

Schema Evolution in Relational, NoSQL and Graph Databases

Prof. Dr.-Ing. habil. Meike Klettke
Lehrstuhl für Data Engineering

FAKULTÄT FÜR INFORMATIK UND DATA SCIENCE



Universität Regensburg

Structure of this Talk

1. **Evolution of Relational Databases**

1. Design vs. Evolution
2. Capacity and Information Change of Evolution Operations
3. Single-table and Multi-table Evolution Operations
4. Schema Matching for deriving Evolution Operations
5. Evolution Pattern

2. **Evolution of NoSQL Databases**

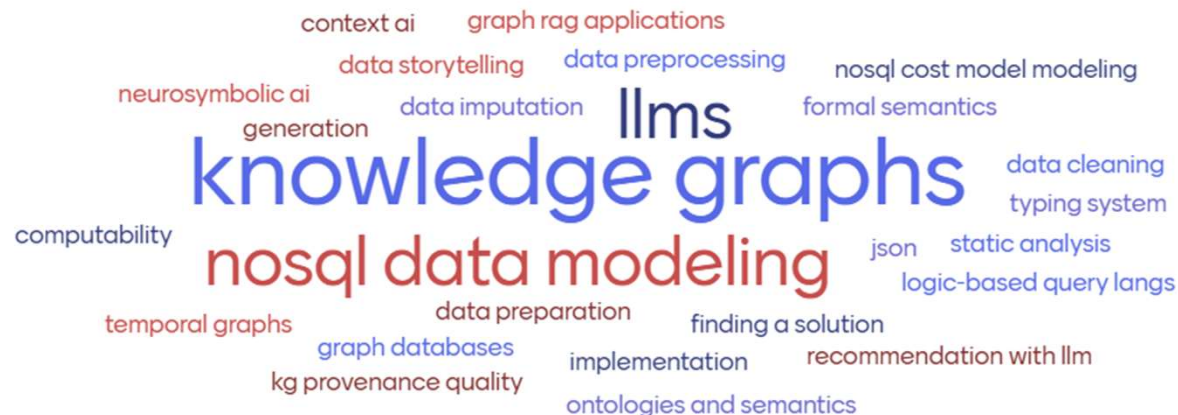
1. Heterogeneity in NoSQL Databases
2. Schemas for NoSQL Databases -- **Taming variable Non-relational Database**
3. Reverse Engineering: Schema Extraction
4. Evolution Operations for heterogeneous data

3. Using **LLMs for Data Engineering Tasks**

all parts with Take-Away Messages

Mentimeter Question 1:

What are your PhD research topics/ your current research topics?



Mentimeter Question 3: How much time do you spend for the different tasks?

Distribute 100 points across your daily scientific work



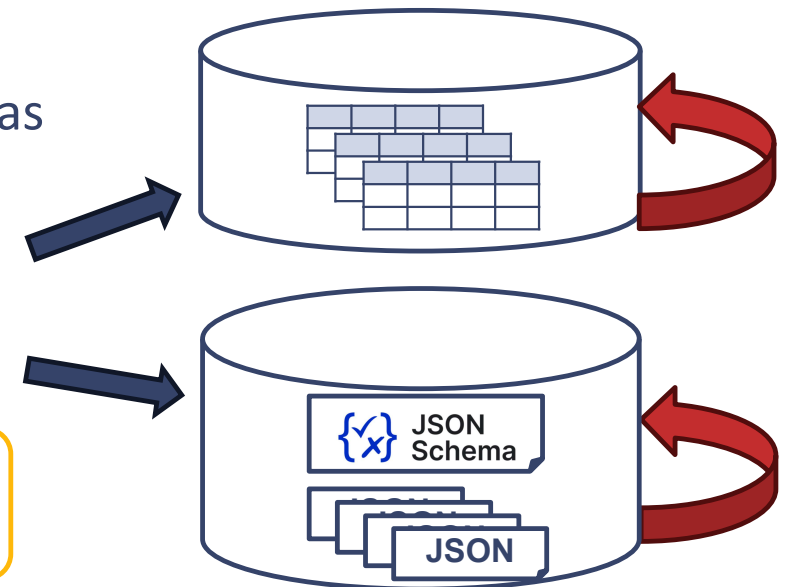
Motivation

Database evolution is the **continuous change** of data structures

- in **relational database**: ALTER TABLE statements as part of SQL/DDL standard
- in **NoSQL databases (JSON, Graph)**: evolution methods are still under development

In this talk:

Handling of **Evolution** in the different
database management systems



1. Database Evolution

"... software aging can, and will occur in all successful products"

(David Parnas: Software aging. Proceedings - International Conference on Software Engineering, 279-287, 10.1109/ICSE.1994.296790)

➡ One way to handle the software aging problem: **Evolution**

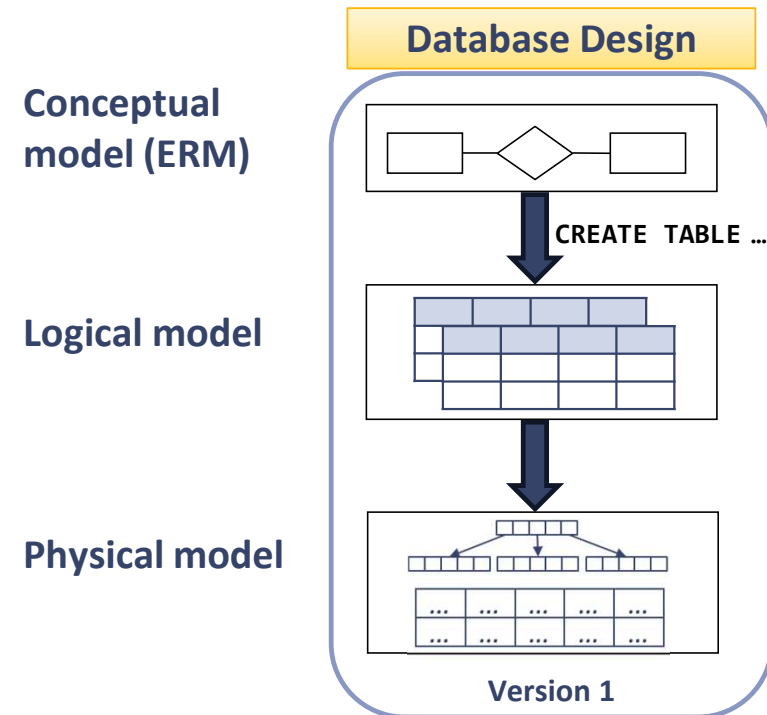
Definition of **Schema Evolution in Databases**:

Schema evolution deals with the need to **retain current data** when **database schema changes** are performed. Formally, Schema Evolution is accommodated when a database system facilitates database **schema modification without the loss of existing data ...** *(John F. Roddick)*

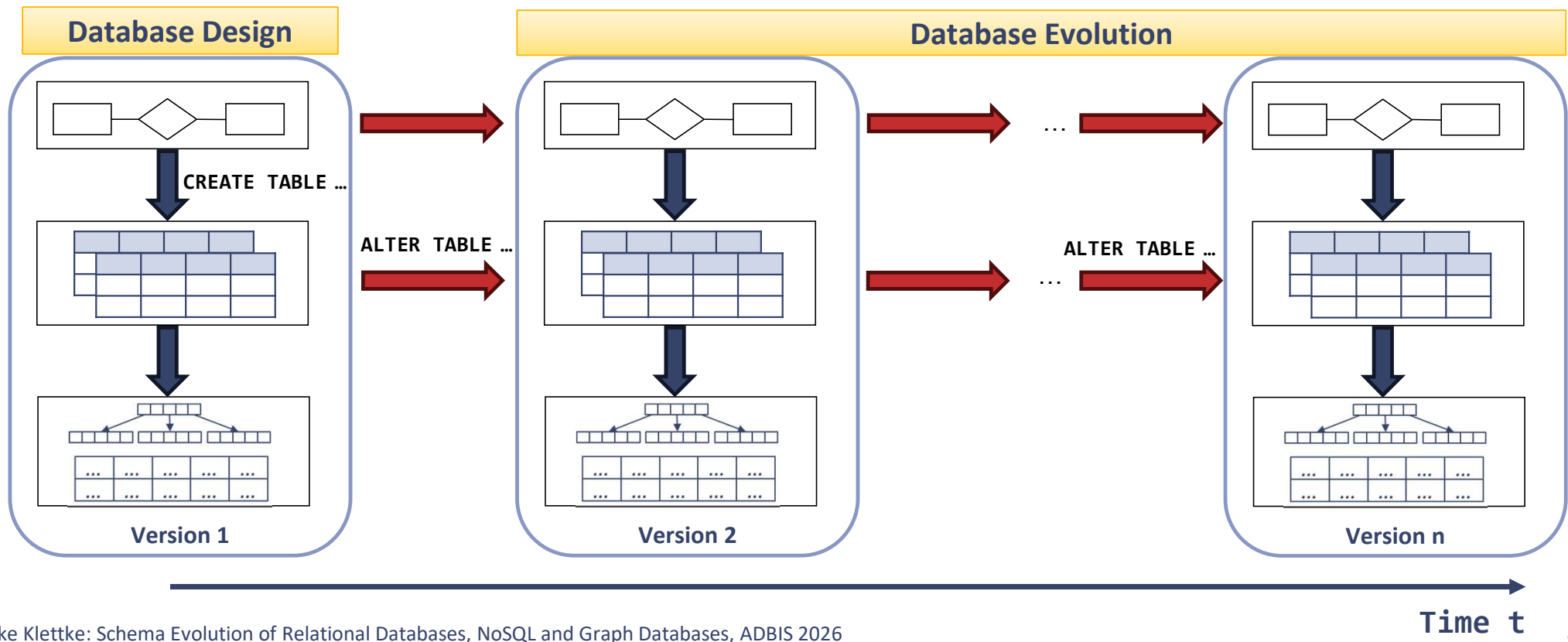
1.1. Design of Relational Databases

- Database design usually starts with a **conceptual model** (Entity-Relationship Model, UML class diagram)
- Rules to translate Entity Relationship Models into **relations**
- Internal** optimal data storage

