University
of Basel

Polyglot Database Management Systems (PolyDBMSs): Beyond Single-Model Solutions

Tutorial at the IEEE Big Data 2025

Marco Vogt <marco.vogt@unibas.ch>, Heiko Schuldt <heiko.schuldt@unibas.ch>

Databases and Information Systems (DBIS) Group
University of Basel, Switzerland

December 9th, 2025

Let Me Introduce Myself

Marco Vogt

- Postdoctoral researcher and lecturer in the Databases and Information Systems group at the **University of Basel**
- Co-founder of **Polypheny GmbH** and initiator of the open-source Polypheny project
- I work on how to make multiple data models and engines feel like one coherent system, from schema and query languages down to execution and optimization

What is this tutorial about?

Topic

Polyglot Database Management Systems (PolyDBMSs) – database systems that expose one logical DBMS over multiple data models and engines

Goal

Show how to manage relational, document, graph, and other data with mixed workloads through a single query layer and control plane

Perspective

Position PolyDBMSs among multi-model DBMSs, polystores, multistores, and polyglot persistence

What we will do

- Introduce the core concepts, architecture, and components of a PolyDBMS (interfaces, schemas, mappings, optimizer, routing, transactions, monitoring)

- Walk through schema modeling and language mappings for cross-model queries on a concrete example

- Demonstrate these ideas hands-on with the open-source Polypheny platform

By the end, you should:

- Understand when a PolyDBMS is useful and when a single engine is enough
- Have a mental model of how such a system is built and operated
- Be able to relate your own “polyglot persistence” setups to the PolyDBMS approach

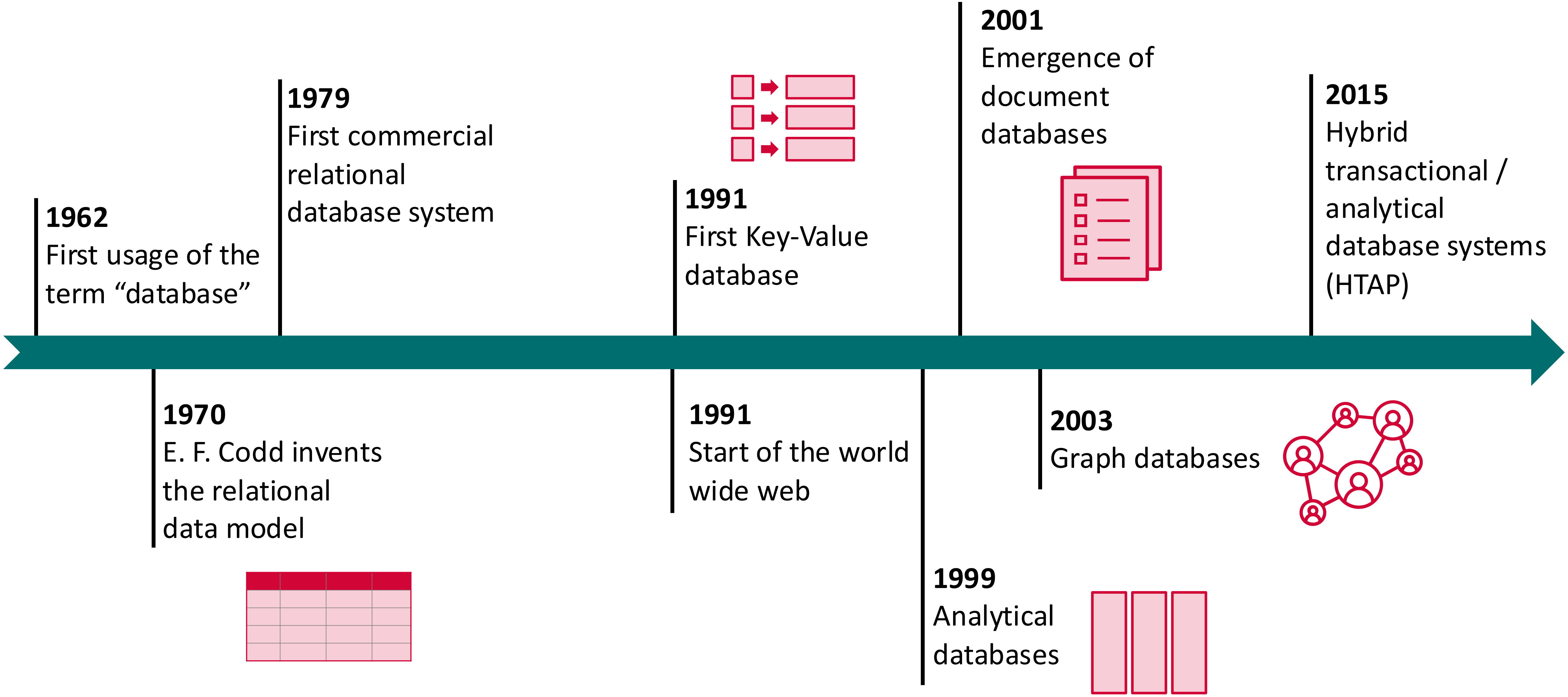
Website

https://bigdata2025.polypheny.com/

Outline

1	Motivation: Why do we need PolyDBMS
2	Schema Model
3	Polypheny
4	Other Approaches and Systems
5	Hands-on: Polypheny
6	Limitations, Open Research Questions, and Outlook

A Brief History of Databases



1962
First usage of the term “database”

1970
E. F. Codd invents the relational data model

1979
First commercial relational database system

1991
First Key-Value database

1991
Start of the world wide web

1999
Analytical databases

2001
Emergence of document databases

2003
Graph databases

2015
Hybrid transactional / analytical database systems (HTAP)

One Size Does Not Fit All

Relational

- Data is represented as "tables"
- Structured data / data following a strict schema
- Transactional workloads

Key-Value

- Array-like data structure
- High performance and scalability
- Allows storing arbitrary values

Document

- Data is represented as collection of documents
- Unstructured data / following no schema
- Nested data structures

Analytical

- Comes in various shapes, e.g., based on the relational model
- All items of a column are stored together

Label-Property Graph

- Data is represented using nodes and edges
- Good for storing and querying relationships

Vector DB

- Collection of high-dimensional vectors
- Similarity search (nearest neighbors)

Motivation: The Gavel Auction House

Auction System

Auction Item Catalog

Payment System

Recommendation System

Business Analytics

Heterogeneous Data

- Different data models
- Structured and unstructured data
- Interconnected

Different Query Languages

- Applications use different query languages
- Data needs to be accessible using all query languages

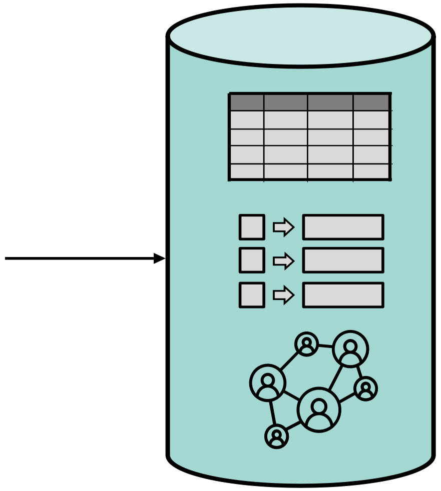
Mixed Workloads

- Transactional workloads
- Analytical workloads
- Special functions

Consistent Modification of Data

- Changes should be available to all applications
- ACID compliant transactions

Multimodel DBMS, Polystores and HTAP Systems

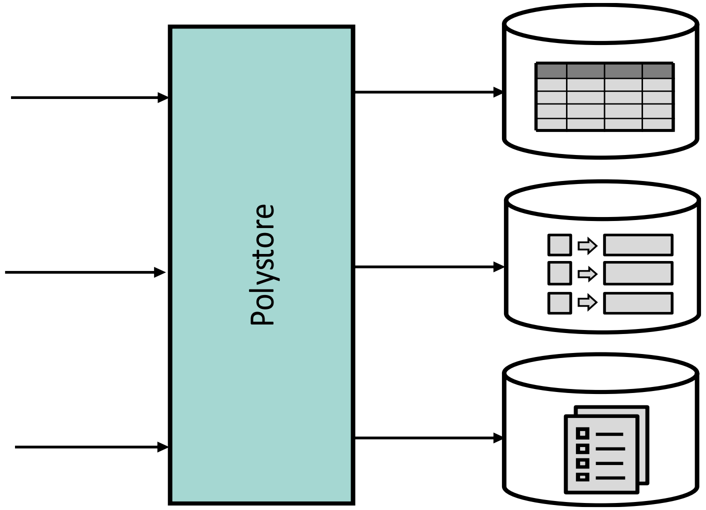


Multimodel DBMS

+ Heterogenous data / multiple data models

- Need to reimplement existing work

- Typically one query language



Polystore

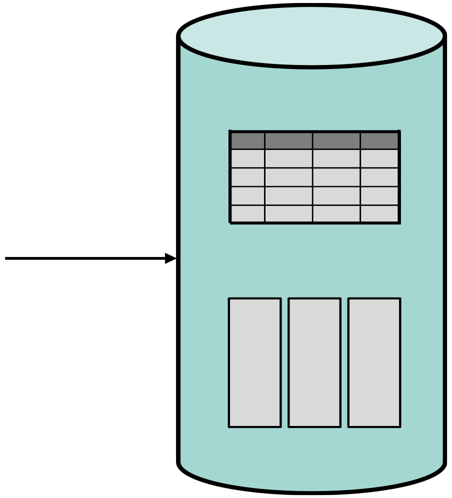
+ Multiple query languages

+ Heterogenous data

+ Large variety of (analytical) workloads

- No data manipulation support

- No DBMS functionality



HTAP System

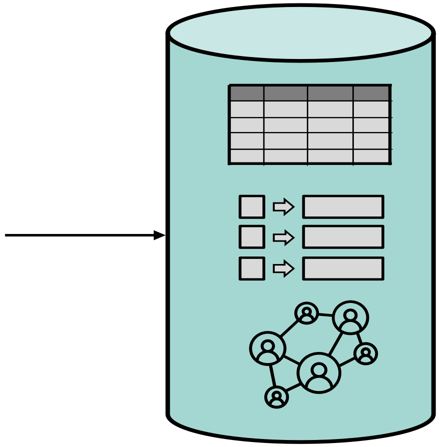
+ DML queries

+ Mixed transactional and analytical workloads

- Only structured data / one data model

- No support for multiple query languages

Multimodel DBMS



Heterogeneous Data ✓

- Different data models
- Structured and unstructured data
- Interconnected

Different Query Languages ✗

- Applications use different query languages
- Data needs to be accessible using all query languages

