CSCI-620

Data Integration

Data integration

- We often need to combine data from multiple sources

- There are multiple strategies for this
  - Data warehousing
  - Federation
  - Data lakes

Data Warehouse

- Where data is stored in a form suitable for analysis and reporting
- Data is often denormalized to make reports faster to generate
- Warehouses usually contain historical data (possibly from multiple sources) that is refreshed periodically

Extract Transform Load

- **Extract** data from multiple sources useful for reporting
- **Transform** the data into a format more suitable for analysis
- **Load** the data into the warehouse

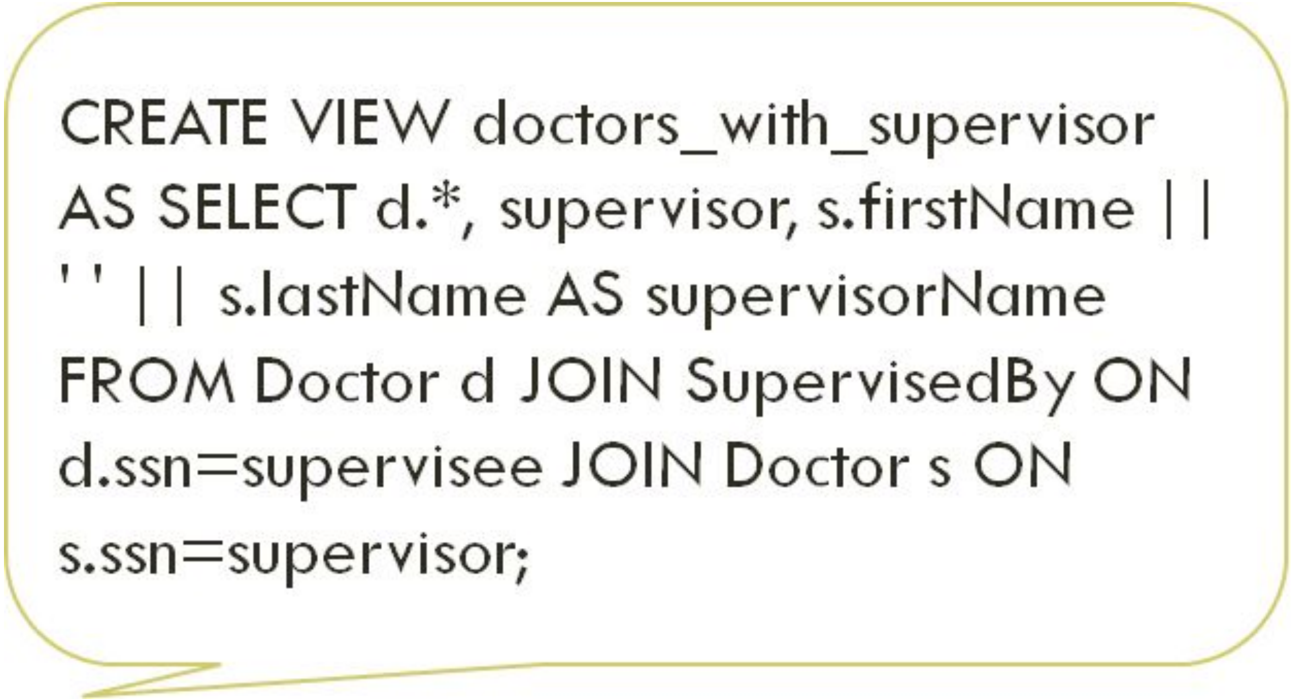
Views

- Creates shortcuts for repeated or common queries

- Views may be *materialized* (stored) to make complex queries faster

- Materialized views must be manually updated or they will return old data

- Some systems will automatically rewrite queries to use materialized views



```
CREATE VIEW doctors_with_supervisor
AS SELECT d.*, supervisor, s.firstName | |
' ' | | s.lastName AS supervisorName
FROM Doctor d JOIN SupervisedBy ON
d.ssn=supervisee JOIN Doctor s ON
s.ssn=supervisor;
```

CREATE VIEW

```
SELECT s.* FROM salaries_by_dept s JOIN
Department d ON d.id=s.department;
```

```
SELECT s.* FROM (SELECT Doctor.department,
MIN(Doctor.salary), MAX(Doctor.salary) FROM
Doctor GROUP BY Doctor.department) s JOIN
Department d ON d.id=s.department;
```

