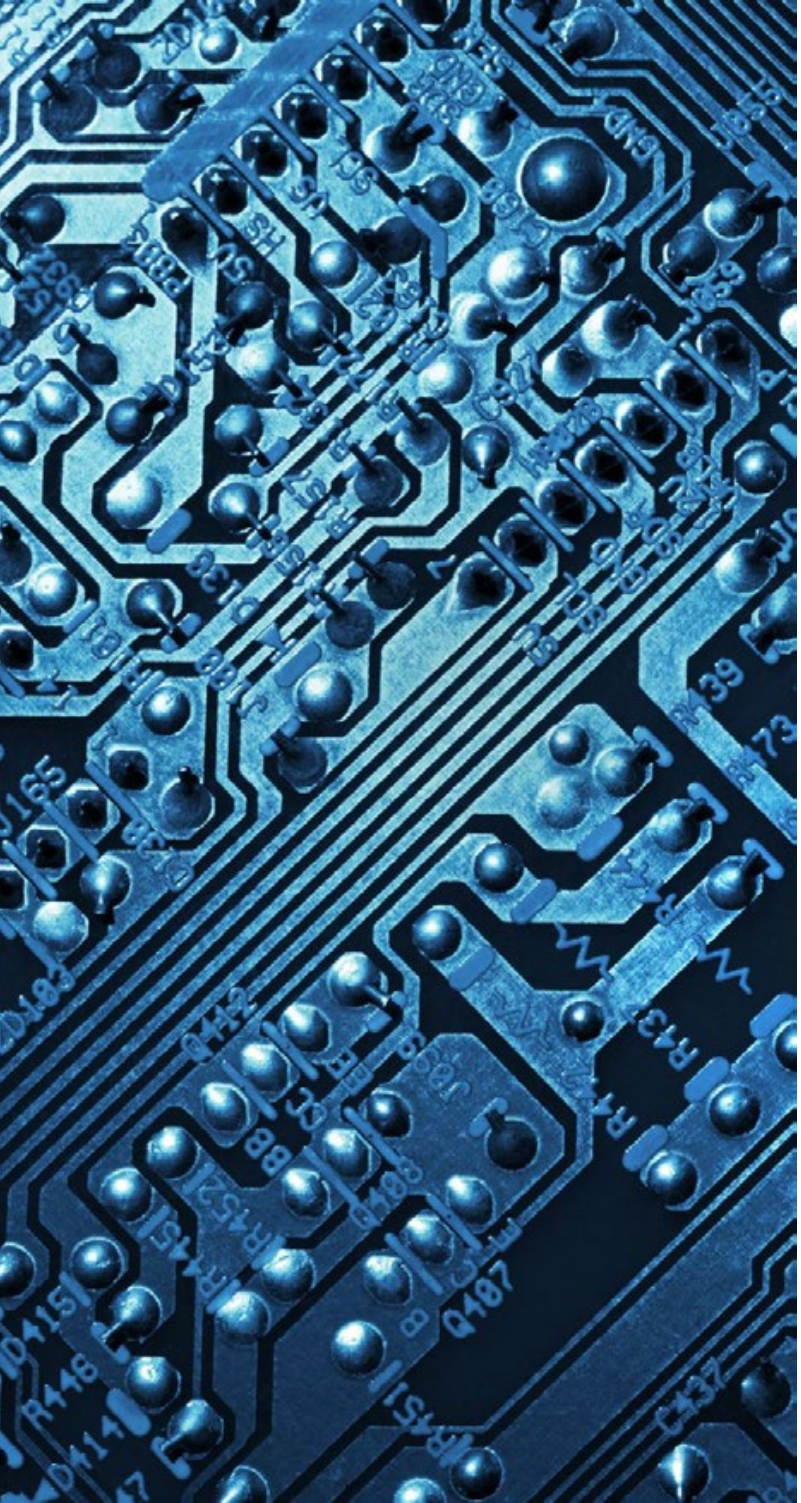
Advanced Databases

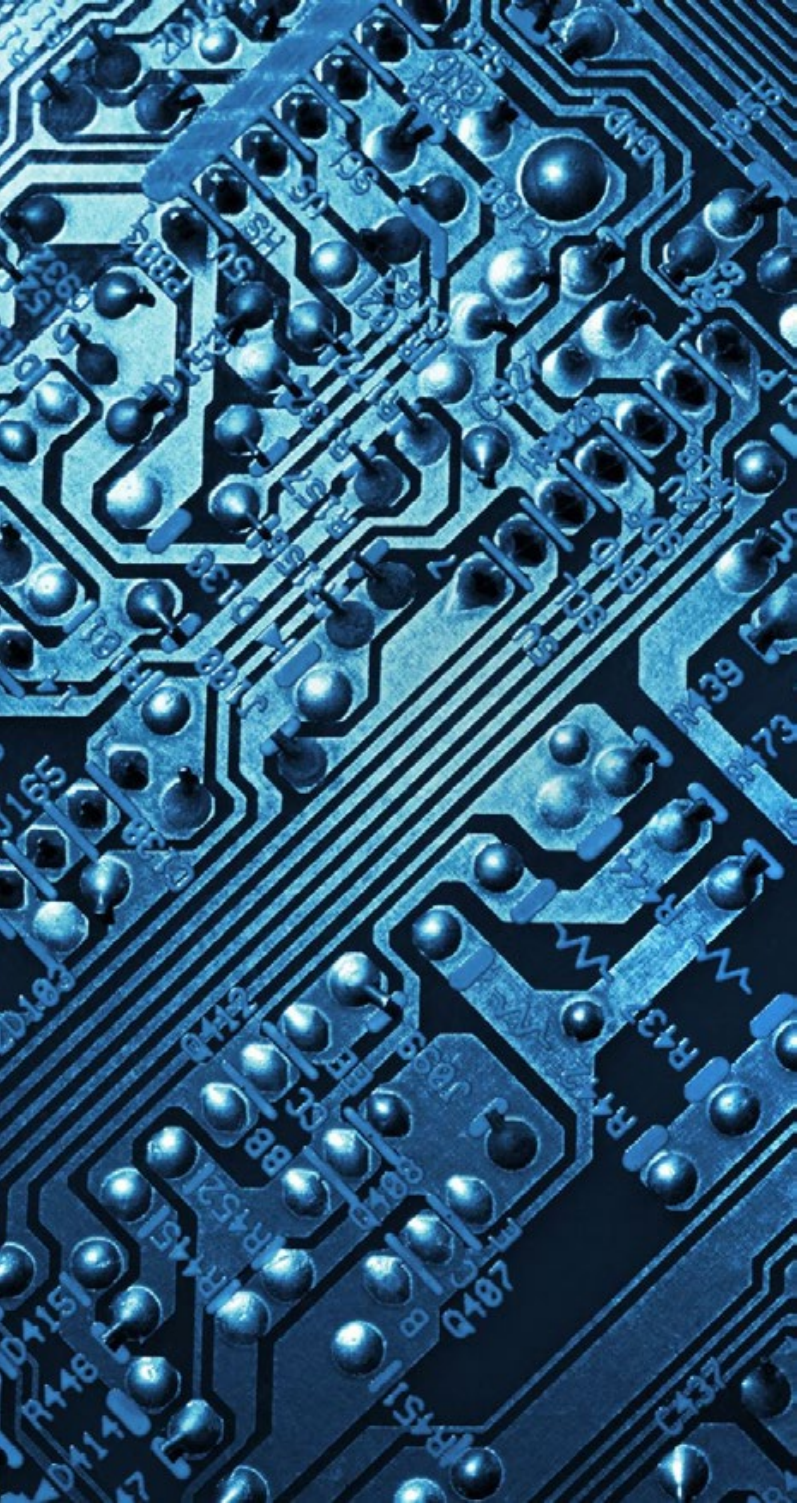
Week 1 – Overview (Part Review)

The reach of Data Science



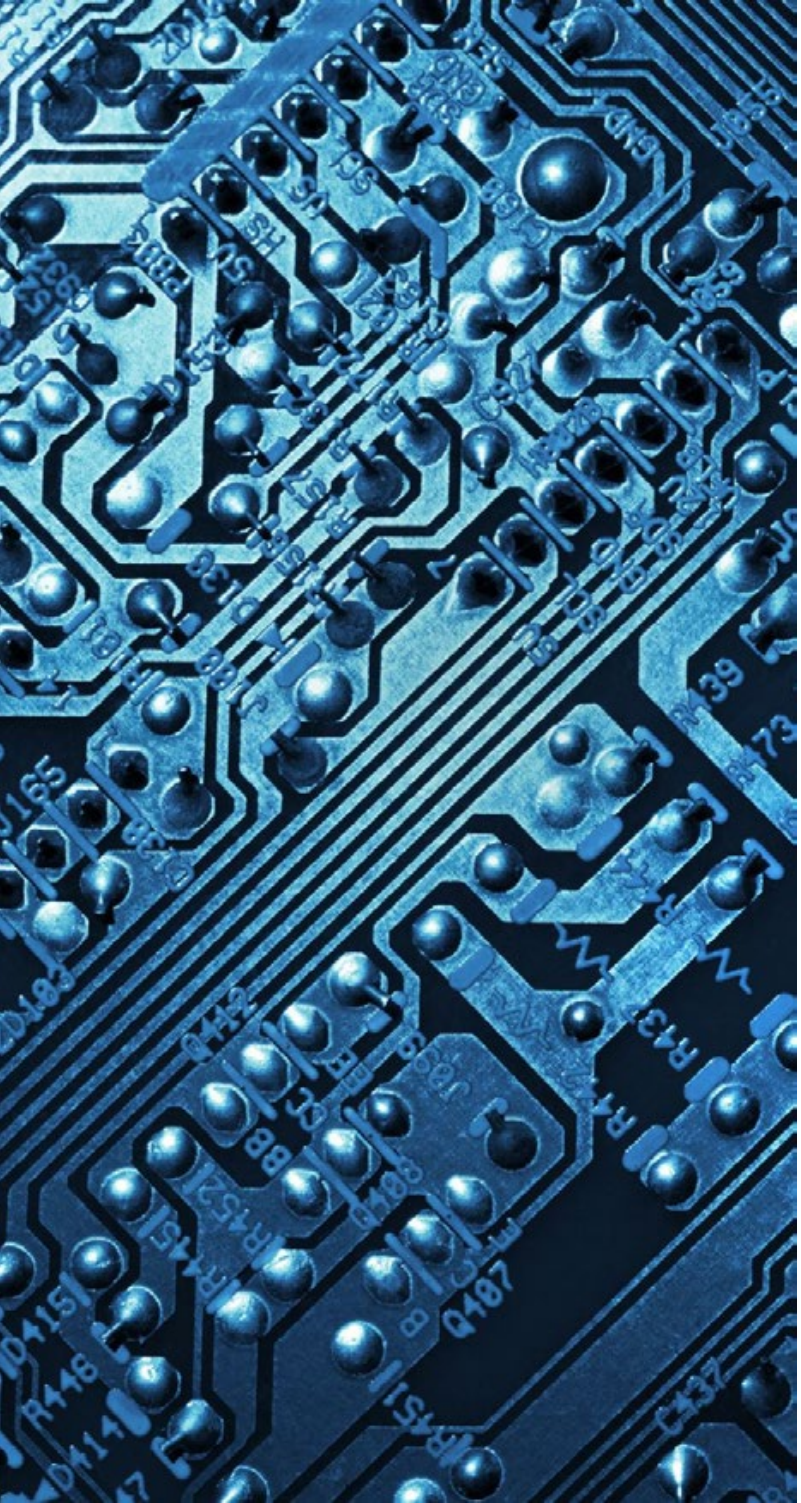
Problem?

- Organisations have lots and lots and lots of data
  - Not necessarily nicely structured and organised
- And are constantly collecting more…



Problem?

- Until recently computer system use expanded extremely fast due to improvements in hardware, particular CPU clock speed
  - This has stalled (at around 3 GHz)
- Moore's Law no longer holds (Observation)
  - "The number of transistors in integrated circuits doubles approximately every two years"
  - Gordon E. Moore (founder of Intel) 1965



Problem?

- But the world has now started to use CPU as units of computation (in clusters)
  - Causes challenges to getting data and software to work efficiently
    - Without disrupting day to day operations

Problem?

- Data (and the ability to use manage and use it) is the main focus of modern computer systems
- Volumes of data are growing rapidly
  - And so is the variation in the type of questions or analysis organisations want to conduct using this data

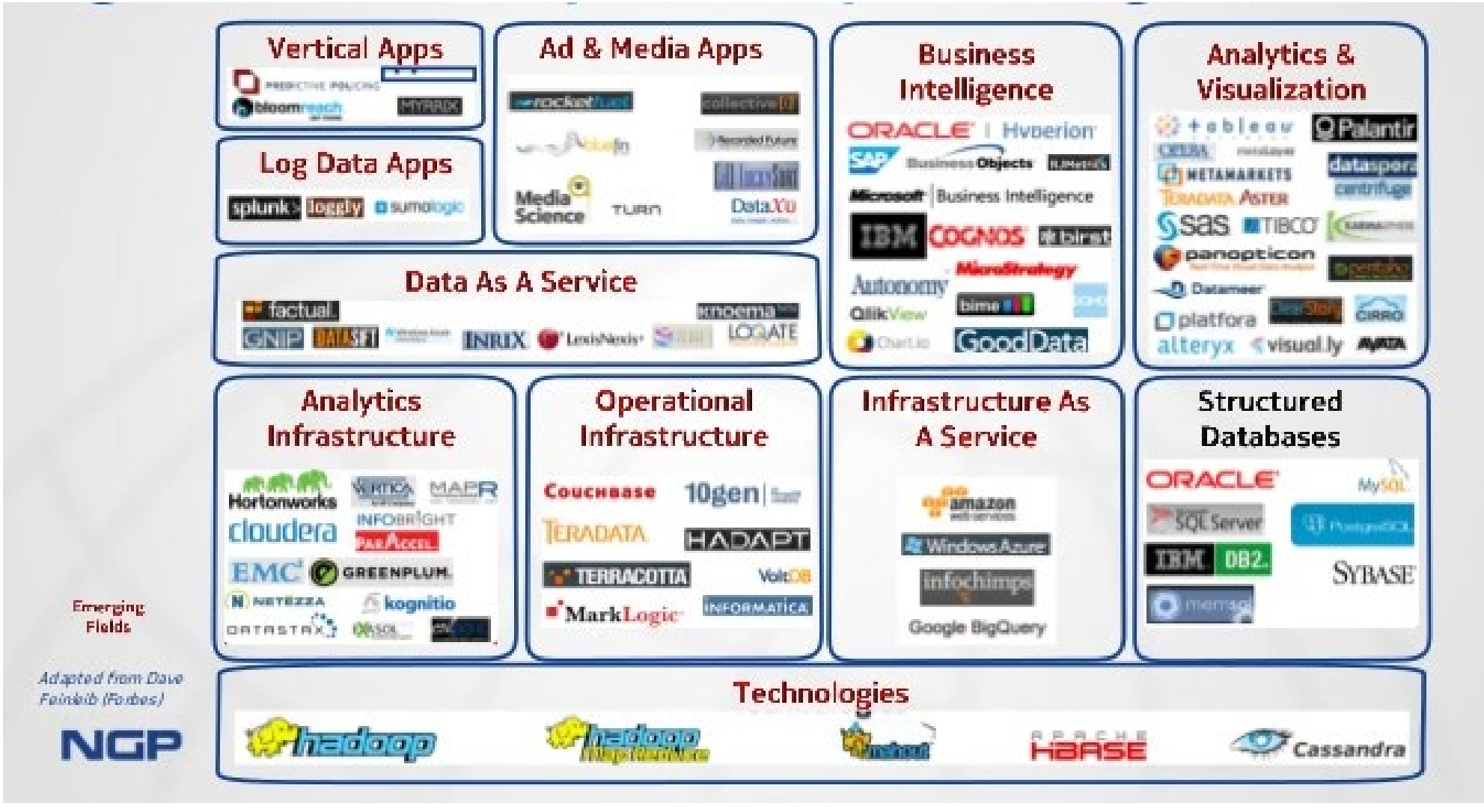
Problem?