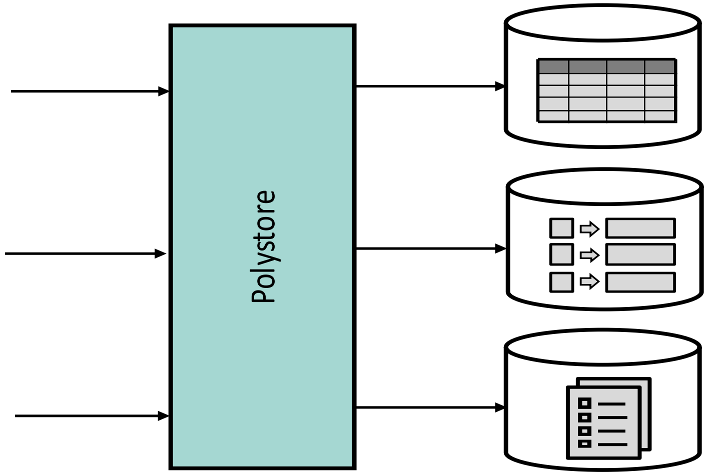
Mixed Workloads ✓

- Transactional workloads
- Analytical workloads
- Special functions

Consistent Modification of Data ✓

- Changes should be available to all applications
- ACID compliant transactions

Polystores



Heterogeneous Data ✓

- Different data models
- Structured and unstructured data
- Interconnected

Different Query Languages ✓

- Applications use different query languages
- Data needs to be accessible using all query languages

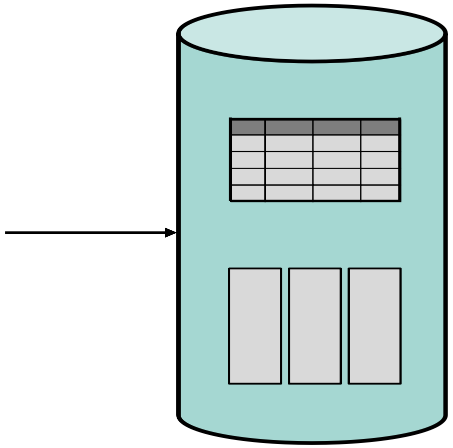
Mixed Workloads ✓

- Transactional workloads
- Analytical workloads
- Special functions

Consistent Modification of Data ✗

- Changes should be available to all applications
- ACID compliant transactions

HTAP Systems



Heterogeneous Data ✗

- Different data models
- Structured and unstructured data
- Interconnected

Different Query Languages ✗

- Applications use different query languages
- Data needs to be accessible using all query languages

Mixed Workloads ✓

- Transactional workloads
- Analytical workloads
- Special functions

Consistent Modification of Data ✓

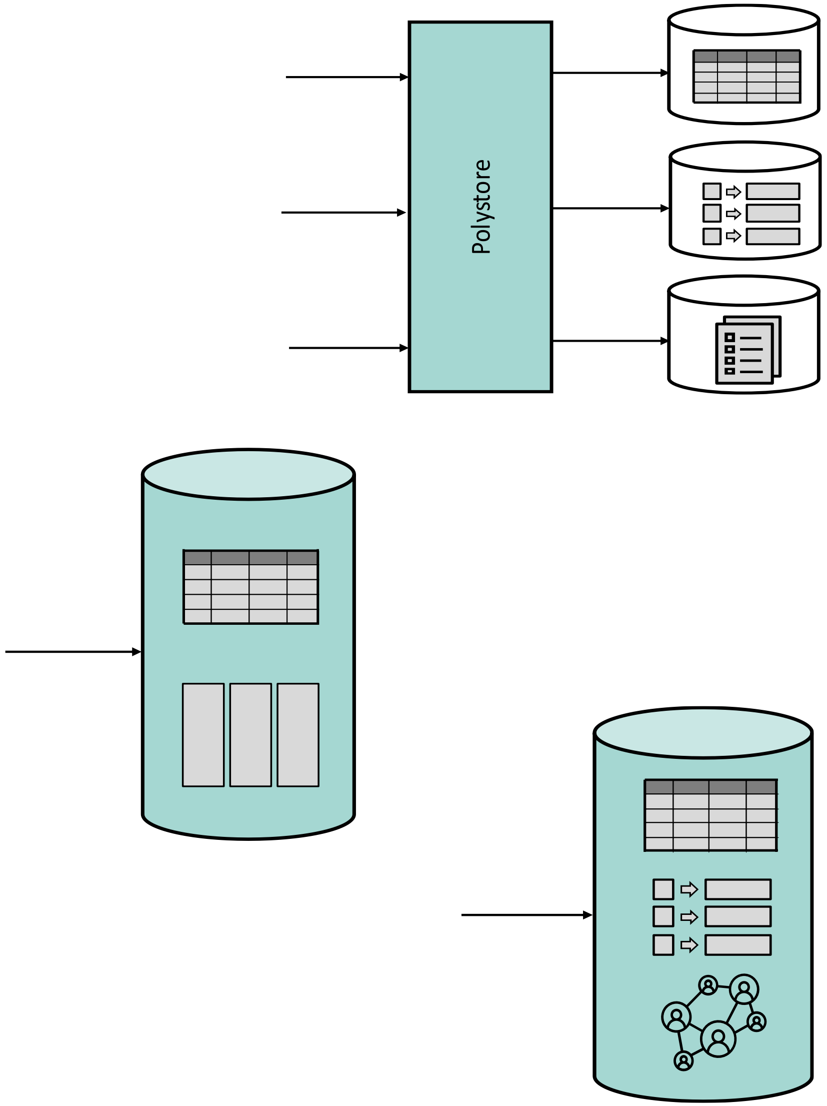
- Changes should be available to all applications
- ACID compliant transactions

A New Kind of Database System

There is the need for a system that …

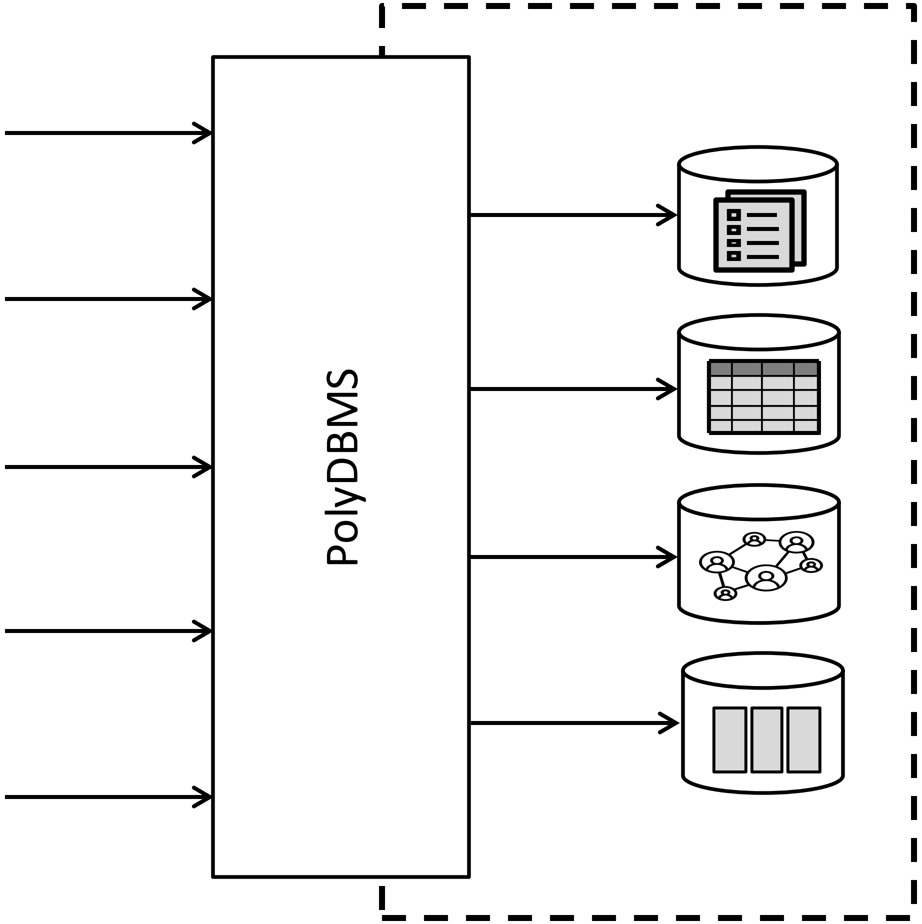
- supports **multiple query languages**
- maintains data according to **multiple data models**
- provides good performance for **mixed workloads**
- enables **cross-model queries**
- supports **data manipulation** queries

Combining the concepts of polystores, multimodel database systems and HTAP systems



PolyDBMS Overview

- A **Polyglot Database Management System** (PolyDBMS) is a DBMS that exposes one logical system over multiple data models and storage engines.
- It provides **multiple query interfaces** (e.g., SQL, document-style, graph-style) for the same underlying data.
- It maintains a **central schema and catalog** that integrate heterogeneous engines and models.
- It **plans, routes, and executes** queries across engines, pushing work down where possible.
- It supports **cross-model queries and heterogeneous workloads**.



