Unified access to all your data points

with Apache MetaModel

Apache MetaModel

Who am I?

Kasper Sørensen, dad, geek, guitarist …

@kaspersor

Long-time developer and
PMC member of:

Apache MetaModel

Founder also of another
nice open source project:

DataCleaner
The premier data quality solution
(commercial) open source

Principal Software Engineer @ Human Inference
a neopost company

Apache MetaModel

Session agenda

- Introduction to Apache MetaModel
- Use case: Query composition
- The role of Apache MetaModel in Big Data
- (New) architectural possibilities with Apache MetaModel

Introduction

to Apache MetaModel

Helicopter view

You can look at Apache MetaModel by it's formal description:

"… a uniform connector and query API to many very different datastore types …"

But let's start with a problem to solve ...

A problem

How to process multiple data sources while:

- Staying agnostic to the database engine.
- Respecting the metadata of the underlying database.
- Not repeating yourself.
- Avoiding fragility towards metadata changes.
- Mutating the data structure when needed.

We used to rely on ORM frameworks to handle the bulk of these ...

Apache MetaModel

ORM - Queries via **domain models**

Metadata (assumed)

```
public class Person {
  public String getName() {...}
}
```

Query (type-safe)

```
ORM.query(Person.class).where(Person::getName).eq("John Doe");
```

Apache MetaModel

Why an **ORM** *might* not work

Requires a domain model

- While many data-oriented apps are agnostic to a domain model.
- Sometimes the data itself is the domain.

Model cannot change at runtime

- Usually needed especially if your app creates new tables/entities

Type-safety through metadata assumptions

- This is quite common, yet it might be a problem that the model is statically mapped and cannot survive e.g. column renames etc.

Alternative to **ORM**: Use JDBC

- Metadata discoverable via java.sql.DatabaseMetaData.
- Queries can be assembled safely with a bit of String magic:

```
"SELECT " + columnNames + " FROM " + tableName + …
```

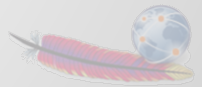
. . .

Alternative to **ORM**: Use JDBC

Wrong!

It turns out that …

- not all databases have the same SQL "dialect".
- they also don't all implement DatabaseMetaData the same way.
- you cannot use this on much else than relational databases that use SQL.
  - What about NoSQL, various file formats, web services etc.



Apache MetaModel

MetaModel - Queries via **metadata model**

Metadata (discovered)

```
DataContext dc = …
Table[] tables = dc.getDefaultSchema().getTables();
Table table = tables[0];
Column[] primaryKeys = table.getPrimaryKeys();
```

Query (metadata-safe)

```
dc.query().from(table).selectAll().where(primaryKeys[0]).eq(42);
```

Query (unsafe)

```
dc.query().from("person").selectAll().where("id").eq(42);
```

MetaModel - Connectivity

Sometimes we have **SQL**, sometimes we have another **native query engine** (e.g. **Lucene**) and sometimes we use MetaModel's **own query engine**.

(A few more connectors available via the MetaModel-extras (LGPL) project)

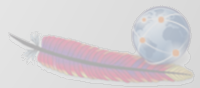
MetaModel updates

```
UpdateableDataContext dc = …

dc.executeUpdate(new UpdateScript() {
  // multiple updates go here - transactional characteristics (ACID, synchronization etc.)
  // as per the capabilities of the concrete DataContext implementation.
});

dc.executeUpdate(new BatchUpdateScript() {
 // multiple "batch/bulk" (non atomic) updates go here
});

// if I only want to do a single update, there are convenience classes for this
InsertInto insert = new InsertInto(table).value(nameColumn, "John Doe");
dc.executeUpdate(insert);
```