- Need to use newer approaches to ensure that the analysis can be performed efficiently
- And that data can be stored and managed securely

Key Drivers

- Increasing volumes of data
  - business data, sensor data, social media data, …
- Increasing importance of data analytics
  - real-time, predictive, constant (no downtime)
- Parallelization paradigm shift
  - multi-core and network speeds increase while CPU clock speeds stall
- Computation resources become more available:
  - IaaS, PaaS, SaaS
  - (Infrastructure-as-a-Service, Platform-as-a-Service, and Software-as-a-Service)
- Increased popularity of free and open source
  - setting standards, utilizing external development resources, improving software quality, avoiding vendor locks …

Big Data Landscape 2021



Adapted from Dave Feinleib (Forbes)

NGP

Database Design

Database Design

- The process of developing a database for a given application aligning with a particular data model type;
- A subtask of the overall software engineering effort.

Database Design

The specification of programs and data is intertwined:

- Need to fit the data needed by the programs.
- Programs are often easy to develop once the structure of the data to be manipulated has been specified.

Database Design

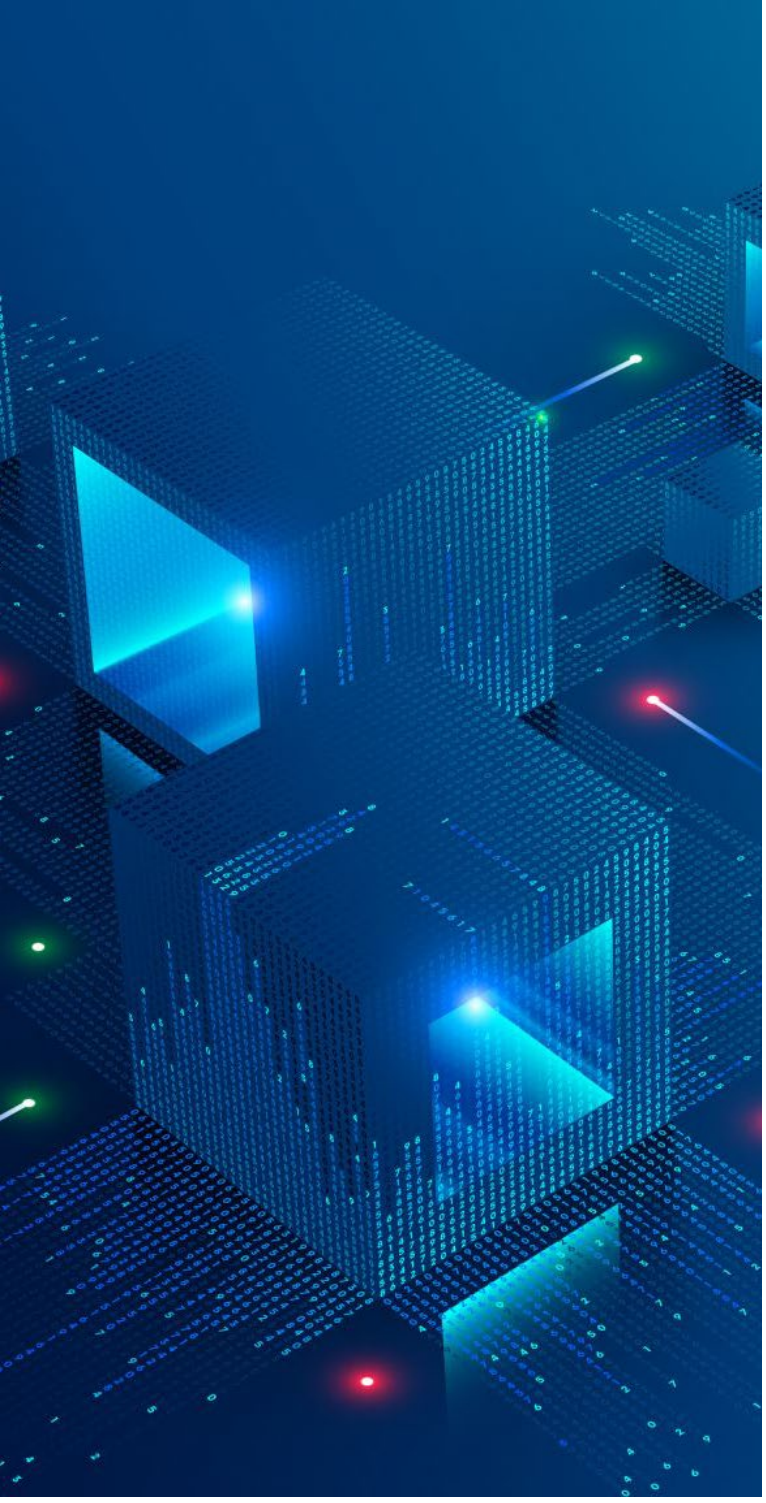
Data, however, is an independent resource:

- Typically, additional programs will be developed later based on the collected data.
- Also, ad-hoc queries will be posed against the DB.

Database Design

Start with Concepts

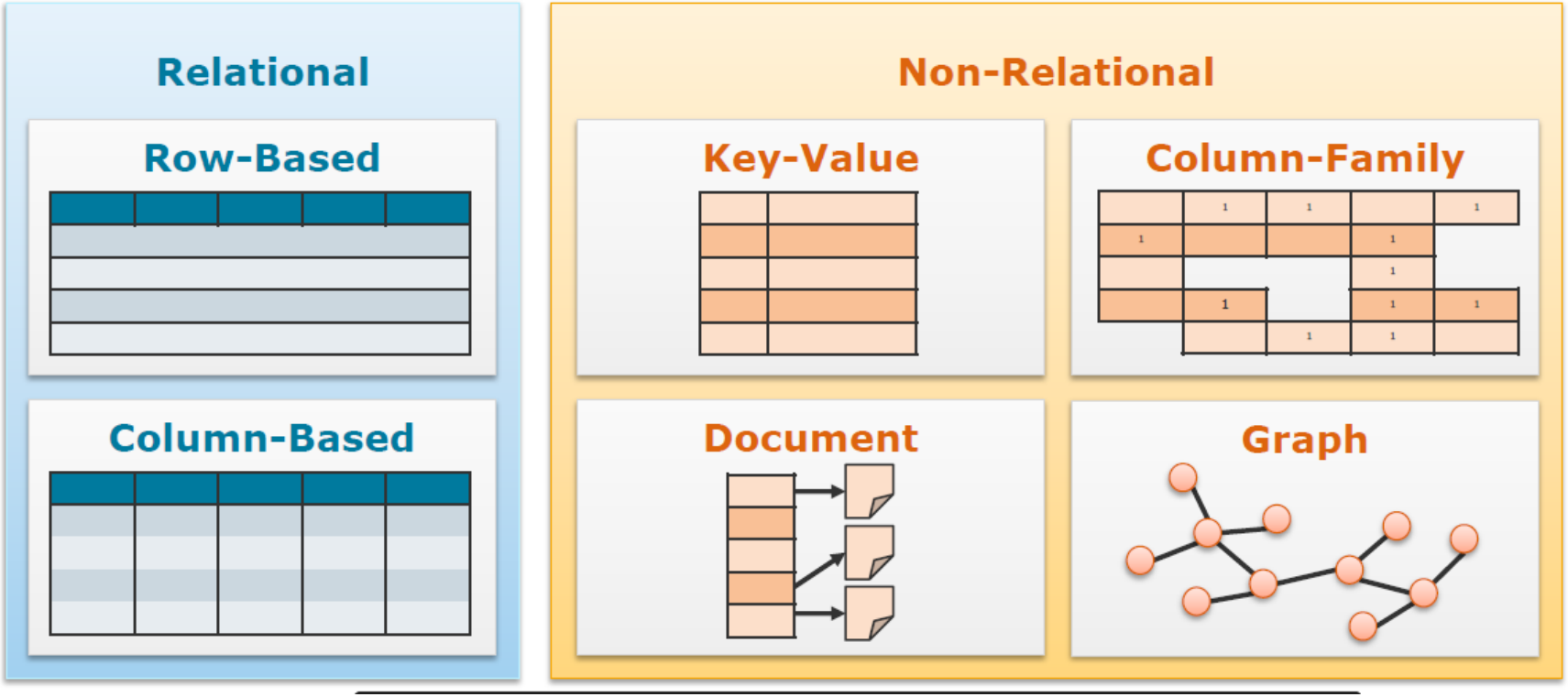
- Build/Design a formal model of the relevant aspects of the real world (“mini world”, “domain of discourse”
- Once the DB is up and running, questions will be posed against the DB.
- Knowledge of these questions beforehand is important input to the DB design process and helps to identify the relevant parts of the real world

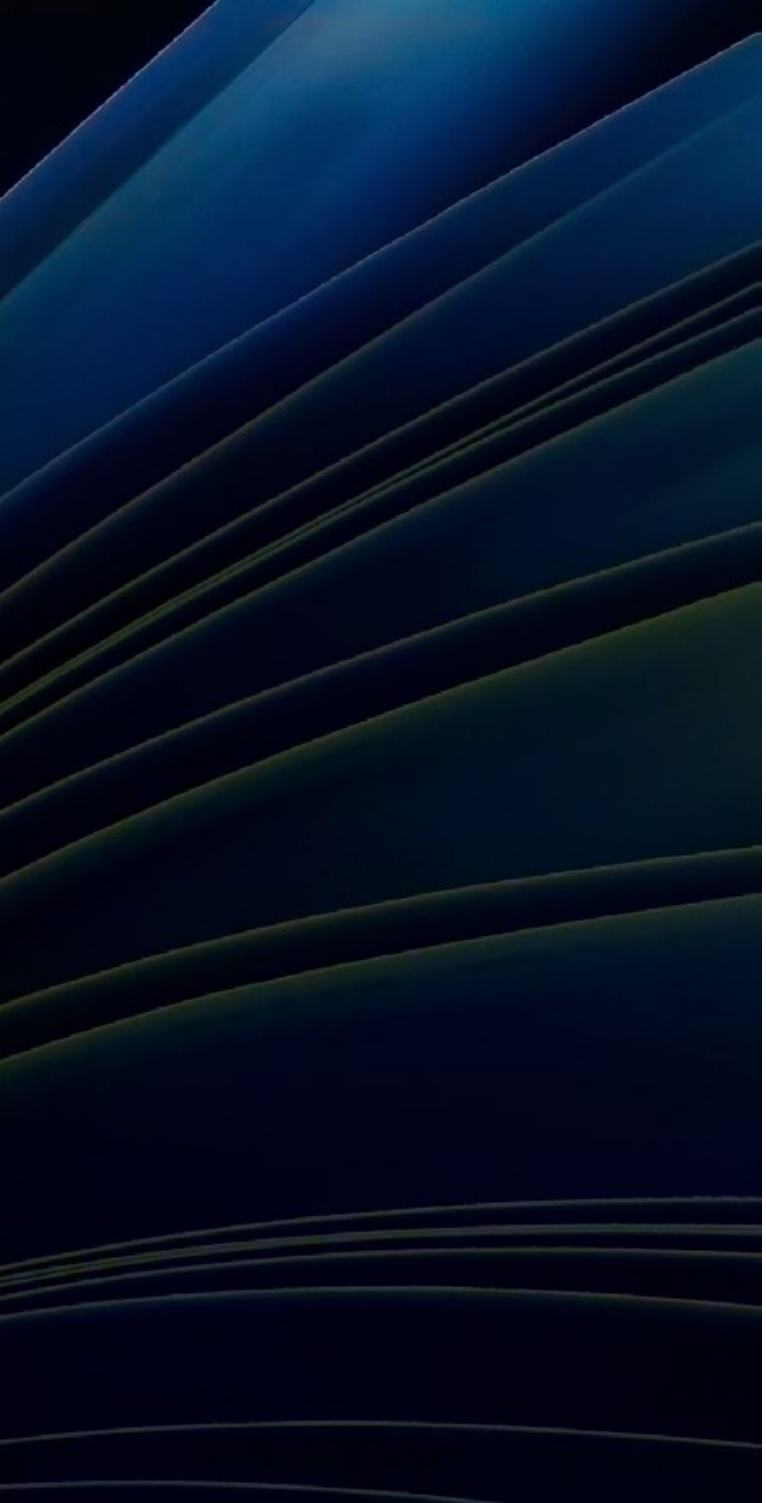


Data Models

- Conceptual layer
  - Data structures, objects, modules, … Application code (plus more APIs etc)
- Logical layer
  - Relational tables, JSON, XML, graphs, … Database management system (DBMS) or storage engine
- Representation layer
  - Bytes in memory, on disk, on network, … Database management system (DBMS) or storage engine
- Physical layer
  - Operating system and hardware drivers

Relational V Non-Relational (Logical)





Data Model – what does it consist of?

Structure	•Physical and conceptual data layout
Constraints	•Inherent limitations and rules
Operations	•Possible query and modification methods

3 Phases of Database Design

Conceptual Database Design.

- Produces the initial model of the mini world in a conceptual data model (e.g., in the ER model).

Logical Database Design.

