
# INF115 Lecture 17:
# *Via Objects to NoSQL*

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.

# Criticism of relational databases

Relational databases are **less suitable** for handling **complex** data structures:

- **Digital maps** for Geographic Information Systems (**GIS**)
- Technical **drawings** (DAK / DAP)
- **Multimedia**
- **Semi-structured** *documents*
- *Social Media*
- **Big data** collected on the « Internet of Things »
- …

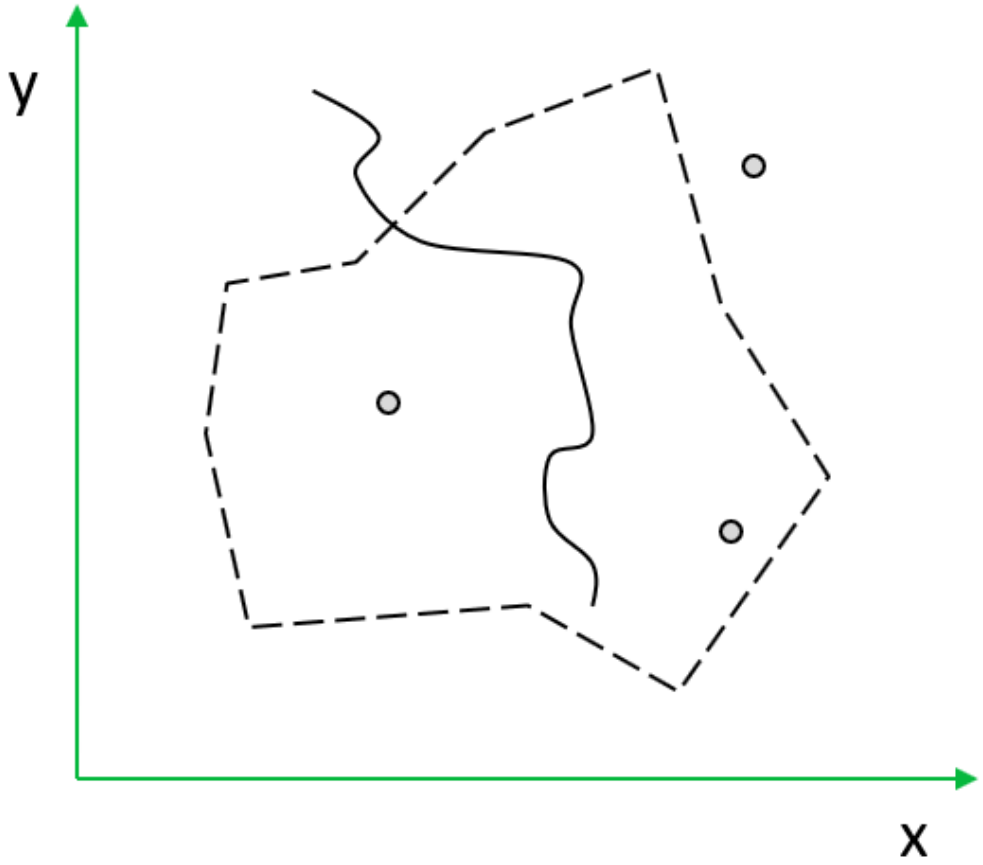
GIS as an example

- Problem to **represent** *digital maps* in **tables** (relational databases)


Hoyanger
2 hr 50 min
130 km
3 hr 28 min
Juterustane
Sognefjord
Viksdalsvatnet
Lustrafjorden
Kvitingsmorki
Bleia
Bleia-Storebotnen
Stølsheimen
Google
Map data ©2021
Norway
Terms
Privacy
Send feedback
5 km

