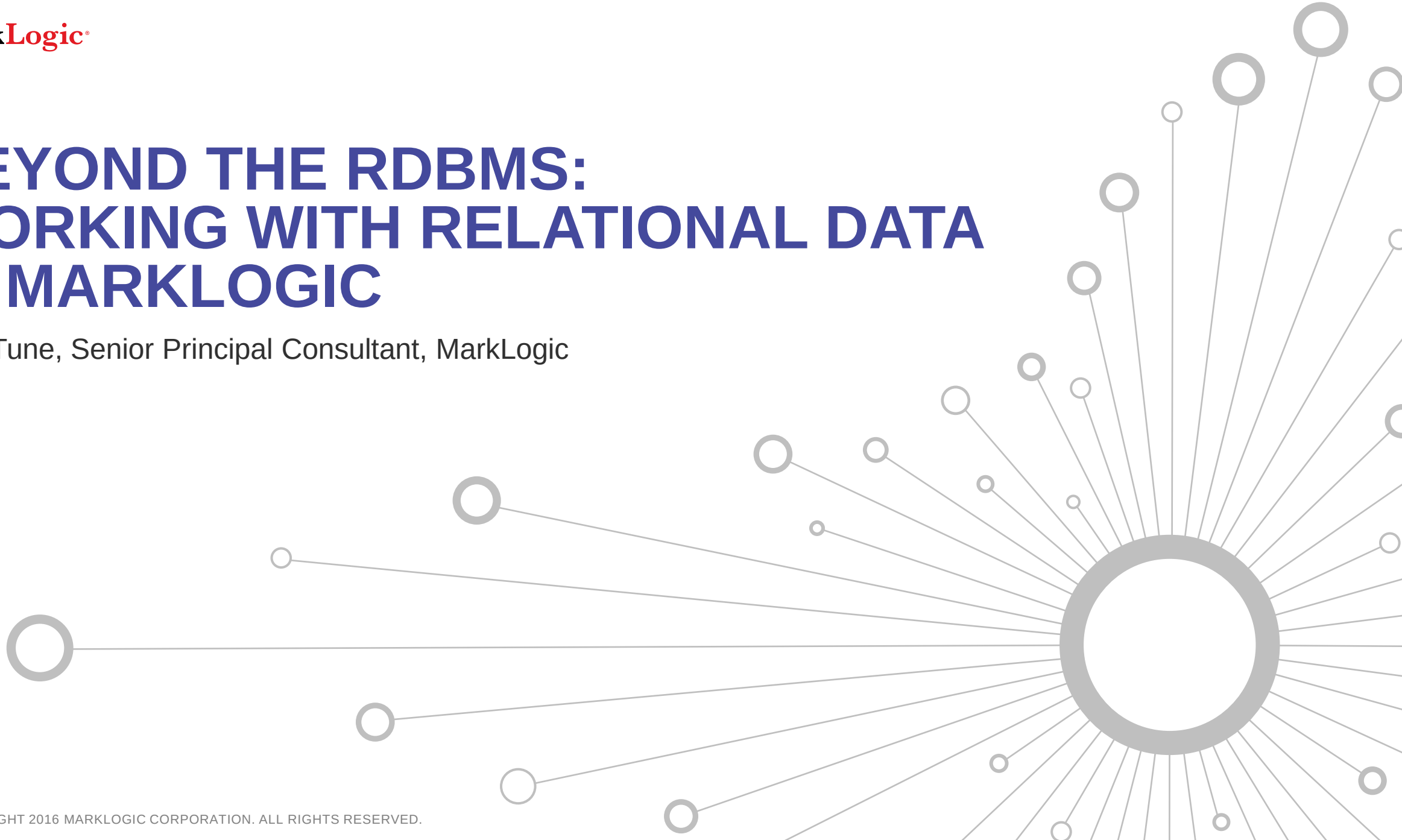
MarkLogic

BEYOND THE RDBMS: WORKING WITH RELATIONAL DATA IN MARKLOGIC

Ken Tune, Senior Principal Consultant, MarkLogic



MarkLogic

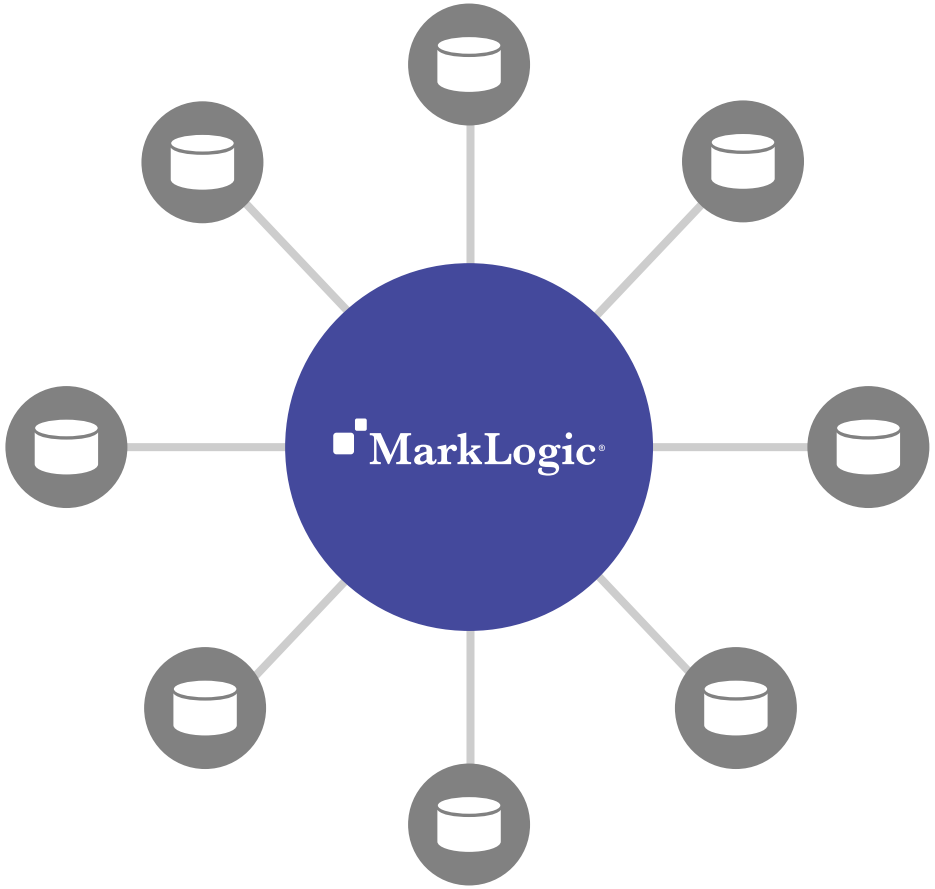
Agenda

- Personal introduction
- Motivation – the Operational Data Hub
- Modeling data in MarkLogic
- Three worked examples demonstrating different migration patterns
- BI Tool integration
- Wrap Up & Q&A

