
# INF115 Lecture 18: *Via Objects to NoSQL (Part 2)*

Adriaan Ludl
Department of Informatics
University of Bergen

Spring Semester 2021

# Chapter 10: *Via Objects to NoSQL*

## Learning Goals:

- **Limitations of relational databases**
- Motivation for ***object relational*** **databases** and ***NoSQL*** **databases**.
- Create user-defined data types in the ***object-relational*** database **PostgreSQL**.
- **Storage principles** and **use cases** of **NoSQL** databases.
- **Design *document* databases** in **MongoDB** and use the JavaScript API to *insert, modify, delete and retrieve* data.
- Design ***graph*** **databases** in **Neo4j**, use the query language **Cypher** to *insert, change, delete and retrieve* data, as well as migrate table data to Neo4j.

# NoSQL and Big Data

**NoSQL** is a collective term for a number of newer alternatives to relational databases.

- NoSQL = **Not Only SQL**
- Complex data structures
- Semi-structured data
- Distributed solutions
- Looser requirements for transaction handling

**Big Data**: datasets that are so large and complex that they push the boundaries of what is possible to handle, both in terms of storage, exchange and processing.

- **Volume + Velocity + Variety + Veracity -> Value**

# NoSQL and «Big» Data

**NoSQL** is a collective term for a number of newer alternatives to relational databases.

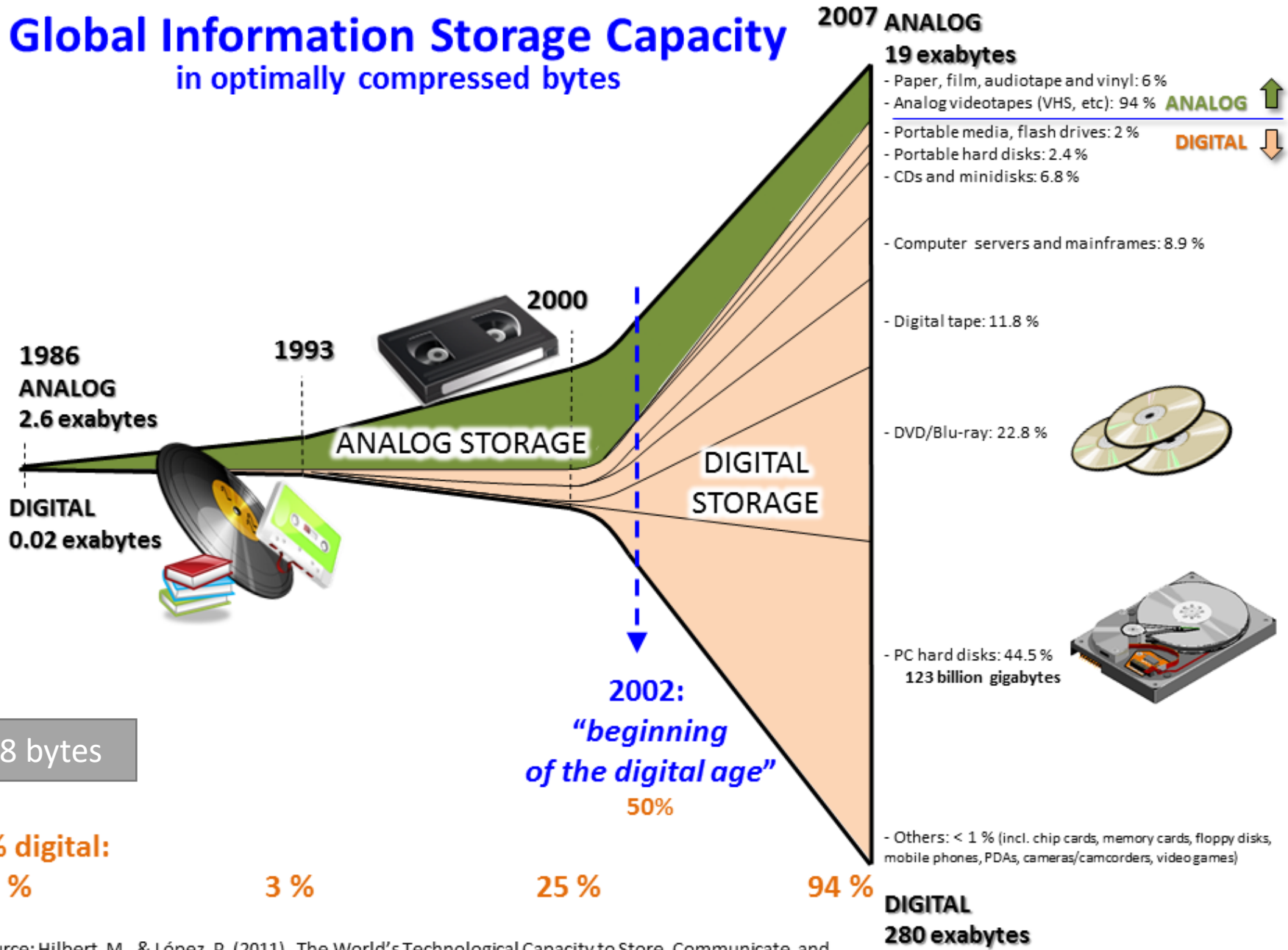
- NoSQL = **Not Only SQL**
- Complex data structures
- Semi-structured data
- Distributed solutions
- Looser requirements for transaction handling