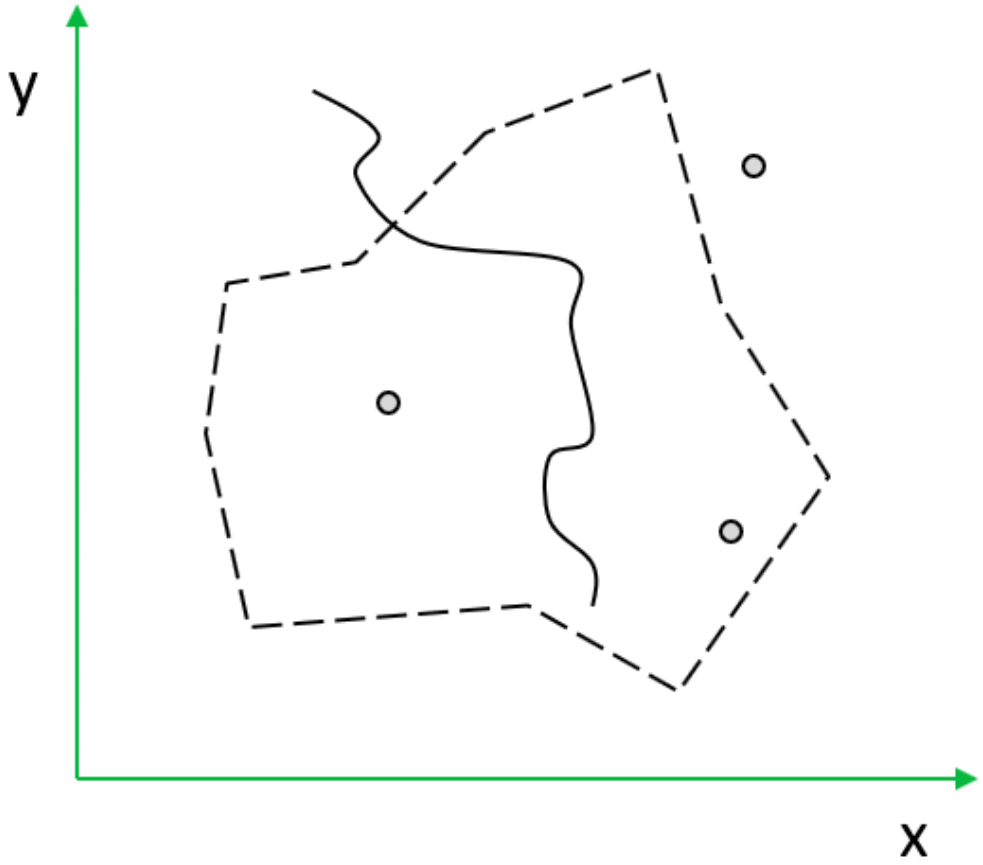

MAPS
Hoyanger
Juterustane
2 hr 50 min
130 km
3 hr 28 min
Sognefjord
Viksdalsvatnet
Lustrafjorden
Kvitingsmorki
Bleia
Bleia-Stolsheimen
Stølsheimen
Google
Map data ©2021
Norway
Terms
Privacy
Send feedback
5 km

# Maps in vector form




- A **map** in **vector form** consists of **points** , **lines** and **surfaces** .
- A **point** can represent a **city**,
- a **polyline** is a road,
- and a **polygon** is a county.

# Maps in vector form




- A **map** in **vector form** consists of **points** , **lines** and **surfaces** .
- A **point** can represent a **city**,
- a **polyline** is a road,
- and a **polygon** is a county.

# Points and Line Segments

- A **point** can be represented as (X, Y).
  - Possible table: Point (X, Y)
- A line **segment** can be represented by the **start** point and **end** point.
  - Possible table: Line segment (X1, Y1, X2, Y2)
- It could be tempting to save **points as values** :
  - Line segment (P1, P2)

- If we do not have a data type «Point» this does not work:

  Values must be **atomic** - we do not get a point "into" a single column.

# Polylines

- A **polyline** can be represented as a **sequence of points** (coordinate pairs).

  Assume a maximum of **3** points.

  - Possible table: Polyline (X1, Y1,  X2, Y2,  X3, Y3)

- In general, we do not want to limit ourselves to 3 points per. polyline.