Independence of Storage Configuration

The Problem

- Data stores do not have the same set of features and capabilities
- Especially problematic with data modification queries

Example: Day of Week (DoW)
Function that takes a timestamp and returns an integer
PostgreSQL: 0-6, Sunday is 0
Oracle: 1-7, Sunday is 1

The solution

- Integrate an execution engine into the PolyDBMS itself
- This engine is able to execute all queries

The result of a query must be independent of

- **how and where** the data is physically stored
- **by which engine** it has been processed

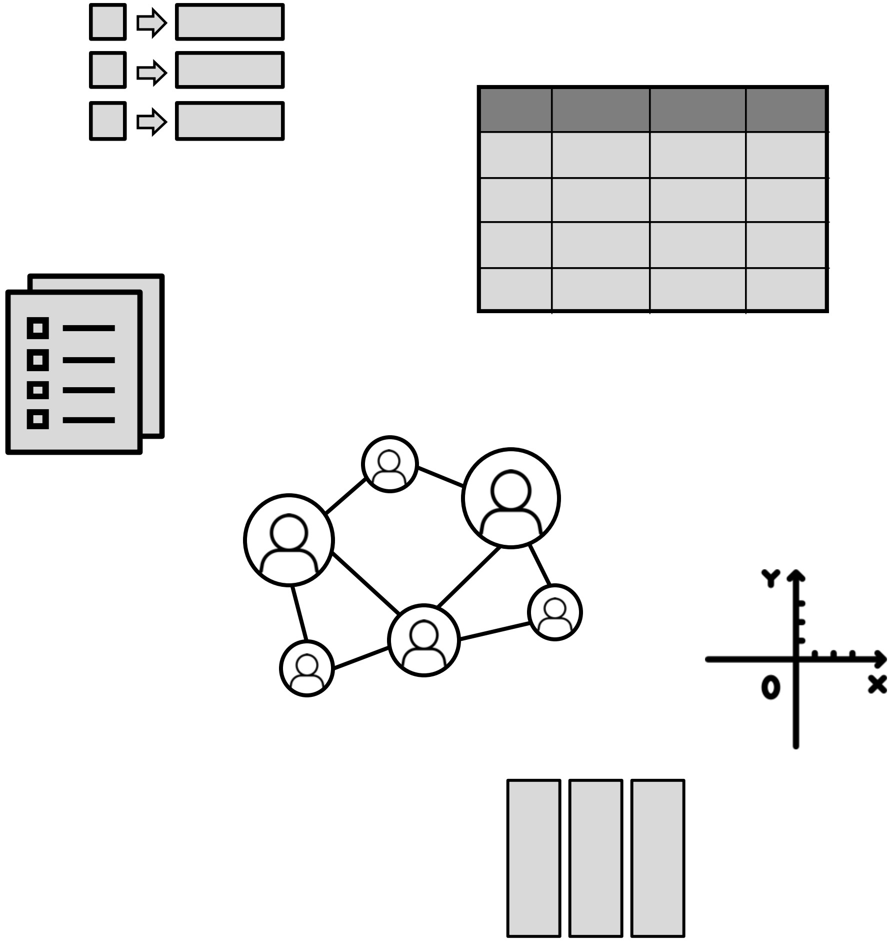
The available query languages, operations and functions must not depend on the physical storage of the data.

Only observable difference between storage configurations should be the execution time

Schema Model

Schema Model: The Multimodel Challenge

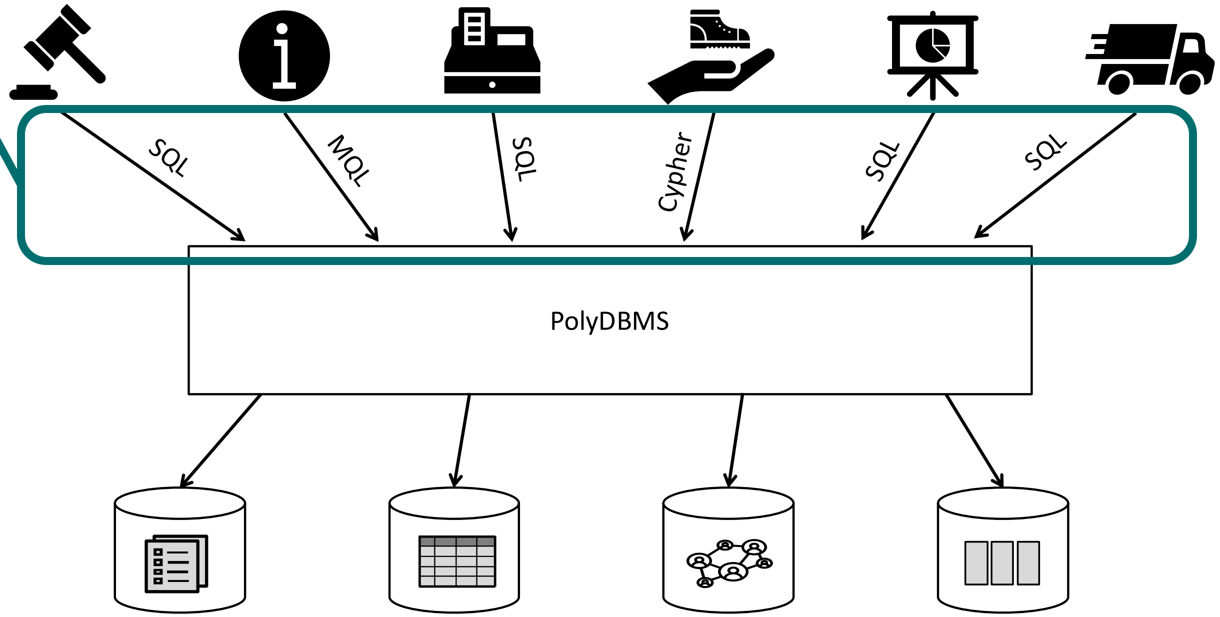
- Each data models defines their own structure and building blocks for organizing the data
- Each query language requires a certain structure
- All this is different!
- Queries across data models and storing data across heterogenous data stores requires mappings between schemas!



Different Kinds of Schemas

Exposed Schema

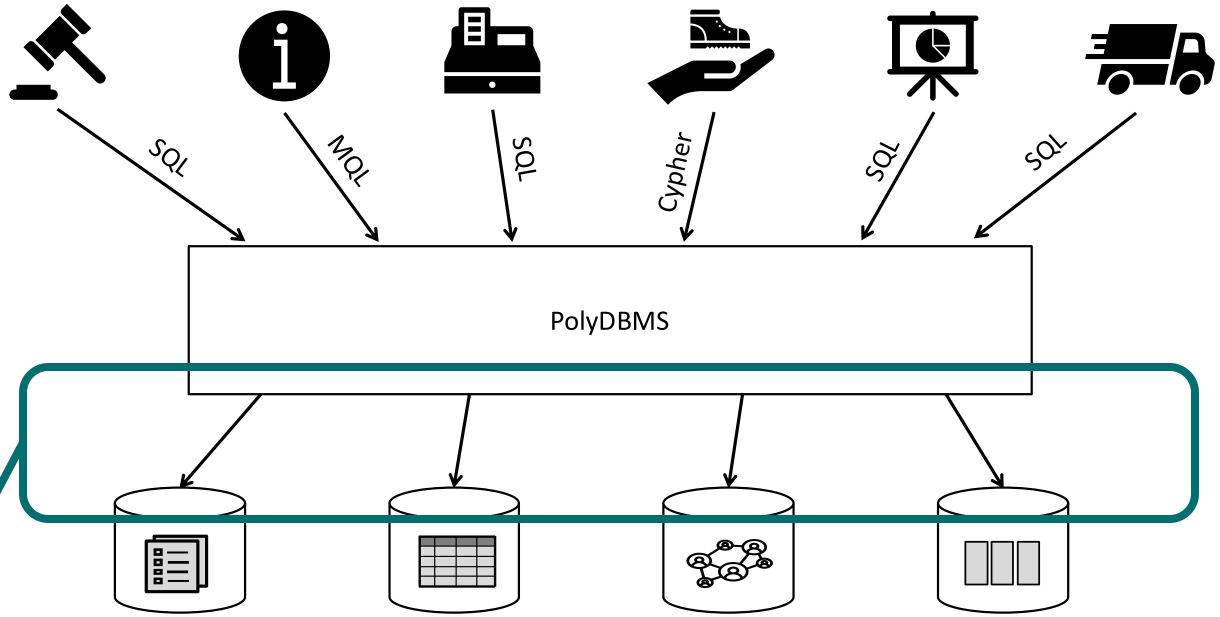
- Building blocks are defined by the query language
- Making semantic concepts from other data models available



Different Kinds of Schemas

Exposed Schema

- Building blocks are defined by the query language
- Making semantic concepts from other data models available



Physical Schema

- Building blocks are defined by the data store
- Storing data for efficient querying
- Utilizing features of the data store

Different Kinds of Schemas

Exposed Schema

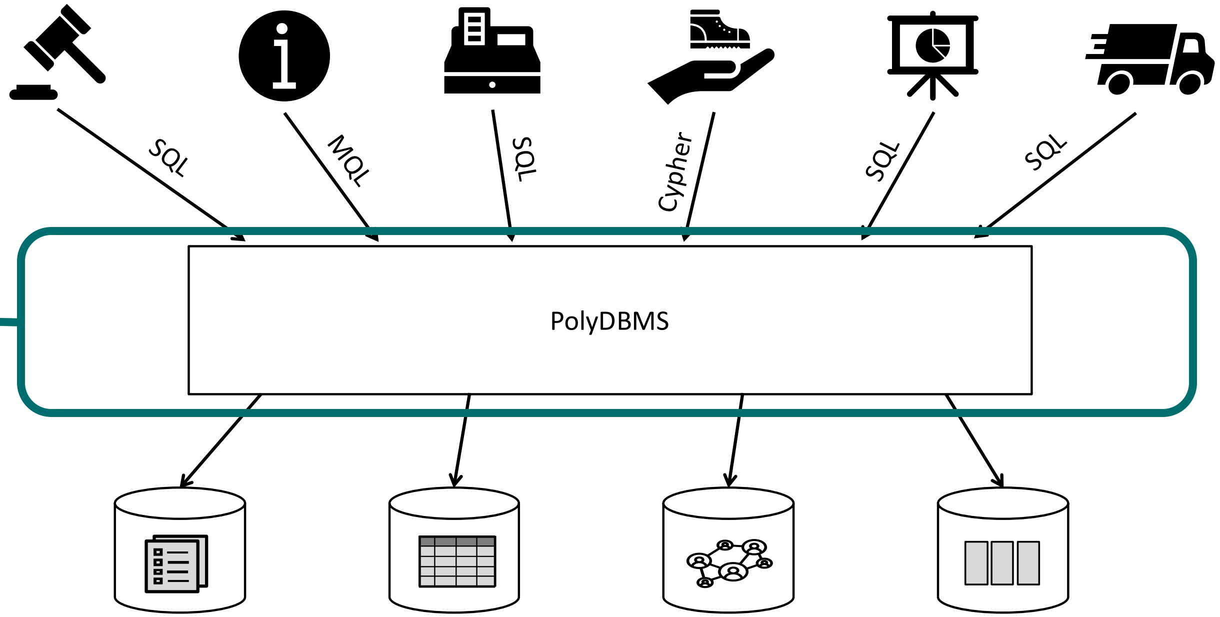
- Building blocks are defined by the query language
- Making semantic concepts from other data models available

Logical Schema

- The central schema of the PolyDBMS
- Building blocks from all supported data models
- Enabling cross-model queries

Physical Schema

- Building blocks are defined by the data store
- Storing data for efficient querying
- Utilizing features of the data store