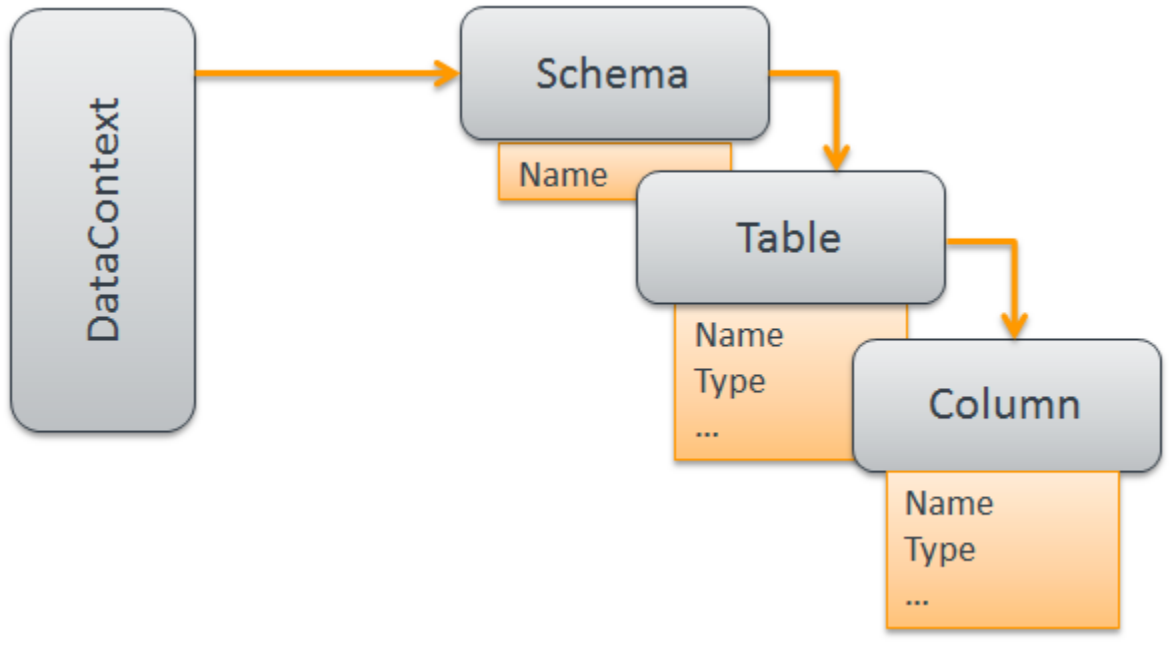
Apache MetaModel

Use case:

Query composition

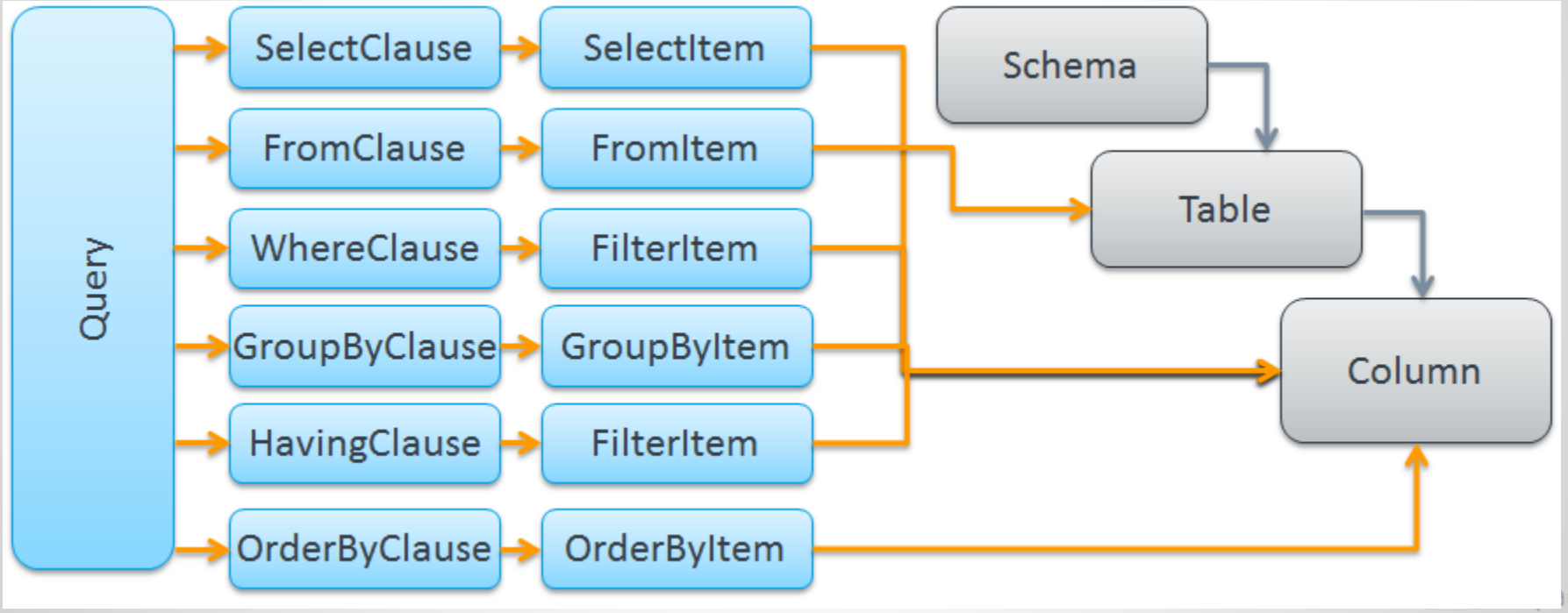
Apache MetaModel

MetaModel schema model



Apache MetaModel

MetaModel query and schema model



Query representation for developers?

```
"SELECT * FROM foo"
```

Query as a string

- Easy to write.
- Easy to read.
- Prone to parsing errors.
- Prone to typos etc.
- Fails at runtime.

```
Query q = new Query();
q.from(table).selectAll();
```

Query as an object

- More involved code to read and write.
- Lends itself to inspection and mutation.
- Fails at compile time.