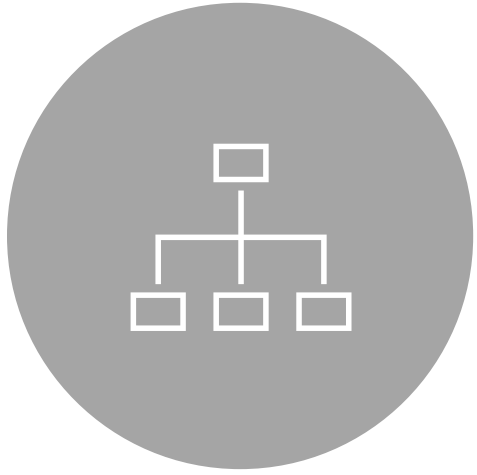
- Transforms the conceptual schema into the data model supported by the DBMS (e.g., the relational model).

Physical Database Design.

- Design indexes, table distribution, buffer size, etc., to maximize performance of the final system

Database Design

WHY MULTIPLE DESIGN PHASES?



ABSTRACTION AND DECOMPOSITION

Database Design

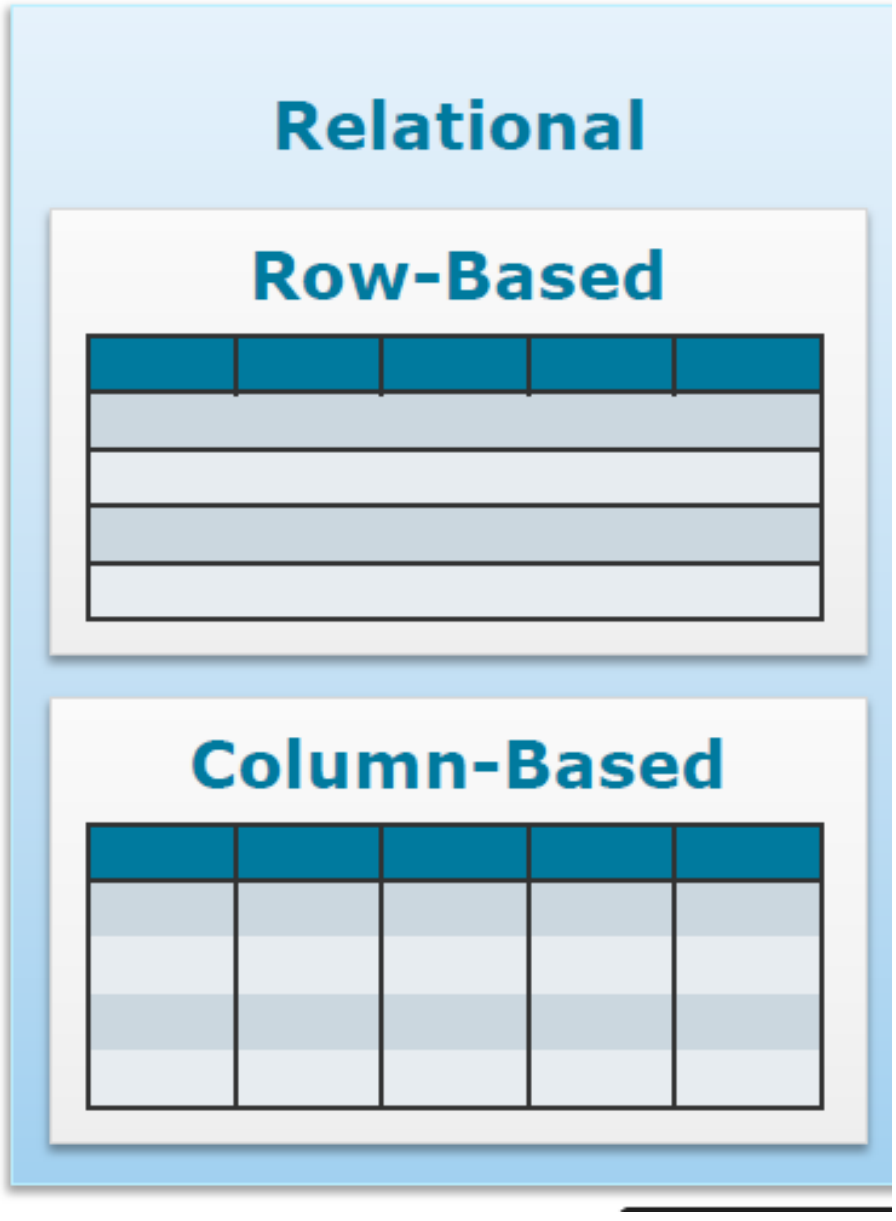
Partition the problem, attack one sub-problem after the other.

For example, during conceptual design there is no need to worry about performance aspects or limitations of the specific SQL dialect of the RDBMSs.

Why multiple design phases?

DBMS FEATURES DO NOT INFLUENCE THE CONCEPTUAL DESIGN AND ONLY PARTIALLY INFLUENCE THE LOGICAL DESIGN.

THUS, THE CONCEPTUAL DESIGN WORK *REMAINS VALID AND USEFUL*, IF A DIFFERENT DBMS IS USED LATER ON.


Relational
Row-Based
Column-Based

Natural Relational Data

Transactional Data

Statistical Data

Basic Social Media Data

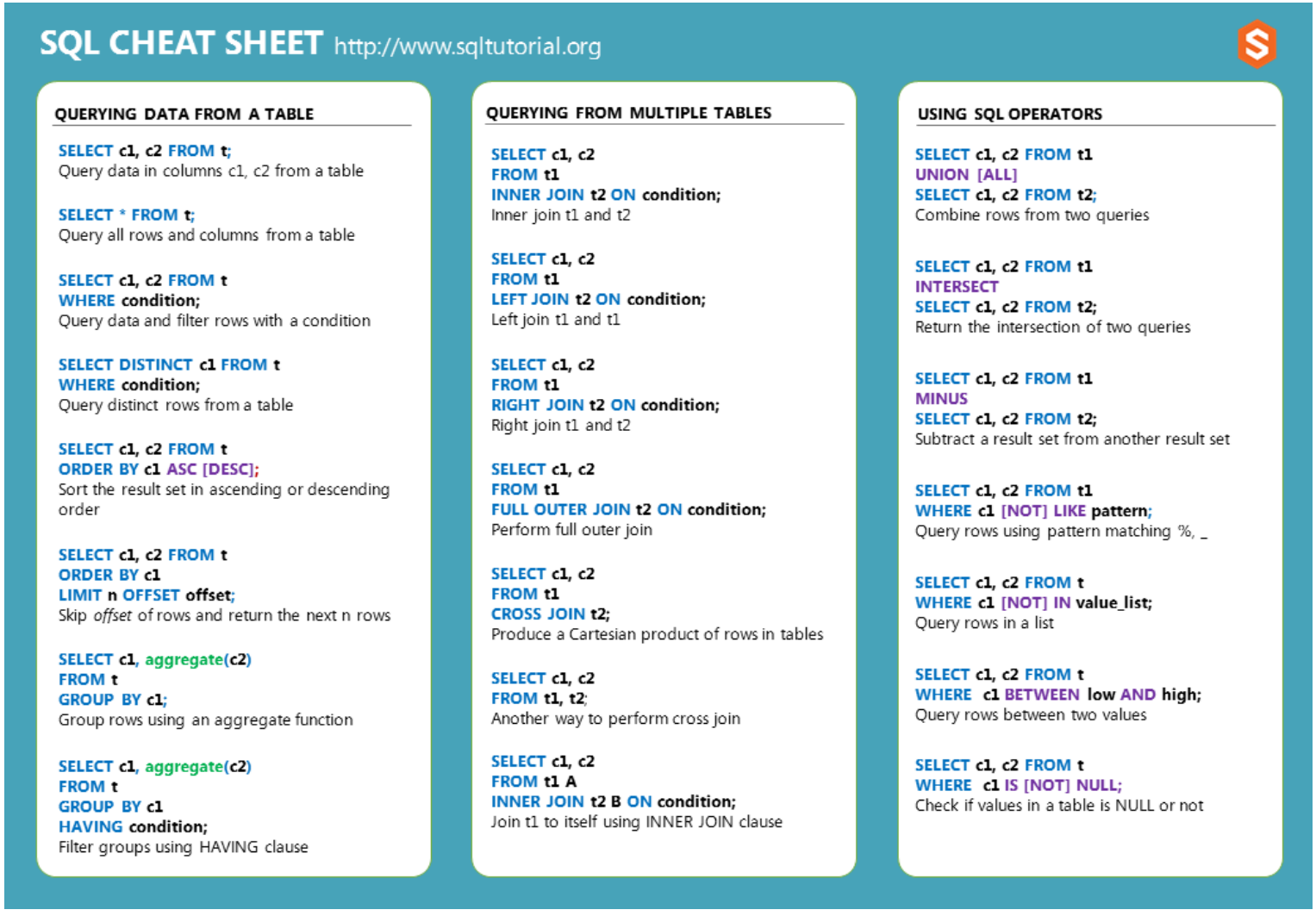
Relational Database – Rankings (https://db-engines.com/en/ranking)

Rank			DBMS	Database Model	Score		
Sep 2022	Aug 2022	Sep 2021			Sep 2022	Aug 2022	Sep 2021
1.	1.	1.	Oracle	Relational, Multi-model	1238.25	-22.54	-33.29
2.	2.	2.	MySQL	Relational, Multi-model	1212.47	+9.61	-0.06
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	926.30	-18.66	-44.55
4.	4.	4.	PostgreSQL	Relational, Multi-model	620.46	+2.46	+42.95
5.	5.	5.	MongoDB	Document, Multi-model	489.64	+11.97	-6.87
6.	6.	6.	Redis	Key-value, Multi-model	181.47	+5.08	+9.53
7.	↑ 8.	↑ 8.	Elasticsearch	Search engine, Multi-model	151.44	-3.64	-8.80
8.	↓ 7.	↓ 7.	IBM Db2	Relational, Multi-model	151.39	-5.83	-15.16
9.	9.	↑ 11.	Microsoft Access	Relational	140.03	-6.47	+23.09
10.	10.	↓ 9.	SQLite	Relational	138.82	-0.05	+10.17
11.	11.	↓ 10.	Cassandra	Wide column	119.11	+0.97	+0.12
12.	12.	12.	MariaDB	Relational, Multi-model	110.16	-3.74	+9.46
13.	13.	↑ 21.	Snowflake	Relational	103.50	+0.38	+51.43
14.	14.	↓ 13.	Splunk	Search engine	94.05	-3.39	+2.45
15.	15.	↑ 16.	Amazon DynamoDB	Multi-model	87.42	+0.16	+10.49
16.	16.	↓ 15.	Microsoft Azure SQL Database	Relational, Multi-model	84.42	-1.75	+6.16
17.	17.	↓ 14.	Hive	Relational	78.43	-0.22	-7.14
18.	18.	↓ 17.	Teradata	Relational, Multi-model	66.58	-2.49	-3.09

Relational Data Model

Structure	Instances	Constraints	Operations
• Schemata: named, non-empty, typed, and unordered sets of attributes • Example: Person(ID, Surname, Name, Address)	• Sets of records, i.e., functions that assign values to attributes • Example: 12345,'Lawless',`Deirdre´,'TU Dublin, Central Quad')	• Integrity constraints: data types, keys, foreign-keys, …	• Relational algebra (and relational calculus) • Usually implemented as Structured Query Language (SQL)

SQL Cheat Sheet - https://www.sqltutorial.org/sql-cheat-sheet/



SQL CHEAT SHEET http://www.sqltutorial.org

QUERYING DATA FROM A TABLE

```
SELECT c1, c2 FROM t;
```
Query data in columns c1, c2 from a table

```
SELECT * FROM t;
```
Query all rows and columns from a table

```
SELECT c1, c2 FROM t
WHERE condition;
```
Query data and filter rows with a condition

```
SELECT DISTINCT c1 FROM t
WHERE condition;
```
Query distinct rows from a table

```
SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
```
Sort the result set in ascending or descending order

```
SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
```
Skip *offset* of rows and return the next n rows

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
```
Group rows using an aggregate function

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
```
Filter groups using HAVING clause

QUERYING FROM MULTIPLE TABLES

```
SELECT c1, c2
FROM t1
INNER JOIN t2 ON condition;
```
Inner join t1 and t2

```
SELECT c1, c2
FROM t1
LEFT JOIN t2 ON condition;
```
Left join t1 and t1

```
SELECT c1, c2
FROM t1
RIGHT JOIN t2 ON condition;
```
Right join t1 and t2

```
SELECT c1, c2
FROM t1
FULL OUTER JOIN t2 ON condition;
```
Perform full outer join

```
SELECT c1, c2
FROM t1
CROSS JOIN t2;
```
Produce a Cartesian product of rows in tables

```
SELECT c1, c2
FROM t1, t2;
```
Another way to perform cross join

```
SELECT c1, c2
FROM t1 A
INNER JOIN t2 B ON condition;
```
Join t1 to itself using INNER JOIN clause

USING SQL OPERATORS

```
SELECT c1, c2 FROM t1
UNION [ALL]
SELECT c1, c2 FROM t2;
```
Combine rows from two queries

```
SELECT c1, c2 FROM t1
INTERSECT
SELECT c1, c2 FROM t2;
```
Return the intersection of two queries

```
SELECT c1, c2 FROM t1
MINUS
SELECT c1, c2 FROM t2;
```
Subtract a result set from another result set

```
SELECT c1, c2 FROM t1
WHERE c1 [NOT] LIKE pattern;
```
Query rows using pattern matching %, _

```
SELECT c1, c2 FROM t
WHERE c1 [NOT] IN value_list;
```
Query rows in a list

```
SELECT c1, c2 FROM t
WHERE c1 BETWEEN low AND high;
```
Query rows between two values

```
SELECT c1, c2 FROM t
WHERE c1 IS [NOT] NULL;
```
Check if values in a table is NULL or not

```
SELECT      <attribute list>
FROM        <relation list>
WHERE       <conditions>
GROUP BY    <grouping attributes>
HAVING      <grouping conditions>
ORDER BY    <attribute list>;
```

Additional Keywords

DISTINCT, AS, JOIN

AND, OR

MIN, MAX, AVG, SUM, COUNT

NOT, IN, LIKE, ANY, ALL, EXISTS

UNION, EXCEPT, INTERSECT

SQL

An example of a declarative query language

You specify the result of a query and not how it should be obtained:

- Easier to understand
- Transparently optimizable
- Implementation independent

Example

- Schema:
- Product(maker, model, type)
- PC(model, speed, ram, hd, rd)
- Laptop(model, speed, ram, hd, screen)

```
SELECT COUNT(hd)
FROM PC
GROUP BY hd
HAVING COUNT(model) > 2;
```

"How many hard disk sizes are built into more than two PCs?"

```
SELECT *
FROM PC PC1, PC PC2
WHERE PC1.speed = PC2.speed
AND PC1.ram = PC2.ram
AND PC1.model < PC2.model;
```

"Find all pairs of PCs with same speed and ram sizes."

```
   (SELECT DISTINCT maker
    FROM Product, Laptop
    WHERE Product.model = Laptop.model)
EXCEPT
   (SELECT DISTINCT maker
    FROM Product, PC
    WHERE Product.model = PC.model);
```

"Find all makers that produce Laptops but no PCs."