MarkLogic

Introduction

- Ken Tune
- Senior Principal Consultant at MarkLogic for ~5 years
- Background : dev lead / system architecture and design
- Strong background in relational technology
- Source code for this talk - https://github.com/rjrudin/marklogic-sakila-demo



OVERVIEW

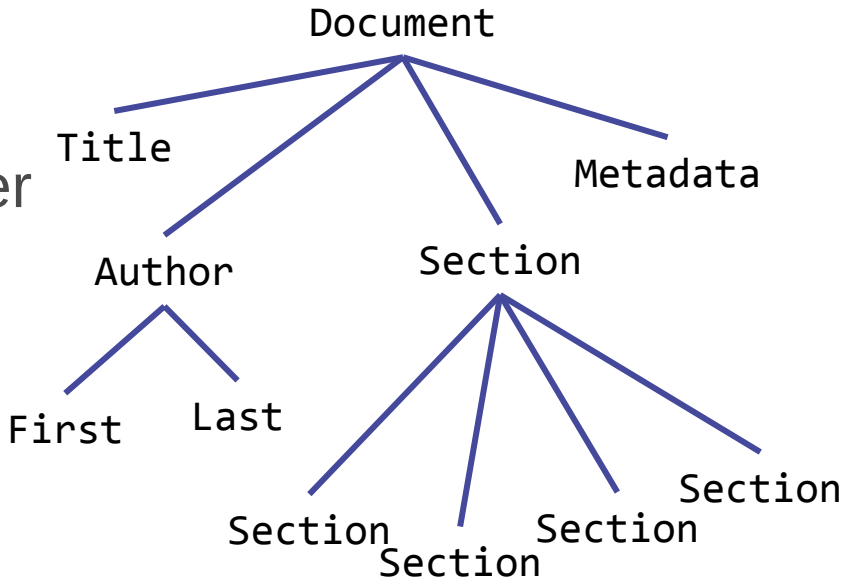
MarkLogic as an Operational Data Hub

- Integrate data from multiple heterogeneous stand-alone sources
- Do more with that data in aggregate
- Use extensive MarkLogic feature set
- Some of those silos will be RDBMSs

MarkLogic

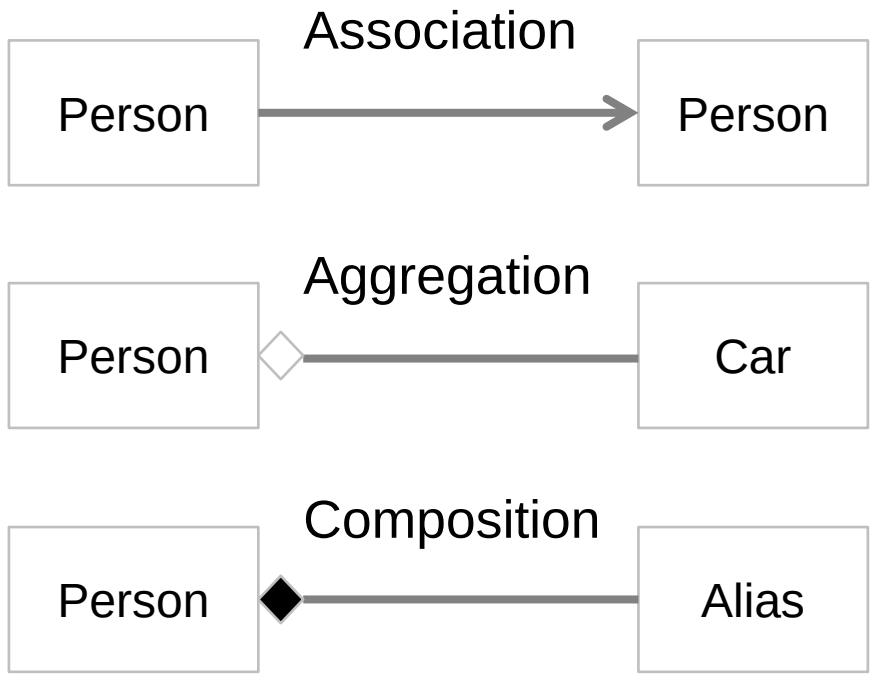
Tables and Documents

- Tables – Tabular!
- Documents – Tree Structures – hierarchical & nested
- Higher dimension implies greater representative power
- We can intuitively transform tables into documents
- Documents offer additional possibilities



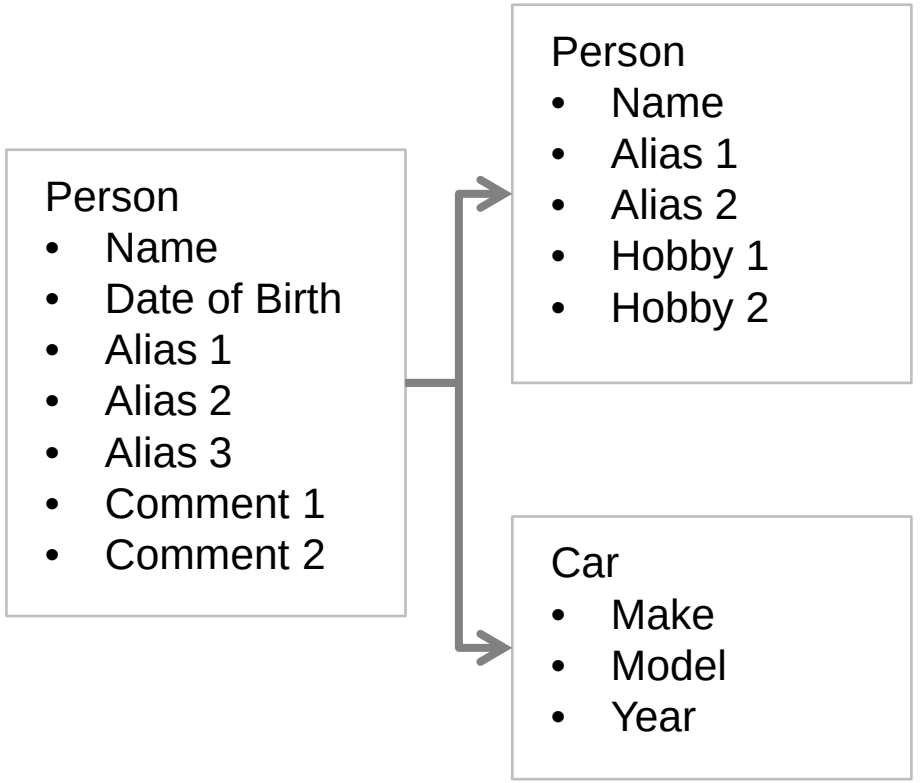
Entities and relationships

- Our applications have conceptual models
- A conceptual model has entities with different kinds of relationships
- Relational forces us to break up our model into separate tables for every 1:many relation
  - Sometimes makes sense, but not always
  - We never have a choice



Documents – greater flexibility

- Choose how you model relationships
- Model is more intuitive
- Model has functional benefits
- Model accommodates heterogeneous data
- Triples!