**Big Data**: datasets that are so large and complex that they push the boundaries of what is possible to handle, both in terms of storage, exchange and processing.

- **Volume** + **Velocity** + **Variety** + **Veracity** -> **Value**

# Global Information Storage Capacity

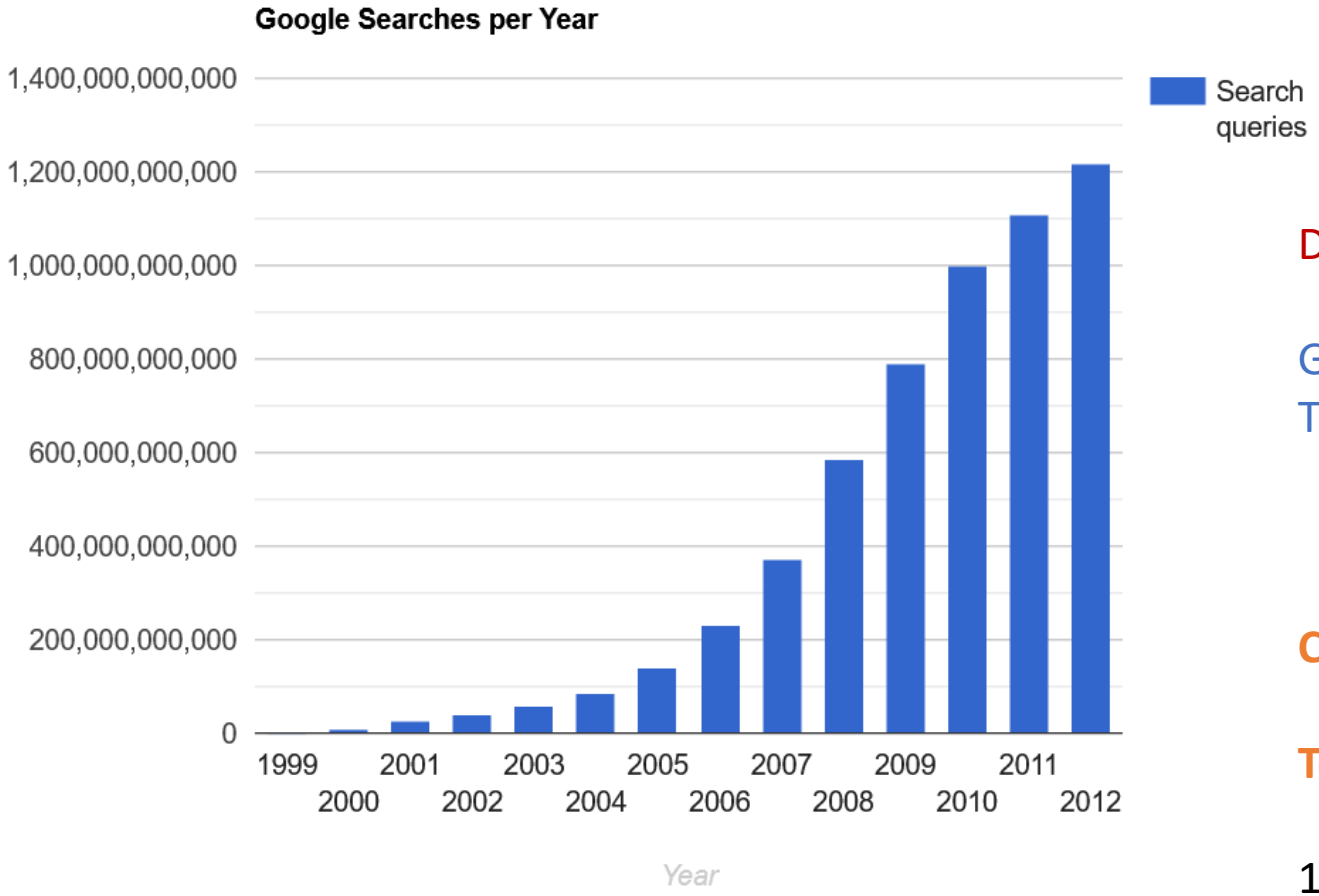
## in optimally compressed bytes




1 exabyte = 10^18 bytes

Source: Hilbert, M., & López, P. (2011). The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332(6025), 60 –65. http://www.martinhilbert.net/WorldInfoCapacity.html

# Example: Google Searches vs CERN data




Data access -> costs **energy** and generates **heat** [4, 5].

Google ~ 1.2 * 10^12 searches per year.
The **Google Search index** > 100 PB in size [2].

**vs**

**CERN** Particle Physics Experiments
generate **one petabyte per day**
**Total storage ca 256 PB.** [6]

1PB = one million gigabytes ~ 10^15


1. https://www.internetlivestats.com/google-search-statistics/#ref-7
2. https://www.google.com/search/howsearchworks/crawling-indexing/
3. https://www.forbes.com/sites/robertbryce/2020/10/21/googles-dominance-is-fueled-by-zambia-size-amounts-of-electricity/
4. https://www.nrk.no/rogaland/sosiale-medier-krever-enorme-mengder-strom-1.15461068