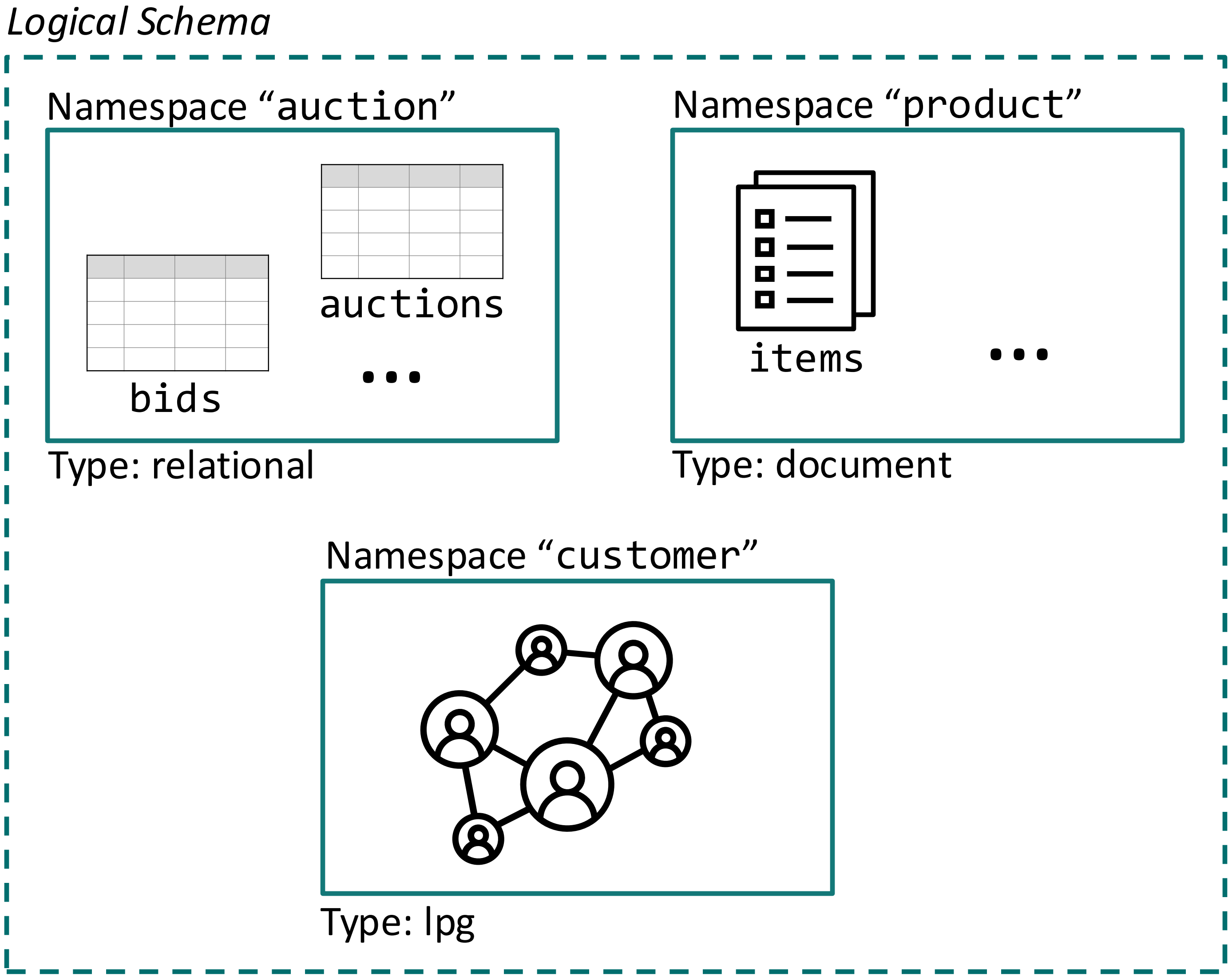
Namespaces

The logical schema S is a set of namespaces N:

$$S := \{N_1, N_2, \dots, N_m\} \text{ with } m \in \mathbb{N}$$

- Every namespace has a unique name and is of a specific data model
- This model defines the available set of schema building blocks

The Gavel Schema



Exposed Schema

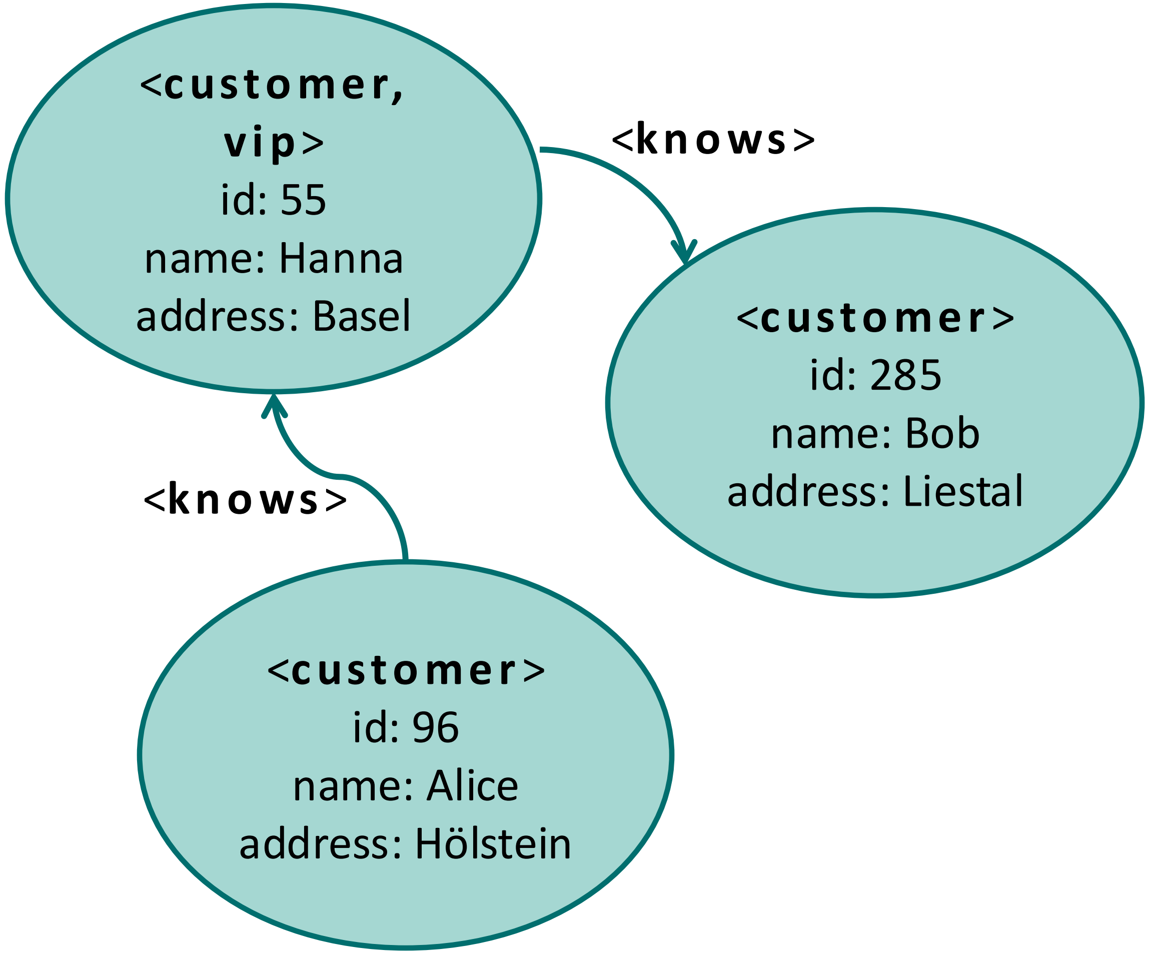
Example: The shipment application needs to access the address of a customer through a relational query interface.

$$N^{LPG} \coloneqq \langle \text{name}, G \rangle$$

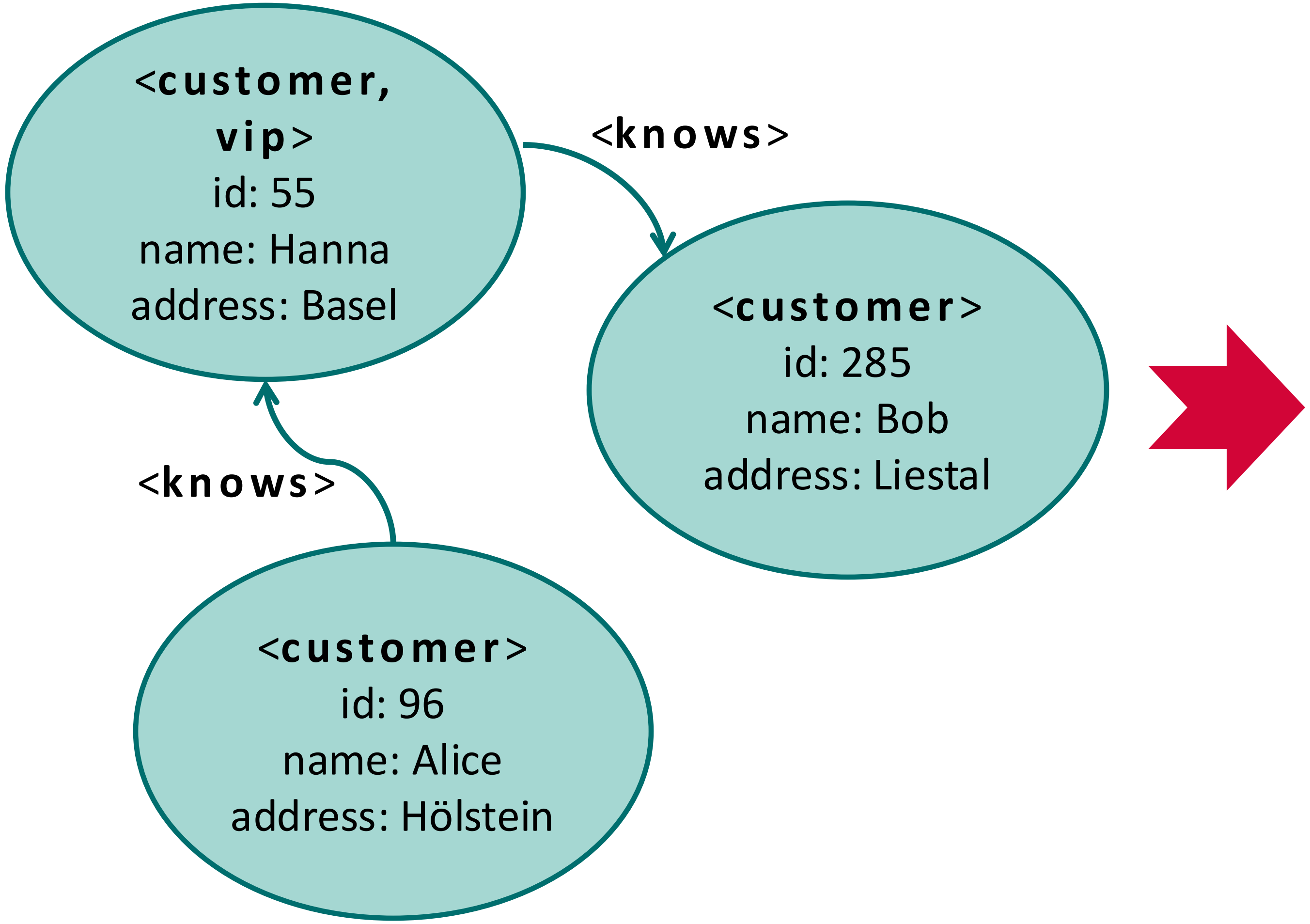
$$N^{REL} \coloneqq \langle \text{name}, \{T_1, \dots, T_n\} \rangle \text{ with } n \in \mathbb{N}$$