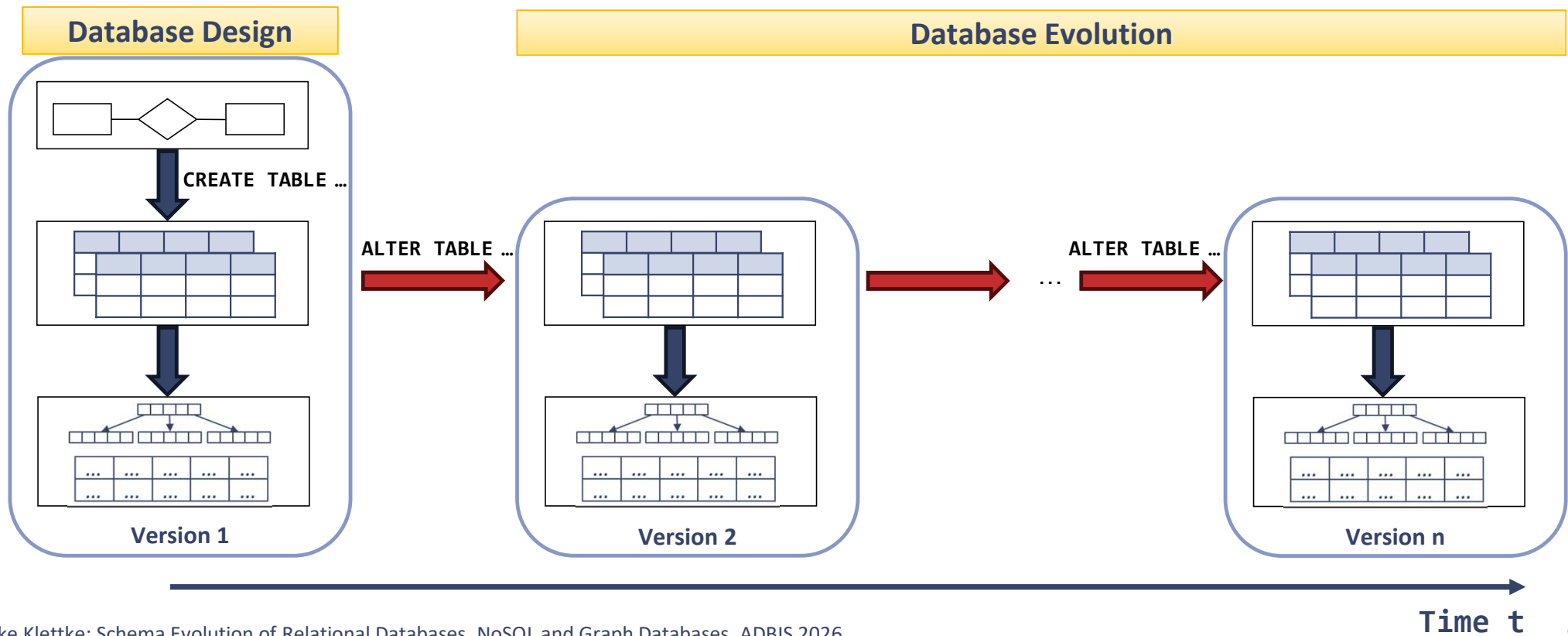
*(This process is taught in each **DB1 lecture** and is frequently **used in practice**.)*



1.1. Database Design and Database Evolution in Theory



1.1. Database Design and Database Evolution in **Practice**



1.1. Schema Evolution of Relational Database

In relational database systems:

- operations to **extend or modify relations** and **database constraints**
 - specify the **operations** on the **logical level**
 - standardized in the Data Definition Language (**SQL/DDL**) for relational databases
-  in all database management systems available in similar form

The evolution operations change

- the **schema** (also in the database catalogues)
- the **data values** accordingly



Data always fulfil the schema constraints:

- **defined structure** (attribute names, data types, not null constraints) and
- semantic constraints (key, foreign key)

1.1. Schema Evolution of Relational Database

Evolution Operations for:

- **Attributes** (add column, rename, change column definition, drop column)

```
ALTER TABLE table_name ...
    ... ADD COLUMN column_name column_definition
    ... ALTER COLUMN column_name column_definition
    ... RENAME COLUMN column_name TO new_column_name
    ... DROP COLUMN column_name
```

- **Constraints** (add or change constraints)

```
ALTER TABLE table_name ...
    ... ADD CONSTRAINT constraint_name UNIQUE(attribute(s))
    ... ADD FOREIGN KEY (attribute)
        REFERENCES table_name(attribute_name)
    ... DROP CONSTRAINT constraint_name
```

- **Tables** (rename tables)

```
ALTER TABLE table_name RENAME TO new_table_name
```

1.2. Characteristics of Evolution Operations

Evolution Operations change:

- the **capacity** of a database (ability to store data) and
- the **information content** of a database

Evolution operations that are **information preserving** guarantee

- old versions can be kept in the databases without changes
- forward compatibility

Schema Level:

capacity extending

capacity preserving

capacity reducing

capacity changing

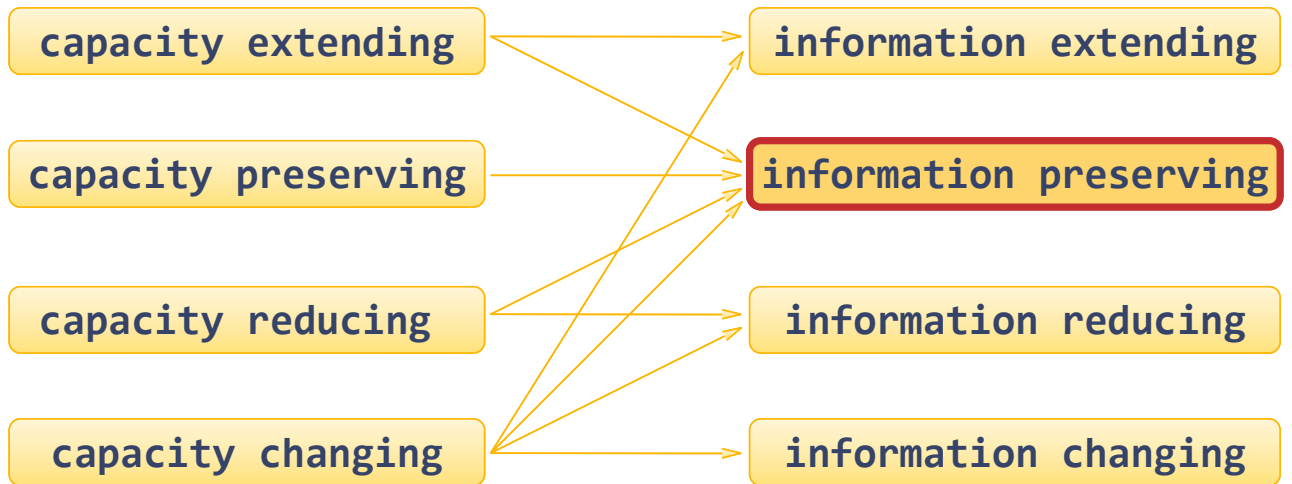
Data Level:

information extending

information preserving

information reducing

information changing



1.3. Single-table and Multi-table Evolution Operations

- Single-table Evolution Operations:
 - add
 - delete
 - rename
 - change

} part of the **SQL/DDL standard** (and available in all systems)
- Multi-table Evolution Operations:
 - split
 - merge
 - move
 - copy

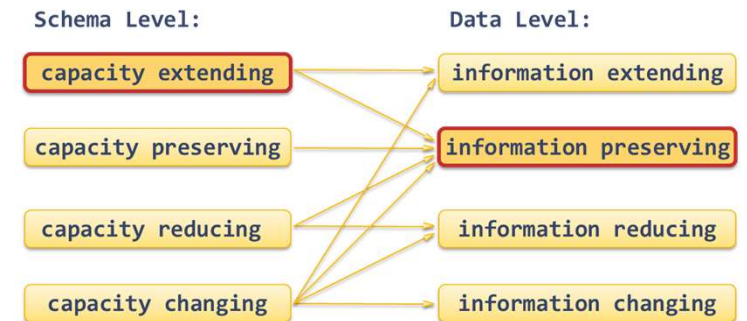
} need additional data **migration scripts** or transactions consisting of **several transformation steps**

1.3. Relational Evolution Operations

Single-table Operations

DDL evolution operations

- support **information preserving operations**
- prevents **information reducing, information changing, information extending** operations



Examples for **capacity extending/ information preserving operation**:

```
ALTER TABLE table_name ADD new_column datatype default
```

e.g.

```
ALTER TABLE T ADD E VARCHAR(10) NULL
```

T:

A	B	C	D	E
a1	b1	c1	d1	NULL
a2	b2	c2	d2	NULL

1.3. Relational Evolution Operations

Multi-table Operations

Database in Version 1:

<u>CID</u>	Name	FirstName	City	<u>ProductID</u>	Amount	ProductName	Product Price	Total Price	<u>OrderDate</u>	PaymentDate
C019	Lambert	Claire	Paris	P1001	2	Router X2000	89.99	179.98	2026-08-29	2026-08-31
C132	Dupont	Philippe	La Rochelle	P3400	10	Presenter	19,95	199.50	2026-08-30	2026-08-30
C004	Martin	Marie	Orleans	P1120	1	WiFi Access Point	59.90	59.90	2026-08-30	2026-09-01
C004	Martin	Marie	Orleans	P4001	1	Notebook	998.00	998.00	2026-08-20	NULL
C132	Dupont	Philippe	La Rochelle	P1120	1	WiFi Access Point	59.90	59.90	2026-07-20	NULL

Evolution

Operation (split):

split Orders into
 Customer(CID, Name, FirstName, City),
 Products(ProductID, ProductName, ProductPrice),
 Orders(CID, ProductID, Amount, TotalPrice, OrderDate, PaymentDate)

1.3. Relational Evolution Operations

Multi-table Operations

Step 1:

- **create table** Customers
- **insert data** (without duplicates)

```
CREATE TABLE Customers (  
    CustomerID VARCHAR(10) PRIMARY KEY,  
    Name VARCHAR(50),  
    FirstName VARCHAR(50),  
    City VARCHAR(50)  
);
```

```
INSERT INTO Customers (CustomerID, Name, FirstName, City)  
SELECT DISTINCT(CustomerID, Name, FirstName, City)  
FROM Orders;
```



<u>Customer ID</u>	Name	FirstName	City
C019	Lambert	Claire	Paris
C132	Dupont	Philippe	La Rochelle
C004	Martin	Marie	Orleans

1.3. Relational Evolution Operations

Multi-table Operations

Step 2:

- **create table** Products
- **insert data** (without duplicates)

```
CREATE TABLE Products (  
    ProductID VARCHAR(10) PRIMARY KEY,  
    ProductName VARCHAR(100),  
    ProductPrice DECIMAL(10,2)  
);
```

```
INSERT INTO Products (ProductID, ProductName, ProductPrice)  
SELECT DISTINCT(ProductID, ProductName, ProductPrice)  
FROM Orders;
```



<u>Product ID</u>	ProductName	ProductPrice
P1001	Router X2000	89.99
P3400	Presenter	19.95
P1120	WiFi Access Point	59.90
P4001	Notebook	998.00

1.3. Relational Evolution Operations

Multi-table Operations

Step 3:

- **delete attributes from table** Orders
- define new **foreign key constraints**

```
ALTER TABLE Orders DROP COLUMN Name,  
                     DROP COLUMN FirstName,  
                     DROP COLUMN City,  
                     DROP COLUMN ProductName,  
                     DROP COLUMN ProductPrice;
```



<u>Customer ID</u>	<u>Product ID</u>	Amount	Total Price	<u>OrderDate</u>	PaymentDate
C019	P1001	2	179.98	2026-08-29	2026-08-31
C132	P3400	10	199.50	2026-08-30	2026-08-30
C004	P1120	1	59.90	2026-08-30	2026-09-01
C004	P4001	1	998.00	2026-08-20	NULL
C132	P1120	1	59.90	2026-07-20	NULL

```
ALTER TABLE Orders ADD FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),  
                     ADD FOREIGN KEY (ProductID) REFERENCES Products(ProductID);
```

Advantages of Evolution Operations in Relational Databases

- Part of the **SQL/ DDL standard**  available in all relational database systems
- Database management systems **guarantee** in each evolution step **data consistency**

Limitations of Evolution Operations in Relational Databases

- Evolution operations on the **logical model** (not on the conceptual model)
- Evolution operation **only on one table at a time**
- **Workarounds** needed for
 - information extending, information reducing and information changing operations
 - **complex refactoring (involving two or more tables)**

Take-Away Message 1

Evolution is an important problem in database applications:

- **re-design database schemas** and
- **migrate all data values** (ideally **information preserving**)

... and for each Data Engineering Tasks

- **methods for redevelopment** and
- development of **incremental solutions are interesting research**

Take-Away Message 2

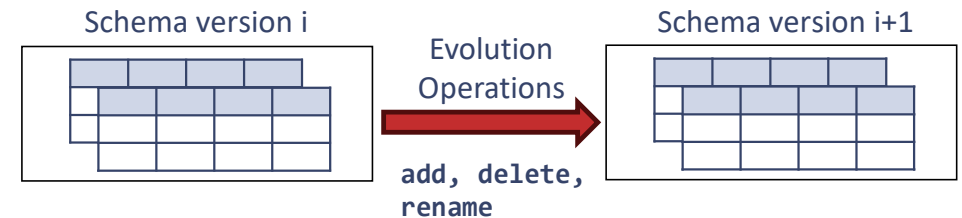
More precise:

for **your concrete PhD project:**

- beside the **design** also plan the **continious re-design**
- beside the **complete process** also plan an **incremental approach**

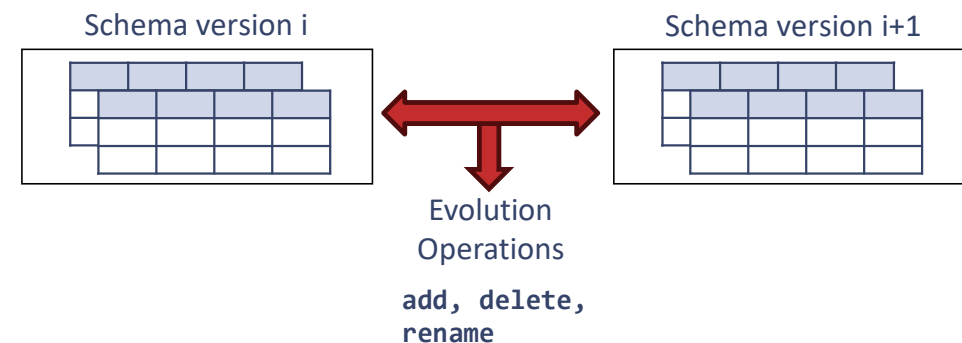
1.4. Evolution Operations vs. Schema Differences

Instead of Schema in version i and
Evolution Operations:



If two different schemas are available –
Extraction of evolution operations:

1. **matching** of identical components (with heuristics)
2. determining the **differences**
3. deriving **evolution operations**, based on heuristics



1.4. Schema Matching and Mapping in Detail

this task is one of the evergreens in Data Engineering

Step 1: Matching, based on Heuristics

Schema version i

<u>CustomerID</u>	<u>ProductID</u>	Amount	Price	TotalPrice	<u>OrderDate</u>	PaymentDate
C019	P1001	2	89.99	179.98	2026-08-29	2026-08-31
C132	P3400	10	19.95	199.50	2026-08-30	2026-08-30
C004	P1120	1	59.90	59.90	2026-08-30	2026-09-01
C004	P4001	1	998.00	998.00	2026-08-20	NULL
C132	P1120	1	59.00	59.90	2026-07-20	NULL

Schema version i+1

<u>CustomerID</u>	<u>ProductID</u>	Amount	<u>OrderDate</u>	PaymentID

<u>PaymentID</u>	Date	TotalSum	CreditcardNo	SecurityCode

1.4. Schema Matching and Mapping in Detail

this task is one of the
evergreens in Data
Engineering

Step 1: Matching,
based on Heuristics
(same data types,
similar property
names, similar data
characteristics)

Schema version i

<u>CustomerID</u>	<u>ProductID</u>	Amount	Price	TotalPrice	<u>OrderDate</u>	<u>PaymentDate</u>
C019	P1001	2	89.99	179.98	2026-08-29	2026-08-31
C132	P3400	10	19.95	199.50	2026-08-30	2026-08-30
C004	P1120	1	59.90	59.90	2026-08-30	2026-09-01
C004	P4001	1	998.00	998.00	2026-08-20	NULL
C132	P1120	1	59.00	59.90	2026-07-20	NULL

Schema version i+1

<u>CustomerID</u>	<u>ProductID</u>	Amount	<u>OrderDate</u>	<u>PaymentID</u>

<u>PaymentID</u>	Date	TotalSum	CreditcardNo	SecurityCode

1.4. Schema Matching and Mapping in Detail

this task is one of the evergreens in Data Engineering

Step 2: Identification of differences, missing and additional parts

Schema version i

<u>CustomerID</u>	<u>ProductID</u>	Amount	Price	TotalPrice	<u>OrderDate</u>	<u>PaymentDate</u>
C019	P1001	2	89.99	179.98	2026-08-29	2026-08-31
C132	P3400	10	19.95	199.50	2026-08-30	2026-08-30
C004	P1120	1	59.90	59.90	2026-08-30	2026-09-01
C004	P4001	1	998.00	998.00	2026-08-20	NULL
C132	P1120	1	59.00	59.90	2026-07-20	NULL

Schema version i+1

<u>CustomerID</u>	<u>ProductID</u>	Amount	<u>OrderDate</u>	<u>PaymentID</u>	<u>PaymentID</u>	Date	TotalSum	CreditcardNo	SecurityCode

1.4. Schema Matching and Mapping in Detail

Step 3: Deriving evolution operations, e.g.

```
delete T.Price
split T into ...
rename TotalPrice to TotalSum
add T1.PaymentID
add T2.PaymentID
add T2.CreditCardNo
add T2.SecurityCode
```

Schema version i

<u>CustomerID</u>	<u>ProductID</u>	Amount	Price	TotalPrice	<u>OrderDate</u>	PaymentDate
C019	P1001	2	89.99	179.98	2026-08-29	2026-08-31
C132	P3400	10	19.95	199.50	2026-08-30	2026-08-30
C004	P1120	1	59.90	59.90	2026-08-30	2026-09-01
C004	P4001	1	998.00	998.00	2026-08-20	NULL
C132	P1120	1	59.00	59.90	2026-07-20	NULL

<u>CustomerID</u>	<u>ProductID</u>	Amount	<u>OrderDate</u>	PaymentID

<u>PaymentID</u>	Date	TotalSum	CreditCardNo	SecurityCode

Evolution Operations vs. Schema Differences

Deriving Evolution operations between two different schema versions (i and $i+1$) needs **heuristics**:

1. **in the matching process** (to identify the identical parts of the schemas)
2. **to derive evolution operations** (based on the differences of the schemas)

Heuristics always means that

- the algorithms can fail
- (do not find the optimal evolution operations)

This method should only be applied if no evolution operations are known

Take-Away Message 3

For **your concrete PhD project**:

- is matching between two different datasets/ solutions and
- transformation between these a promising approach in your research

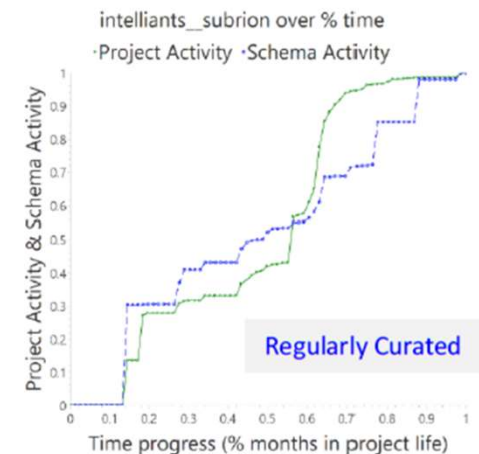
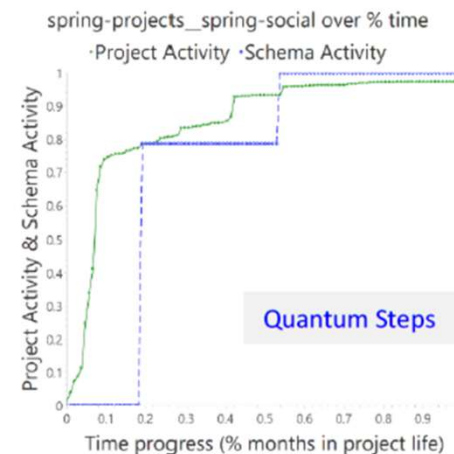
1.5. Pattern of Evolution