5. https://www.bbc.com/news/business-29423535
6. https://home.cern/science/computing/storage

# NoSQL and «Big» Data

**NoSQL** is a collective term for a number of newer alternatives to relational databases.

- NoSQL = **Not Only SQL**
- Complex data structures
- Semi-structured data
- Distributed solutions
- Looser requirements for transaction handling

**Big Data**: datasets that are so large and complex that they push the boundaries of what is possible to handle, both in terms of storage, exchange and processing.

- **Volume + Velocity + Variety + Veracity -> Value**

# Key / value databases

- Data is organized by a collection of **key / value pairs**, such as First name = 'Ola'.
- Suitable for applications that make many, simple lookups based on keys.
- Examples: Redis, Memcached, Riak.

| Nøkkel | Verdi |
|---|---|
| Elev:1:Fornavn | Ailin |
| Elev:1:Etternavn | Liane |
| Elev:1:Fødselsdato | 09.10.2010 |
| Elev:2:Fornavn | Gorm |
| Elev:2:Etternavn | Syrstad |

# Document databases

- Data is stored as a number of **semi-structured** **documents**, often in the form of JSON or XML.
- Each document has a **unique ID** (key).
- Suitable as a database behind **web solutions**.
- Examples: **MongoDB**, CouchDB, Couchbase.