$$N^{LPG} \longmapsto N^{REL}$$

Map node labels as tables and mimic the concept of join tables



LPG ⟼ Relational



customer

id	properties	labels
55	*name*: Hanna, *address*: Basel	customer, vip
285	*name*: Bob, *address*: Liestal	customer
96	*name*: Alice, *address*: Hölstein	customer

vip

id	properties	labels
55	*name*: Hanna, address: Basel	customer, vip

customer->customer

src	tgt	properties	labels
55	285		knows
96	55		knows

customer->vip

src	tgt	properties	labels
96	55		knows

vip->customer

src	tgt	properties	labels
55	285		knows

LPG ⟼ Relational

$$N^{LPG} \coloneqq \langle \text{name}, G \rangle$$

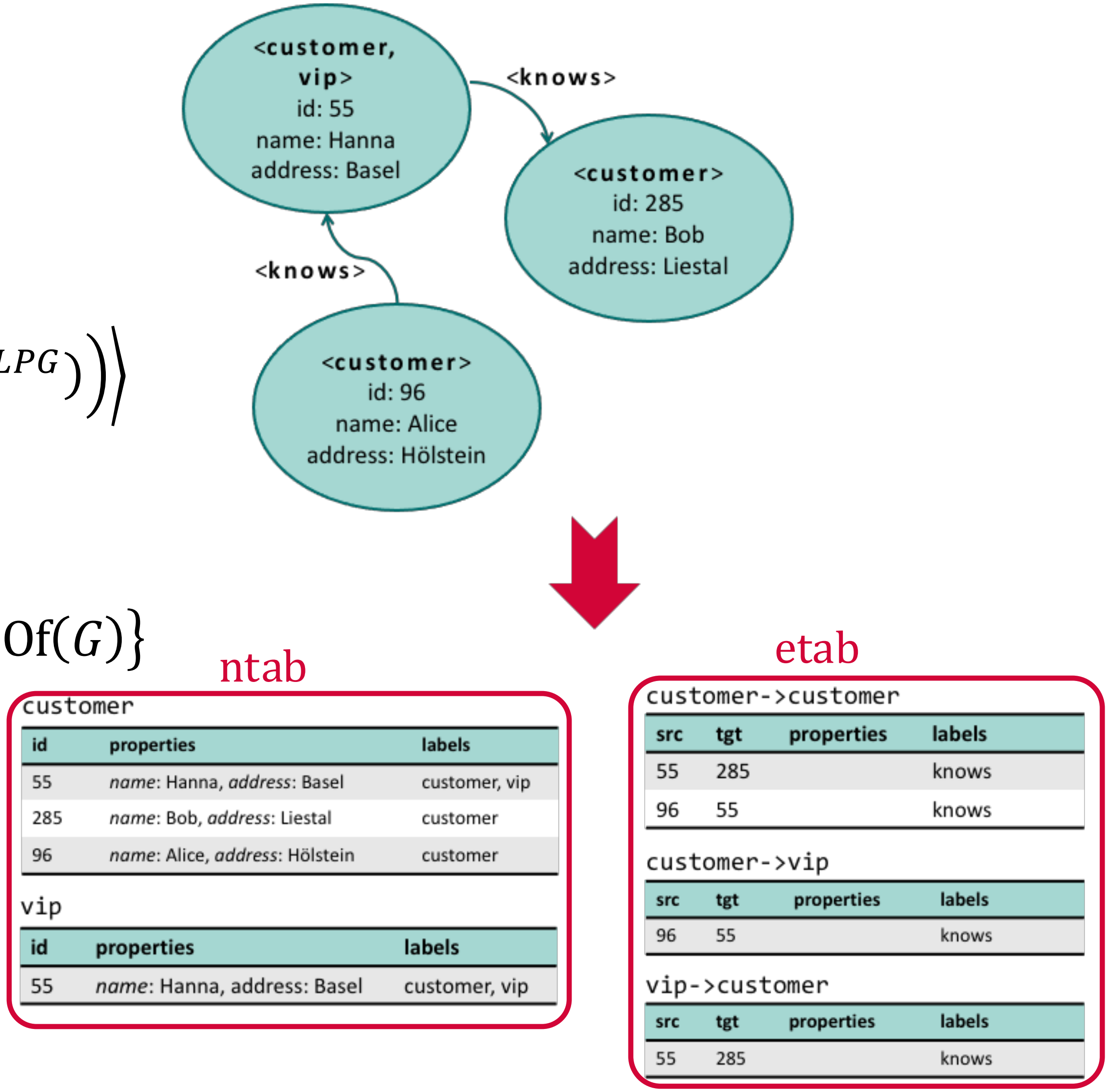
$$N^{REL} \coloneqq \langle \text{name}, \{T_1, \ldots, T_n\} \rangle \;\text{ with } n \in \mathbb{N}$$

$$N^{LPG} \longmapsto N^{REL}$$

$$\langle \text{name}, G \rangle \longmapsto \left\langle \pi_1(N^{LPG}), \text{ntab}\left(\pi_2(N^{LPG})\right) \cup \text{etab}\left(\pi_2(N^{LPG})\right) \right\rangle$$

$$\text{ntab}(G) \coloneqq \left\{ \left(x, (\text{id}, \text{props}, \text{labels})\right) \,\middle|\, x \in \text{labelsOf}(G) \right\}$$

$$\text{etab}(G) \coloneqq \left\{ \left(x{\to}y, (\text{src}, \text{tgt}, \text{props}, \text{labels})\right) \,\middle|\, x, y \in \text{labelsOf}(G) \right\}$$



ntab

customer

id	properties	labels
55	*name*: Hanna, *address*: Basel	customer, vip
285	*name*: Bob, *address*: Liestal	customer
96	*name*: Alice, *address*: Hölstein	customer

vip

id	properties	labels
55	*name*: Hanna, address: Basel	customer, vip

etab

customer->customer

src	tgt	properties	labels
55	285		knows
96	55		knows

customer->vip

src	tgt	properties	labels
96	55		knows

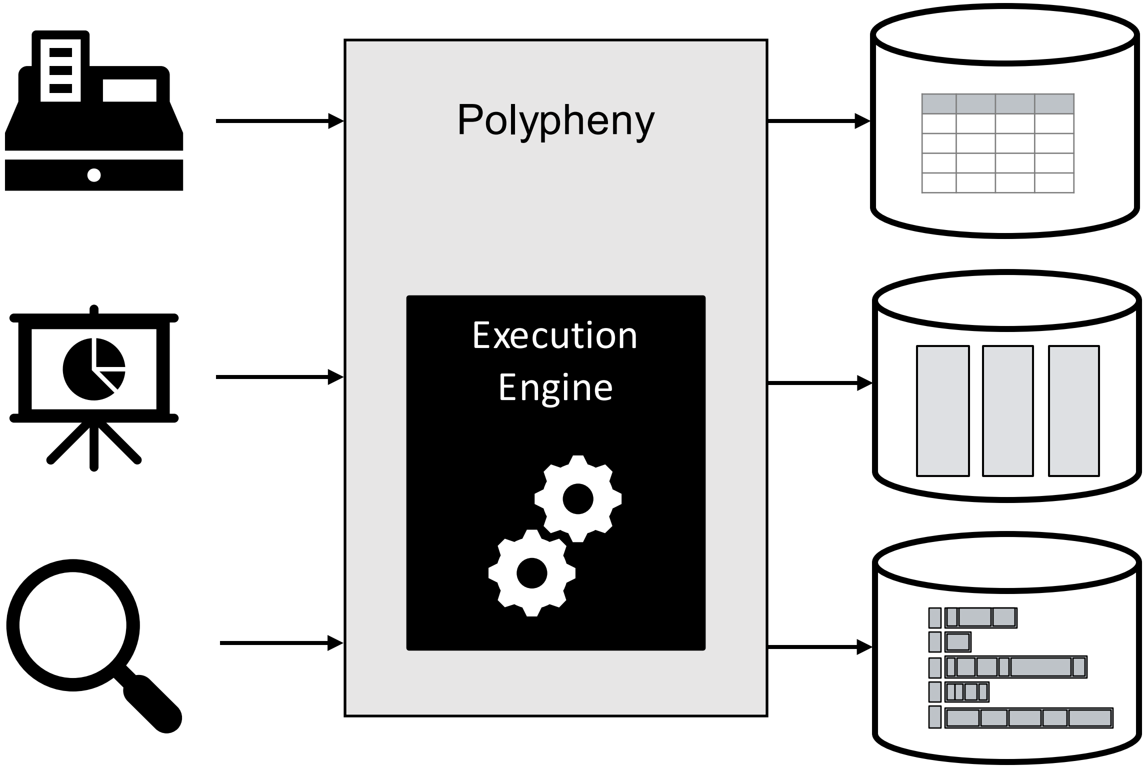
vip->customer

src	tgt	properties	labels
55	285		knows

→ There are similar mappings for the other pairs of data models and for mapping to the physical schema