MarkLogic

Document database - advantages

- Can query the entire database, not limited to a single table
- Collections allow creation of non-invasive taxonomies
- Schema-agnostic means we can ingest different datasets without modeling them the same way - **Key aspect of the data integration story**
- Strong support for metadata – including properties fragments and triples
- Transactional support allows integrity enforcement and distributed transaction support.
- Extensive support for data transformation

MarkLogic

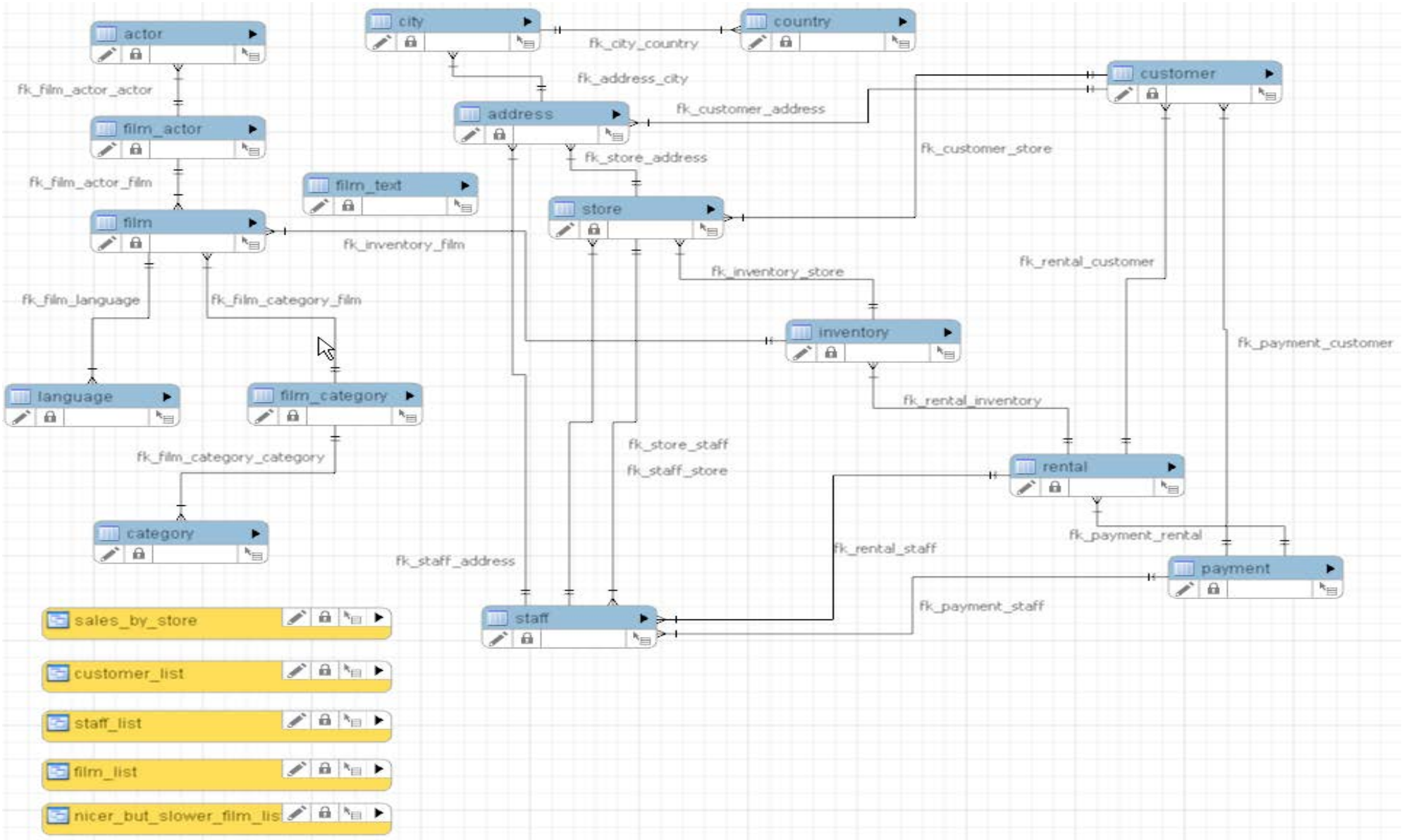
Scenario summary

- You have multiple stand-alone databases (silos)
- You'd like to integrate that data
- You can do it by bringing it into MarkLogic
- MarkLogic holds documents
- Next step : how do we go from tables to documents?

MarkLogic

Our sample data

- MySQL Sakila dataset
  - https://dev.mysql.com/doc/sakila/en/
  - Represents a DVD rental store
  - Used for testing MySQL-specific functionality and new features
  - Pre-loaded in MySQL install
- Perfect for testing out tables -> documents