| Id | Dokument |
|---|---|
| 1 | { Fornavn: Ailin, Etternavn: Liane, Fødselsdato: 09.10.2010 } |
| 2 | { Fornavn: Gorm, Etternavn: Syrstad } |

# Column databases

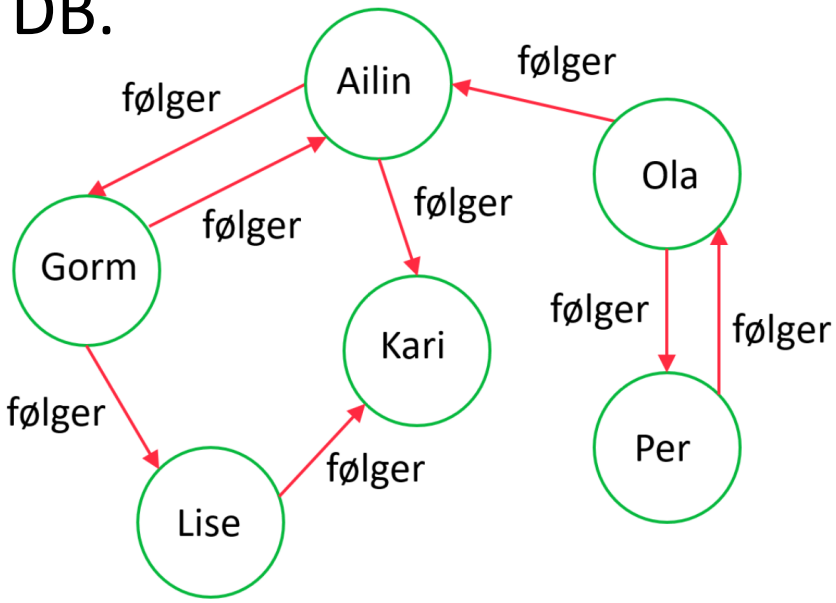
- Data is stored as **key / value pairs**, but different **columns** or column families are **stored separately**.
- Tables in a relational database often contain many columns, but typical queries use only a few of these. By splitting tables into columns ("belonging together"), an efficiency gain can be achieved .
- Examples: Cassandra, HBase.

| Id | Navn |
|---|---|
| 1 | Fornavn:Ailin, Etternavn: Liane |
| 2 | Fornavn: Gorm, Etternavn: Syrstad |

| Id | Alder |
|---|---|
| 1 | Fødselsdato: 09.10.2010 |

# Graph databases

- A **graph** (a "network") is made up of **nodes** and **edges** .
- Such databases are suitable for domains that are naturally graph-like, e.g. social networks: who follows who others on Twitter.
- Examples: **Neo4J**, Microsoft Azure Cosmos DB.



# Quizz on *NoSQL Databases* (part 1)

Please answer the practice quizz on mitt.uib now ☺

(you can take it again later if you want)

**Link:**

- https://mitt.uib.no/courses/27455/quizzes

# MongoDB


- MongoDB is a popular **document** **database** .
- The name is an abbreviation of the English word humongous, which in turn seems to be a creative juxtaposition of huge and monstrous, i.e. something big.
- Presented as a **platform-independent** and **scalable** database for handling large amounts of data, often in distributed solutions.
- Data is stored in **BSON** format – a **binary** “variant” of **JSON** that supports slightly more data types.

# Documents

- A **document** is represented as a BSON object and is organized into **document collections**.
- A document collection can be compared to a table in a relational database.
- A document consists of a number of **fields** , and each field is a **key / value pair** .

```
{
    VNr: "10830",
    Designation: "Santa beard",
    Price: 66.50,
    Number: 42
}
```

# CRUD operations

**CRUD**: create, read, update, and delete

Example of inserting two items:

```
db.Vare.insertMany([
 {_id: "10820", Betegnelse: "Dukkehår, hvitt",
  Pris : 46.50, Antall: 106, Hylle: "E12", KatNr: "13"},
 {_id: "80692", Betegnelse: "Rørespatel",
  Pris : 45.50, Antall: 602, Hylle: "B17", KatNr: "4"}
]);
```

- The **dot notation** db.Vare refers to the document collection *Vare* in database db.
- Each *Vare* is built up as a BSON object and put together in a table. The last *Vare* is missing the *Hylle* field .