Polypheny

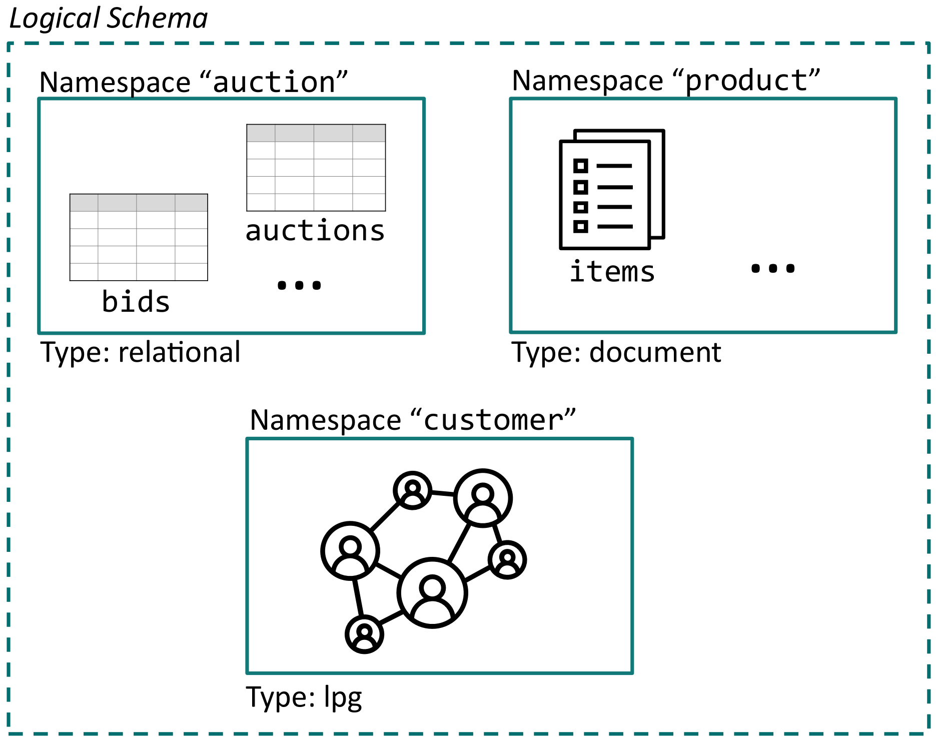
Our Implementation of a PolyDBMS: Polypheny



- Using existing established and domain-optimized DBMS as **storage and execution engines**
- An **integrated execution engine** that compensates missing features and processes joins
- Supports cross-model queries and replication
- Enforces constraints across stores
- Utilizes the optimization and domain-knowledge of specialized systems

An Example Query

```
SELECT
      i.id as "Item",
      c.properties[id] as "Address",
      i.weight as "Weight"
FROM
      product.items i,
      customer.customer c,
      auction.auctions a,
      auction.bids b
WHERE
      i.id = a.item and
      c.id = b.customer and
      a.id = b.auction and
      b.winner = true and
      a.paid = true and
      a.shipped = false
```



Logical Schema

Namespace “auction”

bids

auctions

...

Type: relational

Namespace “product”

items

...

Type: document

Namespace “customer”

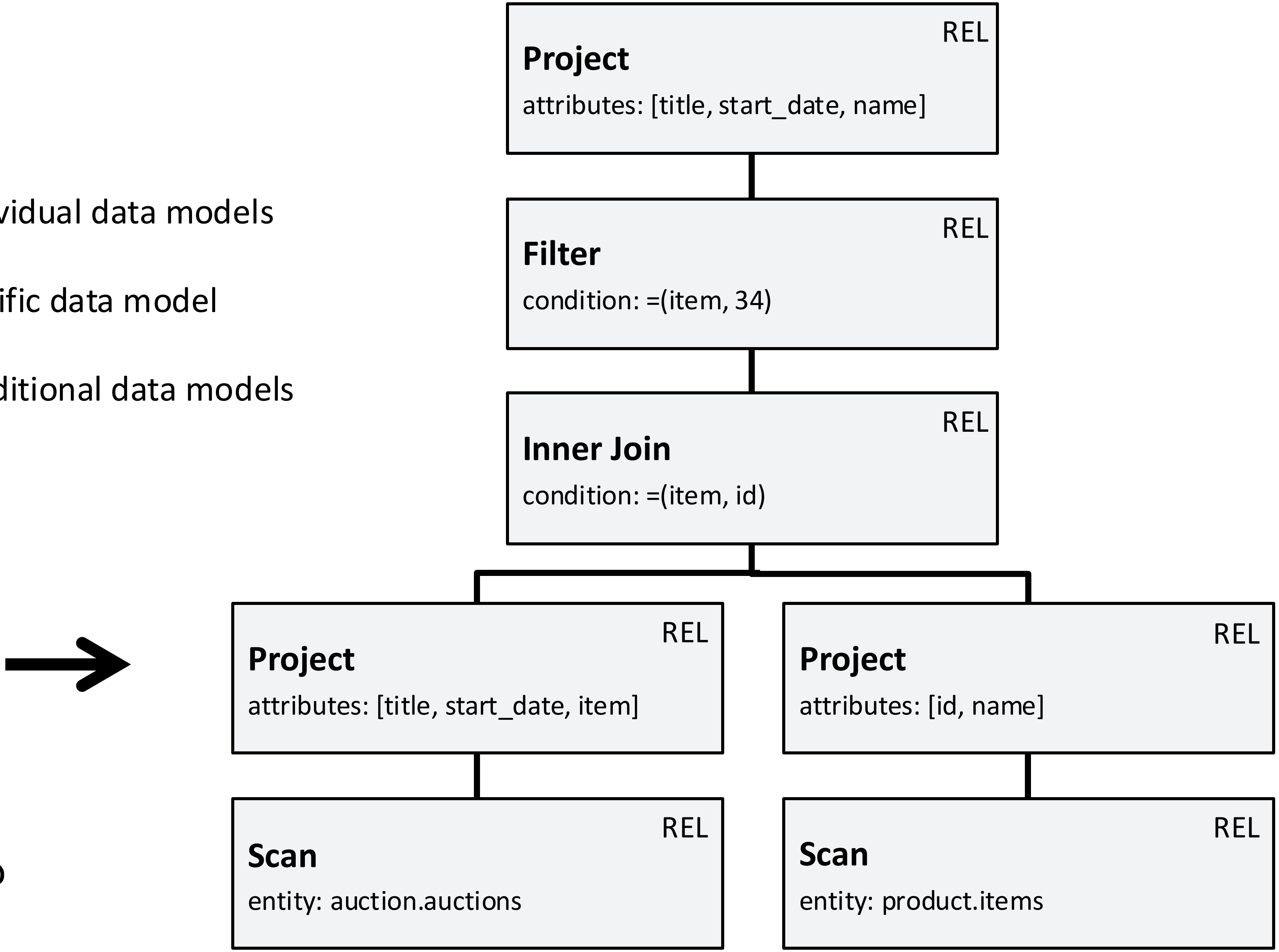
Type: lpg

Item	Address	Weight
56895	Polypheny-Str. 1, Basel	2.3
89626	PolyDBMS Weg 5, Liestal	0.5
59648	Chronos-Str. 3, Hölstein	1.7

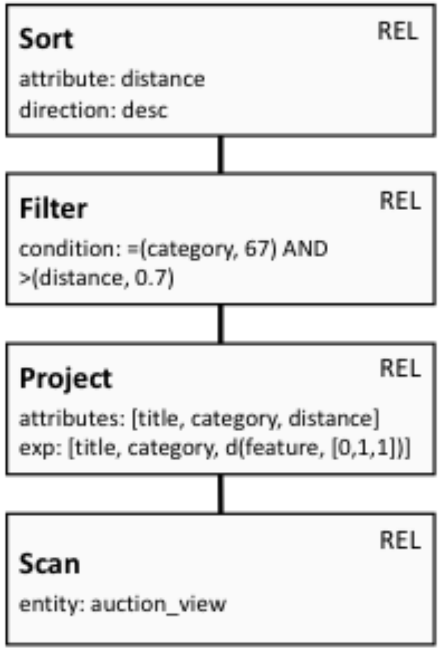
PolyAlgebra

- Preserves the semantics of the individual data models
- Avoids mapping queries into a specific data model
- Can be extended to incorporate additional data models

```
SELECT
   a.title,
   a.start_date,
   u.email
FROM
   auctions a,
   customer.customer c
WHERE
   a.customer = c.id AND
   c.id = 34
```



Logical



Sort REL

attribute: distance
direction: desc

Filter REL

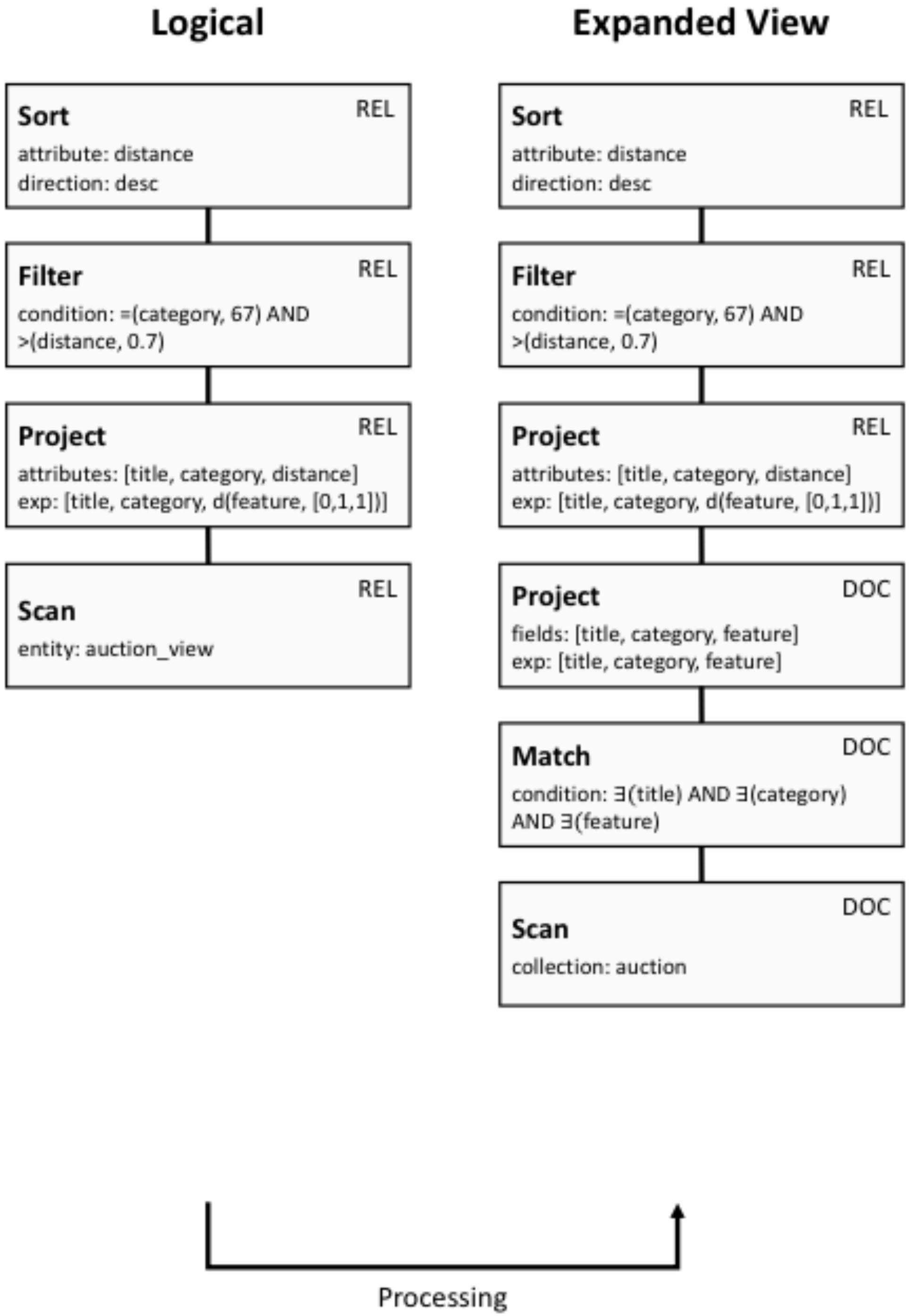
condition: =(category, 67) AND >(distance, 0.7)