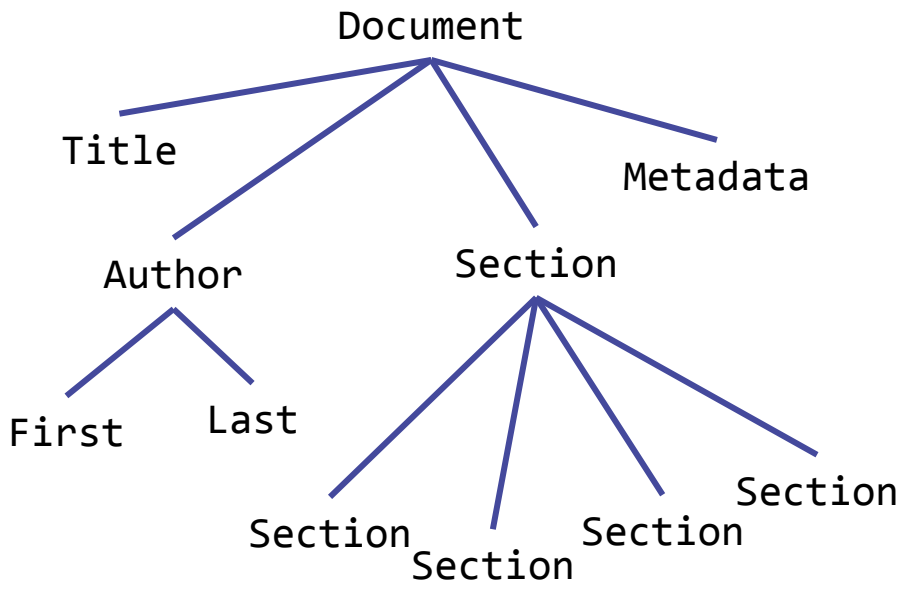
MarkLogic

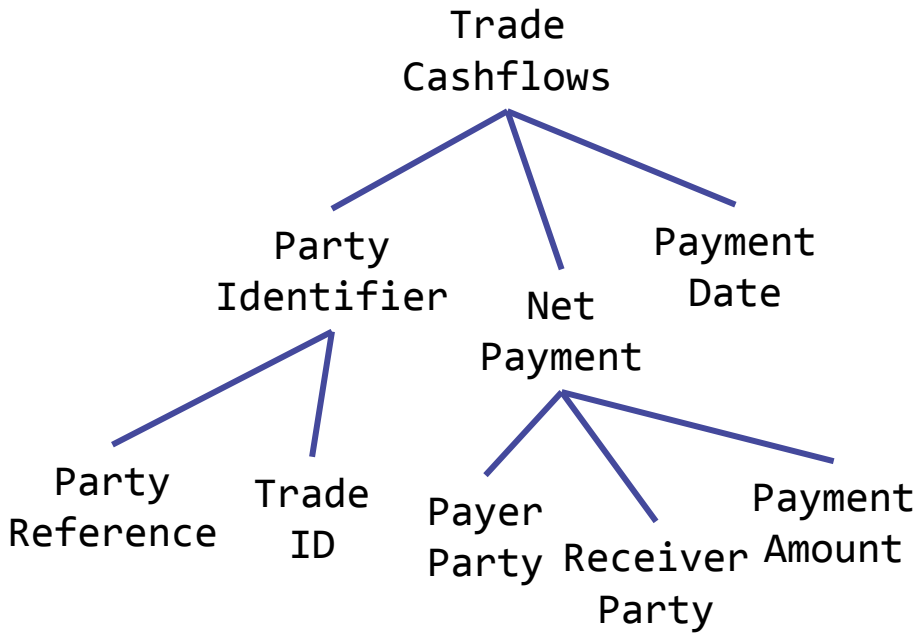
Moving from tables to documents

- Need to understand data modeling options in MarkLogic
- Need to identify entities
  - What are the high level nouns in our system?
  - Those typically drive the types of documents we'll have

MarkLogic Data Modeling 101

- MarkLogic is a document-oriented database
  - Supports any-structured data via hierarchical data model





MarkLogic

MarkLogic Data Modeling 101

- Documents have URIs
  - Can be any string, they just need to be unique
  - Analogous to a primary key, but scoped to the entire database as opposed to a table

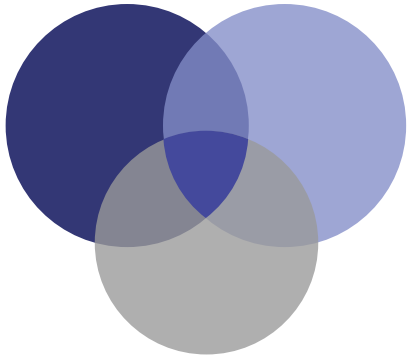
MarkLogic

MarkLogic Data Modeling 101

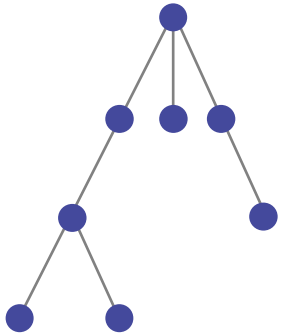
- Prefer UUIDs over sequence numbers
  - Can use sem:uuid-string()
  - Avoids namespacing problems; i.e. ID of “14” from different databases
- Often looks like a path, but not required to be
  - Example - /film/123.xml or just 123.xml or just 123

MarkLogic

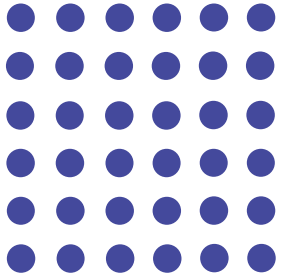
MarkLogic Data Modeling 101



COLLECTIONS
Set-based, n:m relationship



DIRECTORIES
Exclusive, hierarchical, analogous to file system
Based on URI



SECURITY
Roles and document-level permissions invisible to your app

MarkLogic

Identifying entities

- What are the main nouns that users talk and ask questions about?
- For our DVD rental store:
  - As a customer, I want to search for films with certain actors
  - As a customer, I want to find a store with a particular film
  - As a staff member, I want to find overdue rentals
- Nouns – films, actors, stores, rentals, customers, staff members
- Those are the collections of documents we'll have

MarkLogic

Let’s start migrating data!