Strengths of Relational Model

- Strict schemas – good for to the point queries, error prevention, compression, …
- Universal data model serving linked and unconnected data, all data types, …
- Consistency checking (ACID) with support for different consistency levels
  - Atomicity, Consistency, Isolation, and Durability

Weaknesses of Relational Model

- Schemas need to be altered globally if certain records require additional attributes
- Object-Relational Impedance Mismatch:
  - Objects, structs, pointers vs. relations, records, attributes
  - Object-relational mapping (ORM) frameworks like ActiveRecord or Hibernate are used to overcome this
  - Complicates and slows data access; source for potential error

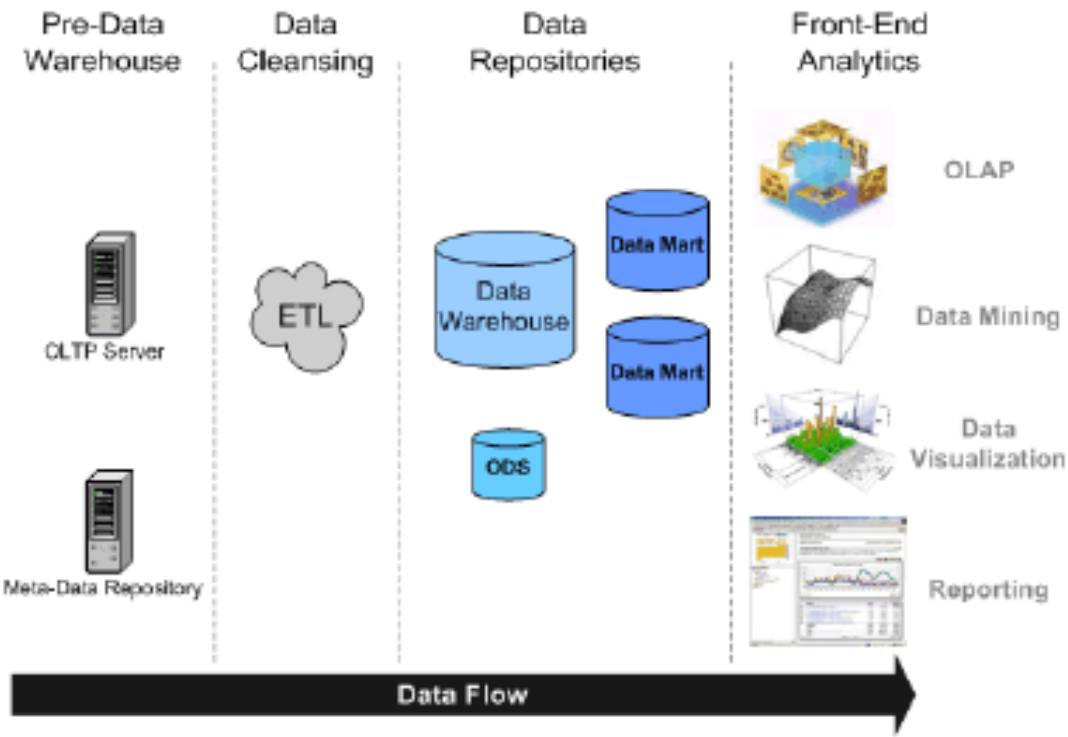
Data Warehouse

- Basically a very large database…
  - Not all very large databases are DW, but all data warehouses are pretty large databases
- Usually start at around a TB and go up to several PB
- It spans over several servers and needs an impressive amount of computing power

Data Warehouse

- Better described as a collective data repository
  - Containing snapshots of the operational data (history)
  - Obtained through data cleansing (Extract-Transform-Load (ETL) process)
  - Useful for analytics

Data Warehouse



OLTP: Online transaction processing; OLAP: Online Analytical Processing

- Experts say…
  - **Ralph Kimball:** "a copy of transaction data specifically structured for query and analysis"
  - **Bill Inmon:** "A data warehouse is a:
    - Subject oriented
    - Integrated
    - Non-volatile
    - Time variant

    collection of data in support of management's decisions."

Row v Column Based

Row-Based

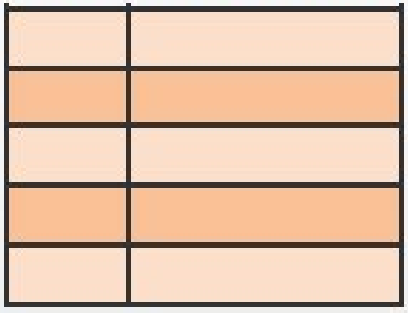
- Store rows continuously
- Examples :
  - Oracle
  - MySQL (open source)
  - Microsoft SQL Server
  - PostgreSQL (open source)
  - DB2
  - Microsoft Access

Column-Based

- Store columns continuously
- Examples :
  - Teradata
  - SAP HANA
  - SAP Sybase IQ
  - Vertica
  - MonetDB (open source)
  - C-Store (open source)

Non-Relational

Key-Value



Column-Family

Document

Graph

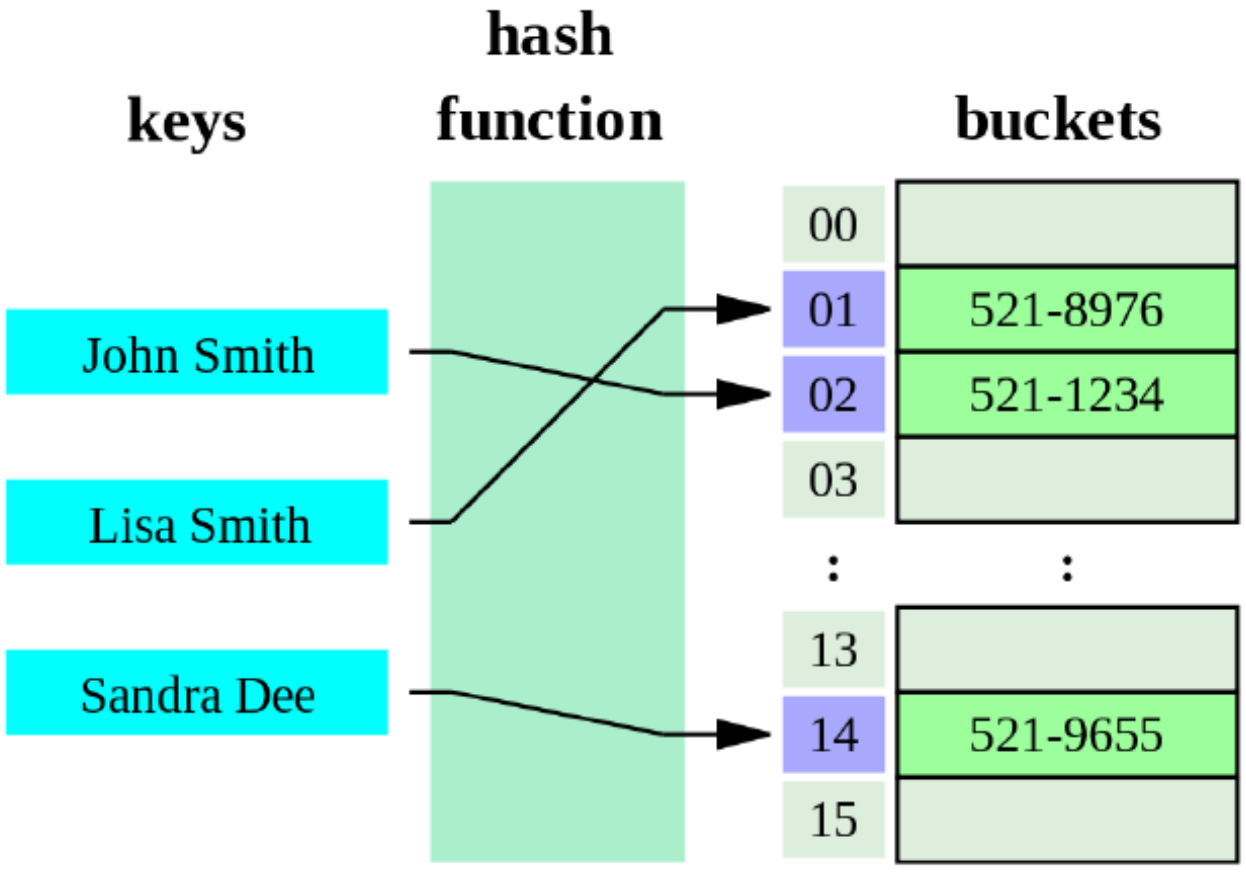
Key-Value Stores– Rankings (https://db-engines.com/en/ranking)

Rank			DBMS	Database Model	Score		
Sep 2022	Aug 2022	Sep 2021			Sep 2022	Aug 2022	Sep 2021
1.	1.	1.	Redis	Key-value, Multi-model	181.47	+5.08	+9.53
2.	2.	2.	Amazon DynamoDB	Multi-model	87.42	+0.16	+10.49
3.	3.	3.	Microsoft Azure Cosmos DB	Multi-model	40.67	-0.70	+2.15
4.	4.	4.	Memcached	Key-value	25.02	+0.38	-0.64
5.	5.	↑ 6.	Hazelcast	Key-value, Multi-model	10.06	-0.63	+0.25
6.	6.	↓ 5.	etcd	Key-value	9.50	-0.81	-0.82
7.	↑ 8.	7.	Ehcache	Key-value	6.87	-0.16	-0.14
8.	↓ 7.	↑ 10.	Ignite	Multi-model	6.71	-0.34	+1.87
9.	9.	9.	Aerospike	Multi-model	6.65	-0.20	+1.32
10.	10.	↓ 8.	Riak KV	Key-value	6.25	-0.04	+0.66
11.	11.	11.	ArangoDB	Multi-model	6.02	+0.11	+1.23
12.	↑ 13.	↑ 14.	Google Cloud Bigtable	Multi-model	5.03	+0.20	+1.05
13.	↓ 12.	↓ 12.	OrientDB	Multi-model	4.81	-0.39	+0.57
14.	14.	↓ 13.	Oracle NoSQL	Multi-model	4.41	+0.03	+0.18
15.	↑ 16.	↑ 16.	ScyllaDB	Multi-model	4.31	+0.50	+0.77
16.	↓ 15.	↓ 15.	RocksDB	Key-value	4.26	-0.03	+0.53
17.	↑ 18.	↑ 18.	LevelDB	Key-value	3.38	+0.05	+0.41
18.	↓ 17.	↑ 20.	Infinispan	Key-value	3.30	-0.05	+0.50

Key-Value Data Model

- Structure
  - Hash map: (mostly large, distributed) key-value data structure
- Constraints
  - Each value is associated with a unique key
- Operations
  - Store key-value pair
  - Retrieve value by key
  - Remove key-value mapping

Example



©Jorge Stolfi (https://commons.wikimedia.org/wiki/File:Hash_table_3_1_1_0_1_0_0_SP.svg)

Querying – Redis https://redis.io/

- In-memory key-value store with file persistence on disk
- Supports five data structures for values:
  - Strings: byte arrays that may represent actual strings or integers, binary serialized objects, …
  - Hashes: dictionaries that map secondary keys to strings
  - Lists: sequences of strings that support insert, append, pop, push, trim, and many further operations
  - Sets: duplicate free collections of strings that support set operations such as diff, union, intersect, …
  - Ordered sets: duplicate free, sorted collections of strings that use explicitly defined scores for sorting and support range operations

Querying – Redis API

- Strings:
  SET hello "hello world"
  GET hello
  → "hello world"
  SET users:goku {race: 'sayan', power: 9001}
  GET users:goku
  → {race: 'sayan', power: 9001}
- Hashes:
  HSET users:goku race 'sayan'
  HSET users:goku power 9001
  HGET users:goku power
  → 9001

"<group>:<entity>"
is a naming convention.

- Lists:
  LPUSH mylist a // [a]
  LPUSH mylist b // [b,a]
  RPUSH mylist c // [b,a,c]
  LRANGE mylist 0 1
  → b, a
  RPOP mylist
  → c
- Sets:
  SADD friends:lisa paul
  SADD friends:lisa duncan
  SADD friends:paul duncan
  SADD friends:paul gurney
  SINTER friends:lisa friends:paul
  → duncan

- Ordered sets:
  ZADD lisa 8 paul
  ZADD lisa 7 duncan
  ZADD lisa 2 faradin
  ZRANGEBYSCORE lisa 5 8
  → duncan
  → paul

Strengths of Key-Value Model

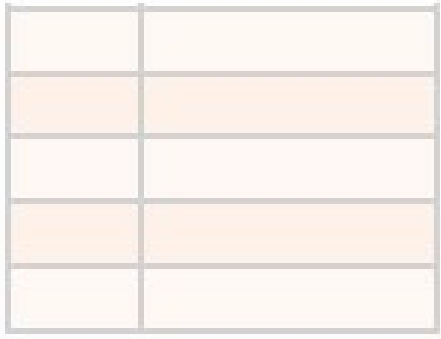
- Efficient storage: fast inserts of key-value pairs
- Efficient retrieval: fast point queries, i.e., value look-ups
- Key-value pairs are easy to distribute across multiple machines
- Key-value pairs can be replicated for fault-tolerance and load balancing

Weaknesses of Key-Value Model

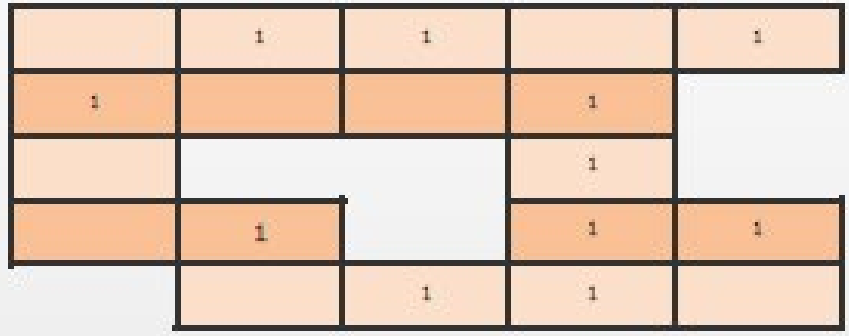
- No filtering, aggregation, or joining of values/entries
- Must be done by the application (or distributed computing framework)

Non-Relational

Key-Value



Column-Family



Document

Graph

Column-Oriented Family (https://db-engines.com/en/ranking)