# CRUD operations

**CRUD**: create, read, update, and delete

Example of inserting two items:

```
db.Vare.insertMany([
 {_id: "10820", Betegnelse: "Dukkehår, hvitt",
  Pris : 46.50, Antall: 106, Hylle: "E12", KatNr: "13"},
 {_id: "80692", Betegnelse: "Rørespatel",
  Pris : 45.50, Antall: 602, Hylle: "B17", KatNr: "4"}
]);
```

- The **dot notation** db.Vare refers to the document collection *Vare* in database db.
- Each *Vare* is built up as a BSON object and put together in a table. The last *Vare* is missing the *Hylle* field .

15 minute break!
Lecture resumes at 15:00

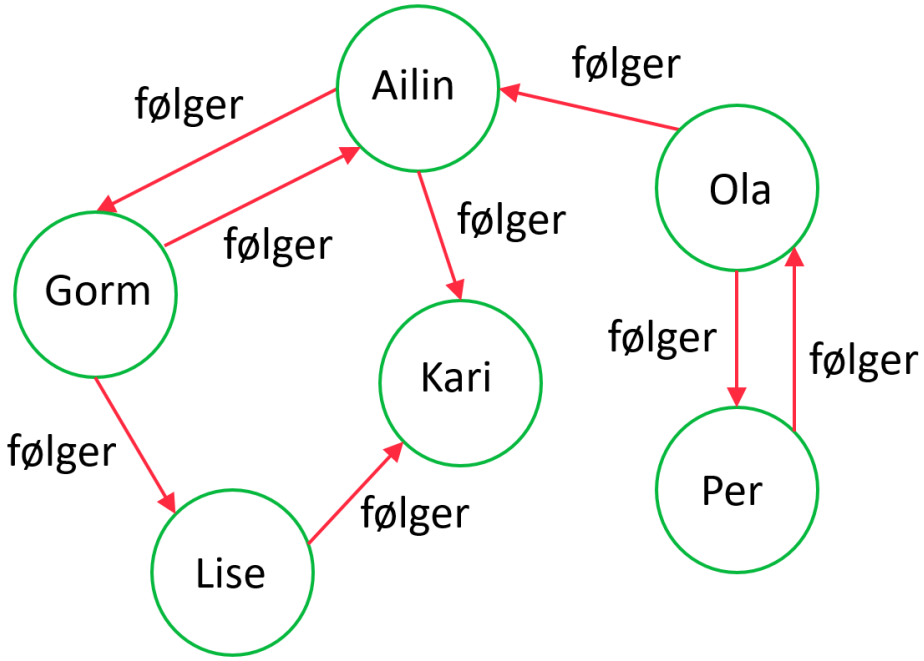
# Queries with the JavaScript API

- This API has methods such as **find** to search the database.
- Conditions are built up as **BSON values** .
- Uses special **$ operators** for comparison (equality $eq, greater than $gt, less than $lt) and **logical operators** (and, or, not).

```
db.vare.find({
   $and:[
     {Pris : {$gte: 30}},        note: $gte means greater or equal
     {Antall: {$lt : 100}}
   ]
}).limit(3);
```

# Neo4J

- Neo4J is a **graph database**.
- A graph consists of **nodes** and **edges**.
- Nodes and edges have **properties** and each property has a value.
- Nodes can be given one or more **node tags** (or “labels”).