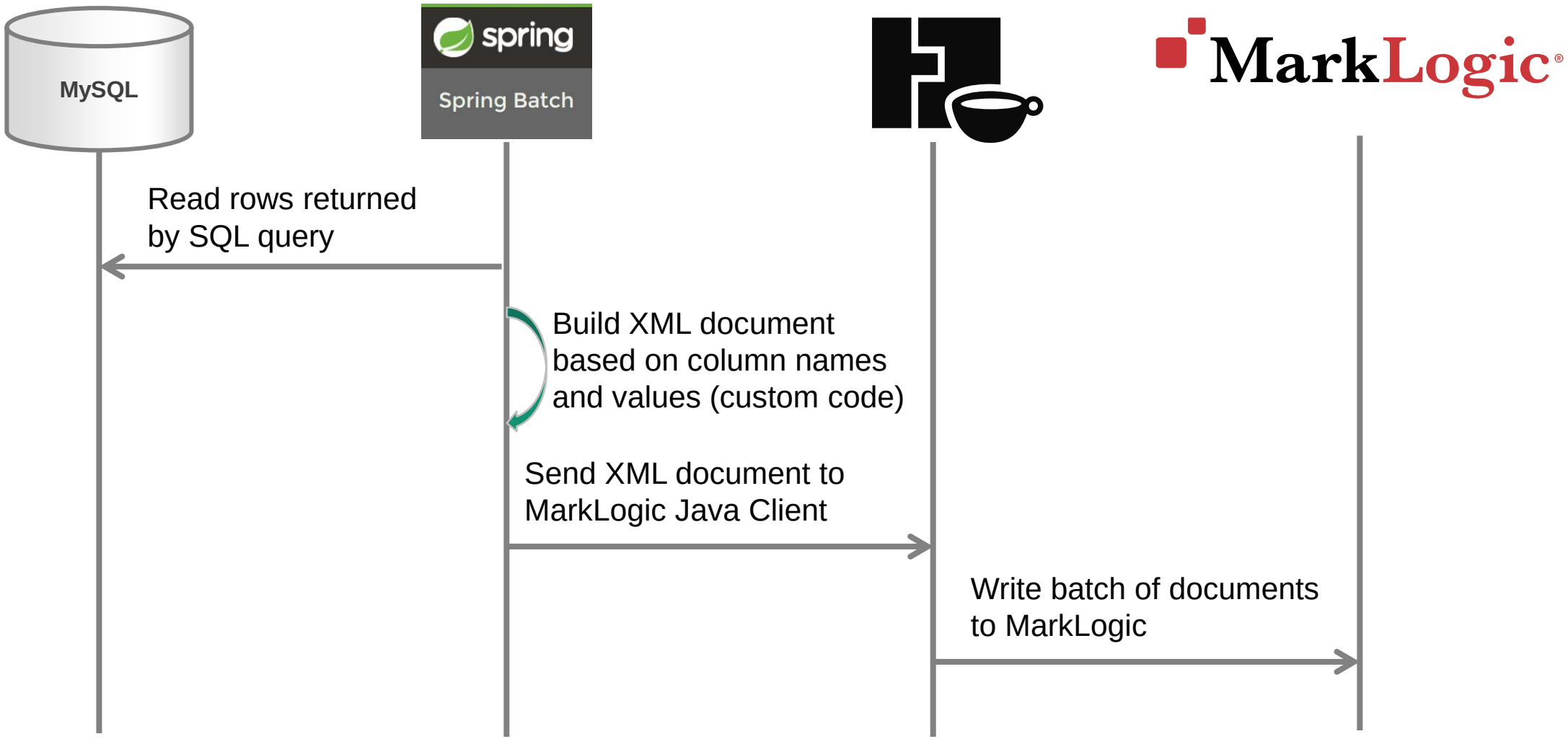
- Demo created using a slush generator
  - https://github.com/marklogic/slush-marklogic-node (slush)
  - https://github.com/rjrudin/slush-marklogic-spring-boot (adapted slush)
  - Angular / Spring Boot / MarkLogic
- Spring Boot middle tier uses Spring Batch to migrate data

MarkLogic

Spring Batch for bulk migrations

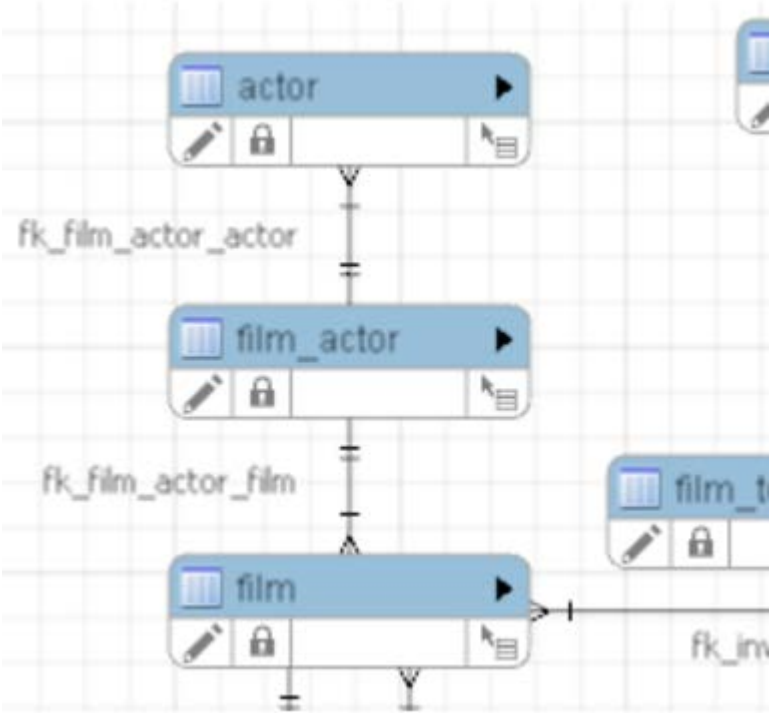
- Many commercial and open source tools exist to help out
- We'll use Spring Batch for a demo here
- It's Java, so it integrates easily with MarkLogic tools
  - MarkLogic Java API
  - Content Pump

Demo architecture

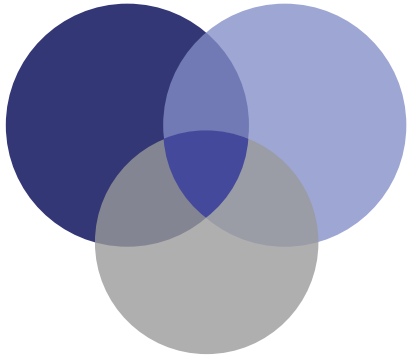


Analyzing actors

- Actor is an entity
  - So create an “actor” document
- SQL query:
  - SELECT * FROM actor

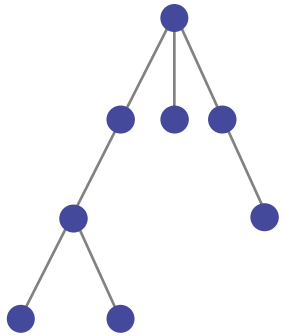


Modeling actors



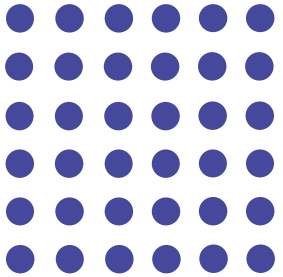
COLLECTIONS

“actor”, “sakila”



DIRECTORIES

“/actor/(uuid).xml”



SECURITY

“dvd-store-reader/read” and “dvd-store-writer/update”