Vassiliadis and Karakasidis compared

- **project change activities** and
- **schema change activities**

They observed: *"Almost 25% (37/151) projects evolve with regular change steps, distributed across time, progressively climbing to the top -band over a long period of time. "*

Paper: Panos Vassiliadis, Alexandros Karakasidis:
Time-Related Patterns Of Schema Evolution. 310-323



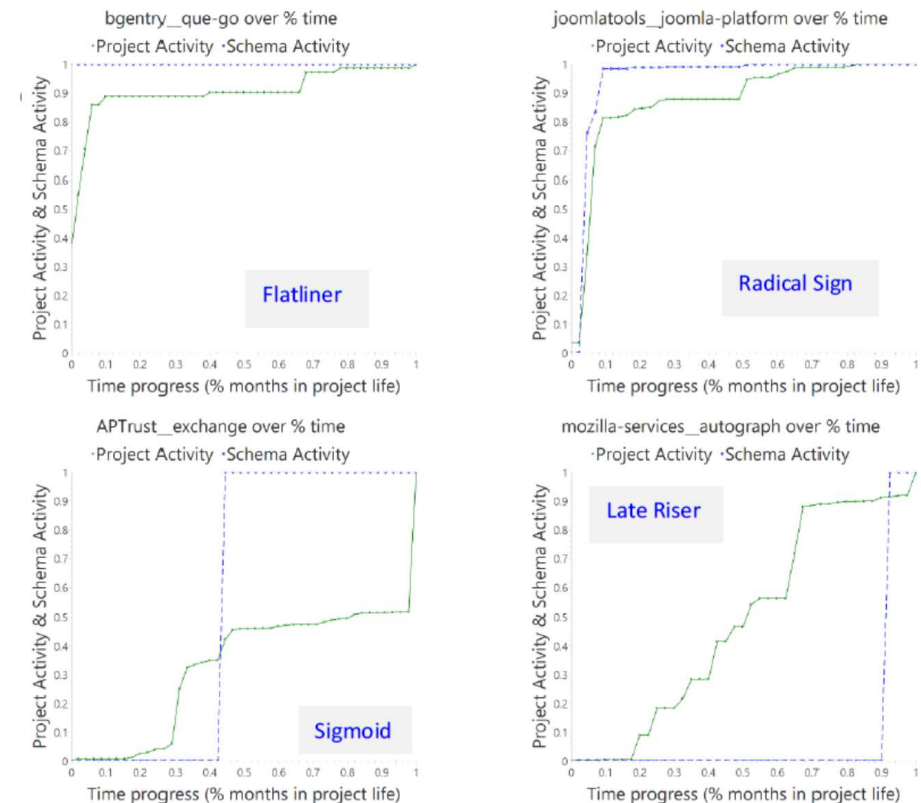
Source of the figure:

https://www.cs.uoi.gr/~pvassil/publications/2025_EDBT/EDBT_2025_TimeEvoPatterns_PresentationLong.pdf

1.5. Pattern of Evolution

But they also noticed an **aversion to change the databases schema**: *"Almost 2/3 of the corpus, 97/151 evolve with very focused change very close to the point of schema birth - the only difference of the involved patterns is when schema birth takes place"*

Paper: Panos Vassiliadis, Alexandros Karakasidis:
Time-Related Patterns Of Schema Evolution. 310-323



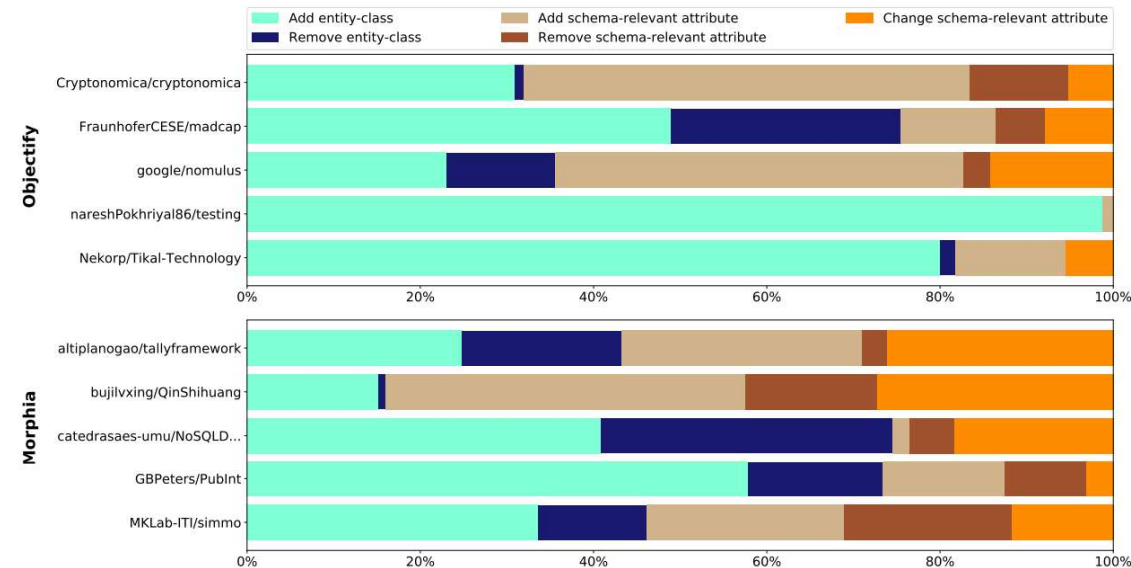
Source of the figure: https://www.cs.uoi.gr/~pvassil/publications/2025_EDBT/EDBT_2025_TimeEvoPatterns_PresentationLong.pdf

1.5. Evolution Operations in Real-world Projects

- Scherzinger et. al. made several studies on **distribution of evolution operations** for projects with Object-NoSQL mappers (to map Java objects to entities)
- two mappers are compared:
 - Objectify
 - Morphia
- Majority of operations are:
 - add entity class** and
 - add attribute**

Source: Stefanie Scherzinger, Sebastian Sidortschuck: *An Empirical Study on the Design and Evolution of NoSQL Database Schemas*, ER 2020

Meike Klettke: Schema Evolution of Relational Databases, NoSQL and Graph Database



(a) By project.

	Entity-class		Schema-relevant attribute		
	add	remove	add	remove	change
Objectify	56.3%	8.4%	24.7%	4.0%	6.5%
Morphia	34.5%	16.2%	21.6%	10.4%	17.2%
Overall	34.8%	15.1%	27.3%	7.4%	15.4 %

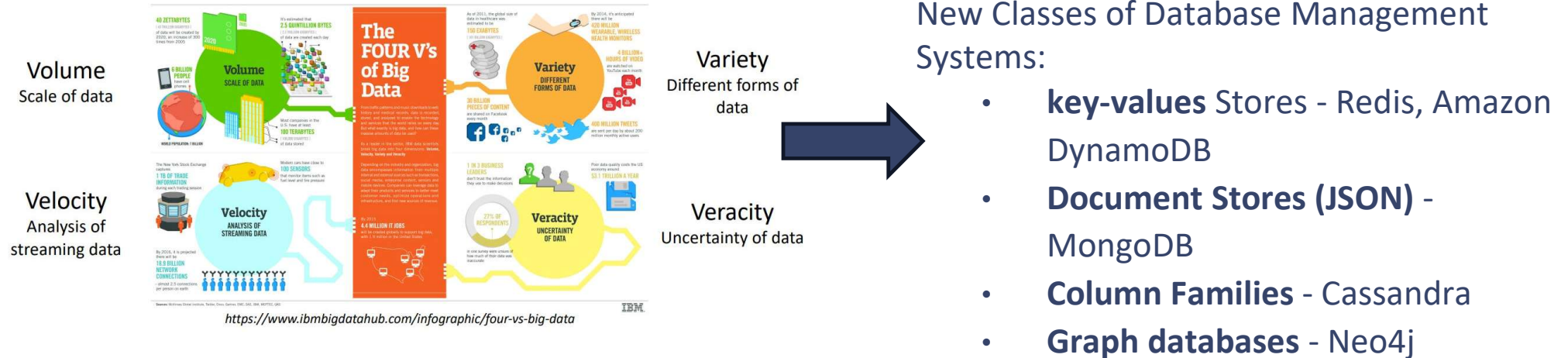
(b) Objectify-based vs. Morphia-based projects.

Take-Away Message 4

The professional community is very interested in **well-conducted empirical studies**, even though it is not so easy to get them published.

2. Big Data Movement as Driver for the Development of NoSQL databases

The **lack of Variety** was one of the drivers for the development of **NoSQL databases** ...