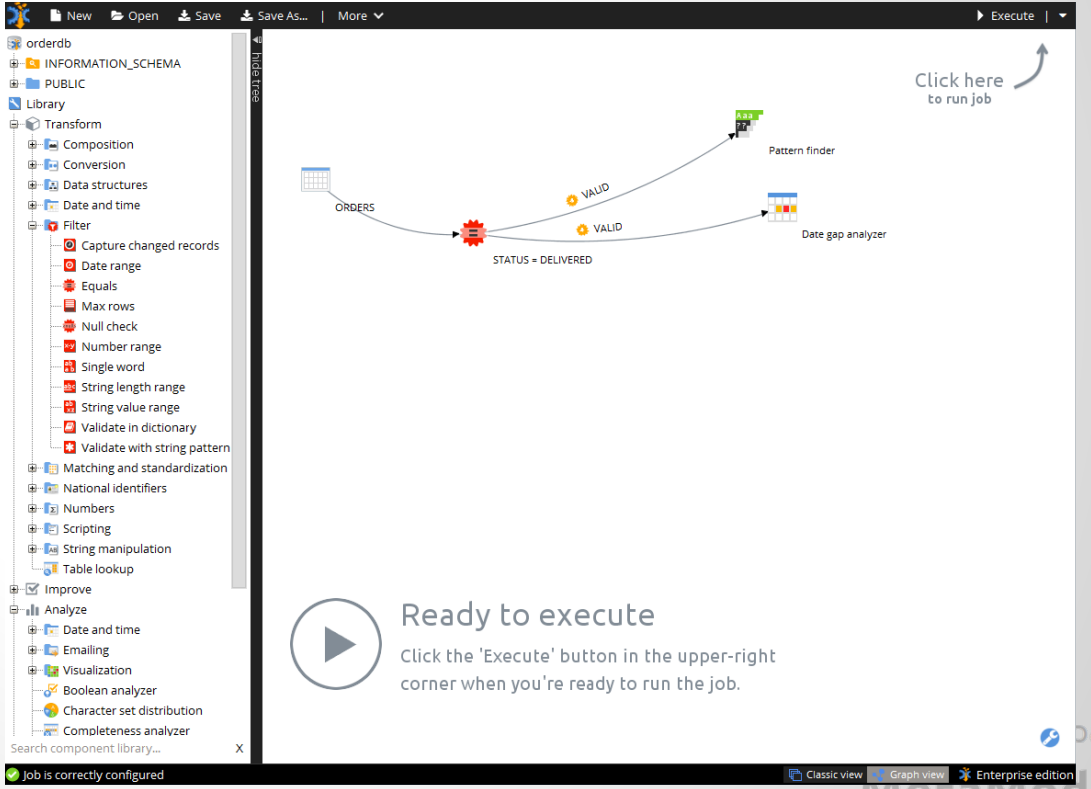
Apache MetaModel

Context-based **Query** optimization

You might not always need to pass data around between components …

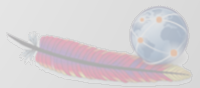
Sometimes you can just pass the query around!

This `STATUS=DELIVERED` filter never actually executes, it just updates the query on the `ORDERS` table.



The role of Apache MetaModel in Big Data

So how does this all **relate** to Big Data**?**

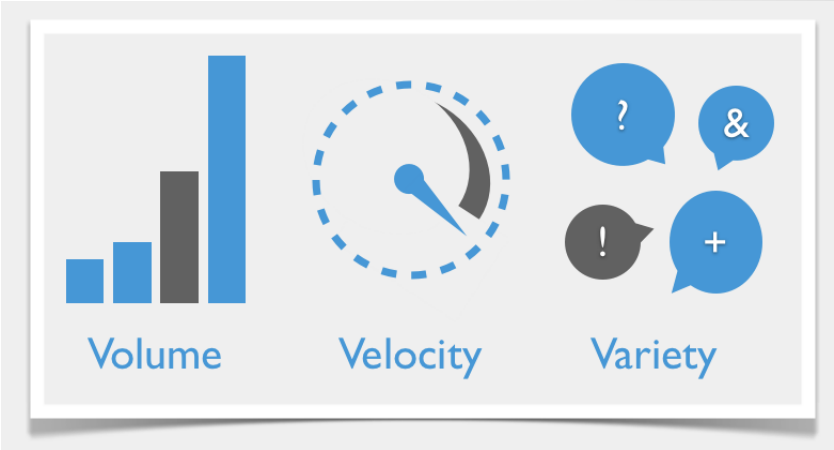


Apache MetaModel

Big Data and the need for metadata

Variety

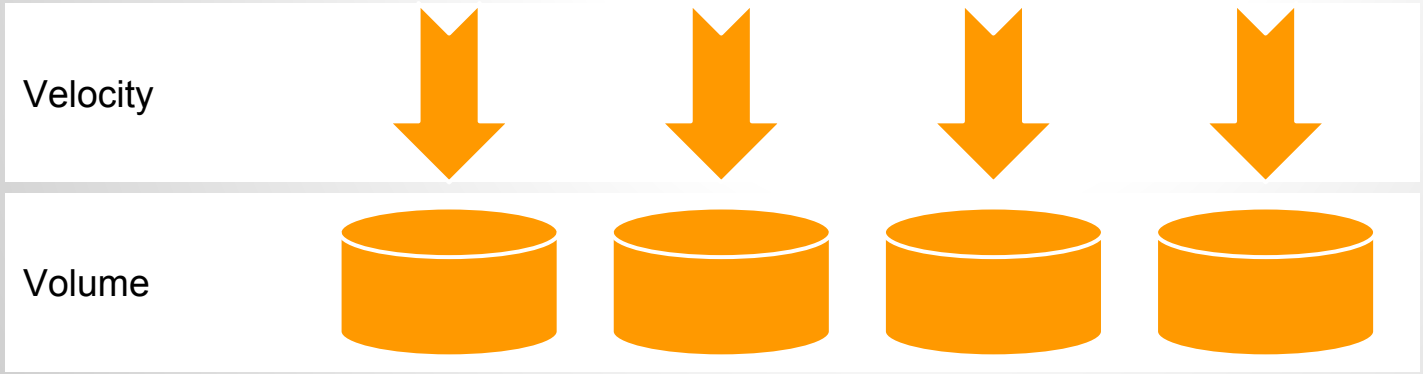
- Not only structured data
- Social
- Sensors
- Many new sources, but also the “old”:
  - Relational
  - NoSQL
  - Files
  - Cloud