# Neo4J

- ASCII art is a way of creating "images" by putting together the symbols in the ASCII character set. Example: **:-)**
- **Cypher** is the **query language** of Neo4j. It is used to extract data from graph databases. It is a declarative, SQL-inspired language for describing visual patterns in graphs using *ASCII-Art syntax.*
- The **pattern** that describes that **two nodes are connected** by an edge is:

```
(  )-->(  )
```

- The query that returns all customer pairs, whe… **query result is also a graph** :

```
MATCH (k1:Kunde)-[f:FØLGER]->(
RETURN k1, f, k2
```

https://neo4j.com/developer/cypher/
https://en.wikipedia.org/wiki/ASCII_art

# Neo4J

- ASCII art is a way of creating "images" by putting together the symbols in the ASCII character set. Example: :-)
- **Cypher** is the **query language** of Neo4j. It is used to extract data from graph databases. It is a declarative, SQL-inspired language for describing visual patterns in graphs using *ASCII-Art syntax.*
- The **pattern** that describes that **two nodes are connected** by a *directed* edge is:

```
( )-->( )
```

- The query that returns all customer pairs, where one follows the other – **the query result is also a graph** :

```
MATCH (k1:Kunde)-[f:FØLGER]->(k2:Kunde)
RETURN k1, f, k2
```

https://neo4j.com/developer/cypher/
https://en.wikipedia.org/wiki/ASCII_art

# ORDER BY and WHERE

- **ORDER BY** works similarly to SQL.

- By retrieving properties (Surname) in **RETURN** , the query result is in tabular form.

```
MATCH (k1:Kunde)−[f:FØLGER]−>(k2:Kunde)
RETURN k1.Etternavn, k2.Etternavn
ORDER BY k1.Etternavn
```

- We can make comparisons and combine conditions with **AND** , **OR** and **NOT** as in SQL.

```
MATCH (v:Vare)
WHERE v.Hylle="E12" AND v.Pris>50
RETURN v
```

# CRUD operations

- Insertion:

```
CREATE (:Vare {VNr:"10830",
Betegnelse:"Dukkehår, hvitt",
Pris :46.50, Antall:106, Hylle:"E12"})
```

- Updating:

```
MATCH (v:Vare {VNr:"10830"})
SET v.Betegnelse = "Dukkehår, grått"
```

- Removing:

```
MATCH (v:Vare {VNr:"10830"})
DELETE v
```

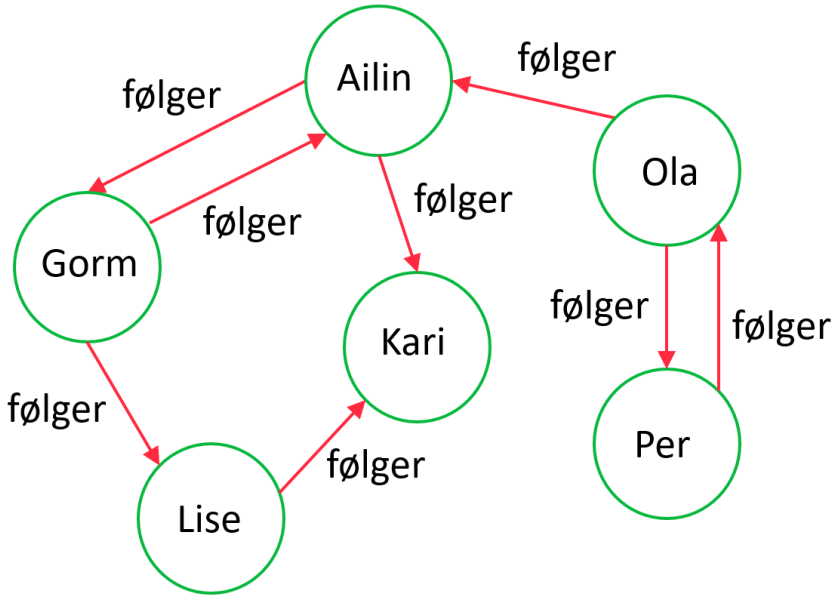
# Aggregation

- Match a **cycle** – k1 follows k2 which follows k3 which follows k1.
- Uses **two** patterns in **MATCH** :

```
MATCH (k1:Kunde)-->(k2:Kunde)-->(k3:Kunde),
(k3:Kunde)-->(k1:Kunde)
RETURN k1, k2, k3
```