  - Option **1** : Polyline (X1, Y1,…, X20, Y20)
  - Option **2** : Create a table that can express that a given point is point number nine, the point sequence for a given line (see next slide).
  - Option **3** : Save Polylines as Binary Large Objects (BLOBs).
  - Option **4** : Introduce a data type Polyline.


Hoyanger
2 hr 50 min
130 km
3 hr 28 min
Juterustane
Sognefjord
Viksdalsvatnet
Lustrafjorden
Kvitingsmorki
Bleia
Bleia-Storebotnen
Stølsheimen
Google
Map data ©2021
Norway
Terms
Privacy
Send feedback
5 km

# Polylines

- A **polyline** can be represented as a **sequence of points** (coordinate pairs).

  Assume a maximum of 3 points.

  - Possible table: Polyline (X1, Y1, X2, Y2, X3, Y3)

- In general, we **do not want to limit** ourselves to 3 points per polyline.
  - Option **1** : Polyline (X1, Y1, … , X20, Y20)
  - Option **2** : Create a table that can express that a given point is point number nine, the point sequence for a given line (see next slide).
  - Option **3** : Save Polylines as Binary Large Objects (BLOBs).
  - Option **4** : Introduce a **data type** Polyline.

# Normalized data model for map data

- A polyline can consist of **many** points. A point can be included in many polylines.
- We get a **table** *Point*, a table *Line* and a linking table *PointLine*. (koblingstabell)
- It may be appropriate to assign points and lines a serial number (Pid and Lid):
  - Point ( Pid , X, Y,…)
  - Line ( Lid ,…)
  - **PointLine ( Lid *, Pid *, SequenceNo )**
- One **polyline** with **3 points** will now be represented by
  - 3 rows in *Point*,
  - 1 row in *Line* and
  - 3 rows in *DotLine*,

  where the points have **serial numbers** 1, 2 and 3.

# Normalized data model for map data

- A polyline can consist of many points. A point can be included in many polylines.
- We get a **table** *Point*, a table *Line* and a link table *PointLine*. (koblingstabell)
- It may be appropriate to assign points and lines a serial number (Pid and Lid):
  - Point ( Pid , X, Y,…)
  - Line ( Lid ,…)
  - **PointLine ( Lid *, Pid *, SequenceNo )**
- One **polyline** with **3 points** will now be represented by
  - 3 rows in *Point*,
  - 1 row in *Line* and
  - 3 rows in *PointLine*,

  where the points have **serial numbers** 1, 2 and 3.

# Quizz on *Via Objects to NoSQL* (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

# The problem of table organization

- **Polygons** can be represented in the same way as polylines, where we require that the last point has the same coordinates as the first point.
- **Geometric** objects **can** thus be **represented** in **tables** by means of simple data types.
- But (simple) "***map operations***" are **difficult to code with SQL**. Example:
  - Is a specific point (a click in a map) within a polygon?
  - Do two polylines intersect?


But conceptually simple "*map operations*" are **difficult to code with SQL**.

Examples:

- Is a specific point (a click in a map) within a polygon?
- Do two polylines intersect?

# Data types for geometry

- The *County* (fylke) entity has attributes Name and Area content.
- *County* also has a **geometric property** , namely the **polygon** that describes the county boundary.
  - County ( FId , PolygonID *, Name, Area Content)
- Instead of a foreign key against a "geometry table", one could wish for the polygon to be represented directly in the county table, as follows:
  - County ( FId , **Boundary** , Name, Area Content)

➢ This requires that we can provide an appropriate data type for **Boundary**.

➢ We might need a **data type** *Polygon* .


# Geometric operations

We want to make a **query** that finds all cities located ***along*** the Glomma.