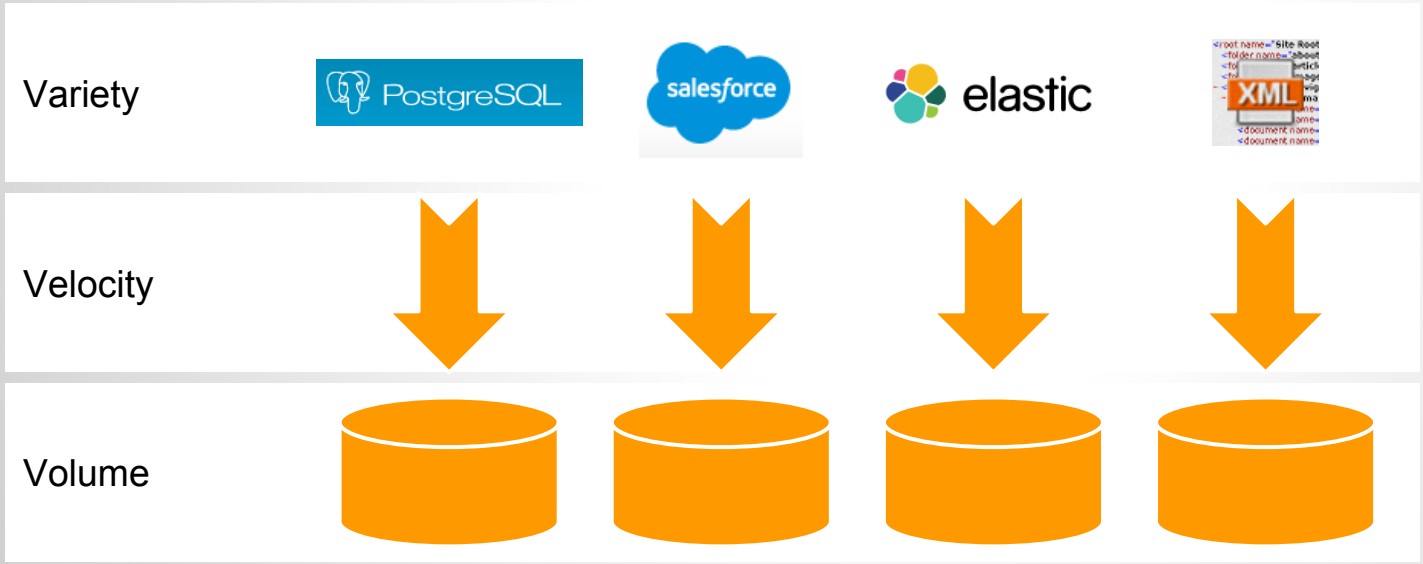
Apache MetaModel

Volume

Apache MetaModel



Apache
MetaModel



Apache MetaModel

Veracity ?

Variety

PostgreSQL

salesforce

elastic

Velocity

Volume

Apache MetaModel

Query language examples

SQL

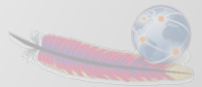
```
SELECT * FROM customers
WHERE country_code = 'GB'
OR country_code IS NULL
```

MongoDB

```
db.customers.find({
        $or: [
                {"country.code": "GB"},
                {"country.code": {$exists: false}}
        ]
})
```

CSV

```
for (line : customers.csv) {
        values = parse(line);
        country = values[country_index];
        if (country == null || "GB".equals(country) {
                emit(line);
        }
}
```



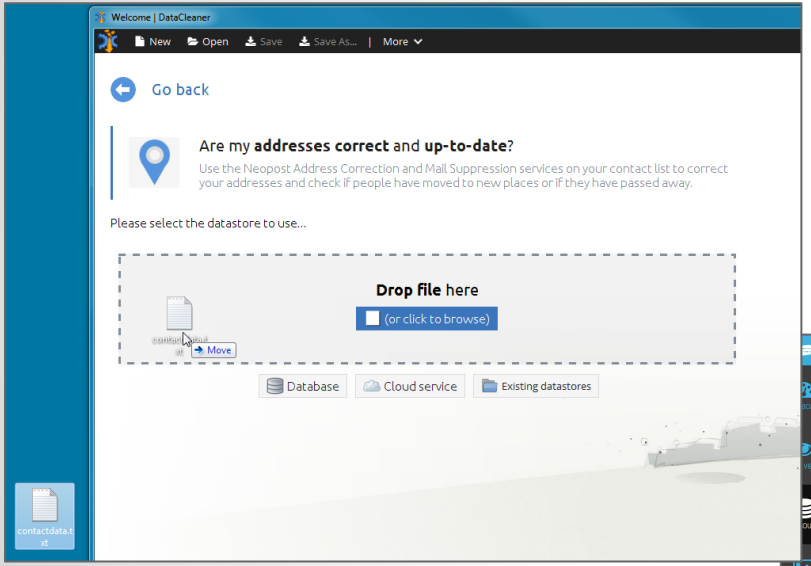
Apache MetaModel

Query language examples

Any datastore

```
dataContext.query()
      .from(customers)
      .selectAll()
      .where(countryCode).eq("GB")
      .or(countryCode).isNull();
```