Project REL

attributes: [title, category, distance]
exp: [title, category, d(feature, [0,1,1])]

Scan REL

entity: auction_view



Logical

Sort REL
attribute: distance
direction: desc

Filter REL
condition: =(category, 67) AND >(distance, 0.7)

Project REL
attributes: [title, category, distance]
exp: [title, category, d(feature, [0,1,1])]

Scan REL
entity: auction_view

Expanded View

Sort REL
attribute: distance
direction: desc

Filter REL
condition: =(category, 67) AND >(distance, 0.7)

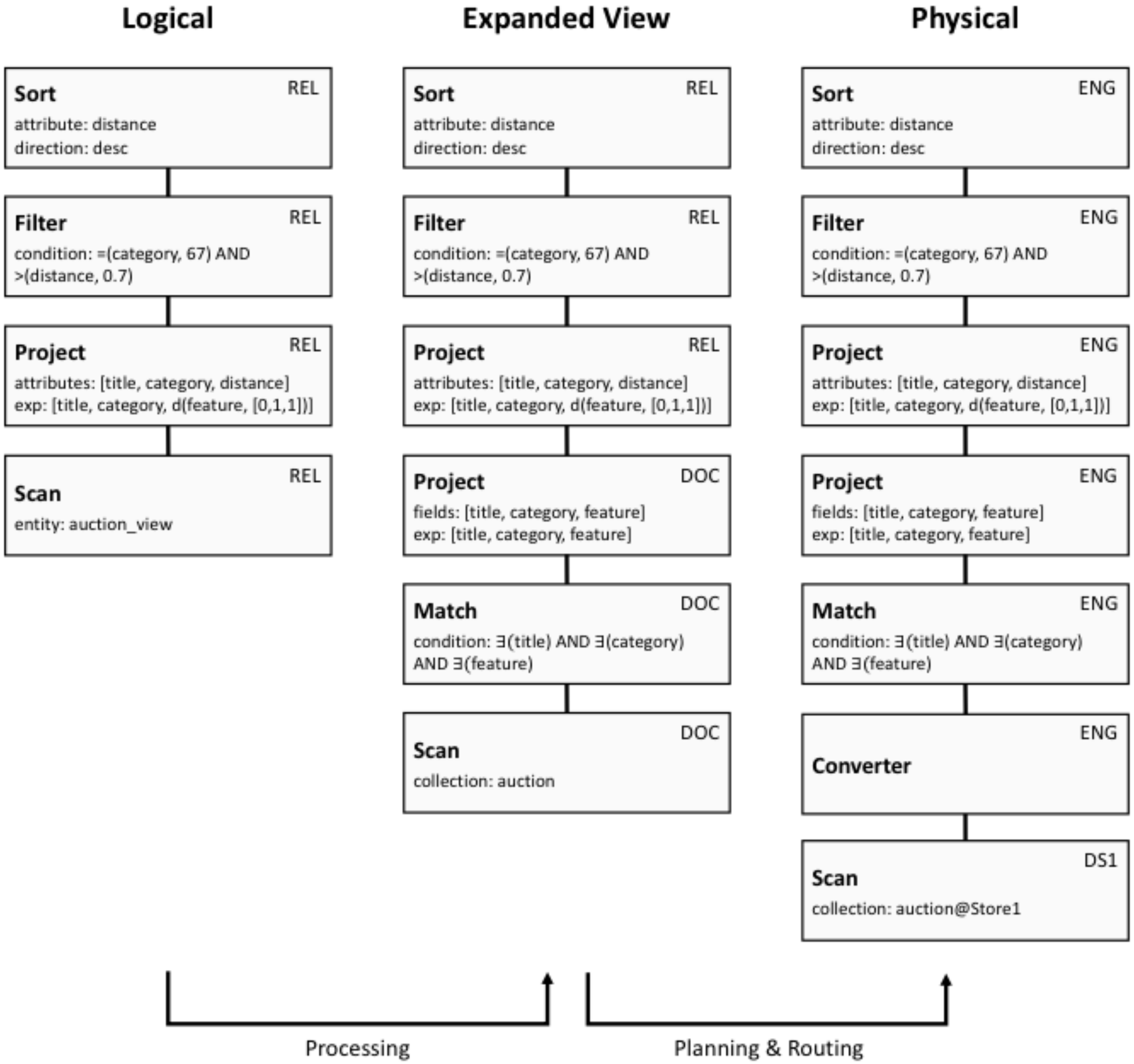
Project REL
attributes: [title, category, distance]
exp: [title, category, d(feature, [0,1,1])]

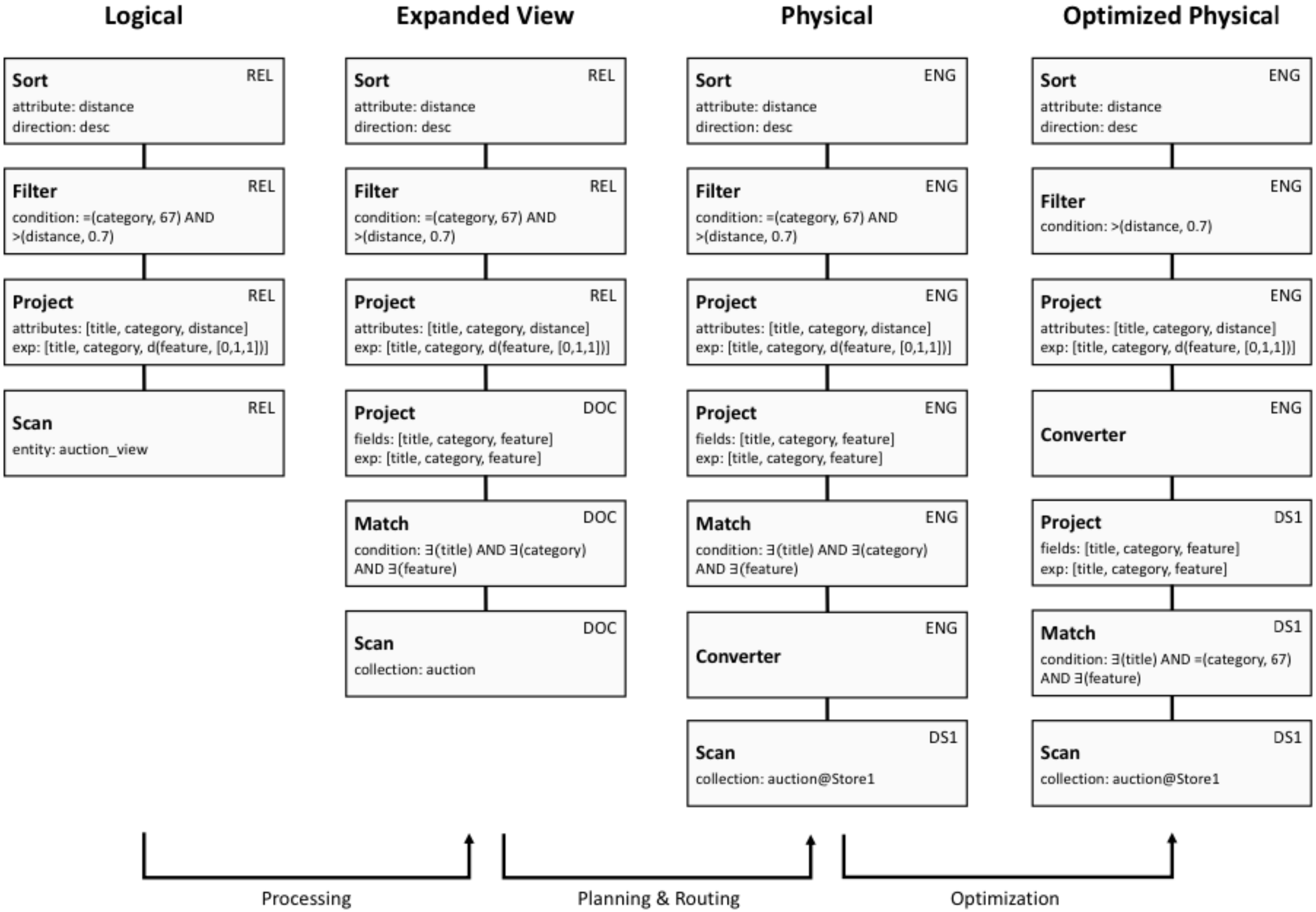
Project DOC
fields: [title, category, feature]
exp: [title, category, feature]

Match DOC
condition: ∃(title) AND ∃(category) AND ∃(feature)