Rank			DBMS	Database Model	Score		
Sep 2022	Aug 2022	Sep 2021			Sep 2022	Aug 2022	Sep 2021
1.	1.	1.	Cassandra	Wide column	119.11	+0.97	+0.12
2.	2.	2.	HBase	Wide column	41.97	-0.48	-3.08
3.	3.	3.	Microsoft Azure Cosmos DB	Multi-model	40.67	-0.70	+2.15
4.	4.	4.	Datastax Enterprise	Wide column, Multi-model	8.18	-0.28	-0.77
5.	5.	5.	Microsoft Azure Table Storage	Wide column	5.67	+0.14	+0.16
6.	6.	6.	Google Cloud Bigtable	Multi-model	5.03	+0.20	+1.05
7.	↑ 8.	↑ 8.	ScyllaDB	Multi-model	4.31	+0.50	+0.77
8.	↓ 7.	↓ 7.	Accumulo	Wide column	4.12	-0.04	+0.18
9.	9.	9.	HPE Ezmeral Data Fabric	Multi-model	1.09	+0.14	+0.18
10.	10.	↑ 11.	Amazon Keyspaces	Wide column	0.61	+0.05	+0.10
11.	11.	↓ 10.	Elassandra	Wide column, Multi-model	0.36	-0.04	-0.27
12.	12.	12.	Alibaba Cloud Table Store	Wide column	0.34	+0.00	-0.06
13.	13.	13.	SWC-DB	Wide column, Multi-model	0.06	+0.01	+0.06

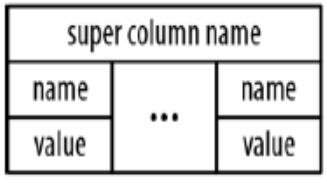
Column-Oriented Family Data Model

- Structure
  - Multi-dimensional hash map
    - (large, distributed) key-value data structure that uses a hierarchy of up to three keys for one typed value
  - Conceptually equivalent to sparse relational tables, i.e., each row supports arbitrary subsets of attributes
- Constraints
  - Each value is associated with a unique key
  - Hierarchy of keys is a tree
  - Integrity constraints: keys, foreign-keys, cluster-keys (for distribution), …
- Operations
  - At least: store key-value pair; retrieve value by key; remove key-value pair
  - Usually: relational algebra support without joins (with own SQL dialect)

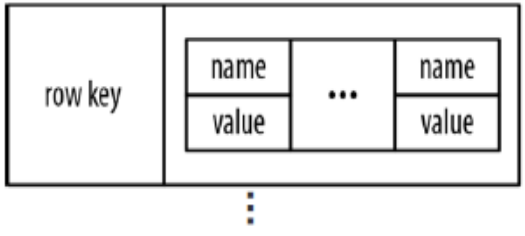
Column-Oriented Family Example

Column = key-value pair

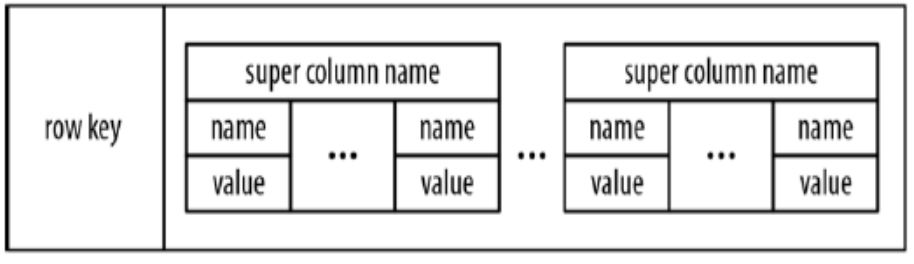
©https://neo4j.com/blog/aggregate-stores-tour/



Super Column = key-hashmap pair

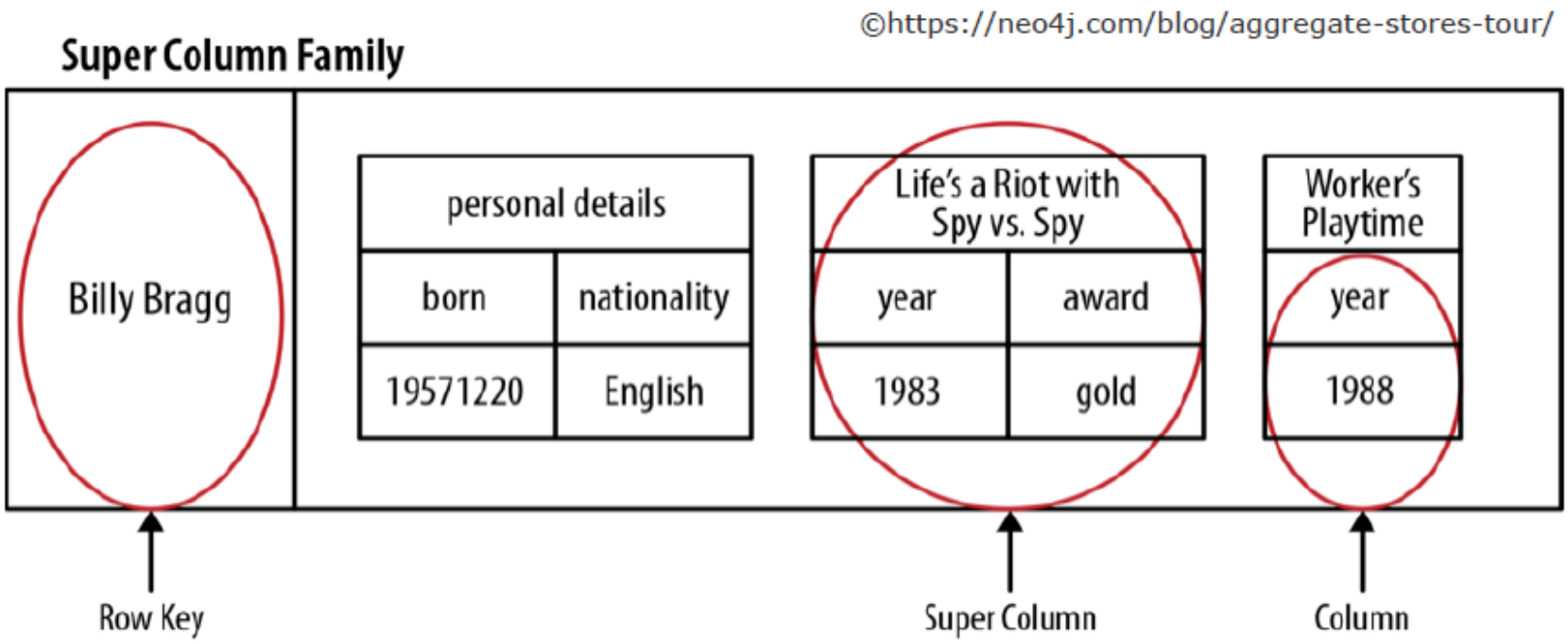


Column Family = Map<RowKey, SortedMap<ColumnKey, ColumnValue>>
≈ relational table



Super Column Family = Map<RowKey,
SortedMap<SuperColumnKey,
SortedMap<ColumnKey, ColumnValue>>>

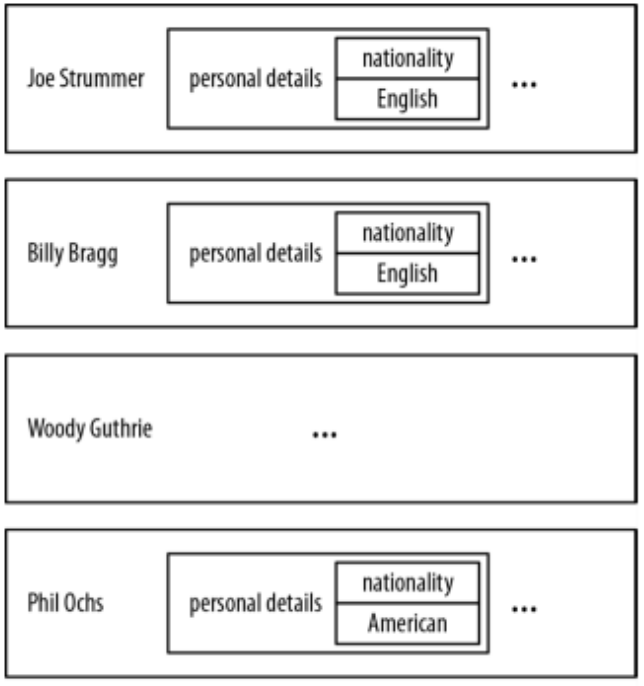
Column-Oriented Family Example



Hierarchy of keys enables:

- Flexible schemata (column names model attributes and row keys records)
- Value groupings (by super column names and row keys)

Column-Oriented Family Example



Analogy:

Relational Model	Cassandra Model
Database	Keyspace
Table	Column Family (CF)
Primary key	Row key
Column name	Column name/key
Column value	Column value

Hierarchy of keys enables:

- Flexible schemata (column names model attributes and row keys records)
- Value groupings (by super column names and row keys)

Column-Oriented Family Example

- Cassandra Query Language CQL …is an SQL dialect (same syntax)
  - https://cassandra.apache.org/_/index.html
  - Supports all DML and DDL functionalities
  - Does not support:
    - joins, group by, triggers, cursors, transactions, or (stored) procedures
    - OR and NOT logical operators (only AND)
  - Subqueries
- With the following key differences:
  - WHERE conditions should be applied only on columns with an index
  - Timestamps are comparable only with the equal operator (not <,>,<>)
  - UPDATE statements only work with a primary key (they do not work based on other columns or as mass update)
  - INSERT can override existing records, UPDATE can creates new records

CQL Example

Schema:

first key attribute(s) = **partition key** (determines which node stores the data)

- **Playlists(id, song_order, album, artist, song_id, title)**

further key attribute(s) = **cluster key** (keys within a partition/node)

Query:

```
SELECT *
FROM Playlists
WHERE id = 62c36092-82a1-3a00-93d1-46196ee77204
ORDER BY song_order DESC
LIMIT 4;
```

= key attribute

= key attribute

Result:

id	song_order	album	artist	song_id	title
62c36092...	4	No One Rides for Free	Fu Manchu	7db1a490...	Ojo Rojo
62c36092...	3	Roll Away	Back Door Slam	2b09185b...	Outside Woman Blues
62c36092...	2	We Must Obey	Fu Manchu	8a172618...	Moving in Stereo
62c36092...	1	Tres Hombres	ZZ Top	a3e63f8f...	La Grange

CQL Example

SQL:

```
CREATE DATABASE myDatabase;
```

```
SELECT *
FROM myTable
WHERE myField > 5000
AND myField < 100000;
```

CQL:

```
CREATE KEYSPACE myDatabase
WITH replication = {
  'class': 'SimpleStrategy',
  'replication_factor': 1};
```

```
SELECT *
FROM myTable
WHERE myField > 5000
AND myField < 100000
ALLOW FILTERING;
```

Otherwise:

Bad Request: Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute it despite the performance unpredictability, use ALLOW FILTERING.

Strengths of Column-Oriented model

- Efficient storage: fast inserts of data items
- Efficient retrieval: fast point queries, i.e., value look-ups
- Data structure is easy to distribute across multiple machines
- Data structure can be replicated for fault-tolerance and load balancing
- Flexible schemas

Weaknesses of Column-Oriented model

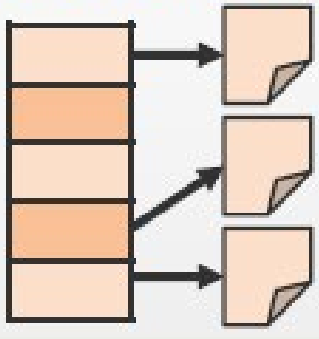
- No join and limited filtering support (filtering might also be super slow)
  - Must be done by the application (or distributed computing framework)
- Multi-key structure groups values to entities but general groupings and aggregations are not supported
- Non-point queries, i.e., those that read more than one mapping, are costly

Non-Relational

Key-Value

Column-Family

Document



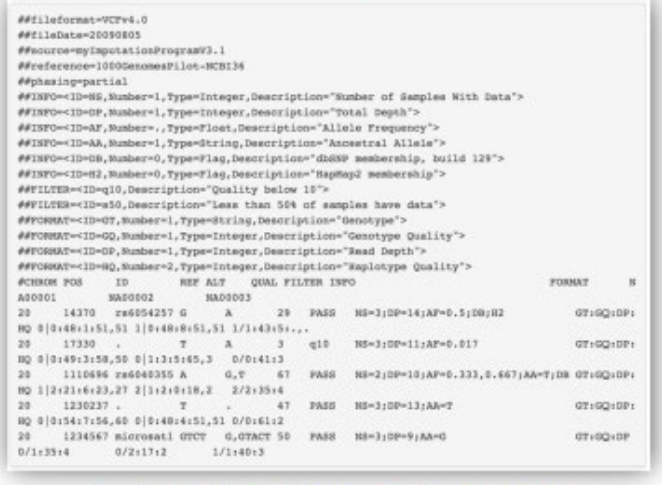
Graph

Document Data model
Natural Document Data

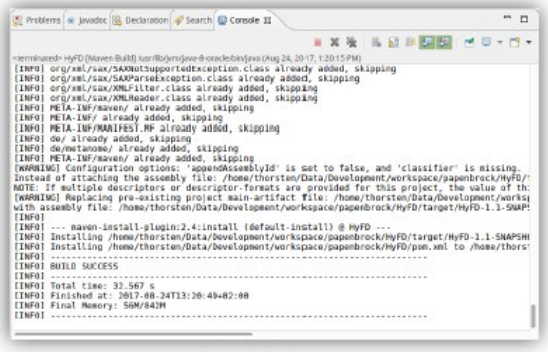
Digital Documents

Web Pages

Structured Data



Scientific Data Formats



Log Data

Document Oriented Stores (https://db-engines.com/en/ranking)

Rank			DBMS	Database Model	Score		
Sep 2022	Aug 2022	Sep 2021			Sep 2022	Aug 2022	Sep 2021
1.	1.	1.	MongoDB	Document, Multi-model	489.64	+11.97	-6.87
2.	2.	2.	Amazon DynamoDB	Multi-model	87.42	+0.16	+10.49
3.	3.		Databricks	Multi-model	55.62	+1.00	
4.	4.	↓ 3.	Microsoft Azure Cosmos DB	Multi-model	40.67	-0.70	+2.15
5.	5.	↓ 4.	Couchbase	Document, Multi-model	28.69	-0.10	+1.01
6.	6.	↓ 5.	Firebase Realtime Database	Document	19.08	+0.28	+1.05
7.	7.	↓ 6.	CouchDB	Document, Multi-model	16.56	-0.46	+1.09
8.	8.	↑ 9.	Google Cloud Firestore	Document	10.27	+0.24	+2.17
9.	↑ 10.	↓ 7.	MarkLogic	Multi-model	9.49	-0.14	-0.11
10.	↓ 9.	↓ 8.	Realm	Document	9.31	-0.51	-0.20
11.	11.	↓ 10.	Aerospike	Multi-model	6.65	-0.20	+1.32
12.	12.	↑ 13.	Google Cloud Datastore	Document	6.61	+0.17	+2.24
13.	↑ 14.	↓ 11.	ArangoDB	Multi-model	6.02	+0.11	+1.23
14.	↓ 13.	↓ 12.	Virtuoso	Multi-model	5.96	-0.20	+1.54
15.	15.	↓ 14.	OrientDB	Multi-model	4.81	-0.39	+0.57
16.	16.	↓ 15.	Oracle NoSQL	Multi-model	4.41	+0.03	+0.18
17.	17.	17.	RavenDB	Document, Multi-model	4.12	+0.23	+0.66
18.	18.	↓ 16.	IBM Cloudant	Document	3.77	+0.22	+0.28

Document Data Model

- Structure
  - Hash map: (large, distributed) key-value data structure
  - Documents: values are documents or collections of documents that (usually) contain hierarchical data
  - XML, JSON, RDF, HTML, …
- Constraints
  - Each value/document is associated with a unique key
- Operations
  - Store key-value pair
  - Retrieve value by key
  - Remove key-value mapping
- Note:
  - Document stores are often considered to be schemaless, but since the applications usually assume some kind of structure they are rather schema-on-read in contrast to schema-on-write.