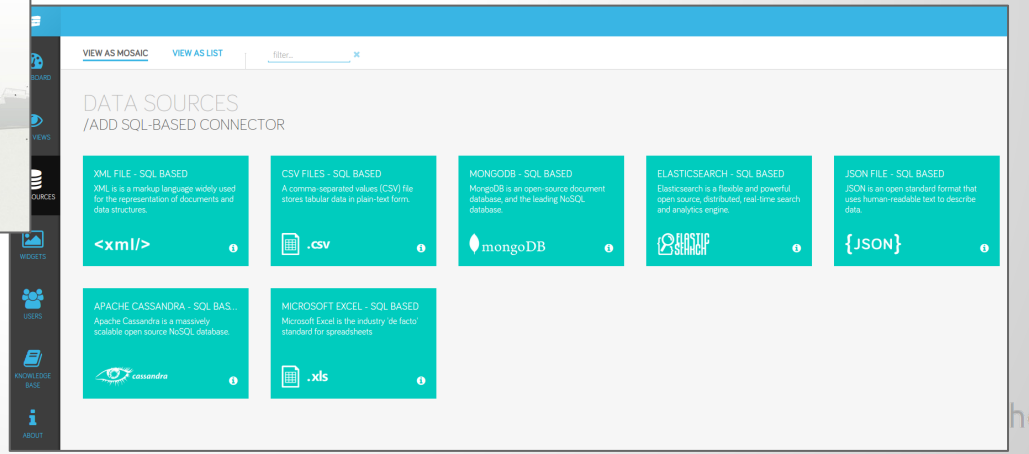
Make it easy to ingest data in your lake



DataCleaner
The premier data quality solution

(commercial) open source

STRATIO



he
MetaModel

Metadata to enable automation

Big Data **Variety** and **Veracity** means we will be handling:

- Different physical formats of the same data
- Different query engines
- Different quality-levels of data

How can we automate data ingestion in such a landscape?

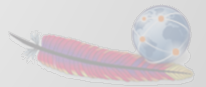
Metadata to enable automation

Big Data **Variety** and **Veracity** means we will be handling:

- Different physical formats of the same data
  We need a uniform metamodel for all the datastores.
  And enough metadata to infer the ingestion transformations needed.

- Different query engines
  We need a uniform query API based on the metamodel.

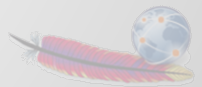
- Different quality-levels of data
  We need our ingestion target to be aware of the ingestion sources.



Apache
MetaModel

Traditional view on metadata

user			
id (pk)	CHAR(32)	java.lang.String	not null
username	VARCHAR(64)	java.lang.String	not null, unique
real_name	VARCHAR(256)	java.lang.String	not null
address	VARCHAR(256)	java.lang.String	nullable
age	INT	int	nullable



Apache MetaModel

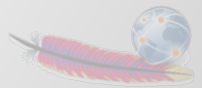
Traditional view on metadata

user

id (pk)	CHAR(32)	java.lang.String	not null
username	VARCHAR(64)	java.lang.String	not null, unique
real_name	VARCHAR(256)	java.lang.String	not null
address	VARCHAR(256)	java.lang.String	nullable
age	INT	int	nullable

customer

id (pk)	BIGINT	long	not null
firstname	VARCHAR(128)	java.lang.String	not null
lastname	VARCHAR(128)	java.lang.String	not null
street	VARCHAR(64)	java.lang.String	not null
house number	INT	int	not null



Apache MetaModel

A more **elaborate** metadata view

user

id (pk)	CHAR(32)	java.lang.String	not null
username	VARCHAR(64)	java.lang.String	not null, unique
real_name	VARCHAR(256)	java.lang.String	not null
address	VARCHAR(256)	java.lang.String	nullable
age	INT	int	nullable

customer

id (pk)	BIGINT	long	not null
firstname	VARCHAR(128)	java.lang.String	not null
lastname	VARCHAR(128)	java.lang.String	not null
street	VARCHAR(64)	java.lang.String	not null
house number	INT	int	not null

32 char UUID

Person full **name**

Address (unstructured)

Person age
Numeric ratio variable

Same real-world entity?