2. Main Characteristics of NoSQL databases

1. Non-relational data model
2. Schema less or flexible (optional) schema
3. horizontal scalable



NoSQL databases solve the "**Variability Requirement**" in a very simple way: **no schema constraints at all**

Advantages:

- everything can be stored
- evolution not necessary
- **different schema versions** in the same databases

Disadvantages:

- problems to use the (heterogeneous) data
- **no schema constraints (no error detection)**

2.1. Heterogeneity in NoSQL Database

- .. but first, we start with some **unwanted side-effects** of **schema-less systems**
- e.g. **different property names**

```
{ "id": 7,
  "name": "Dupont",
  "firstname": "Juliette",
  "address": {
    "city": "Orleans",
    ...
  }
```

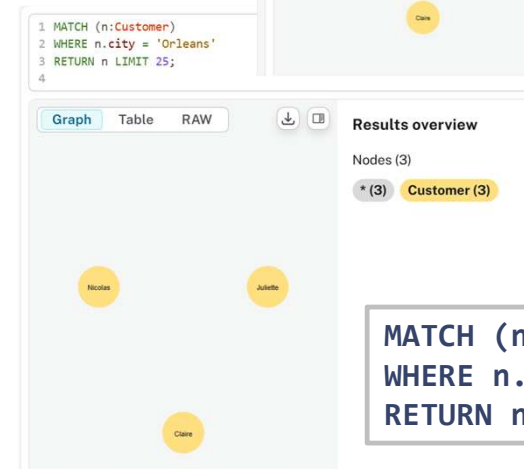
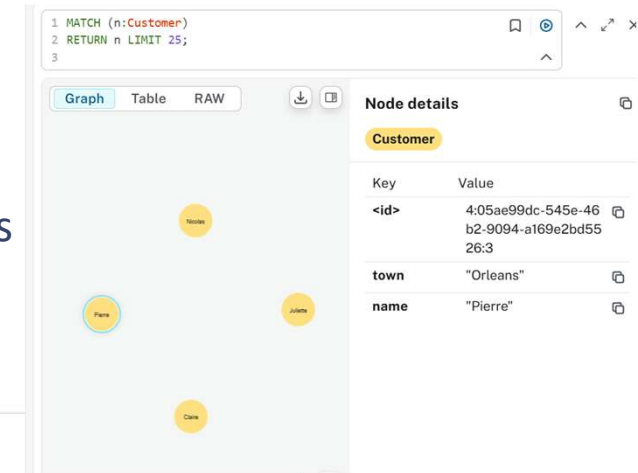
```
{ "id": 9,
  "name": "Simon",
  "firstname": "Pierre",
  "address": {
    "town": "Orleans",
    ...
  }
```

Sample query
in MongoDB:

```
db.Customer.find(
  { city: "Orleans" },
  { name: 1, firstname: 1 }
)
```

These documents are not detected in **queries** (incomplete **selection**, **join** (dangling tuples), **grouping**).

Data Nodes
in Neo4j:



Sample query
in Neo4j:

```
MATCH (n:Customer)
WHERE n.city = 'Orleans'
RETURN n LIMIT 25;
```

2.2. Data models and their schemas

Data models for structured data:

Data Model	Schema
Relational model, since 1970	Since 1970

Data models for semi-structured data:

Data Model	Schema
XML, 1998	DTD 1998 XML Schema, 2001
JSON as data model, ~2009 (MongoDB)	JSON schema 2013 JDT 2026
Graph data, ~2007 (Neo4j)	Currently not standardized

Semi-structured data models:

- started **schema-less**
- **added schema languages** some years later
- database management systems usually allow **optional** and **partial schema control**

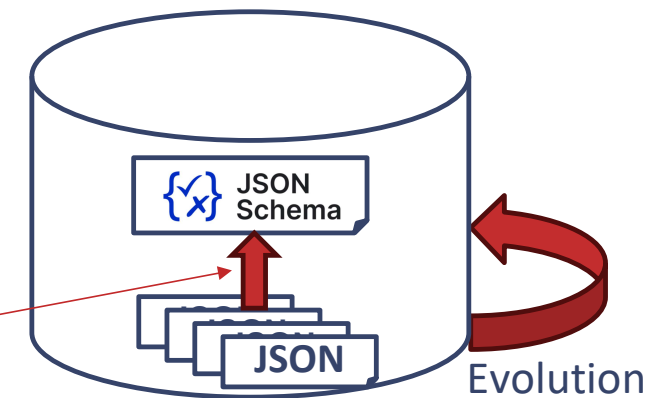
Evolution of NoSQL Databases (JSON, Graph)

- **Data** and their **implicit structures** are also **changing** in schema-less databases
- **Evolution operations are needed** to
 - evolve the schema (if available)
 - adapting the data (data migration)

At least simple evolution operations are needed:

- **insert, update, delete***

If schema not available  **reverse engineering –
schema extraction**



*same operations: create, update, delete as in: Maíra Marques, Jocelyn Simmonds, Pedro O. Rossel, María Cecilia Bastarrica: Software product line evolution: A systematic literature review, Information and Software Technology, Volume 105, 2019

2.2. Schema Languages for JSON

JSON Schema

- Schema definition for JSON Documents
- in **JSON Syntax**
- contains
 - property names,
 - datatypes,
 - constraints
 - nesting of objects

```
{ "$schema": "https://json-schema.org/draft/2020-12/schema",  
  "type": "object",  
  "properties": {  
    "id": {"type": "integer"},  
    "name": {"type": "string"},  
    "firstname": {"type": "string"},  
    "address": {  
      "type": "object",  
      "properties": {  
        "city": {"type": "string"},  
        "ZIP": {"type": "integer"},  
        "street": {"type": "string"}}},  
    "email": {"type": "string"}},  
  "required": ["id", "name", "address"],  
  "additionalProperties": false  
}
```

2.2. Schema Languages for JSON

JSON Schema

JSON schema allows:

- **complete schema**
(structured relational-like datasets)
- definition of **variants**
- **heterogeneity**
- **extensibility** (definition of a partial schema that can be extended)

```
"required": ["id", "name", "address"],  
"additionalProperties": false,
```

```
"oneOf" : [{ "required" : ["city"]},  
            { "required" : ["state"]}],
```

```
"required": ["id", "name"],
```

```
"additionalProperties": true,
```

2.2. Schema Languages for JSON

JDT (will be published next week at the ER conference in St. Johns, Canada)

- Schema definition for JSON Documents
- in **Pythonic Style**
- contains
 - property names,
 - datatypes,
 - constraints
 - nesting of objects
- *(but a much simpler syntax than JSON schema)*

```
is type object
user:
  is closed
  id is number, required, not null
  name is required, string
  firstname is string
  address:
    is closed
    city is string
    zip is number
    street is string
  email is string
```

2.2. Schema Languages for JSON

JDT (will be published next week at the ER conference in St. Johns, Canada)

- Creating Schemas: Editor (with autocomplete function)
- Validating JSON documents against a JDT definition
- Ecosystem (Schema Extraction, Evolution) are planned in the next steps



Website: go.ur.de/jdt

JDT Schema Validator

Validate JSON files against JDT schemas

JDT Schema

[Browse](#)

```
is type object
user:
  is closed
  id is number, required, not null
  name is required, not boolean
  age is optional, (number or null)
  hobbies is array(string), required
  eyecolor is "blue" or "green" or "brown"
```

JSON File

[Browse](#)

```
{
  "user": {
    "id": 1,
    "name": "Jane Doe",
    "age": 28,
    "hobbies": [
      "reading",
      "coding",
      "hiking"
    ],
    "eyecolor": "blue"
  }
}
```

[Load Example](#)

[Clear](#)

[Validate](#)

2.3. Schema Extraction

Schema Extraction from JSON databases
(= reverse engineering, data profiling)

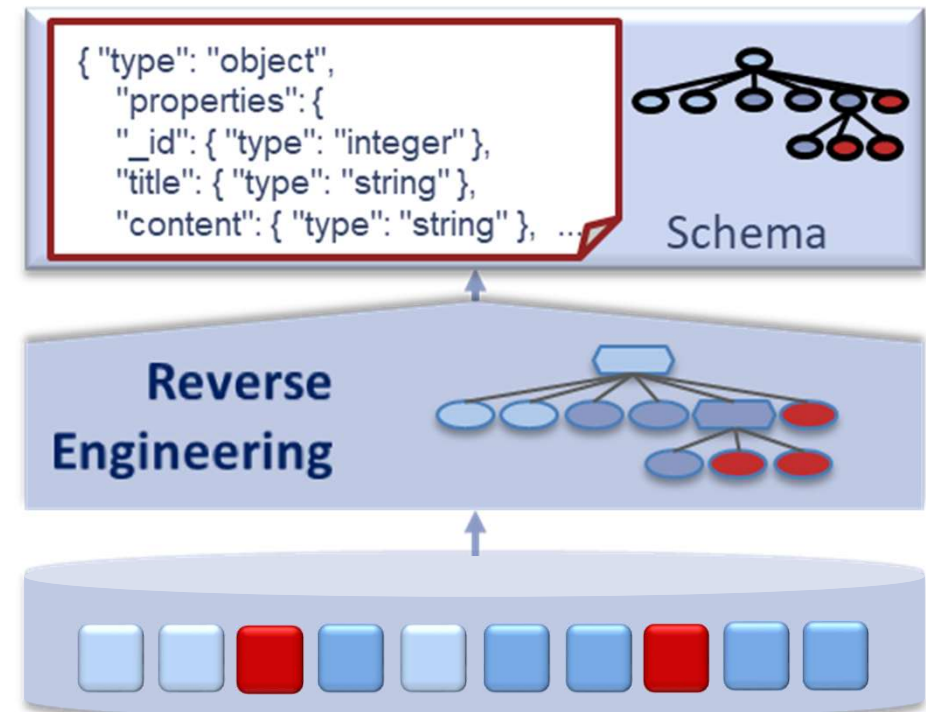
- NoSQL database **without explicit schema**
- schema extraction is the first task to **understand** NoSQL databases

Algorithm for Extracting the Schema

- **Input:** Set of JSON documents
- **Method:** Using an internal **Graph Structure** (*Structure Identification Graph (SG)*) to represents the schema
- **Output:** JSON schema

suggested in:

Meike Klettke, Uta Störl, Stefanie Scherzinger: *Schema Extraction and Structural Outlier Detection for NoSQL Data Stores*. BTW 2015

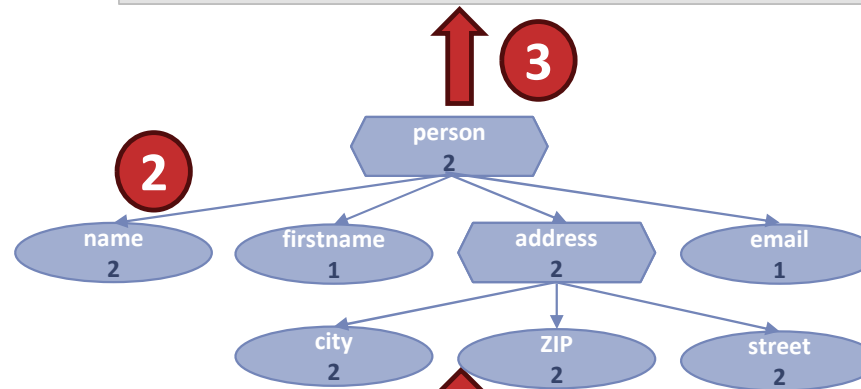


2.3. Schema Extraction Example

Substeps:

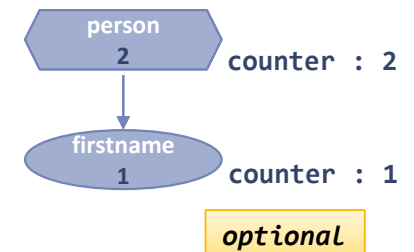
- 1 - parsing all JSON documents
- extracting the **implicit structure** (property names, data types)
- 2 - storing a **counter** in a graph
- 3 - generate the JSON schema from the graph
 - with **property names, data types, nesting** and
 - **required and optional elements**

```
{ "$schema": "https://json-schema.org/...",
  "type": "object",
  "properties": {
    "name": { "type": "string"},
    "firstname": { "type": "string"},
    "address": { "type": "object",
      "properties": { ... } }, },
  "required": ["name", "address" ],
  "additionalProperties": false }
```

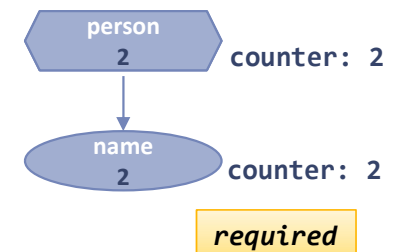


```
{ "id": 7,
  "name": "Garcia",
  "firstname": "Claire",
  "address": {
    "city": "Orleans",
    ...
  }
}
```

Optional properties:



Required properties:

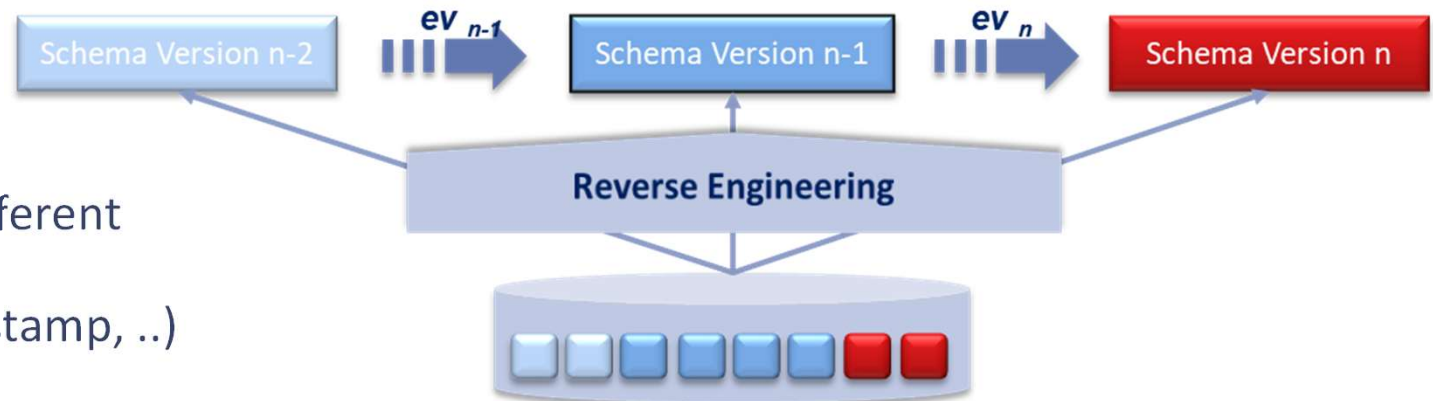


2.3. Schema Version Extraction

Next level:

Usually, **NoSQL database**

- have evolved over time
- can contain datasets in different structural versions
- timeline is available (timestamp, ..)



Aim: extracting

- **series of schema versions** and
- changes over time, specified through **evolution operations**

M. Klettke, H. Awolin, U. Störl, D. Müller, S. Scherzinger: *Uncovering the evolution history of data lakes*. SCDM@IEEE Big Data Conf., 2017
Uta Störl, Meike Klettke: *Darwin: A Data Platform for Schema Evolution Management and Data Migration*, DataPlat@EDBT 2022

2.3. Schema Version Extraction

Algorithm for Extracting Schema Versions

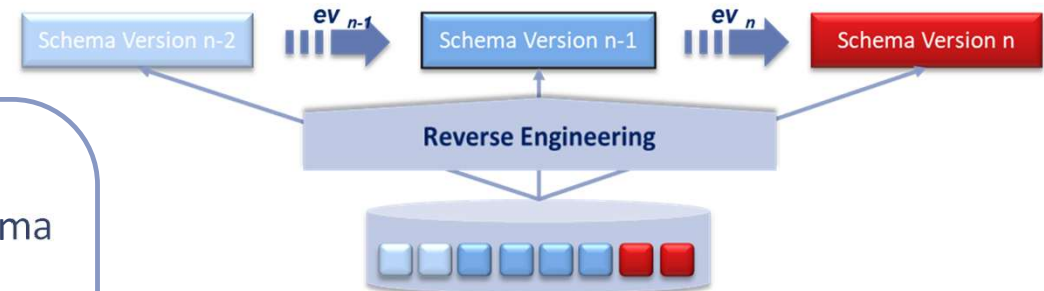
Input: Set of JSON documents (with timestamps)

Method:

- Create a Graph structure that represents the schema
- Detecting **structural changes** between two consecutive datasets
- In case of such structural changes:
 - **new schema version** is generated
 - **evolution operations** are extracted

Output:

- Series of JSON schemas and
- evolution operations



Determining of Evolution operations is not trivial: e.g. between two versions:

- **rename** or a combination of **delete** + **add**?
- **add** or **copy**?
- **move** or a combination of **delete** + **add**?

M. Klettke, H. Awolin, U. Störl, D. Müller, S. Scherzinger: *Uncovering the evolution history of data lakes*. SCDM@IEEE Big Data Conf., 2017
Uta Störl, Meike Klettke: Darwin: A Data Platform for Schema Evolution Management and Data Migration, *DataPlat@EDBT 2022*

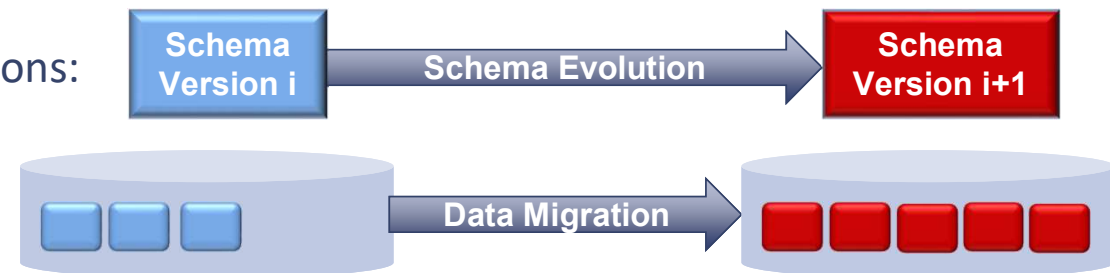
2.4. Evolution of NoSQL Databases (JSON, Graph)

No standardized NoSQL Evolution language available yet.
We still need the "**ALTER TABLE ..**" for **NoSQL**

Some research papers suggest evolution operations:

- **Single-type operations:**
 - **add** property
 - **rename** property
 - **delete** property
- **Multi-type operations:**
 - **copy** property (denormalization)
 - **move** property (refactoring)
 - **split/merge** (normalization/denormalization)

In our ongoing work: with **filter operations** to **consider variants**



Implementation:

- Specifying changes on **schema level** (**Evolution Operations**)
- **Data Migration** accordingly (**eager, lazy, predictive**)

Stefanie Scherzinger, Meike Klettke, and Uta Störl: Managing Schema Evolution in NoSQL Data Store, DBPL@VLDB, Italy, 2013

Andrea Hillenbrand, Maksym Levchenko, Uta Störl, Stefanie Scherzinger, Meike Klettke: MigCast: Putting a Price Tag on Data Model Evolution in NoSQL Data Stores. SIGMOD Conference 2019, Demo Program

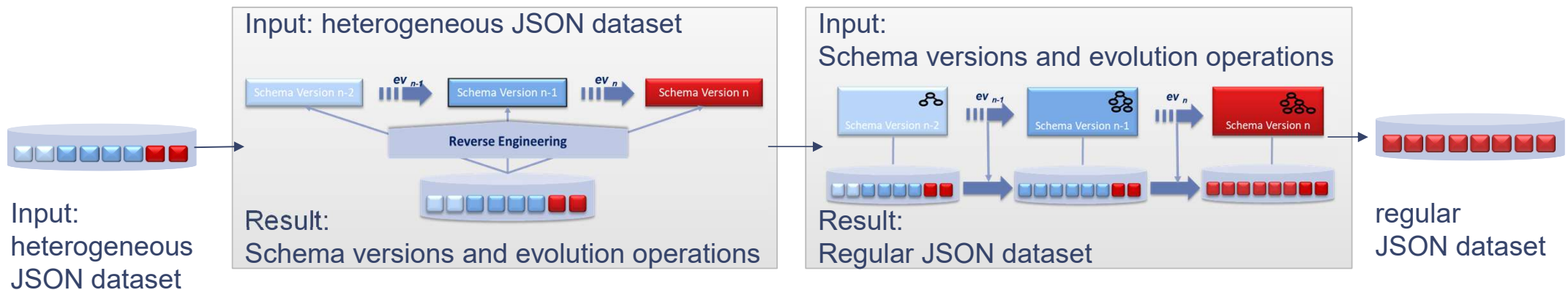
2.4. Combining Schema Version Extraction and Evolution

Step 1: Reverse Engineering

- **Schema Version Extraction** and
- Derivation of **Evolution Operations**

Step 2: Data Migration

- Generating data migration operations from the evolution operations
- **migration of all datasets** into the latest version



Take-Away Message 5

If you are publishing something, take **all relevant keywords (potential search terms)** in the **title!**

People are not scanning all publications of the database conferences, they are searching for publications (google scholar, dblp)

Schema Management and Evolution for Graph Databases

In our ongoing work

- definition of an **evolution language for graph databases**
- realizing **evolution** and **data transformation** for graph data
- building a **user interface** for supporting these processes

All information are provided here:

- Dominique Hausler: Nautilus - Evolution Tool for Graph Databases - Universität Regensburg
- <https://www.uni-regensburg.de/informatik-data-science/fakultaet/einrichtungen/data-engineering/forschung/nautilus>

Evolution Operation(s) [?]

