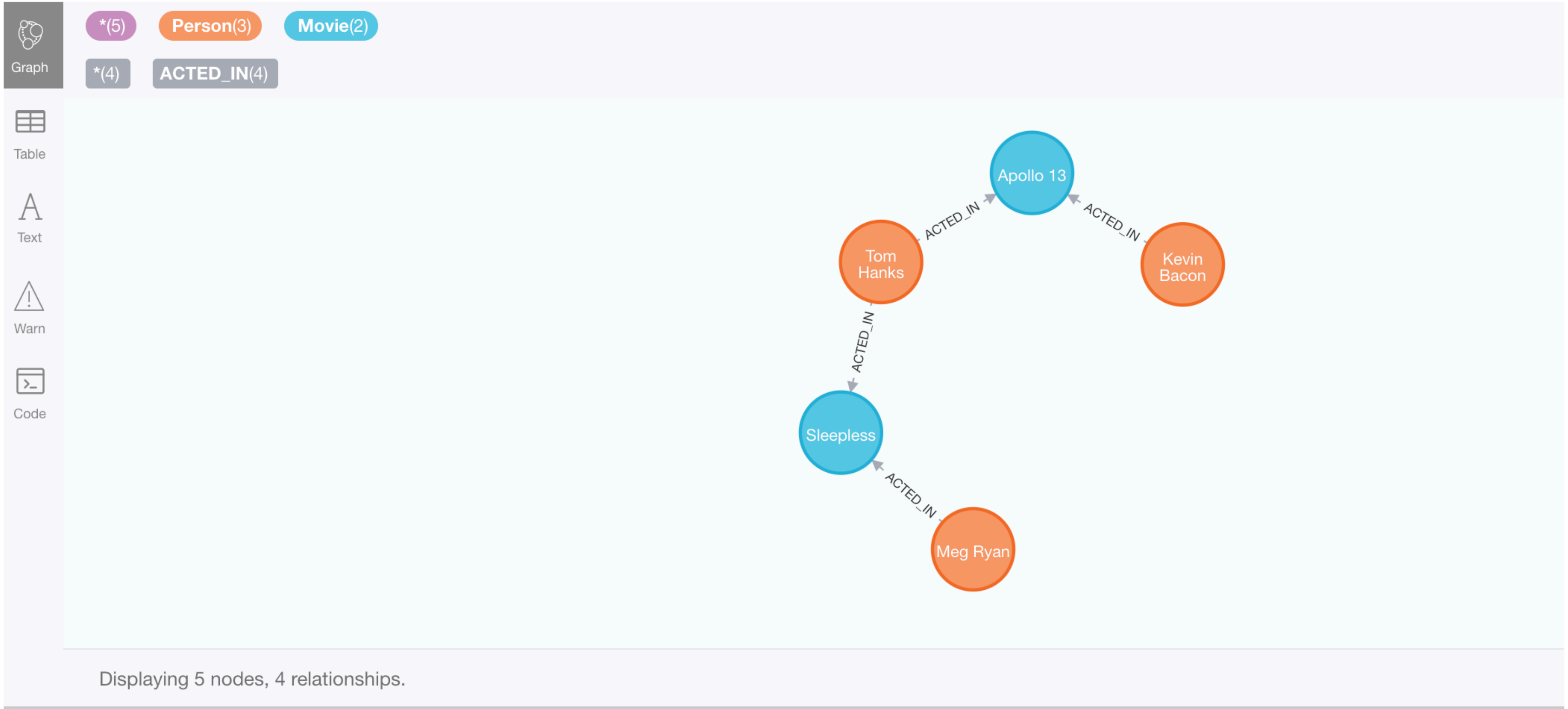
```
CREATE (p:Book
  {name:'This Book'})-[r:IS_WRITTEN_BY]->
    (p:Person {name:'Tibs'}
)
```

From Neo4J's own examples:

```
MATCH p=shortestPath(
  (bacon:Person {name:"Kevin Bacon"})-[*]-
    (meg:Person {name:"Meg Ryan"})
) RETURN p
```

Graph

```
neo4j$ MATCH p=shortestPath( (bacon:Person {name:"Kevin Bacon"})-[*]-(meg:Person {name:"Meg Ryan"}) ) RETURN p
```



When to use a graph database

- You have a knowledge graph shaped puzzle
- Neo4J: You want to build structures as you learn them
- Neo4J: You want to leverage existing techniques & solutions

Things just about all the shapes give you

- Transactions (not really OpenSearch)
- JSON support
- Vector embeddings (Valkey with a module; SQLite has an extension)
- Extensibility

What we've looked at

Five different kinds (shapes) of database

Relational	PostgreSQL®	Use for just about anything
	SQLite	Use in your programs, use locally
Columnar	ClickHouse®	Use for analytics, historical data
Document	OpenSearch®	Use for text corpuses, semi-structured data, indexing
Key Value	Valkey™	Use for caching, pub/sub, simple queues
Graph	Neo4J®	Use for graph/network data

Acknowledgements

Postgres, PostgreSQL and the Slonik Logo are trademarks or registered trademarks of the PostgreSQL Community Association of Canada, and used with their permission.

ClickHouse is a registered trademark of ClickHouse, Inc. https://clickhouse.com.

The OpenSearch Project is a project of The Linux Foundation.

Valkey and the Valkey logo are trademarks of LF Projects, LLC.

Neo4j®, Neo Technology®, Cypher®, Neo4j® Bloom™, Neo4j® AuraDS™ and Neo4j® AuraDB™ are registered trademarks of Neo4j, Inc. All other marks are owned by their respective companies.

Aiven

I work for Aiven *Your AI-ready Open Source Data Platform*, and we provide managed versions of PostgreSQL, ClickHouse, Valkey and OpenSearch (and free versions of PG and Valkey).

Fin

Get a free trial of Aiven services at
http://go.aiven.io/tibs-signup

Also, we're hiring! See https://aiven.io/careers

Slides created using typst and polylux, and available at
https://github.com/Aiven-Labs/the-5-types-of-database,
licensed