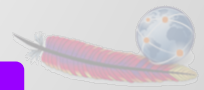
Person first **name**

Person last **name**

Address part - street

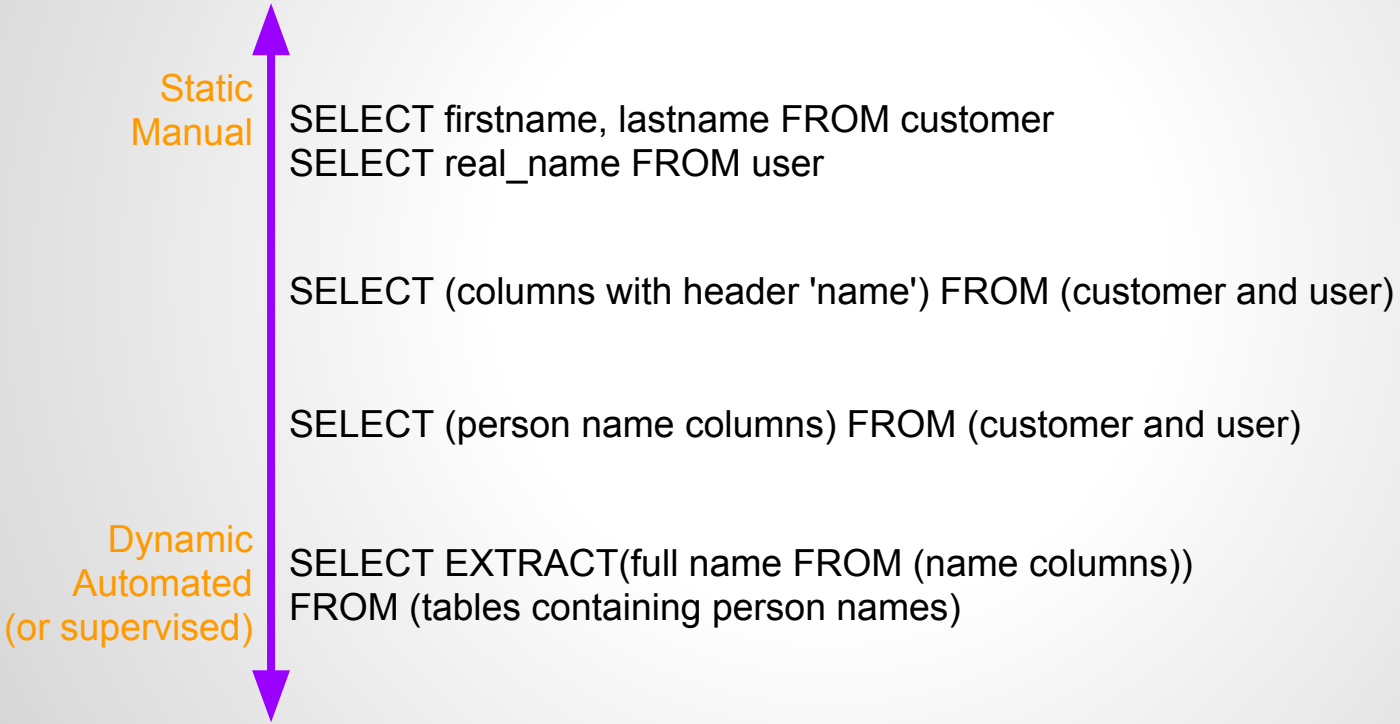
Address part - house number

Numeric nominal variable



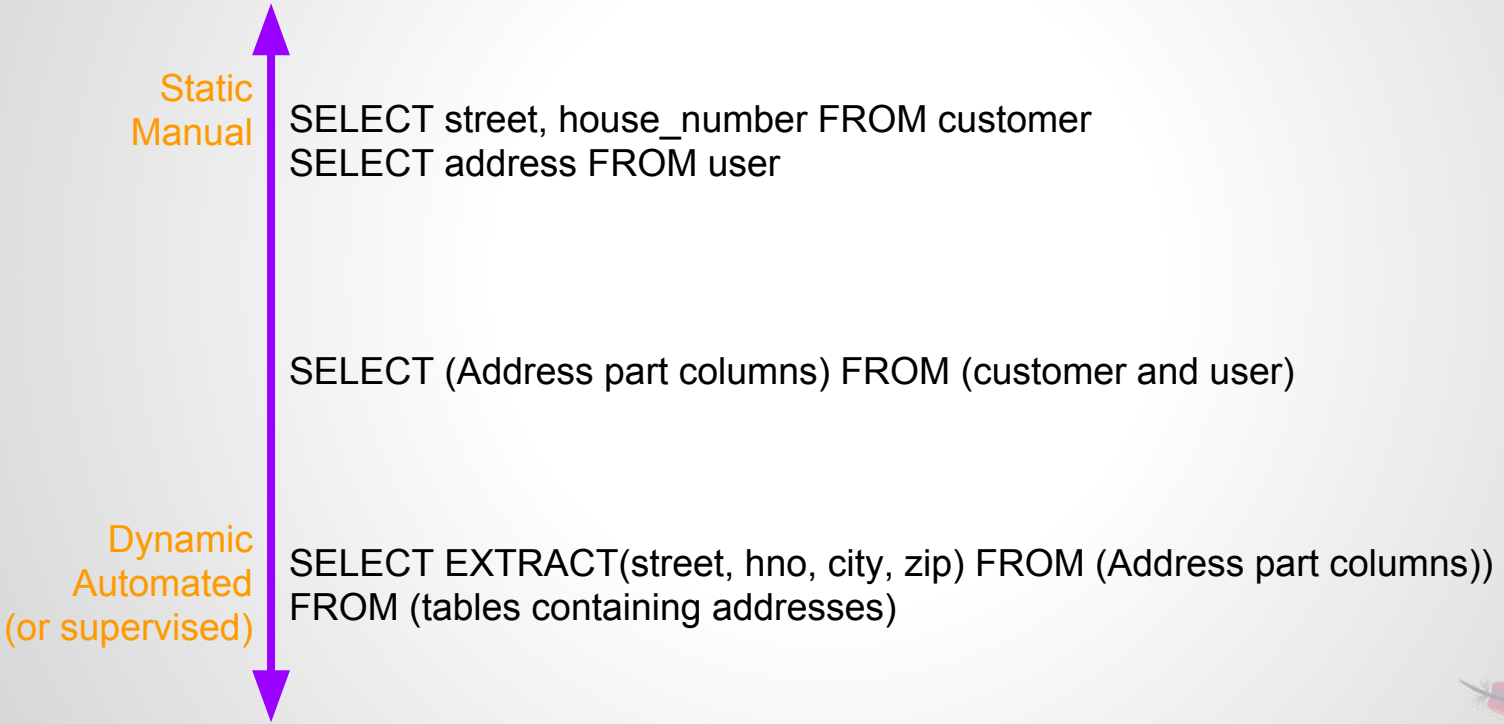
Apache MetaModel

Elaborate **metadata** and querying



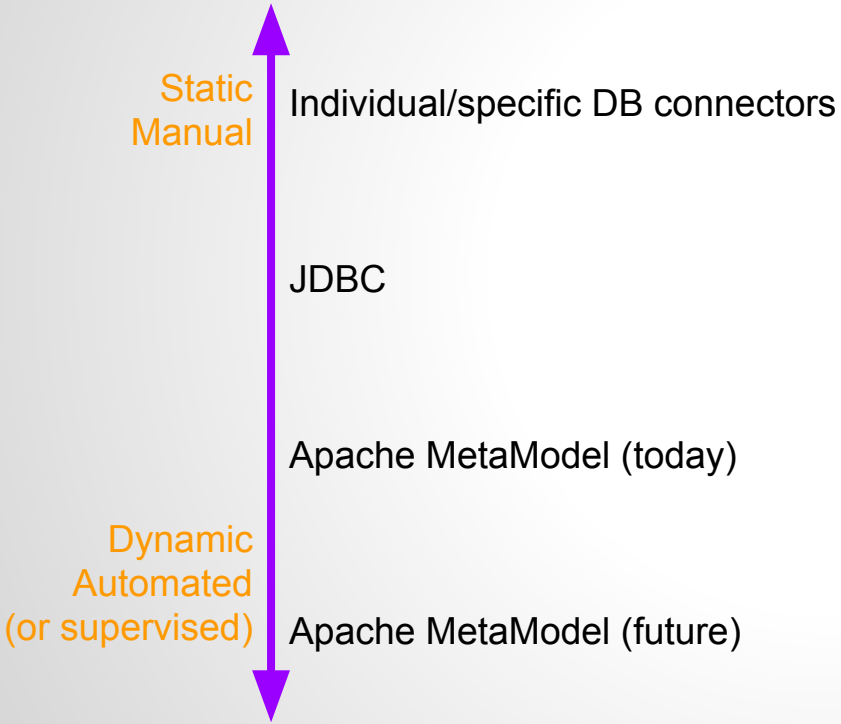
Apache MetaModel

Elaborate **metadata** and querying



Apache MetaModel

Elaborate **metadata** and querying



Apache MetaModel

What about **speed**?

Performance

- Performance *is* important in development of MetaModel.
- But not in favor of uniformness.
- In some cases the metadata may benefit performance by automatically tweaking query parameters (e.g. fetching strategies).
- We usually expose the native connector objects too.