View expansion

```
EXPLAIN SELECT s.* FROM salaries_by_dept s JOIN
Department d ON d.id=s.department;
```

```
 Hash Join
   Hash Cond: (d.id = doctor.department)
   ->  Seq Scan on department d
   ->  Hash
       ->  HashAggregate
             Group Key: doctor.department
             ->  Seq Scan on doctor
```

View expansion

CREATE MATERIALIZED VIEW

```
CREATE MATERIALIZED VIEW
salaries_by_dept AS SELECT
department, MIN(salary),
MAX(salary) FROM Doctor GROUP BY
department;
```

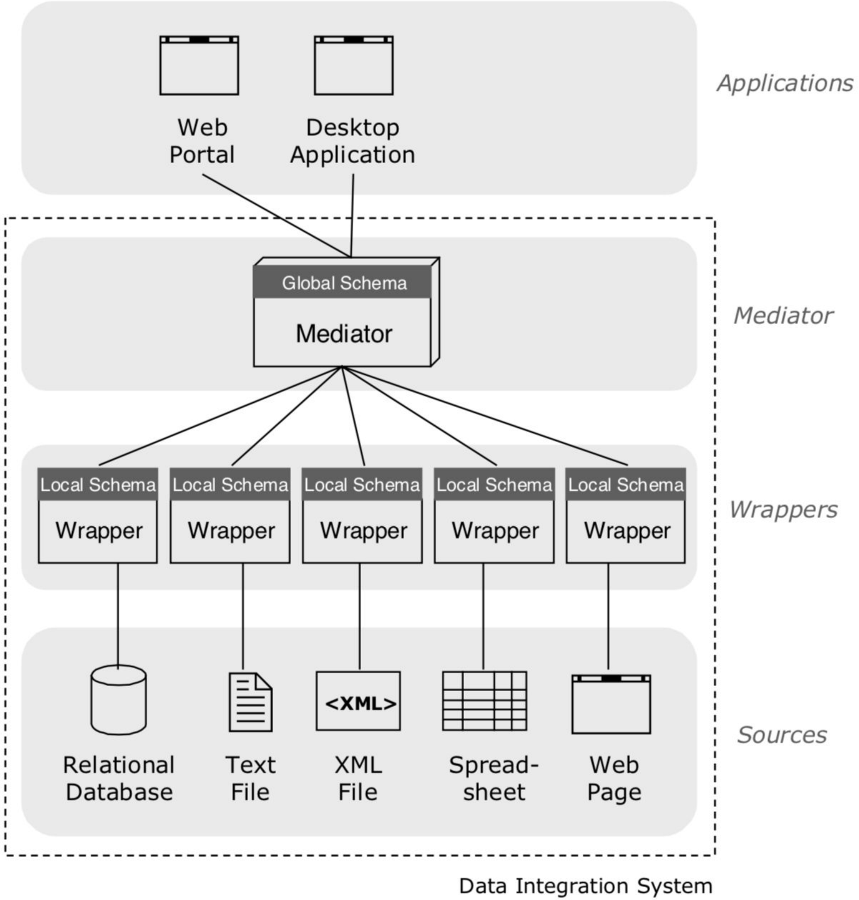
```
EXPLAIN SELECT s.* FROM salaries_by_dept s JOIN
Department d ON d.id=s.department;
```

```
 Hash Join
   Hash Cond: (s.department = d.id)
   ->  Seq Scan on salaries_by_dept s
   ->  Hash
         ->  Seq Scan on department d
```