---

- We must for each city (with given coordinates) calculate the distance to the polyline Glomma, which means that we must **calculate the distance** from each city to each of the line segments that Glomma is made up of.
- If we **expand** SQL with **data types** *Point* and *Polyline*, it will be natural that some **basic operations** on points and lines are also built in, e.g. distance:

```
SELECT Byer.*
FROM Byer, Elv
WHERE Distance(By.Punkt, Elv) < 0.5
AND Elv.Navn = "Glomma"
```

15 minute break!
Lecture resumes at 11:00

# What was the point?

- Processing of map data shows the *need* for

  **complex**, **user-defined** (=custom) ***data types***.

- In ***object-oriented*** database systems and programming languages, you can define **your own data types** with **associated operations**.

- Map data is only one example - but a good example - of the need for user-defined data types.
  - Geometry data types are described in the standard:
    ISO/IEC 13249 SQL/MM:
    *SQL multimedia and application packages.*

# What was the point?

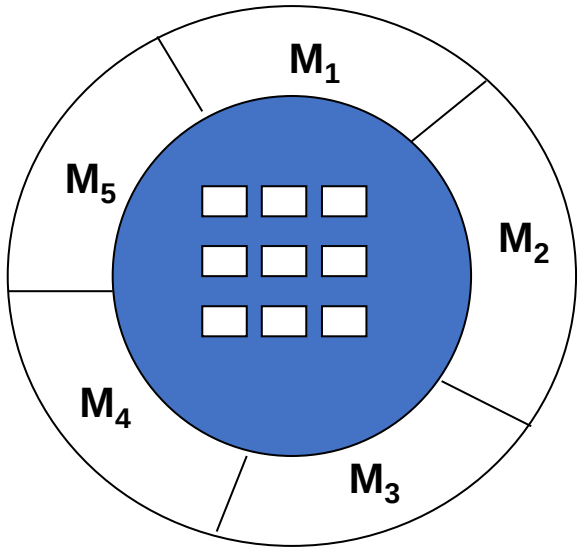
- Processing of map data shows the *need* for

  **complex**, **user-defined** (=custom) ***data types***.

- In ***object-oriented*** database systems and programming languages, you can define **your own data types** with **associated operations**.

- Map data is only one example - but a good example - of the need for user-defined data types.
  - Geometry data types are described in the standard:
    ISO/IEC 13249 **SQL/MM**:
    *SQL multimedia and application packages.*

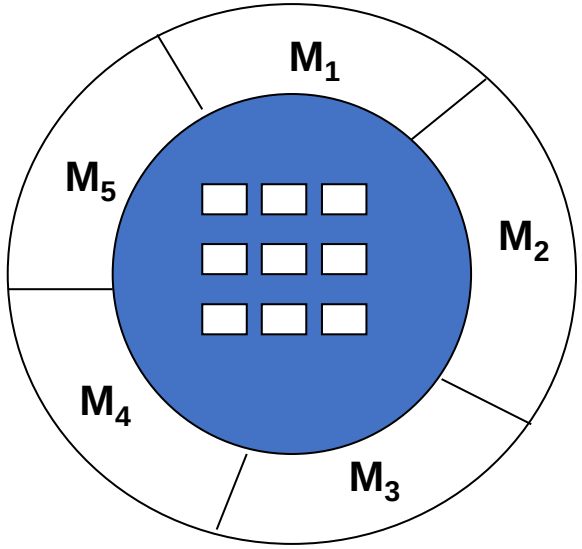
# Object = data + methods




- An **object** is a thing, a concept.
  - Example: Employee
- An object is in a certain **state** at any given time.
  - EmployeeNo = 1, Surname = "Hansen", Salary = 475,000
- An object can go into a new state, but it still remains "the same" object.
  - Hansen is Hansen even after she gets a pay rise.
- An object has **methods** .
  - anEmployee.changePay (newPay)
- Objects **protect** their data elements = **encapsulation**.

  All access goes via the methods.