- Set functions can be used without explicit grouping:

```
MATCH (v:Vare)-[:TILHØRER]->(k:Kategori)
RETURN k.Navn, count(*) AS AntallVarer
ORDER BY k.Navn
```



# Quizz on *Via Objects to NoSQL* (part 2)

Please answer the practice quizz on mitt.uib now ☺

(you can take it again later if you want)

**Link:**

- https://mitt.uib.no/courses/27455/quizzes

# Examples of patterns:

- Chain:

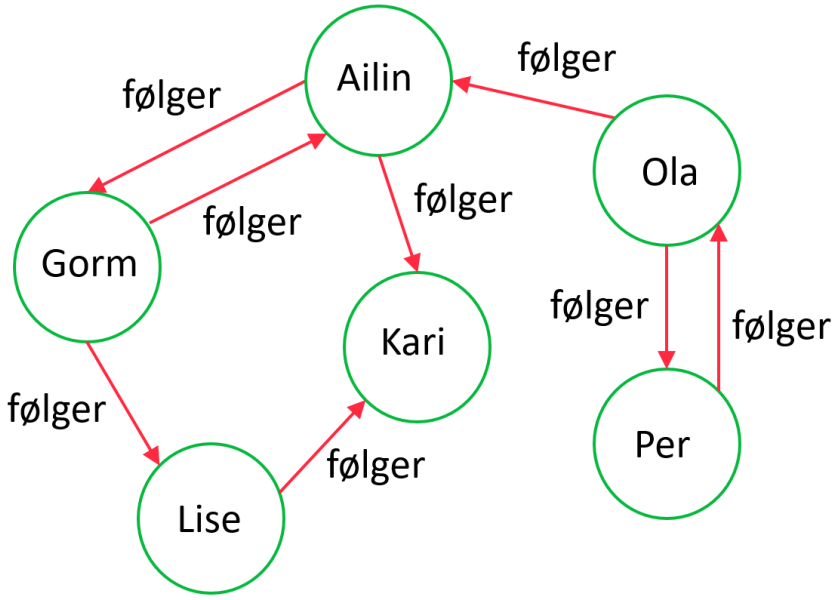
```
MATCH (k1:Kunde)-->(k2:Kunde)-->(k3:Kunde)
```

- Cycle:

```
MATCH (k1:Kunde)-->(k2:Kunde)-->(k3:Kunde),
(k3:Kunde)-->(k1:Kunde)
```

- Fork:

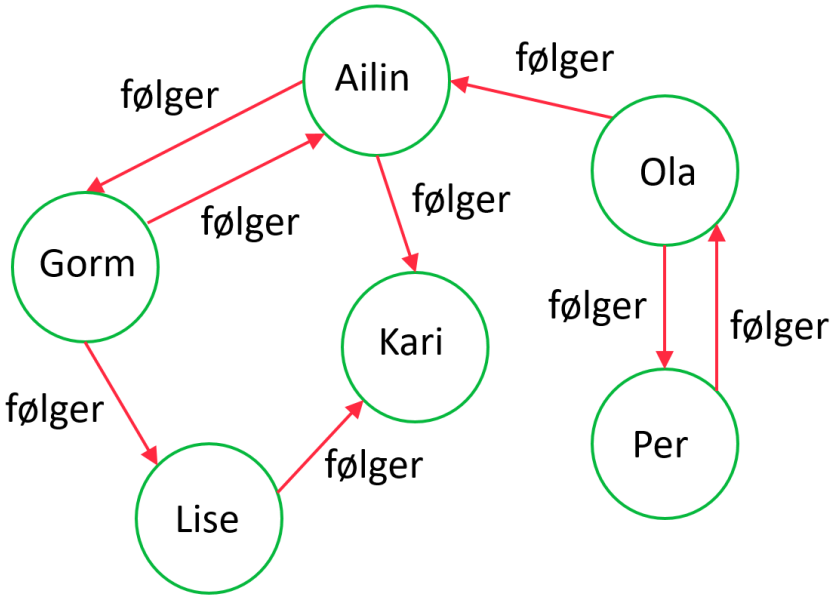
```
MATCH (k1:Kunde)-->(k2:Kunde),
(k1:Kunde)-->(k3:Kunde)
```