GEO: **add** **node** **with label** PlaceOfWork

Operation Type With
add node label PlaceOfWork

GEO: **rename** **label** Schema

Operation Type
rename label

Rename to

SchemaEvolution

Remove

Add new form

Submit

Reset

Schema Evolution
Form

SDS – Structural
Database Statistics

Schema Statistics
Visualization

DataEngineering

Count: 1
Count: 1

Initial schema of entity type

Property key(s): name (STRING)
Relationship(s): MAIN_TOPIC,7,in

Latest schema of entity type

Property key(s): name (STRING)
Relationship(s): MAIN_TOPIC,7,in

Label or Type

Data **before**
evolution

Data **after**
evolution

DataEngineering

Data

Delete/ Rename

Add

Generator Inactive Manager MultiModel MultiModelGroup Person PlaceOfWork

Name of Entities

Number of labels shown: 11

Get initial db statistics

Take-Away Message 6

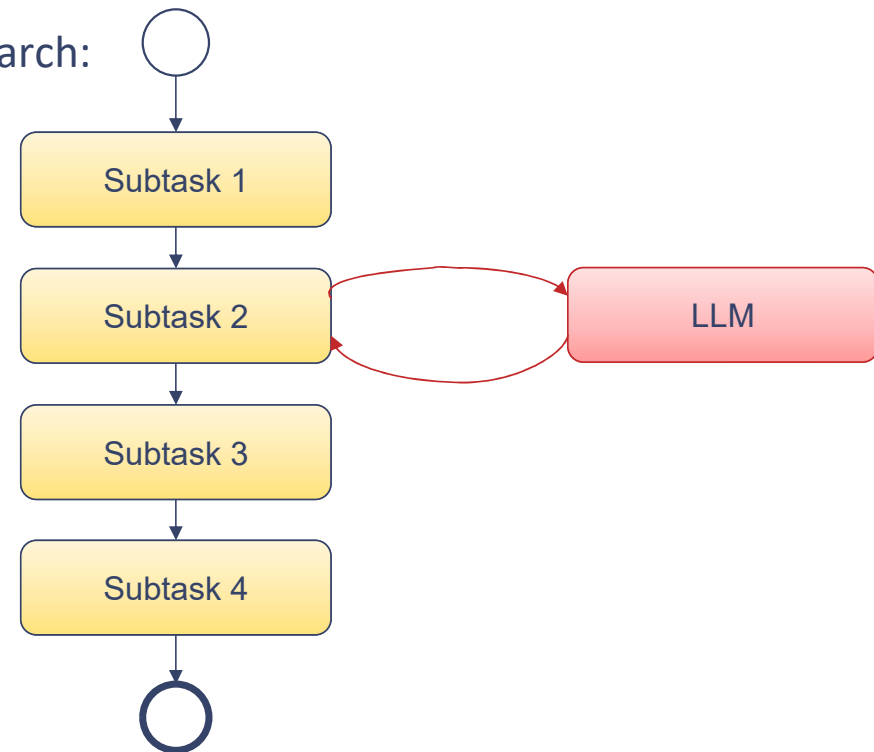
Data heterogeneity has to be considered in many applications and approaches. Usually, the data processing becomes much more complicated – **new scientific challenges** arise.

Therefore, an at least **partial schema control** is recommended.

3. LLMs in Data Engineering Research (1/4)

It seems to be a trend in the Data Engineering Research:

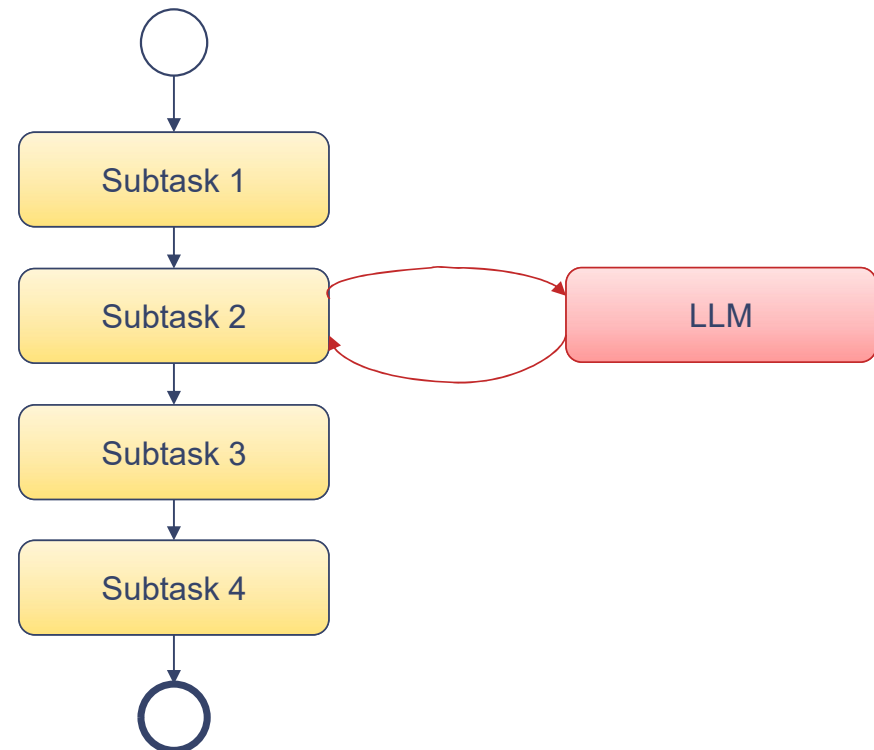
- identify a complicated problem. e.g.
 - data integration, matching,
 - data cleaning,
 - query generation,
 - optimization, ...
- solve it through LLM



3. LLMs in Data Engineering Research (2/4)

Scientific work for LLM usage

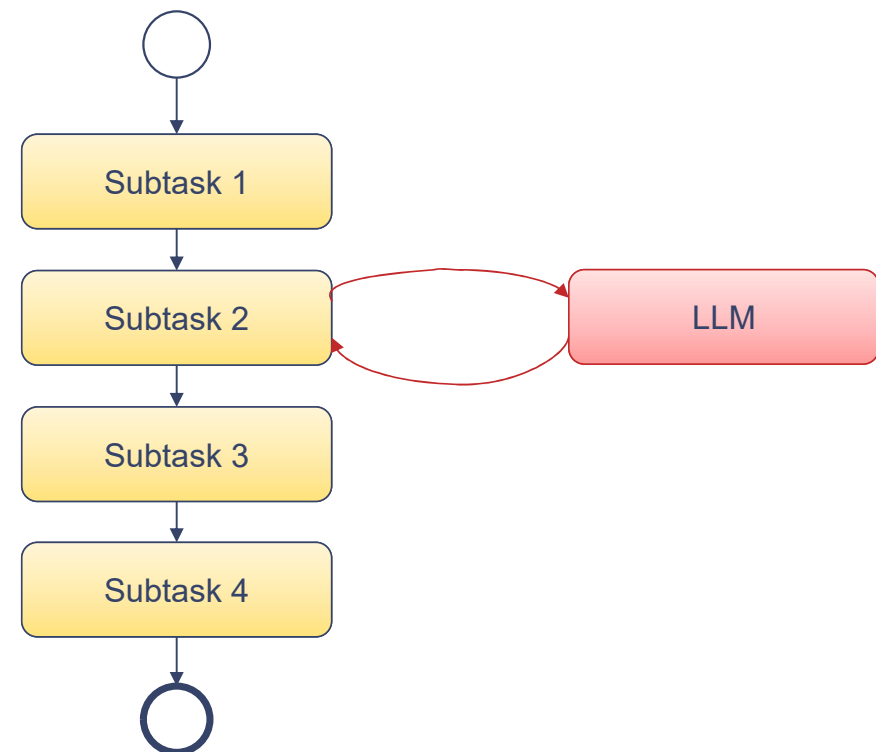
- identify how prompting should be designed
- determine which additional information must be included
- compare results of different LLMs
- prove the results of an LLM with other methods
- estimate the costs
- ... (what else?)
- ...



3. LLMs in Data Engineering Research (3/4)

Problems:

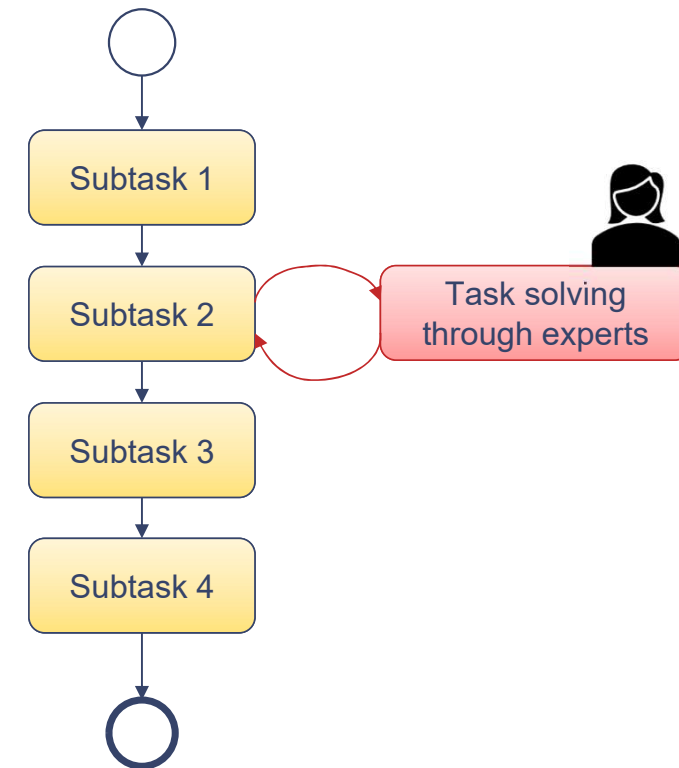
- not reproducible
- results can change, esp. with new versions
- costs can change
- LLMs are included in the operative process (outsourcing of relevant and repeating tasks)



3. Experts in Data Engineering Research

Imagine the same problem as **outsourcing in general**

Scientific work for LLM usage	.. and expert involvement
identify how prompting should be designed	identify how the contract with the expert must be defined
determine which additional information must be included	determine which additional information the expert needs
compare results of different LLMs	compare results of diff. experts
prove the results of an LLM with other methods	prove the results of the experts with other methods
estimate the costs	estimate the costs
... (what else?)	
...	