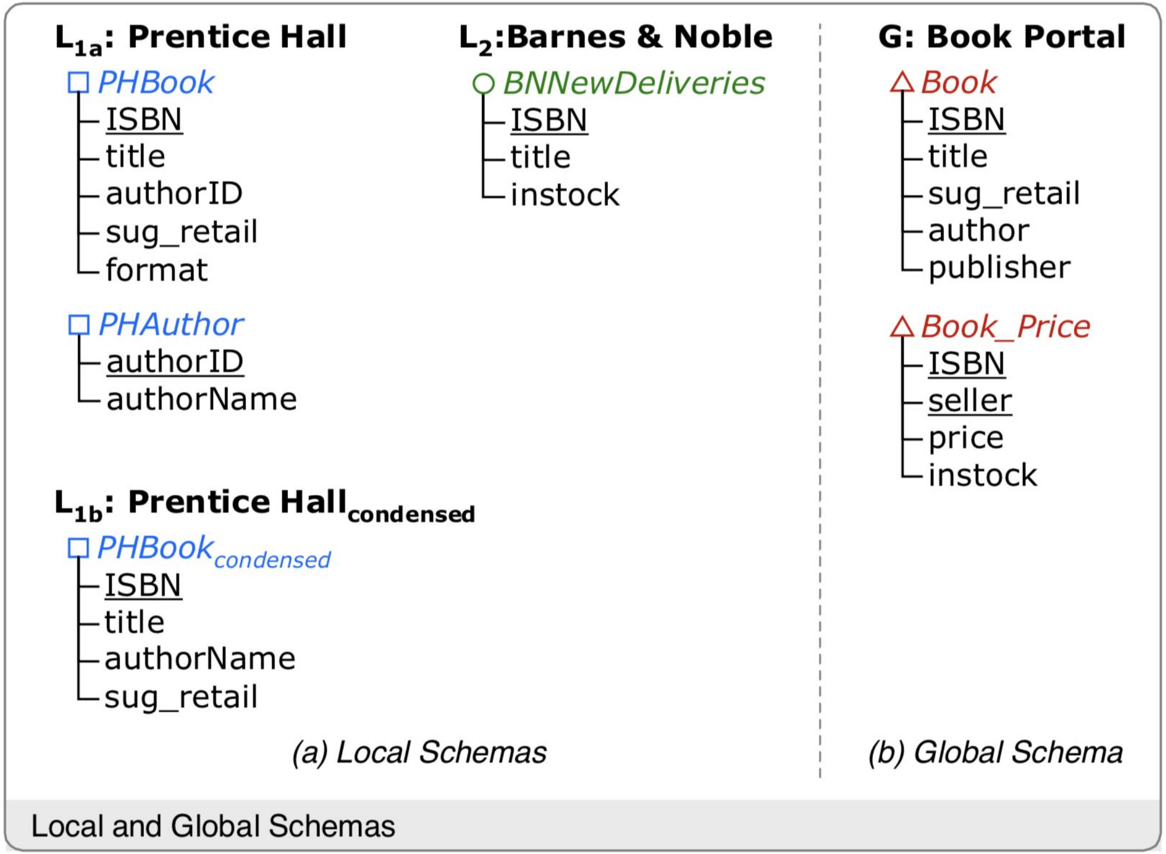
Materialized view querying

Global schema

- When integrating data, our goal is to produce a *global* or shared schema
- We can write the global schema or each of our local schemas as a view



View-based data integration



L_{1a}: Prentice Hall

- □ *PHBook*
  - ISBN
  - title
  - authorID
  - sug_retail
  - format
- □ *PHAuthor*
  - authorID
  - authorName

L_2:Barnes & Noble

- ○ *BNNewDeliveries*
  - ISBN
  - title
  - instock

G: Book Portal

- △ *Book*
  - ISBN
  - title
  - sug_retail
  - author
  - publisher
- △ *Book_Price*
  - ISBN
  - seller
  - price
  - instock