# Graph Algorithms

- The function **shortestPath** calculates the shortest path between nodes in the graph. You can do this for two specific nodes, or for any combination of nodes.
- Example: Find the (smallest) number of "sequences" from a particular customer to all others, where [: FØLGER *] matches all possible directed paths along edges of type [: FØLGER]. ( FØLGER means FOLLOWS )

```
MATCH (k1:Kunde {Fornavn: Ola'}), (k2:Kunde)
WHERE k1.KNr > k2.KNr
MATCH p = shortestPath((k1)–[:FØLGER*]–>(k2))
RETURN p
```



# From Table Data to Graph Databases

- A relational database can be **migrated** to a graph database in several ways. One possible solution:
  - **Rows become nodes .**
  - Table names become node tags.
  - Columns become properties.
  - Some surrogate keys are removed, but natural keys are retained.
  - **Foreign keys become directed edges .**
  - Some connection tables can be replaced by edges with properties.
  - Relationship names become types (on edges).

# From Hierarchies via Relationships to NoSQL

Database history started with **hierarchical** databases and **network** databases

back in the 1960s.

- **Edgar F. Codd** [Codd 1970] **argued against** this way of organizing data (see Chapter 6), **because** it was based on navigation structures, or the use of " **pointers** ".

- With **object-relational** databases and **NoSQL** databases, we work again **with "pointers"**. The *queries* we write to such databases must *follow these pointers*, and thus become **dependent** on the selected data structure. **Is this a problem?**

  - **Relational databases are** still well suited for **structured data**,
  - But that various **NoSQL databases** are better suited for applications that handle **complex** and / or **semi-structured data**.

# From Hierarchies via Relationships to NoSQL

Database history started with **hierarchical** databases and **network** databases

back in the 1960s.

- **Edgar F. Codd** [Codd 1970] **argued against** this way of organizing data (see Chapter 6), **because** it was based on navigation structures, or the use of " **pointers** ".
- With **object-relational** databases and **NoSQL** databases, we work again **with "pointers"**. The ***queries*** we write to such databases must *follow these pointers*, and thus become **dependent** on the selected data structure. **Is this a problem?**

- **Relational databases are** still well suited for **structured data**,
- But that various **NoSQL databases** are better suited for applications that handle **complex** and / or **semi-structured data**.

# From Hierarchies via Relationships to NoSQL

Database history started with **hierarchical** databases and **network** databases

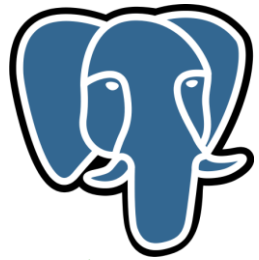
back in the 1960s.

- **Edgar F. Codd** [Codd 1970] **argued against** this way of organizing data (see Chapter 6), **because** it was based on navigation structures, or the use of " **pointers** ".
- With **object-relational** databases and **NoSQL** databases, we work again **with "pointers"**. The *queries* we write to such databases must *follow these pointers*, and thus become **dependent** on the selected data structure. **Is this a problem?**

❖ **Relational databases are** still well suited for **structured data**,

❖ But that various **NoSQL databases** are better suited for applications that handle **complex** and / or **semi-structured data**.

# Summary: *Via Objects to NoSQL*




- **NoSQL**: Not only SQL
- **Value** of Big Data: **Volume** + **Velocity** + **Variety** + **Veracity**
- **Paradigms** and **use cases** of **NoSQL** databases.
- Design ***document*** **databases** in **MongoDB.**
- Design ***graph*** **databases** in **Neo4j**, use the query language **Cypher.**