Document Data Model

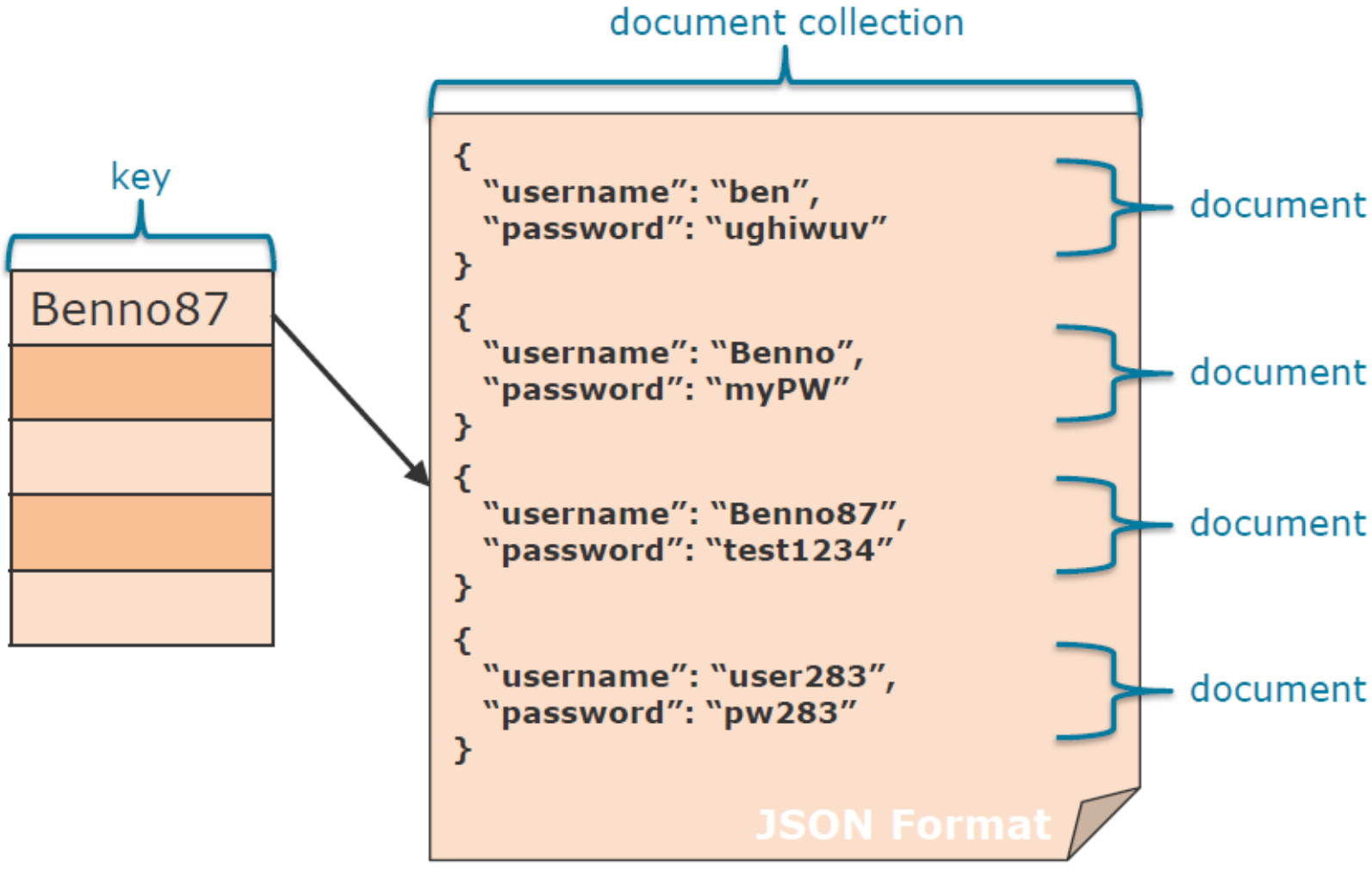
Relational data model

Highly-structured table organization with rigidly-defined data formats and record structure.

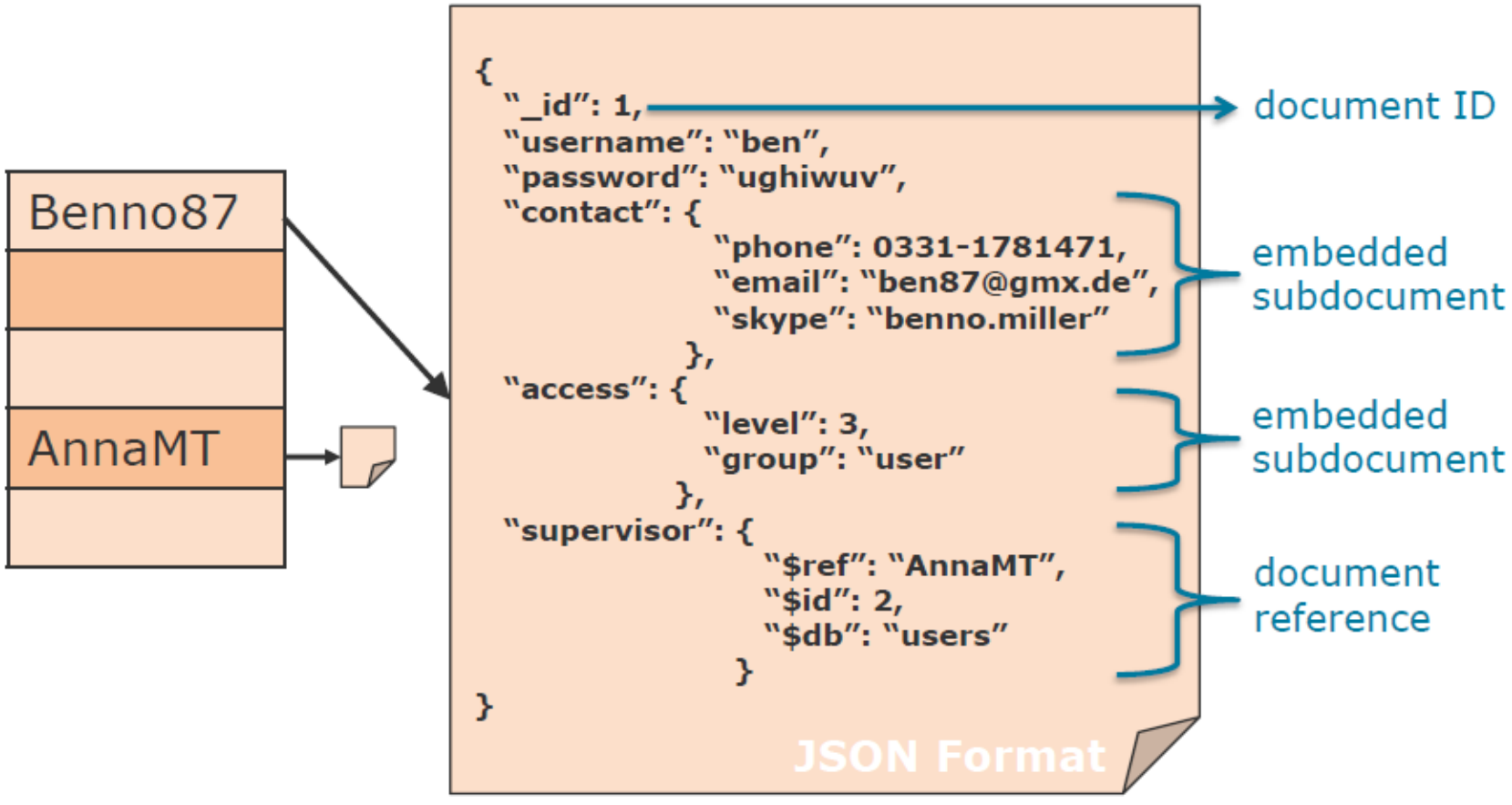
Document data model

Collection of complex documents with arbitrary, nested data formats and varying "record" format.

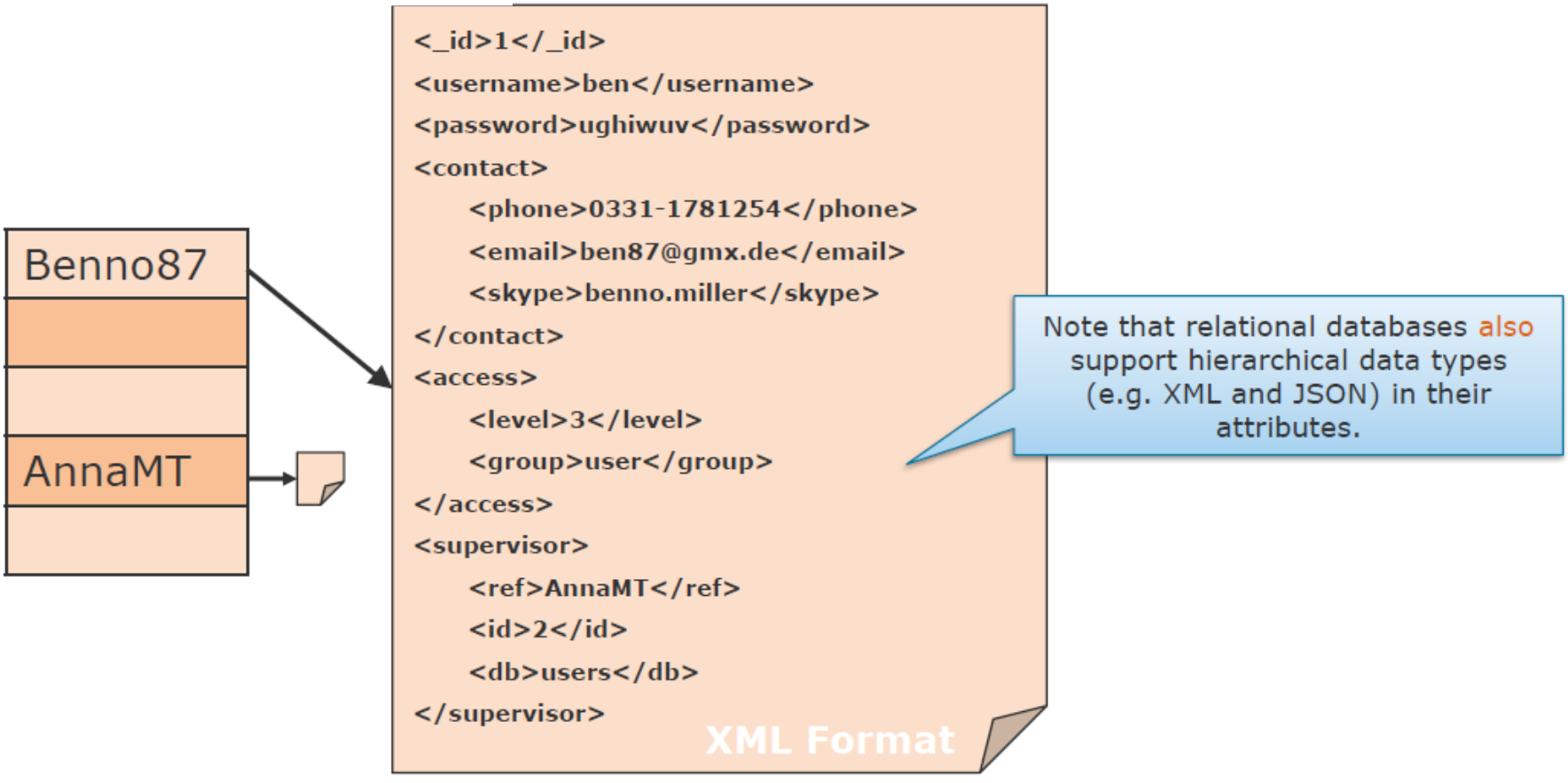
Document Data Model Example



Document Data Model Example



Document Data Model Example



Strengths of Document model

- Efficient storage: fast inserts of key-value pairs
- Efficient retrieval: fast point queries, i.e., document (collection) look-ups
- Document (collections) are easy to distribute across multiple machines
- Document (collections) can be replicated for fault-tolerance and load balancing
- Flexible document formats: documents can use different formats

Weaknesses of Document model

- No filtering, aggregation, or joining of values/entries
  - In general must be done by the application (or distributed computing framework)
  - Some do e.g. MongoDB
- Document parsing must be done by the application
- (Usually) developers need to explicitly/manually plan for distribution of data across instances (key-value and column-family stores do this automatically)
- Updates to documents are expensive if they alter their encoding/size

Document Data Model
Querying

MongoDB …

- is a free and open-source document-oriented DBMS
- uses JSON-like documents with schemata and integrity constraints (keys)

SQL Terms/Concepts	MongoDB Terms/Concepts
database	database
table	collection
row/record	document
column/attribute	field
index	index
table join	$lookup, embedded document
primary key (any column)	primary key (always the _id filed)
aggregation (group by)	aggregation pipeline

https://www.mongodb.com/

Document Data Model Example

Create/Drop

Document:

```
{
 _id: 1,
 user_id: "abc123",
 age: 55,
 status: 'A'
}
```

SQL

```
CREATE TABLE people (
    id MEDIUMINT NOT NULL
        AUTO_INCREMENT,
    user_id Varchar(30),
    age Number,
    status char(1),
    PRIMARY KEY (id)
)
```

```
DROP TABLE people
```

MongoDB

```
db.people.insertOne( {
    user_id: "abc123",
    age: 55,
    status: "A"
} )
```

```
db.people.drop()
```

First insert automatically creates the document collection "people".

https://docs.mongodb.com/manual/

Document Data Model Example

Alter

SQL

```
ALTER TABLE people
ADD join_date DATETIME
```

MongoDB

```
db.people.updateMany(
    { },
    { $set: { join_date: new Date() } }
)
```

"$" introduce operators (= functions)

Document:

```
{
  _id: 1,
  user_id: "abc123",
  age: 55,
  status: 'A'
}
```

SQL

```
ALTER TABLE people
DROP COLUMN join_date
```

MongoDB

```
db.people.updateMany(
    { },
    { $unset: { "join_date": "" } }
)
```

Collections do not describe or enforce the structure of their documents, i.e., no structural alteration at collection level.