L_{1b}: Prentice Hall$_{\text{condensed}}$

- □ *PHBook$_{condensed}$*
  - ISBN
  - title
  - authorName
  - sug_retail

(a) Local Schemas

(b) Global Schema

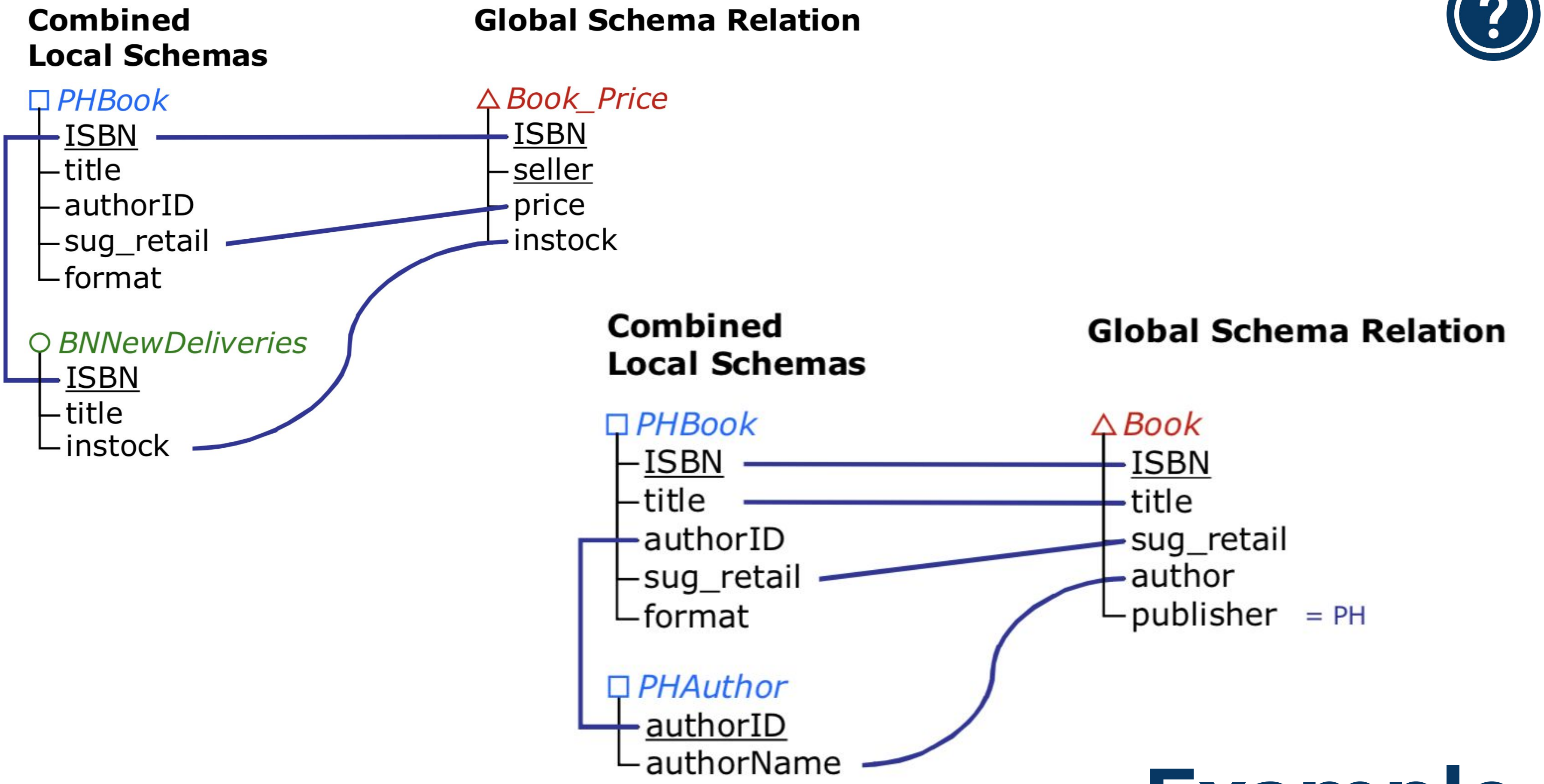
Local and Global Schemas

Example

Global as view (GAV)

- If sources are unlikely to change, we can express our global schema *G* as a view over each data source L_i
- Queries on the global schema are easy to execute based on this mapping
- Adding a new source later is hard