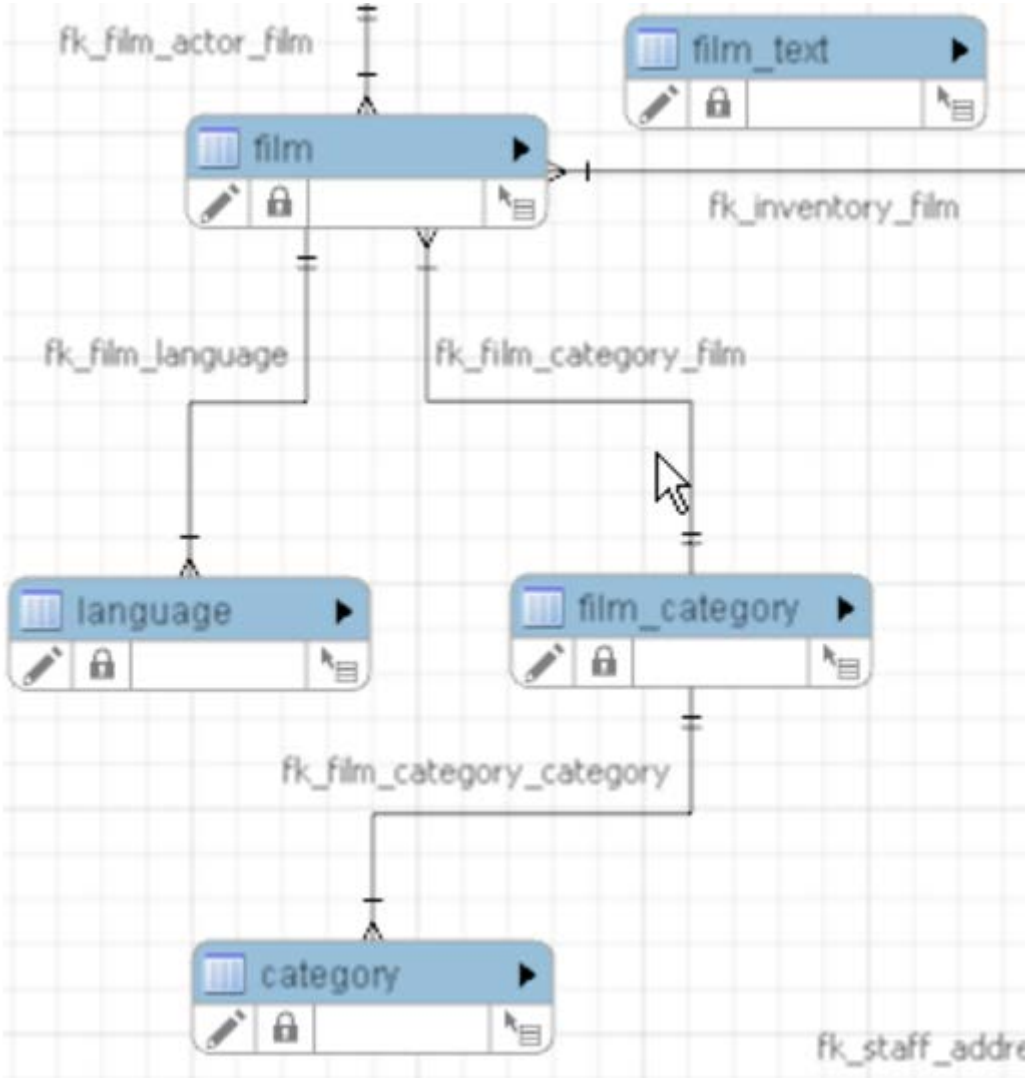
MarkLogic

More about our actor documents

- Each column in a row becomes an element name
  - Simple approach, often a good starting point
- Can analyze it and determine what more to do
- URI and collections allow for future “actor” datasets
  - Example – pull in IMDB set of actors with collections of “actor” and “imdb”
  - Could have completely different structure

Analyzing films

- A film is an entity
- So create a “film” document
- Tables – films, film_text, language, film_category, category
  - Conceptually, we feel all of those belong together

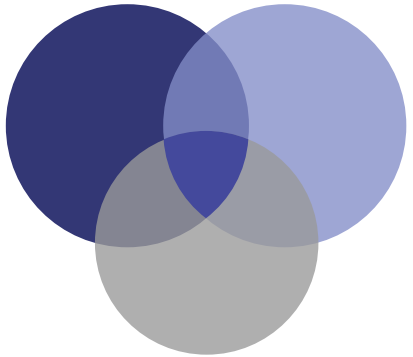


MarkLogic

Migrate films

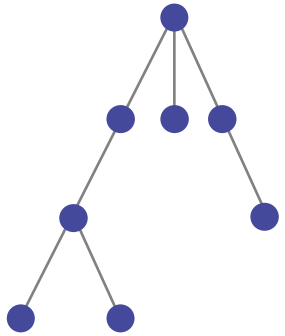
```
SELECT film.*, film_text.description as filmText, category.name as category,
language.name as language

FROM film

LEFT JOIN film_category ON film.film_id = film_category.film_id

LEFT JOIN category ON film_category.category_id = category.category_id

LEFT JOIN film_text ON film.film_id = film_text.film_id

LEFT JOIN language ON film.language_id = language.language_id

ORDER BY film.film_id
```

Modeling films



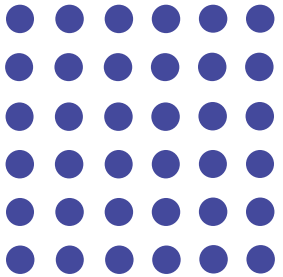
COLLECTIONS

“film”, “sakila”



DIRECTORIES

“/film/(uuid).xml”



SECURITY

“dvd-store-reader/read” and
“dvd-store-writer/update”

Benefits of our film document

- We merged 5 tables into a single document
  - Physical model mirrors the conceptual model
- Viewing and updating doesn't require any joins
- More importantly, searching doesn't either
  - Can easily get word query hits on film text
  - Can easily build facets on rating and category