# Object = data + methods




- An **object** is a thing, a concept.
  - Example: Employee
- An object is in a certain **state** at any given time.
  - EmployeeNo = 1, Surname = "Hansen", Salary = 475,000
- An object can go into a new state, but it still remains "the same" object.
  - Hansen is Hansen even after she gets a pay rise.
- An object has **methods** .
  - anEmployee.changePay (newPay)
- Objects **protect** their data elements = **encapsulation**.

  All access goes via the methods.

# Objects and Classes

- A **class** is a "template" for creating objects.
  - From the class *Employee* we can generate objects for employee Per, employee Lise and employee Jens.
- The class describes **what** kind of **data** and what **methods** ("actions") the objects of the class should contain.
- During program execution, **objects** are **generated** from the classes (**instantiation**).
- All objects of a given class have the same structure.

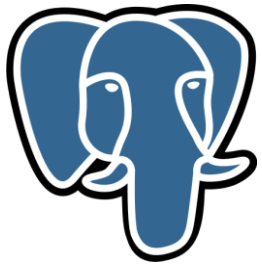
# Object relational databases

- In the 1990s, **object databases** were developed to handle complex data structures, e.g. map data.
- In **response** to this, the **relational databases** were expanded with object-oriented mechanisms:

1. Possibility to ***define*** **own data types** with associated methods, which can be used in table definitions and SQL.
2. "**Collection**" classes: Table, sequence, multiset.
3. **Subclasses** (specialization).

Such **hybrid systems** are called **object-relational databases** and had a greater commercial **impact** than the pure object databases.

# Data types in PostgreSQL



➢ Support for SQL and standard data types.

In addition:

- **Custom** data types
- Array types
- Frame types
- Interval types
- **Geometry** types
- XML and JSON data types

# 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

# Custom data types

```
CREATE TYPE AdresseType AS
(
  Gate VARCHAR(50),
  Nummer INTEGER
);
```

Can be used as data type for a column *Addresse* in a *Person* table:

| Id | Fornavn | Adresse |
|---|---|---|
| 5 | Ole | (Furulia, 28) |

# Custom data types

We use the **ROW** function to build the address "Furulia 28" from a street name and a street number:

```
INSERT INTO Student(Id, Fornavn, Adresse)
VALUES (5, 'Ole', ROW('Furulia', 28));
```

And we use **dot notation** to retrieve the components from such compound values:

```
SELECT Id, Fornavn, (Adresse).Gate
FROM Student;
```

# Arrays and frame types

An **array** is used to store a list of values of the same data type,

e.g. a list of phone numbers.

Example of column definition: `Tlf TEXT[]`

To obtain the first value: `Tlf[0]`

A **frame type** (norsk: oppramstype) is defined by listing all the values:

```
CREATE TYPE StatusType AS
   ENUM ('aktiv', 'permisjon', 'sluttet');
```

| Id | Fornavn | Adresse | Tlf | Status |
|---|---|---|---|---|
| 5 | Ole | (Furulia, 28) | {'22334455','88776655'} | aktiv |

# Geometry data types

**Geometry** data types make it possible to represent points (POINT), rectangles (BOX), circles (CIRCLE) and polygons (POLYGON). Example of column definition:

```
Sirkel CIRCLE
```

Example of use in INSERT:

```
CIRCLE(POINT(300.0, 450.5), 80)
```

**PostGIS** expands PostgreSQL with the data type **Geography** , which can be used to represent points, lines and surfaces in a coordinate system based on longitude and latitude.

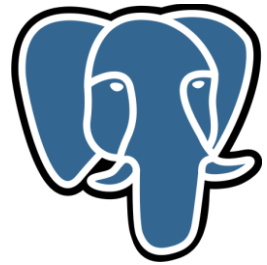
# NoSQL

**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

# Summary: *Via Objects to NoSQL*

- Extending SQL with **custom** **data types**:
  - Point, Line, polyline, polygon …
  - Operations: Distance, intersection …
  - E.g. in the standard ISO/IEC 13249 **SQL/MM**



- **Classes**, objects and methods
- **Object relational** **databases** such as PostgreSQL:
  - Array types etc …