Example



Combined Local Schemas

Global Schema Relation

□ *PHBook*
- ISBN
- title
- authorID
- sug_retail
- format

○ *BNNewDeliveries*
- ISBN
- title
- instock

△ *Book_Price*
- ISBN
- seller
- price
- instock

Combined Local Schemas

Global Schema Relation

□ *PHBook*
- ISBN
- title
- authorID
- sug_retail
- format

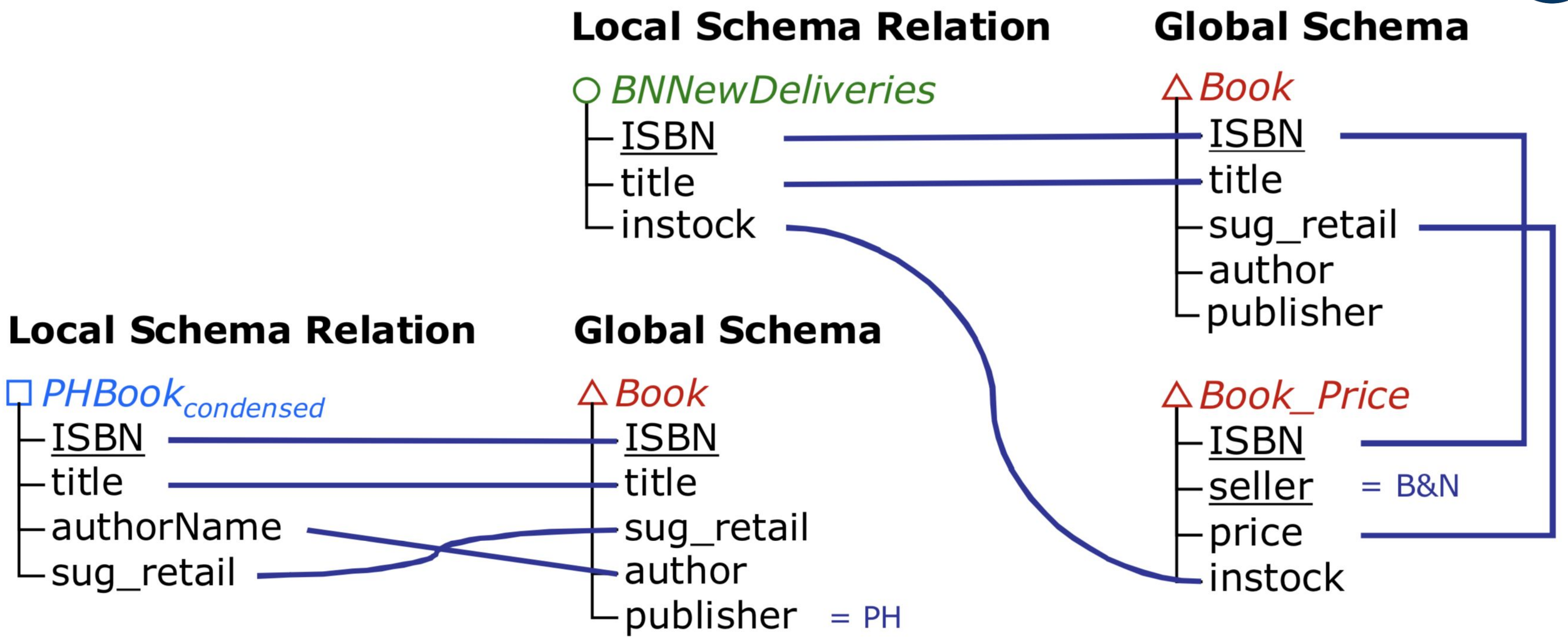
□ *PHAuthor*
- authorID
- authorName

△ *Book*
- ISBN
- title
- sug_retail
- author
- publisher = PH

Local as view (LAV)

- When sources change frequently, we can instead express each source as a view over the global schema
- Processing queries is harder
- Adding new sources is easy since we only need to write a single view

Example



Local Schema Relation

○ *BNNewDeliveries*
- ISBN
- title
- instock

Global Schema

△ *Book*
- ISBN
- title
- sug_retail
- author
- publisher

△ *Book_Price*
- ISBN
- seller = B&N
- price
- instock

Local Schema Relation

□ *PHBook*$_{condensed}$
- ISBN
- title
- authorName
- sug_retail

Global Schema

△ *Book*
- ISBN
- title
- sug_retail
- author
- publisher = PH

Global

Cust (ID, firstName, lastName, …)
CustPhones(ID, Type, PhoneNum, …)

Source 1

Customers (ID, firstName, lastName, homePhone, cellPhone, …)

Source 2

Customers (ID, firstName, lastName, …)
CustomersPhones(ID, Type, PhoneNum)

Another Example

Extract Transform Load

- **Extract** data from multiple sources useful for reporting
- **Transform** the data into a format more suitable for analysis
- **Load** the data into the warehouse

Federation

- A *federated* database system allows one database to connect to other systems
- For example, PostgreSQL has foreign data wrappers for this purpose
- Queries can then join with data on other systems with each executing part of the query

```
CREATE FOREIGN TABLE employees (
    id integer,
    name text,
    address text)
SERVER mysql_svr
OPTIONS (table 'hr.employees');
```