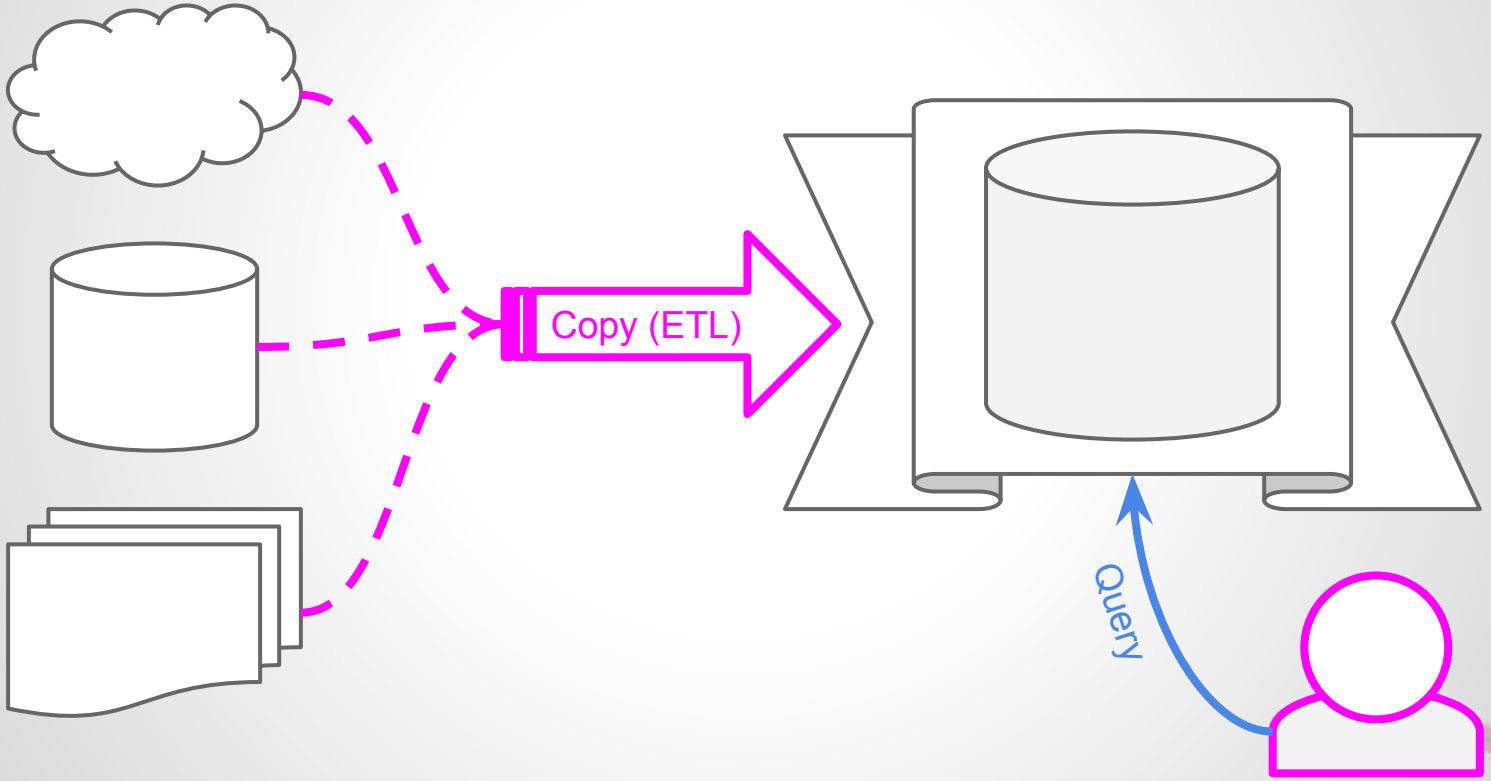
Time to market

- MetaModel makes it easy to cover all the data sources with the *same* codebase.
- Typical 80/20 trade-off scenario.
- Avoid premature optimization.

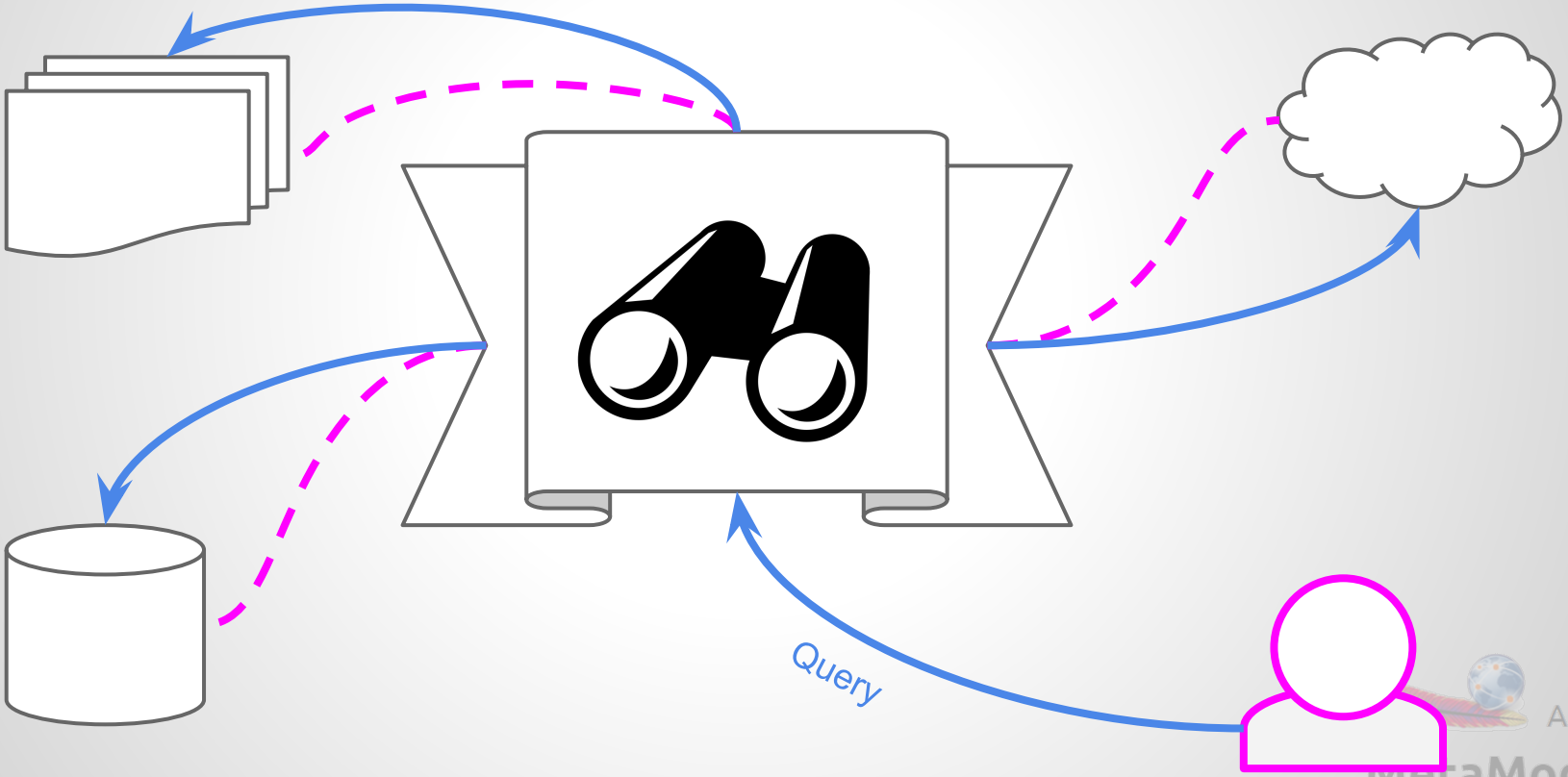
(New) architectural possibilities

with Apache MetaModel

Data **Integration** scenario



Data **Federation** scenario



Apache MetaModel

Let's see some code

Query a remote CSV file

https://github.com/kaspersorensen/ApacheBigDataMetaModelExample

Apache MetaModel

Thank you

Questions?

Apache MetaModel