But: $set and $unset can be used for bulk updates.

https://docs.mongodb.com/manual/

Document Data Model Example

Insert, Update and Delete

Document:

```
{
  _id: 1,
  user_id: "abc123",
  age: 55,
  status: 'A'
}
```

SQL

```
INSERT INTO people(user_id,
                   age,
                   status)
VALUES ("bcd001",
        45,
        "A")
```

MongoDB

```
db.people.insertOne(
   { user_id: "bcd001", age: 45, status: "A" }
)
```

SQL

```
UPDATE people
SET status = "C"
WHERE age > 25
```

MongoDB

```
db.people.updateMany(
   { age: { $gt: 25 } },
   { $set: { status: "C" } }
)
```

SQL

```
DELETE FROM people
WHERE status = "D"
```

MongoDB

```
db.people.deleteMany( { status: "D" } )
```

https://docs.mongodb.com/manual/

Document Data Model Example

Select

Document:

```
{
  _id: 1,
  user_id: "abc123",
  age: 55,
  status: 'A'
}
```

SQL	MongoDB
`SELECT * FROM people`	`db.people.find()`
`SELECT user_id, status FROM people WHERE status = "A"`	`db.people.find( { status: "A" }, { user_id: 1, status: 1, _id: 0 } )`
`SELECT * FROM people WHERE status = "A" OR age = 50`	`db.people.find( { $or: [ { status: "A" } , { age: 50 } ] } )`
`SELECT * FROM people WHERE age > 25 AND age <= 50`	`db.people.find( { age: { $gt: 25, $lte: 50 } } )`

Always selected if not deselected.

https://docs.mongodb.com/manual/

Document Data Model Example

Aggregate

Document:

```
{
  _id: 1,
  user_id: "abc123",
  age: 55,
  status: 'A'
}
```

SQL

```
SELECT COUNT(*)
FROM people
WHERE age > 30
```

MongoDB

```
db.people.count( { age: { $gt: 30 } } )
```

MongoDB's **aggregation pipeline**:
We can add additional operators like $match after the $group to further refine the result.

```
db.sales.aggregate(
  [ { $group : {
        _id : { month: { $month: "$date" },
                year: { $year: "$date" } },
        totalPrice: { $sum: { $multiply: [ "$price", "$quantity" ] } },
        averageQuantity: { $avg: "$quantity" },
        count: { $sum: 1 } } } ] )
```

Group the documents by month and year and calculate the **total price,** the **average quantity,** and the **count of documents** per group.

https://docs.mongodb.com/manual/

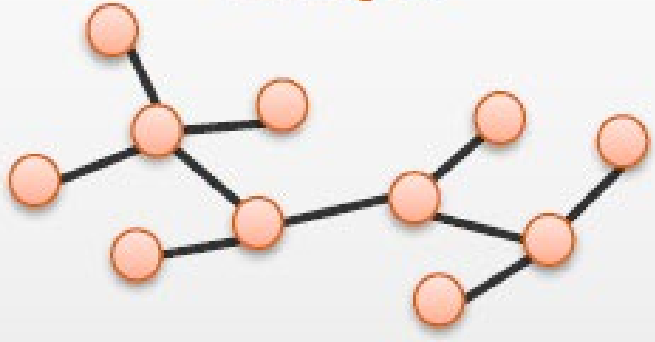
Non-Relational

Key-Value

Column-Family

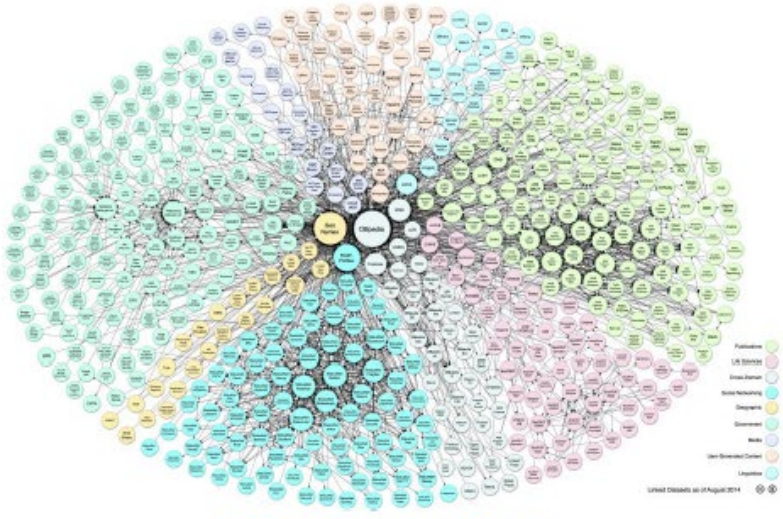
Document

Graph



Graph Data Model
Natural Graph Data

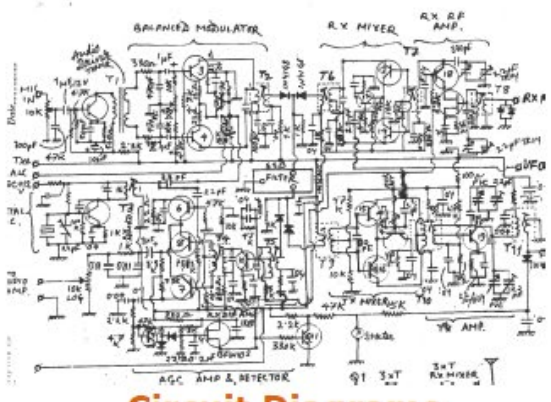
Social Graphs



Linked Open Data

Road and Rail Maps

Network Topologies

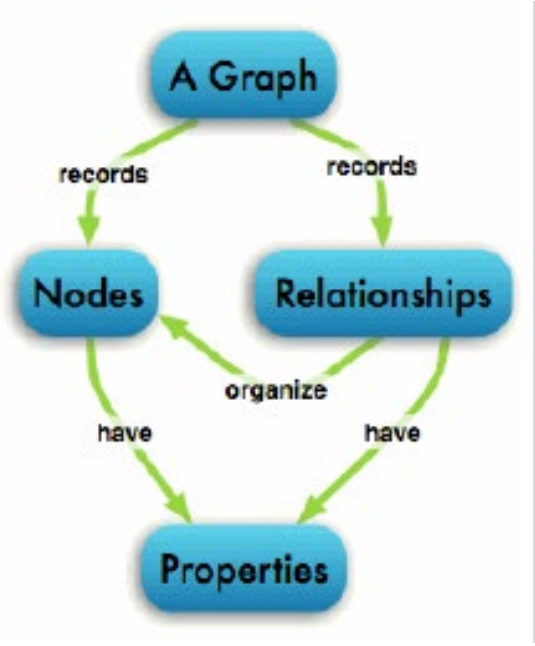


Circuit Diagrams

Graph DBMS (https://db-engines.com/en/ranking)

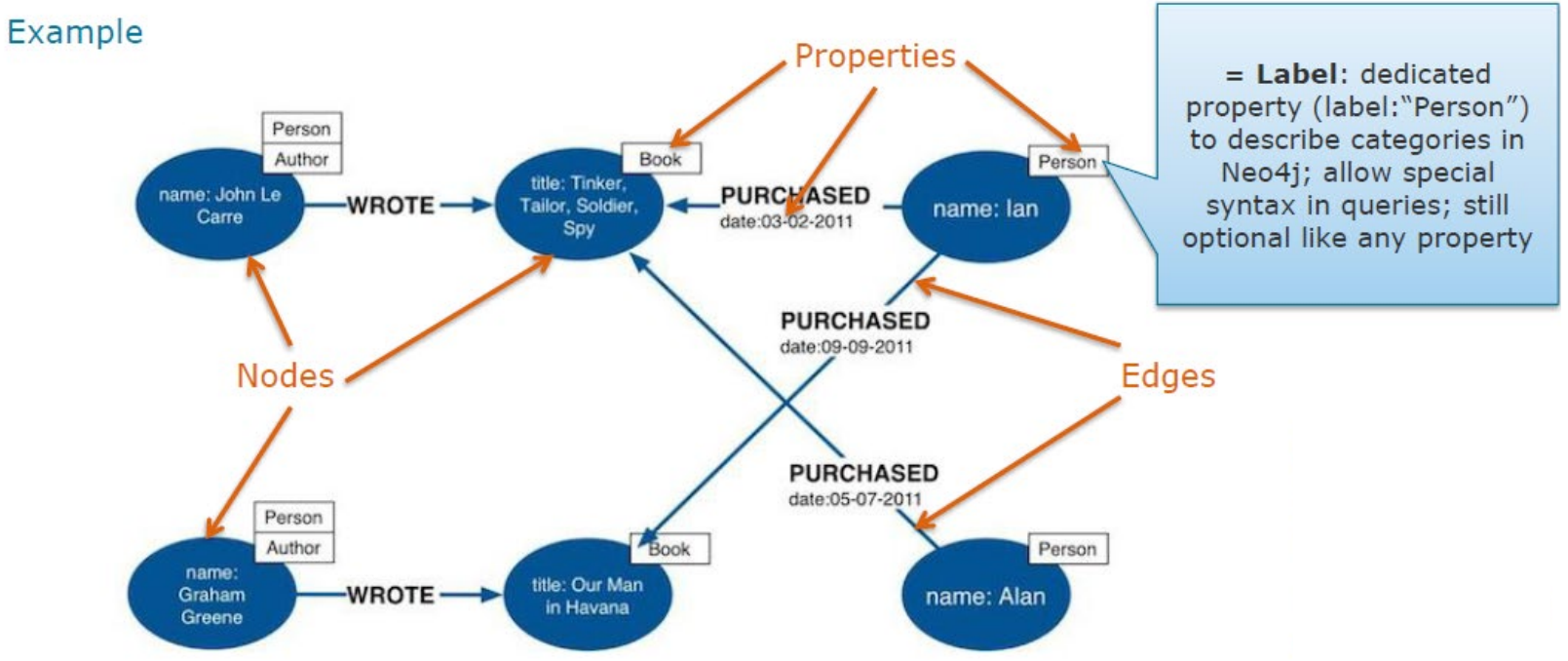
Rank			DBMS	Database Model	Score		
Sep 2022	Aug 2022	Sep 2021			Sep 2022	Aug 2022	Sep 2021
1.	1.	1.	Neo4j	Graph	59.48	+0.12	+1.85
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-model	40.67	-0.70	+2.15
3.	↑ 4.	3.	ArangoDB	Multi-model	6.02	+0.11	+1.23
4.	↓ 3.	4.	Virtuoso	Multi-model	5.96	-0.20	+1.54
5.	5.	5.	OrientDB	Multi-model	4.81	-0.39	+0.57
6.	6.	↑ 8.	Amazon Neptune	Multi-model	3.19	+0.11	+0.78
7.	↑ 8.	7.	JanusGraph	Graph	2.64	+0.12	+0.06
8.	↓ 7.	↓ 6.	GraphDB	Multi-model	2.52	-0.02	-0.08
9.	9.	9.	TigerGraph	Graph	2.14	-0.03	+0.24
10.	10.	↑ 12.	Stardog	Multi-model	1.67	+0.01	-0.03
11.	↑ 12.	11.	Fauna	Multi-model	1.56	+0.18	-0.17
12.	↓ 11.	↓ 10.	Dgraph	Graph	1.48	-0.09	-0.34
13.	13.	13.	Giraph	Graph	1.20	0.00	-0.09
14.	14.	↑ 15.	NebulaGraph	Graph	1.18	+0.10	+0.08
15.	15.	↓ 14.	AllegroGraph	Multi-model	1.12	+0.07	-0.13
16.	16.	↑ 18.	Graph Engine	Multi-model	0.93	+0.05	+0.13
17.	17.	↓ 16.	Blazegraph	Multi-model	0.89	+0.03	-0.02
18.	18.	↓ 17.	TypeDB	Multi-model	0.87	+0.05	+0.05

Graph Data Model (Properties)



- Structure
  - Nodes: entities equivalent to records in the relational model
  - Edges: (un)directed connections between nodes; represent relationships
  - Properties: information relating to nodes (and edges); equivalent to attribute-value or key-value pairs
- Constraints
  - Nodes: consist of a unique identifier, a set of outgoing edges, a set of incoming edges, and a collection of properties
  - Edges: consist of a unique identifier, the end- and start-nodes, a label, and a collection of properties
- Operations
  - Insert/query/update/delete notes, edges, and properties (CRUD)
  - Traverse edges; aggregate queries (avg, min, max, count, sum, …)
  - Most popular query language: Cypher (declarative; uses pattern matching)