MarkLogic

Lookup data is different in MarkLogic

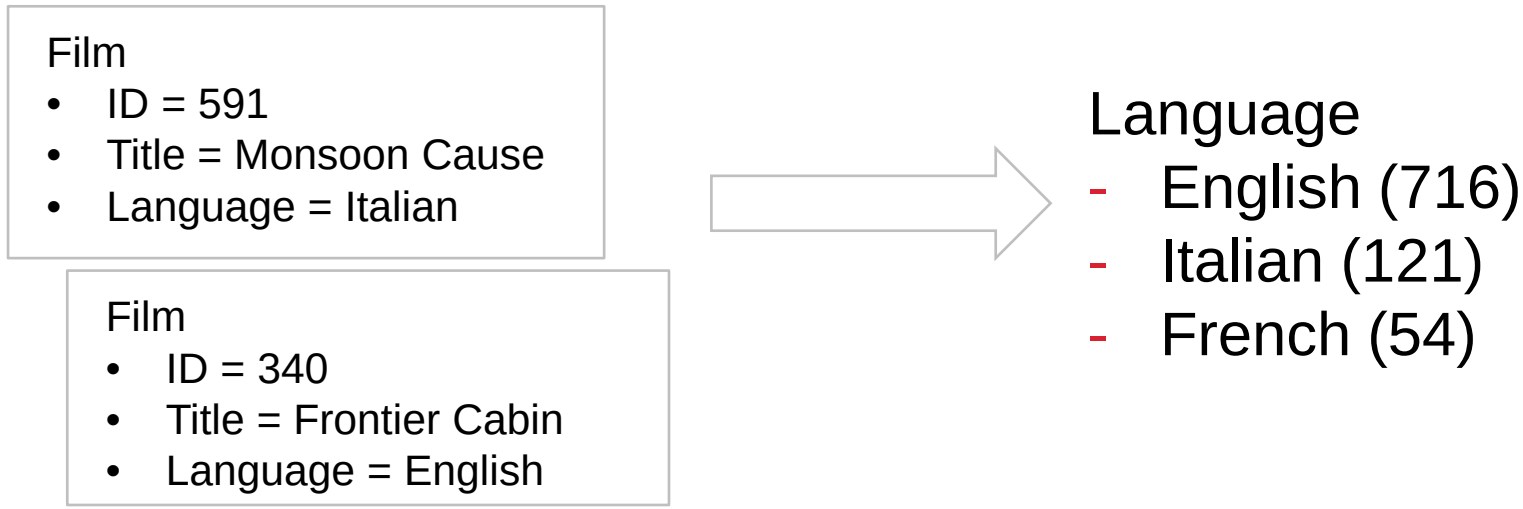
- Common to have lookup tables in relational databases
- How often is this data updated vs how often is it searched?
- Updates are optimized in favor of searches

film_id	title	language_id
133	Chamber Italian	2
285	English Bulworth	1

language_id	name
1	English
2	Italian

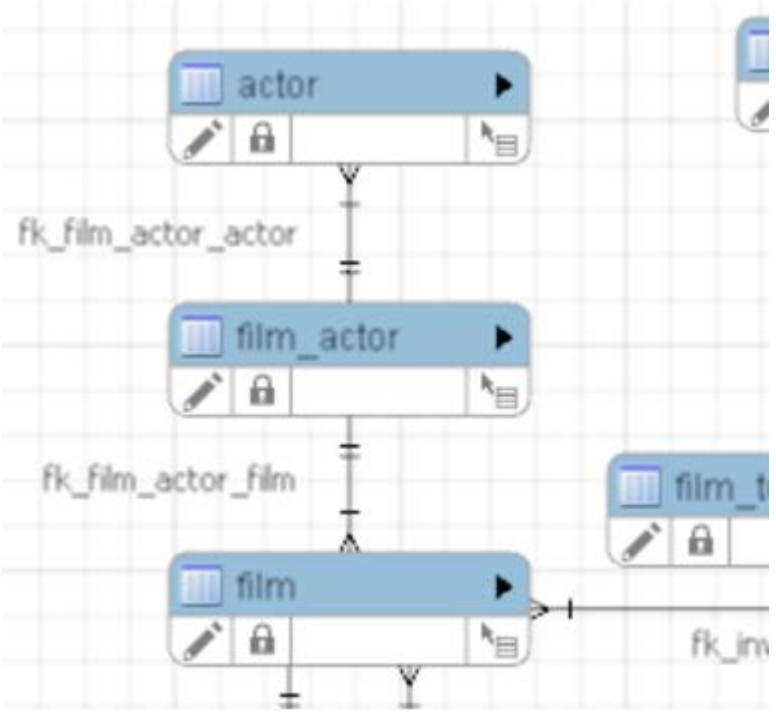
Consider denormalizing lookup data

- Common practice with MarkLogic is to denormalize lookup data
- Favor optimizing search over update
- Much easier to support word queries and facets



Revisiting actors

- We loaded actors “as-is”
- But what about that many to many?



MarkLogic

Analyzing how actors will be queried

- We of course want to support searching “for” actors
  - Find me all actor documents with the word “Uma” in them
- But don’t we also want to search for films “by” actor names?
  - Find me all film documents with the word “Uma” in them
  - Or find me all film documents with a cast member with a first name of “Uma”

MarkLogic

Consider denormalizing some actor data

```
<actors>
  <actor><first-name>Uma</><last-name>Thurman</></>
  <actor><first-name>John</><last-name>Travolta</></>
</actors>
```

- Now we can support efficient word and element queries
- Can still use semantics for other types of questions
  - Find me all films with a cast member who has won a Best Actress Oscar

MarkLogic

Thinking through denormalizing

- Consider the questions your users want to ask
- Consider how often the data that answers those questions will be updated
- Consider the amount of documents affected when the data is updated
- Consider how efficient a join would be
- Consider enriching documents with triples to answer questions via SPARQL
- Get recommendations
  - MarkLogic Professional Services
  - stackoverflow

Migrate films with actors

SELECT film.*, film_text.description as filmText, category.name as category, actor.actor_id as "actor/id", actor.first_name as "actor/firstName", actor.last_name as "actor/lastName"

FROM film LEFT JOIN film_category ON film.film_id = film_category.film_id

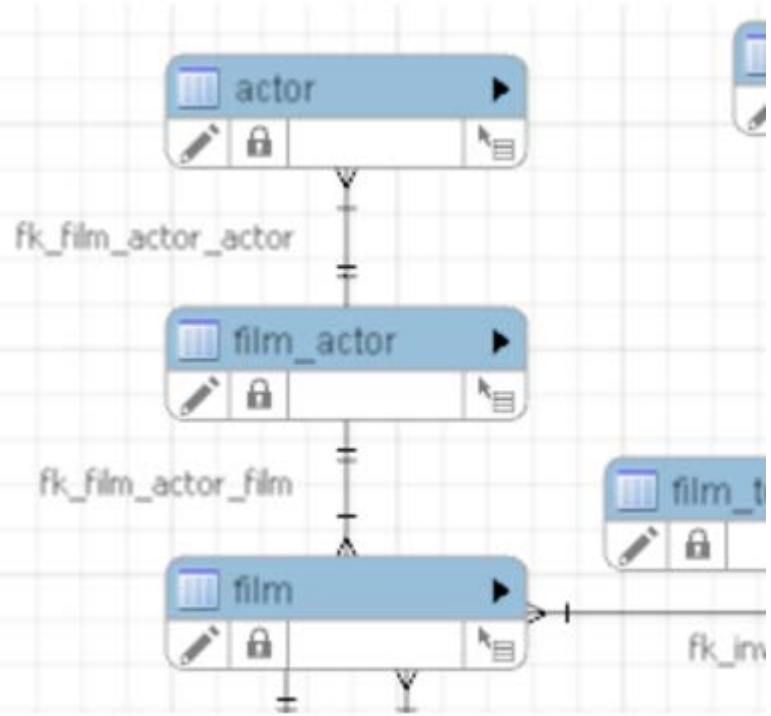
LEFT JOIN category ON film_category.category_id = category.category_id