Foreign data wrapper

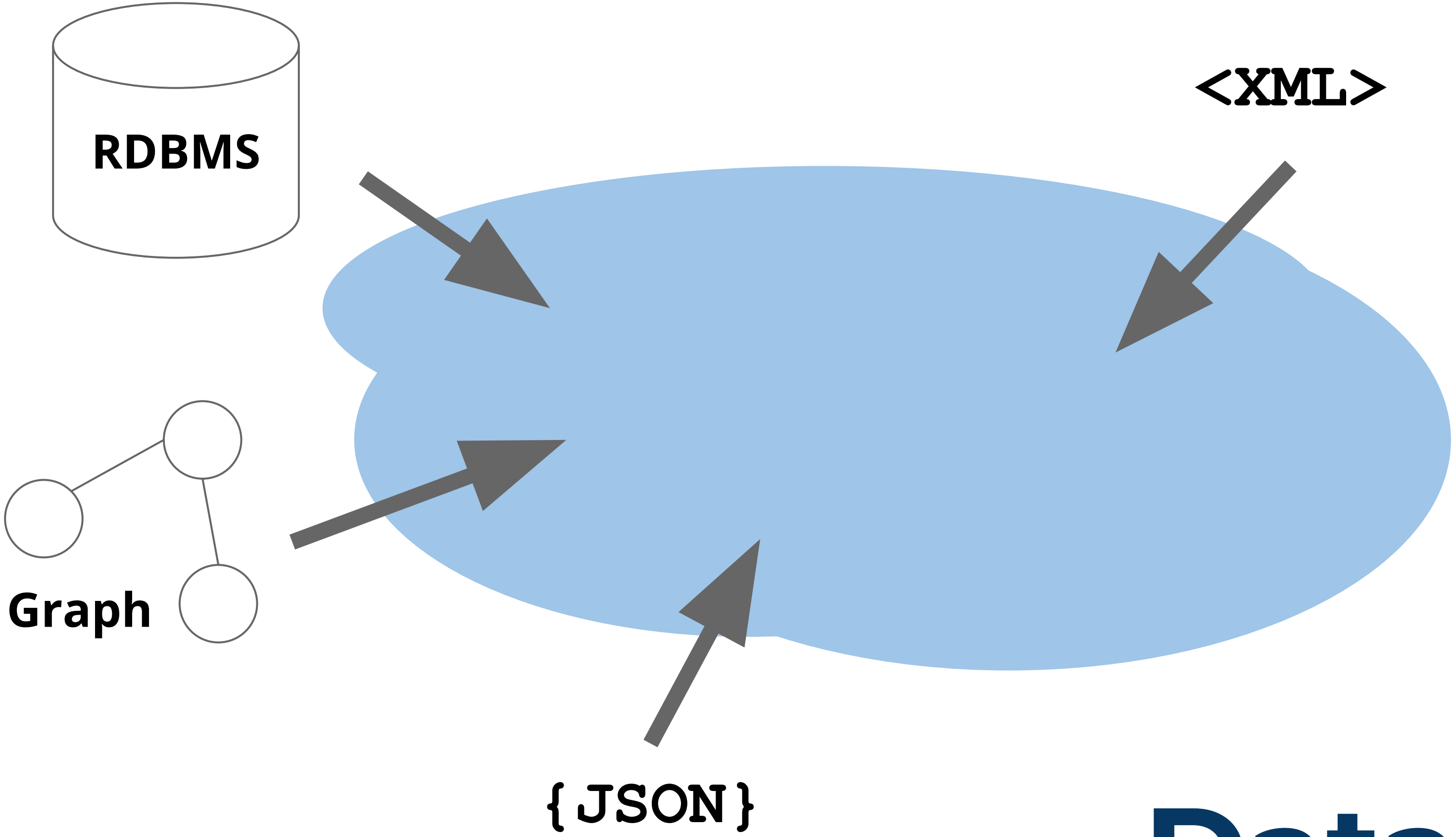
Data lakes

- One problem with data warehousing is that ETL can be time-consuming
- If we have a lot of data, then it may not be possible to keep up
- Instead, a data lake just collects large amounts of data without any transformation step