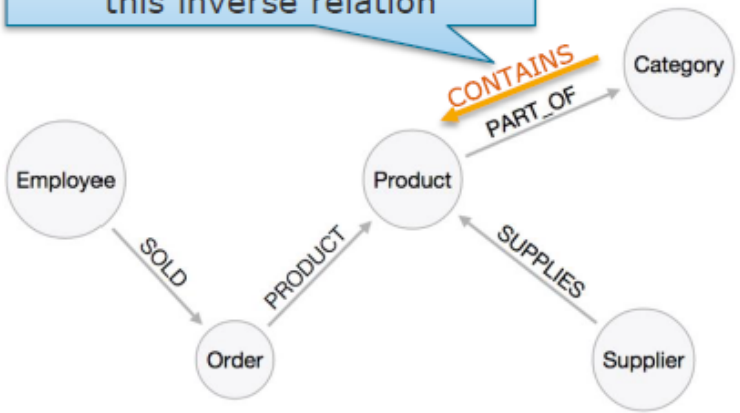
Graph Data Model Example

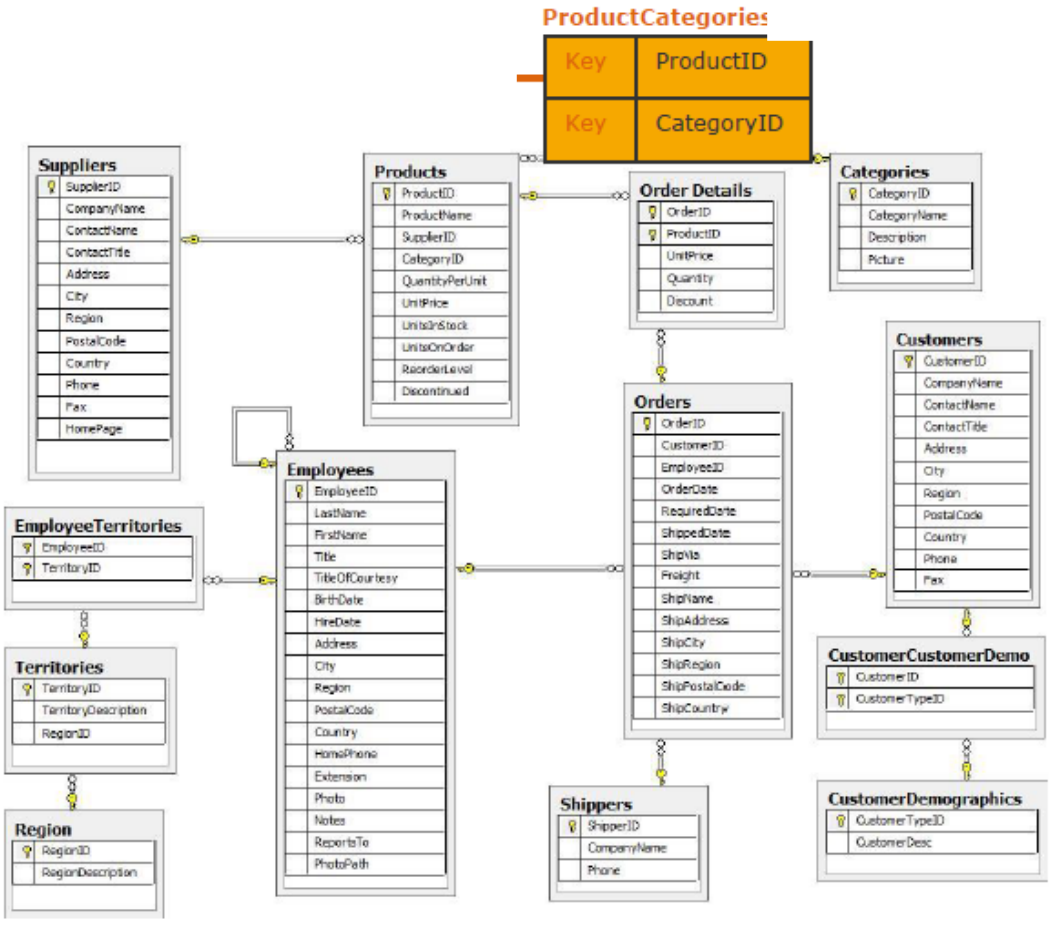


Graph V Relation

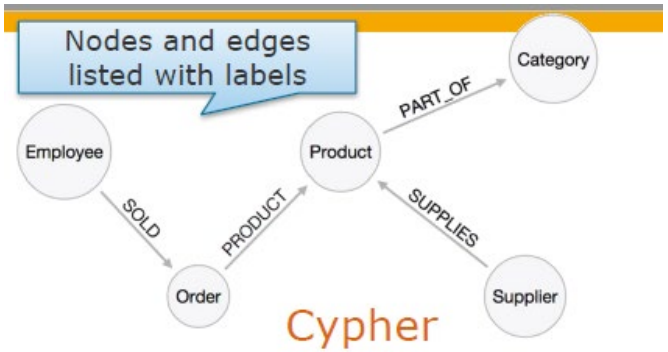
Model that Products can have multiple Categories!

Nothing to be done here; though we **could** also add this inverse relation





Graph Data Model Example



SQL

```
SELECT p.*
FROM products as p;
```

```
SELECT p.ProductName, p.UnitPrice
FROM products as p
ORDER BY p.UnitPrice DESC
LIMIT 10;
```

```
SELECT p.ProductName, p.UnitPrice
FROM products AS p
WHERE p.ProductName = 'Chocolade';
```

Cypher

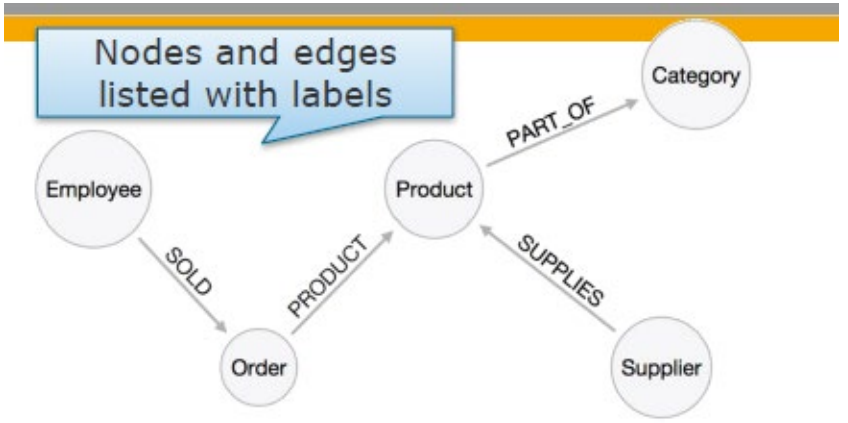
```
MATCH (p:Product)
RETURN p;
```

```
MATCH (p:Product)
RETURN p.productName, p.unitPrice
ORDER BY p.unitPrice DESC
LIMIT 10;
```

```
MATCH (p:Product)
WHERE p.productName = "Chocolade"
RETURN p.productName, p.unitPrice;

MATCH (p:Product {productName:"Chocolade"})
RETURN p.productName, p.unitPrice;
```

Graph Model Example



- **SQL**

```
SELECT DISTINCT c.CompanyName
FROM customers c, orders o, order_details od, products p
WHERE c.CustomerID = o.CustomerID
AND o.OrderID = od.OrderID
AND od.ProductID = p.ProductID
AND p.ProductName = 'Chocolade';
```

- **Cypher**

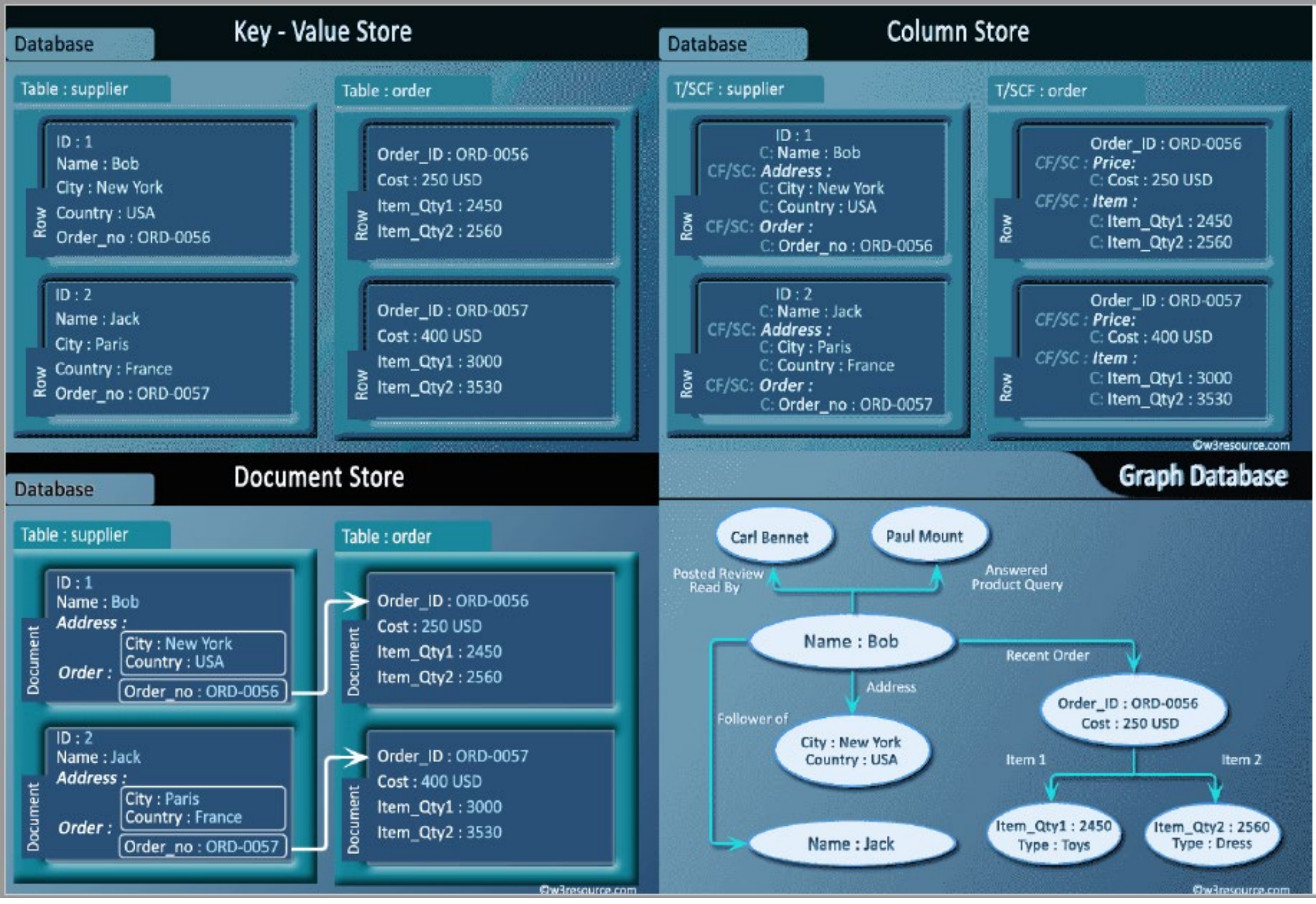
```
MATCH (c:Customer)-[:PURCHASED]->(:Order)-[:PRODUCT]->(p:Product)
WHERE p.productName = "Chocolade"
RETURN distinct c.companyName;
```

Strengths of Graph model

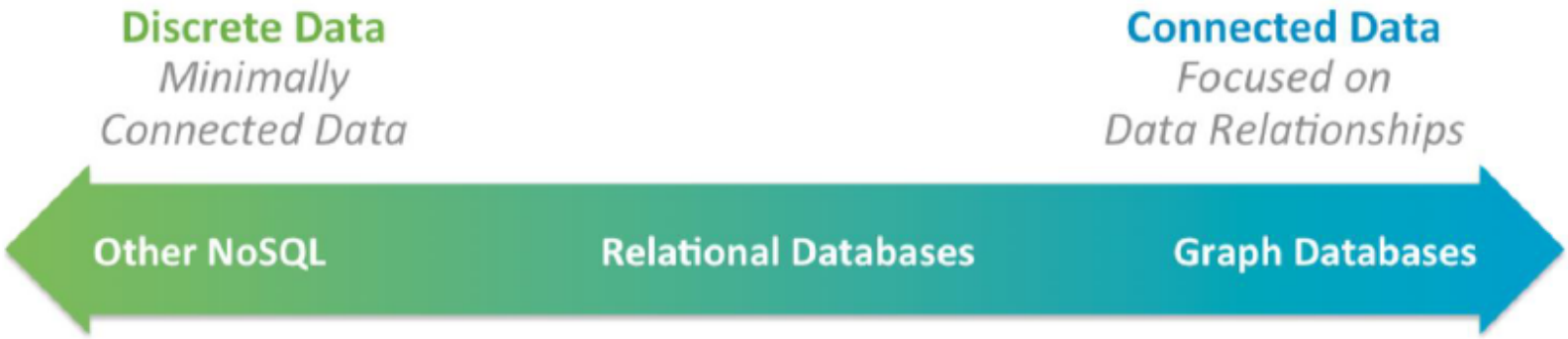
- Many-to-many relationships (other data models heavily prefer one-to-many)
- Efficient traversal of relationships between entities (relationship queries)
  - Traversal costs proportional to the average out-degree of nodes (and not proportional to the overall number of relationships)
  - Join performance scales naturally with the size of the data
- Natural support for graph queries: shortest path, community detection, …
- Flexible schemata due to flexible edge and property definitions
- Direct mapping of nodes/edges to data structures of object-oriented applications
- Support for consistency checking (ACID) and horizontal scalability

Weaknesses of Graph model

- OLTP and CRUD operations on many nodes are comparatively slow


Key - Value Store
Database
Table : supplier
Row
ID : 1
Name : Bob
City : New York
Country : USA
Order_no : ORD-0056
Row
ID : 2
Name : Jack
City : Paris
Country : France
Order_no : ORD-0057
Table : order
Row
Order_ID : ORD-0056
Cost : 250 USD
Item_Qty1 : 2450
Item_Qty2 : 2560
Row
Order_ID : ORD-0057
Cost : 400 USD
Item_Qty1 : 3000
Item_Qty2 : 3530
Column Store
Database
T/SCF : supplier
Row
ID : 1
C: Name : Bob
CF/SC: Address :
C: City : New York
C: Country : USA
CF/SC: Order :
C: Order_no : ORD-0056
Row
ID : 2
C: Name : Jack
CF/SC: Address :
C: City : Paris
C: Country : France
CF/SC: Order :
C: Order_no : ORD-0057
T/SCF : order
Row
Order_ID : ORD-0056
CF/SC : Price:
C: Cost : 250 USD
CF/SC : Item :
C: Item_Qty1 : 2450
C: Item_Qty2 : 2560
Row
Order_ID : ORD-0057
CF/SC : Price:
C: Cost : 400 USD
CF/SC : Item :
C: Item_Qty1 : 3000
C: Item_Qty2 : 3530
©w3resource.com
Document Store
Database
Table : supplier
Document
ID : 1
Name : Bob
Address :
City : New York
Country : USA
Order :
Order_no : ORD-0056
Document
ID : 2
Name : Jack
Address :
City : Paris
Country : France
Order :
Order_no : ORD-0057
Table : order
Document
Order_ID : ORD-0056
Cost : 250 USD
Item_Qty1 : 2450
Item_Qty2 : 2560
Document
Order_ID : ORD-0057
Cost : 400 USD
Item_Qty1 : 3000
Item_Qty2 : 3530
©w3resource.com
Graph Database
Carl Bennet
Paul Mount
Posted Review
Read By
Answered
Product Query
Name : Bob
Recent Order
Address
Follower of
City : New York
Country : USA
Order_ID : ORD-0056
Cost : 250 USD
Item 1
Item 2
Item_Qty1 : 2450
Type : Toys
Item_Qty2 : 2560
Type : Dress
Name : Jack
©w3resource.com

Summary



- inhomogeneous data
- frequent schema changes
- fast growth
- little/no relationship support
- usually sacrifice ACID
- usually horizontal scaling
- data distribution
- throughput
- OLTP focus

- homogeneous data
- relatively reliable schemata
- moderate growth
- full relationship support
- usually comply with ACID
- usually vertical scaling
- data compression
- transactions and security
- OLTP and OLAP

- highly linked data
- frequent schema changes
- moderate growth
- specialized on relationships
- usually comply with ACID
- usually horizontal scaling
- data distribution
- relationship traversal
- OLAP focus

Image: © https://neo4j.com/blog/aggregate-stores-tour/