LEFT JOIN film_actor ON film.film_id = film_actor.film_id

LEFT JOIN actor ON film_actor.actor_id = actor.actor_id

LEFT JOIN film_text ON film.film_id = film_text.film_id

ORDER BY film.film_id



MarkLogic

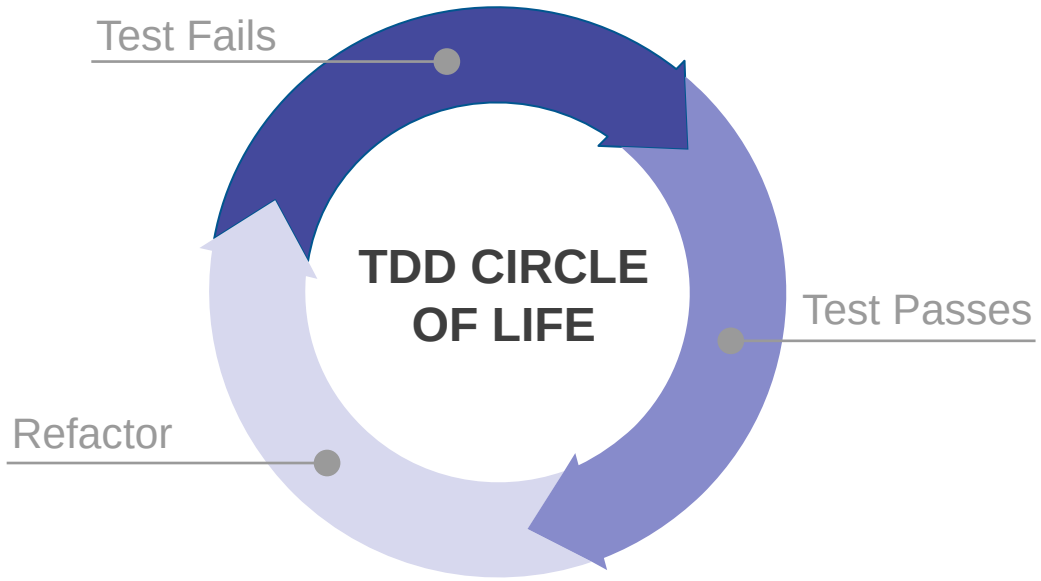
Summary of migrations

- Actors: load as-is, one row per document
- Films: merge multiple tables and rows into one document; denormalize lookup data
- Films and actors: denormalize some actor data onto film documents

- Your migrations will most likely involve all of these techniques

Migrations in the real world

- Focus on migration as soon as possible
- Favor Agile development
  - MarkLogic simplifies loading data
  - Migrate some data, test, iterate



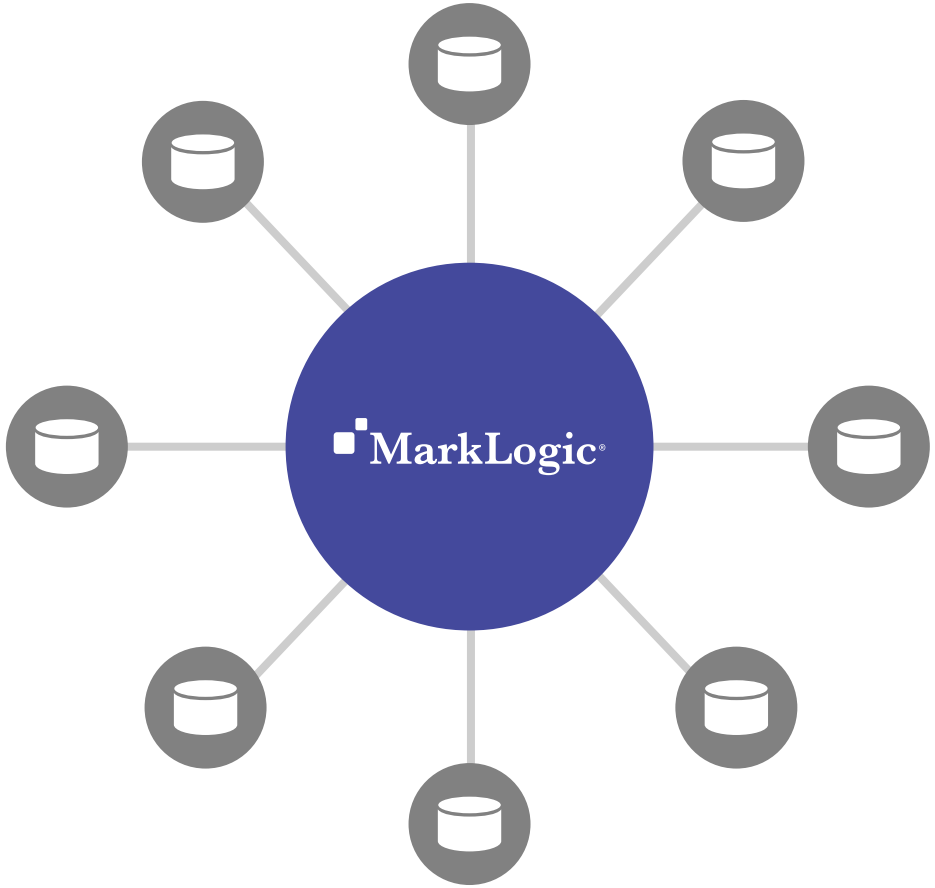
MarkLogic

Export

- Problem
  - BI tools are geared towards relational databases
  - How do we integrate those tools with MarkLogic?
- Solution
  - Use a MarkLogic ODBC server to expose documents as rows and support SQL
- Demo

Summary

- Goal: unified, actionable view of data
- Must migrate data from tables to documents
  - Analyze and identify entities
  - Design a data model for those entities
    - URIs, collections, security
  - Iterate, test, iterate, test
- Realize the benefits of documents in an operational, transactional enterprise NoSQL database



Q&A