Data lakes

- Skipping the transform steps solves the velocity problem
- However, data is now in many different formats making queries challenging
- We need to make sure we maintain appropriate *metadata* for the data

Data lake



Other integration problems

- Schema matching

  Aligning schemas from different sources

- Entity resolution

  Identifying the same real world thing across sources

Schema matching

- Data may come from different systems but we want to connect them together

- This has several challenges
  - Syntax - languages used by the DBs
  - Structure - layout of records
  - Semantics - description used

```
CREATE TABLE users(
  id INT PRIMARY KEY,
  username VARCHAR(50));
```

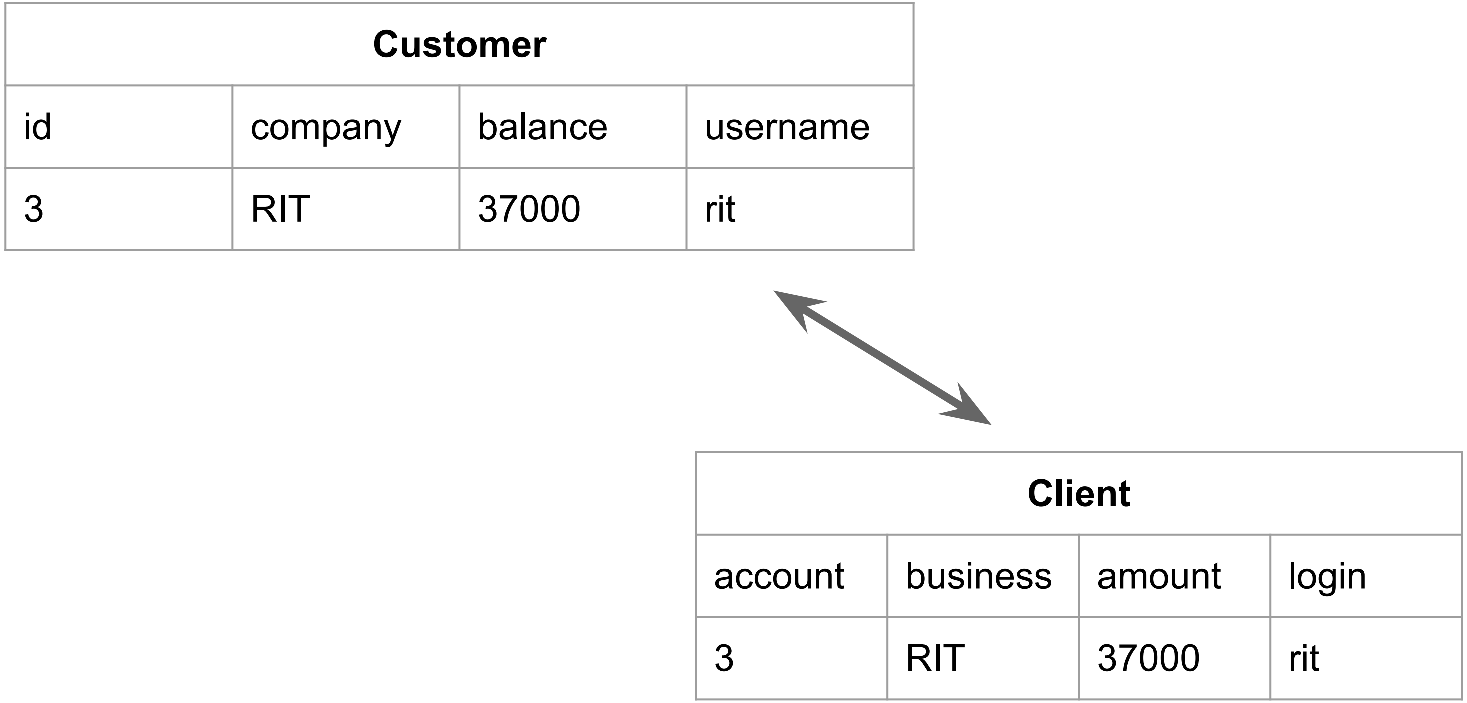
```
db.createCollection("users", {
  validator: {$jsonSchema: {
    bsonType: "object",
    required: ["username"],
    properties: {username: {bsonType: "string"}}
}})
```