Take-Away Message 7

- Develop something that is so important that it is **still known and valid in 10 years**
- maybe so important that **not only other scientists**, but also **GenAI** can learn from it

Conclusion

- **Evolution** in databases is an important task
 - in relational databases: **ALTER TABLE** statements or workarounds
 - in NoSQL databases (JSON, Graph): languages and tools are under development
 - the separation between schema and data as well as the data heterogeneity cause new requirements
 - algorithms for
 - **schema extraction**
 - **schema version extraction**
 - **evolution** of heterogeneous data as well as
 - **preview** of the evolution effect are helpful
- } **ecosystem for non-relational databases**

Mentimeter Question 3:

What part of your scientific work do you enjoy the most?



Take-Away Message 8

**Please write surveys OR
a textbook OR
detailed state-of-the-art-chapters in your PhD!**

Don't let LLM summarize the knowledge in our field.

References (1/5)

Software Aging and Evolution:

- David Parnas: *Software aging*, Proceedings - International Conference on Software Engineering, 279-287. 10.1109/ICSE.1994.296790, 1994
- Maíra Marques, Jocelyn Simmonds, Pedro O. Rossel, María Cecilia Bastarrica: *Software product line evolution: A systematic literature review*, Information and Software Technology, Volume 105, 2019

Evolution of Relational Databases:

- Carlo Curino, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo: *Automating the database schema evolution process*. VLDB J. 22, 1 (2013), pp. 73–98, 2013
- Carlo Curino, Hyun Jin Moon, Carlo Zaniolo: *Graceful database schema evolution: the PRISM workbench*. Proc. VLDB Endow. 1, 1 (August 2008), 761–772. <https://doi.org/10.14778/1453856.1453939>, 2008
- Renee J. Miller, Yannis E. Ioannidis, Raghu Ramakrishnan: *The Use of Information Capacity in Schema Integration and Translation*, VLDB, pp. 120–133, 1993
- Panos Vassiliadis, Alexandros Karakasidis: *Time-Related Patterns Of Schema Evolution*. In EDBT, OpenProceedings.org, 310–323, 2025

Views in Databases:

- Marc H. Scholl, Christian Laasch, Markus Tresch: *Updatable views in object-oriented databases*, International Conference on Deductive and Object-Oriented Databases. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991
- Alon Y. Halevy: *Answering queries using views: A survey*, The VLDB Journal 10.4 (2001): 270-294
- Anthony L. Furtado, Marco A. Casanova: *Updating relational views*. Query Processing in Database Systems (1985): 127-142

References (2/5)

NoSQL Databases:

- Harsha Vyawahare, P. Karde, V. Thakare: *A Hybrid Database Approach Using Graph and Relational Database*, in: RICE (2018)
- Lena Wiese: *Advanced Data Management: For SQL, NoSQL, Cloud And Distributed Databases* (De Gruyter Textbook), 2015
- MongoDB: www.mongodb.com
- Cassandra: <https://cassandra.apache.org>
- Neo4j: <https://neo4j.com>

Schemas for NoSQL Databases:

- Angela Bonifati, *The Quest for Schemas in Graph Databases (Keynote)*, in: DOLAP, volume 3369 of CEUR Workshop Proceedings, CEUR-WS.org (2023)
- Angela Bonifati, Peter Furniss, Alastair Green, Russ Harmer, Eugenia Oshurko, Hannes Voigt. 2019: *Schema Validation and Evolution for Graph Databases*, In ER (Lecture Notes in Computer Science, Vol. 11788). Springer, 448–456
- Benedikt Sennebogen, Antoon Bronselaer, Toon Boeckling, Uta Störl, Meike Klettke: *Simpler is Better: JDT as a New Schema Language for JSON Documents*, Demp paper, ER conference 2026

References (3/5)

Schema Extraction for NoSQL Databases:

- Meike Klettke, Uta Störl, Stefanie Scherzinger: *Schema Extraction and Structural Outlier Detection for NoSQL Data Stores*. BTW 2015
- Meike Klettke, Hannes Awolin, Uta Störl, Daniel Müller, Stefanie Scherzinger: *Uncovering the evolution history of data lakes*. SCDM@IEEE Big Data Conf., 2017
- Uta Störl, Meike Klettke: *Darwin: A Data Platform for Schema Evolution Management and Data Migration*, DataPlat@EDBT 2022
- Diego Sevilla Ruiz, Severino Feliciano Morales, Jesús García Molina: *Inferring versioned schemas from NoSQL databases and its applications*, International conference on Conceptual Modeling. Cham: Springer International Publishing, 2015
- Enrico Gallinucci, Matteo Golfarelli, Stefano Rizzi: *Schema profiling of document-oriented databases*, *Information Systems* 75 (2018): 13-25.
- Ziyi Zhang and Teng Lv: *JSVE: JSON Schema Variant Extraction Based on Clustering and Formal Concept Analysis*, *IEEE Access*, vol. 13, pp. 145517-145527, 2025

References (4/5)

Evolution of NoSQL Databases:

- Stefanie Scherzinger, Meike Klettke, and Uta Störl: *Managing Schema Evolution in NoSQL Data Store*, DBPL@VLDB, Italy, 2013
- Marek Polák, Martin Chytil, Karel Jakubec, Vladimir Kudelas, Peter Piják, Martin Necaský, Irena Holubová: *Data and Query Adaptation Using DaemonX*, Comput. Informatics 34 (2015)
- Meike Klettke, Uta Störl, Manuel Shenavai, Stefanie Scherzinger: *NoSQL schema evolution and big data migration at scale*, in: IEEE BigData, IEEE Computer Society (2016)
- H. Vyawahare, P. Karde, V. Thakare: *A Hybrid Database Approach Using Graph and Relational Database*, in: RICE (2018)
- Mark L. Möller, Meike Klettke, Andrea Hillenbrand, Uta Störl: *Query Rewriting for Continuously Evolving NoSQL Databases*, in: ER, volume 11788 of LNCS, Springer (2019)
- Stefanie Scherzinger, Sebastian Sidortschuck: *An Empirical Study on the Design and Evolution of NoSQL Database Schemas*, ER 2020
- Angela Bonifati, Peter Furniss, Alastair Green, Russ Harmer, Eugenia Oshurko, Hannes Voigt: *Schema Validation and Evolution for Graph Databases*. In ER (Lecture Notes in Computer Science, Vol. 11788). Springer, 448–456, 2019
- Alberto Hernández Chillón, Meike Klettke, Diego Sevilla Ruiz, and Jesús García-Molina: *A Generic Schema Evolution Approach for NoSQL and Relational Databases*. IEEE Trans. Knowledge and Data Eng. 36, 7 (2024), 2774–2789, 2024
- Alberto H. Chillón, Diego S. Ruiz, Jesus Garcia Molina: *Towards a Taxonomy of Schema Changes for NoSQL Databases: The Orion Language*, in: ER, volume 13011 of LNCS, Springer, 2021
- Uta Störl and Meike Klettke: *Darwin: A Data Platform for Schema Evolution Management and Data Migration*. In EDBT/ICDT Workshops (CEUR Workshop Proceedings, Vol. 3135). CEUR-WS.org, 2022

References (5/5)

Evolution of NoSQL Databases, cont.:

- Uta Störl and Meike Klettke. 2022. Darwin: *A Data Platform for Schema Evolution Management and Data Migration*. In EDBT/ICDT Workshops (CEUR Workshop Proceedings, Vol. 3135). CEUR-WS.org
- Pablo Suárez-Otero, Michael J. Mior, María José Suárez Cabal, and Javier Tuya: *CoDEvo: Column family database evolution using model transformations*. J. Syst. Softw. 203 (2023), 111743, 2023
- Pavel Koupil, J. Bártík, Irena Holubová, *MM-evocat: A Tool for Modelling and Evolution Management of Multi-Model Data*, in: CIKM, ACM (2022)
- Andrea Hillenbrand, Maksym Levchenko, Uta Störl, Stefanie Scherzinger, Meike Klettke: *MigCast: Putting a Price Tag on Data Model Evolution in NoSQL Data Stores*. SIGMOD Conference 2019, Demo Program
- Pablo Suárez-Otero, Michael J. Mior, María José Suárez-Cabal, Javier Tuya, *CoDEvo: Column family database evolution using model transformations*, J. Syst. Softw. 203, 2023
- Andrea Hillenbrand, Stefanie Scherzinger, Uta Störl: *Remaining in Control of the Impact of Schema Evolution in NoSQL Databases*. In: Ghose, A., Horkoff, J., Silva Souza, V.E., Parsons, J., Evermann, J. (eds) Conceptual Modeling. ER 2021. Lecture Notes in Computer Science(), vol 13011. Springer, Cham. https://doi.org/10.1007/978-3-030-89022-3_13, 2021
- Dominique Hausler, Meike Klettke: *Nautilus: Implementation of an Evolution Approach for Graph Databases*. MoDELS (Companion) 2024
- Dominique Hausler: *Estimation, Impact and Visualization of Schema Evolution in Graph Databases*. MoDELS (Companion) 2024
- Dominique Hausler, Torben Eckwert, Meike Klettke, Michael Guckert, Gabriele Taentzer: *Model-driven Schema Transformation for Graph Databases*. ER 2025