Scan DOC
collection: auction

Processing


Logical
Expanded View
Physical
Sort REL
attribute: distance
direction: desc
Filter REL
condition: =(category, 67) AND >(distance, 0.7)
Project REL
attributes: [title, category, distance]
exp: [title, category, d(feature, [0,1,1])]
Scan REL
entity: auction_view
Sort REL
attribute: distance
direction: desc
Filter REL
condition: =(category, 67) AND >(distance, 0.7)
Project REL
attributes: [title, category, distance]
exp: [title, category, d(feature, [0,1,1])]
Project DOC
fields: [title, category, feature]
exp: [title, category, feature]
Match DOC
condition: ∃(title) AND ∃(category) AND ∃(feature)
Scan DOC
collection: auction
Sort ENG
attribute: distance
direction: desc
Filter ENG
condition: =(category, 67) AND >(distance, 0.7)
Project ENG
attributes: [title, category, distance]
exp: [title, category, d(feature, [0,1,1])]
Project ENG
fields: [title, category, feature]
exp: [title, category, feature]
Match ENG
condition: ∃(title) AND ∃(category) AND ∃(feature)
Converter ENG
Scan DS1
collection: auction@Store1
Processing
Planning & Routing



Logical

Sort REL
attribute: distance
direction: desc

Filter REL
condition: =(category, 67) AND >(distance, 0.7)

Project REL
attributes: [title, category, distance]
exp: [title, category, d(feature, [0,1,1])]

Scan REL
entity: auction_view

Expanded View

Sort REL
attribute: distance
direction: desc

Filter REL
condition: =(category, 67) AND >(distance, 0.7)

Project REL
attributes: [title, category, distance]
exp: [title, category, d(feature, [0,1,1])]

Project DOC
fields: [title, category, feature]
exp: [title, category, feature]

Match DOC
condition: ∃(title) AND ∃(category) AND ∃(feature)

Scan DOC
collection: auction

Physical

Sort ENG
attribute: distance
direction: desc

Filter ENG
condition: =(category, 67) AND >(distance, 0.7)

Project ENG
attributes: [title, category, distance]
exp: [title, category, d(feature, [0,1,1])]

Project ENG
fields: [title, category, feature]
exp: [title, category, feature]

Match ENG
condition: ∃(title) AND ∃(category) AND ∃(feature)

Converter ENG

Scan DS1
collection: auction@Store1

Optimized Physical

Sort ENG
attribute: distance
direction: desc

Filter ENG
condition: >(distance, 0.7)

Project ENG
attributes: [title, category, distance]
exp: [title, category, d(feature, [0,1,1])]

Converter ENG

Project DS1
fields: [title, category, feature]
exp: [title, category, feature]

Match DS1
condition: ∃(title) AND =(category, 67) AND ∃(feature)

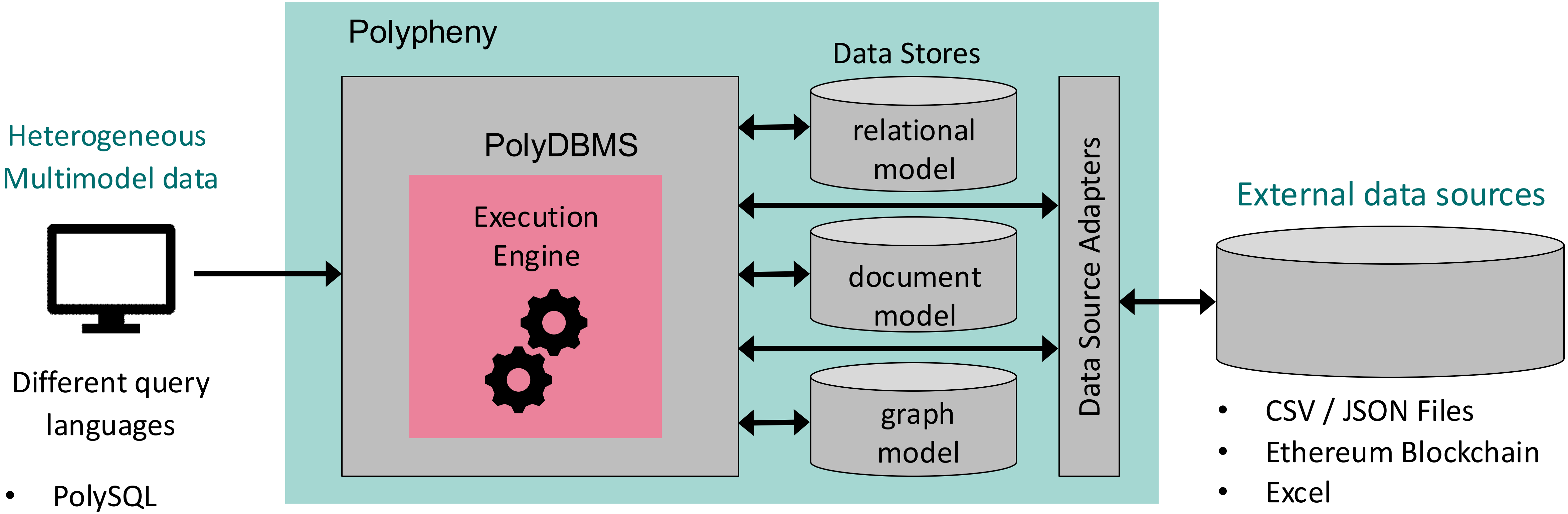
Scan DS1
collection: auction@Store1

Processing

Planning & Routing

Optimization

Polypheny in Detail



- PolySQL
- MongoQL
- REST
- CQL
- PolyPig
- Cypher

- Cassandra
- Cottontail-DB
- DuckDB
- File
- HSQLDB

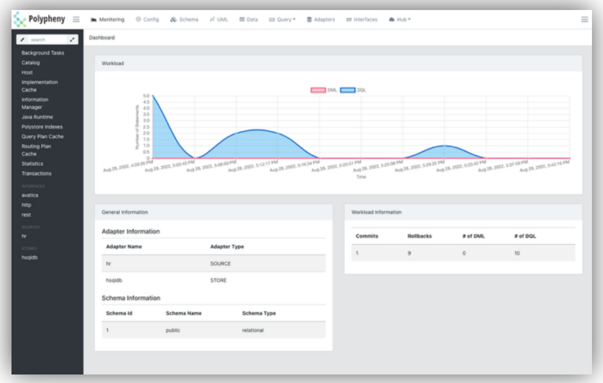
- MonetDB
- MongoDB
- Neo4j
- PostgreSQL

- CSV / JSON Files
- Ethereum Blockchain
- Excel
- Google Docs
- MariaDB / mySQL
- MonetDB
- PostgreSQL
- File System

Other Approaches and Systems

- **Multi-model DBMSs** (e.g., ArangoDB, OrientDB, PostgreSQL+JSONB)
  One engine that supports several data models internally. They unify models, but not multiple heterogeneous engines.
- **Polystores and multistores** (e.g., BigDAWG, research polystores)
  Federate queries over diverse engines. Strong at cross-engine access, usually around one main query language.
- **SQL-on-everything / data virtualization** (e.g., Trino/Presto, Drill, Calcite-based systems, Denodo-like platforms)
  Provide a single SQL interface over many sources, often via virtual views, with limited polyglot query capabilities.
- **HTAP DBMSs** (e.g., hybrid transactional/analytical systems)
  Combine OLTP and OLAP in one engine, typically for a single primary data model.

PolyDBMSs go beyond these by offering polyglot query interfaces, explicit logical schemas and mappings, and a DBMS-style control plane that coordinates multiple engines, models, and HTAP workloads as one system.



Adapter Information

Adapter Name	Adapter Type
hr	SOURCE
hsqldb	STORE

Schema Information

Schema Id	Schema Name	Schema Type
1	public	relational

Commits	Rollbacks	# of DML	# of DQL
1	9	0	10

Google Summer of Code

Interfaces:
- JDBC
- REST
- Python
- HTTP

Languages:
- SQL
- Cypher
- MongoQL
- CQL
- PIG

Release for major platforms and comprehensive documentation available on `polypheny.org`

Polypheny

open source

monetdb

Hands On

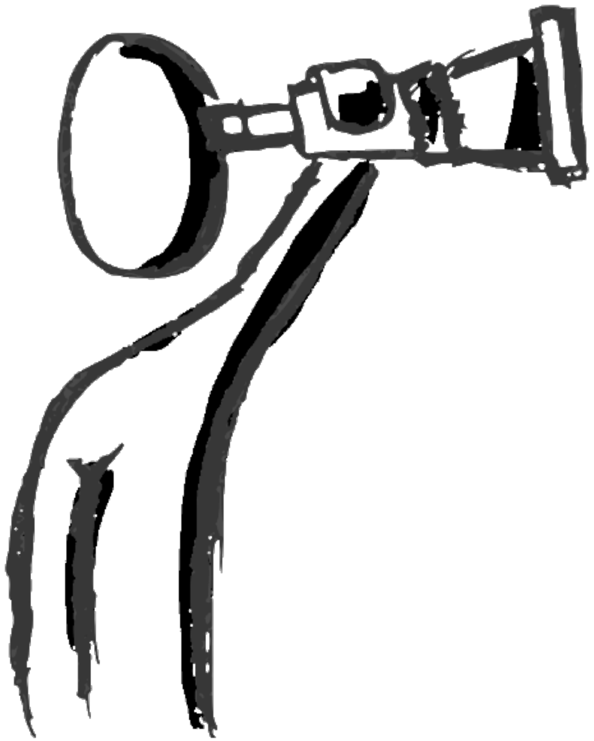
Limitations and when not to use a PolyDBMS

- **Overhead vs. a single engine**
  A PolyDBMS adds parsing, optimization, routing, and coordination. If the workload fits well into one mature engine, that extra layer may just add latency and complexity.
  You pay for flexibility

- **Simple, single-model workloads**
  If there is only one data model (e.g., purely relational OLTP or purely analytical SQL on a warehouse), a specialized DBMS is usually simpler, faster, and easier to operate.

- **Very tight latency or extreme throughput SLAs**
  Cross-engine queries, data movement, and coordination can make it hard to meet ultra-low-latency or hard real-time constraints.

Future Work



- Distributed PolyDBMS
- Self-adaptiveness
- Data streams and continuous queries

Thank you!

```
SQL>    SELECT * FROM questions

MQL>    db.questions.find()

Cypher> MATCH (q:questions) RETURN q
```

- **PolyDBMSs unify many engines and models as one DBMS**
  One logical system exposes polyglot query interfaces (SQL, document, graph, …) over heterogeneous stores, rather than gluing separate databases together.

- **A central schema model and mappings make cross-model queries possible**
  Exposed, logical, and physical schemas, with explicit mappings between them, decouple application schemas from physical storage and data models.

- **PolyDBMSs are powerful but not always the right tool**
  For simple, single-model workloads or ultra-tight SLAs, a single specialized engine remains the better choice.

contact: marco.vogt@unibas.ch