Syntactic heterogeneity

Customer			
id	street	city	zip
3	One Lomb Memorial Drive	Rochester	14623

Customer	
id	address
3	One Lomb Memorial Drive, Rochester, 14623

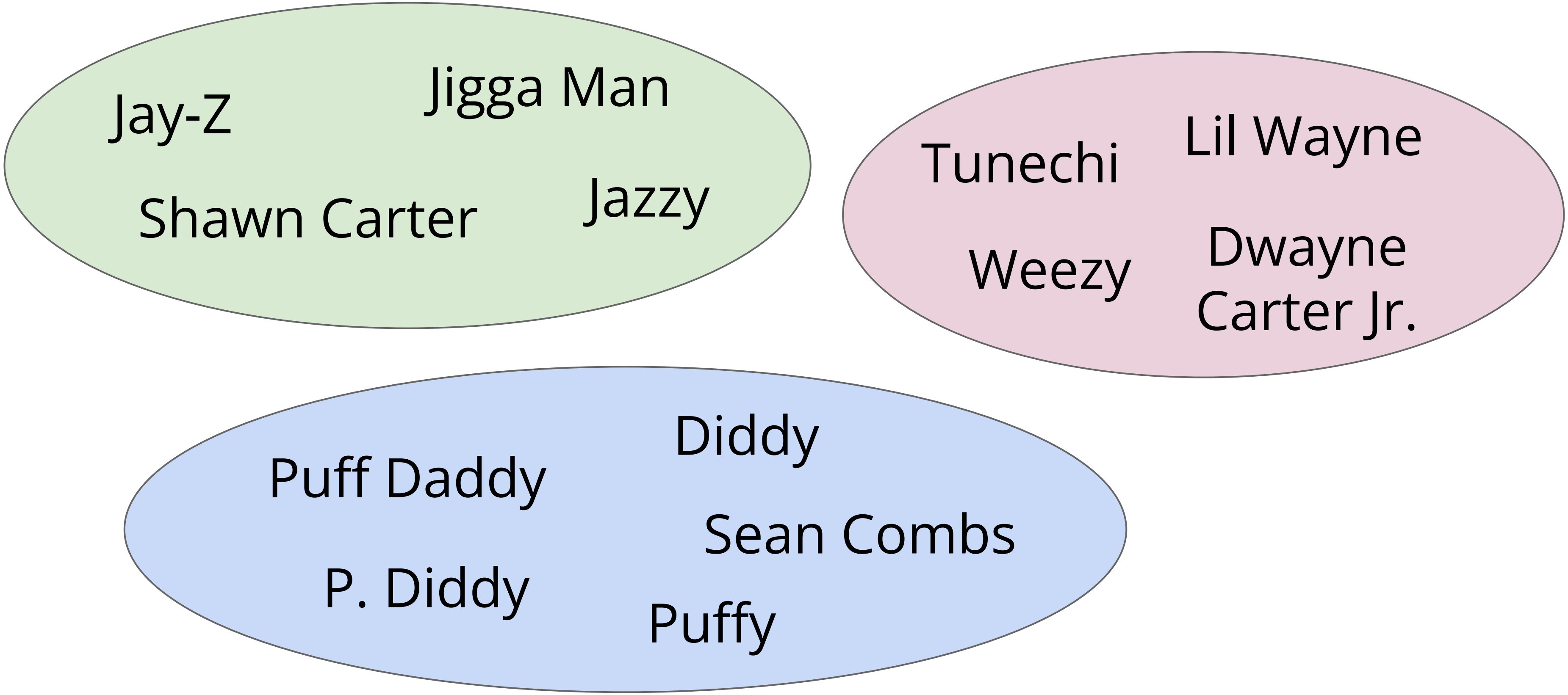
Structural heterogeneity



Customer			
id	company	balance	username
3	RIT	37000	rit

Client			
account	business	amount	login
3	RIT	37000	rit

Semantic heterogeneity



Jay-Z

Jigga Man

Shawn Carter

Jazzy

Tunechi

Lil Wayne

Weezy

Dwayne Carter Jr.

Puff Daddy

Diddy

Sean Combs

P. Diddy

Puffy

Entity Resolution

Ontologies

- Ontologies provide a standardized representation of semantics that can be shared across data sources

- Several large scale instances exist:
  - Google Knowledge Graph
  - DBpedia
  - Schema.org