# No SQL Databases

Thomas Schwarz, SJ

# Relational Model Shortcomings

- Greater Scalability
  - High write throughput / very large datasets
- Independence from few vendors — Move towards Open Source
- Need for different query operations
- Restrictiveness of relational schemas

# NoSQL History

- 2006: Bigtable: distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers
  - Bigtable uses a single key

# NoSQL History

- 2007 Dynamo
  - Primary key
  - Uses consistent hashing to partition and distribute data
  - A global, distributed key-value store

# NoSQL History

- 2007 - 2009: Riak, MongoDB, HBase, Accumulo, Hypertable, Redis, Cassandra, Neo4j
- Non-relational databases using different ideas
- Access without SQL

# NoSQL characteristics

- Schema agnostic
- Non-relational
- Commodity hardware
- Highly distributable

# NoSQL Characteristics

- Schema redesign overhead
  - Example Classic-Models:
    - You suddenly need to restructure so that you can split up an order
    - If you change the scheme you have old and new data
  - Often discover new relationships after working with data for a while

# NoSQL Characteristics

- Unstructured Data Explosion
- Combining from incompatible sources
- Sparse data

# NoSQL Characteristics

- Transactions are expensive
  - World-wide 24/7/365.25 data
  - Use *eventual consistency*

# Data at Very Large Scale Example

- Hush: HBase URL Shortener
  - Hand a URL to a Shortener service
  - Get a shorter URL back
    - E.g. to use in twitter messages
  - Shortener provides counter for each shortened URLs
  - "Vanity URL" that incorporate specific domain names
  - Need to maintain users
    - log in to create short URLs
    - track existing URLs
    - see reports for daily, weekly, or monthly usage

# Data at Very Large Scale Example

- Data is too large to store at a single server
- But:
  - Limited need for transactions
  - Importance of high throughput writes and reads

# Data at Very Large Scale Example

- Columnar Layout
  - A relational database strategy often adopted in No-SQL databases
  - Instead of storing data in tuples
  - Store by attribute

# Data at Very Large Scale Example

- For large HUSH:
  - Can use a relational database
  - Use normalization and obtain a scheme

# Data at Very Large Scale

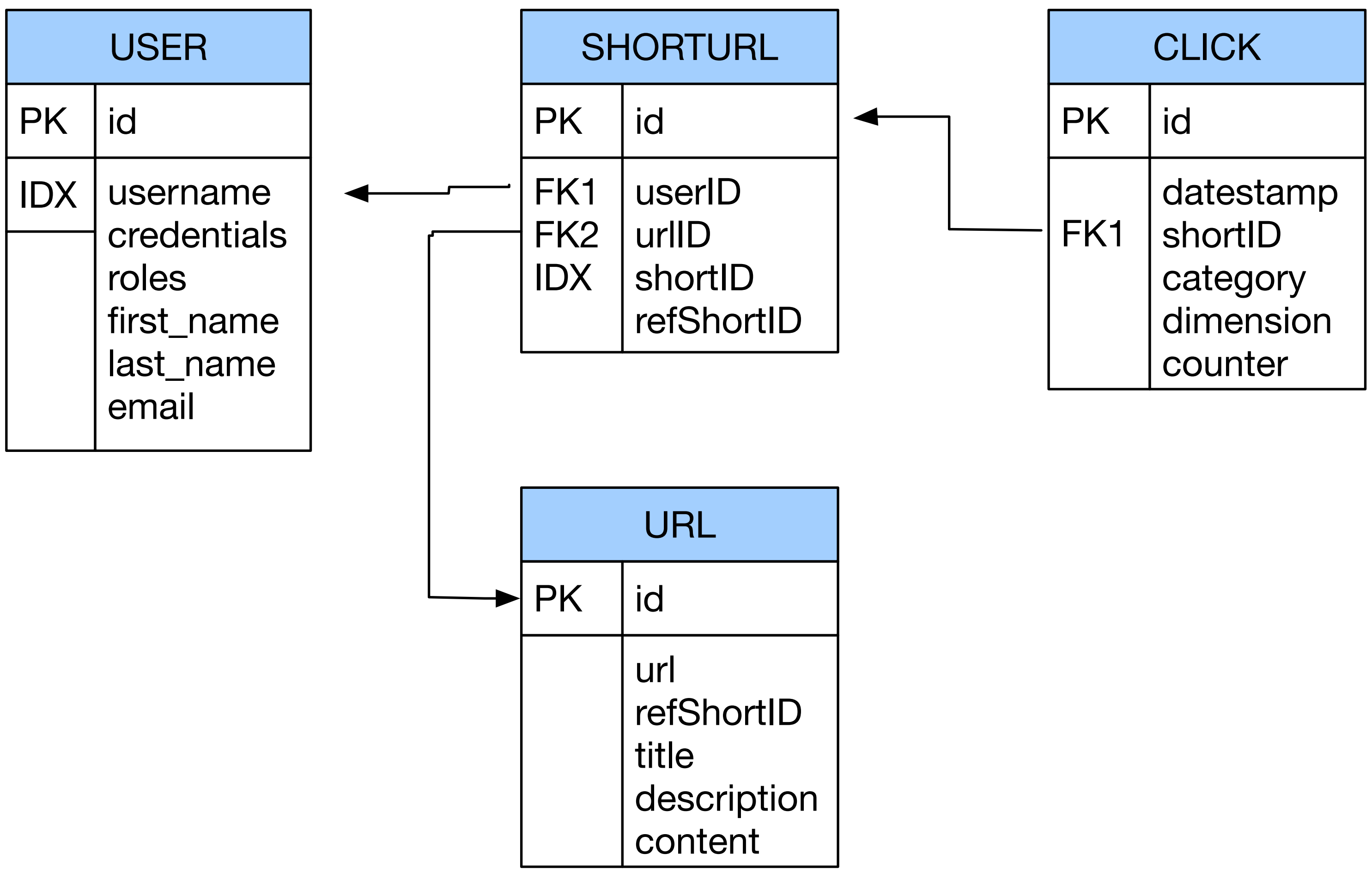
- Principles of Denormalization, Duplication, Intelligent Keys
  - Denormalize by duplicating data in more than one table
    - Avoids aggregation at read time
    - Pre-materialize required views

# Data at Very Large Scale

- Example: HBase URL Shortener (Hush)
  - user(id, username, credentials, rules, first_name, last_name, email) with unique username constraint
  - url(id, url, refShortID, title, description, content)
  - shorturl(id, userID, urlID, shortID, refShortID, description) with unique shortID and F.K. userID and urlID
  - click(id, datestamp, shortID, category, dimension, counter) with F.K. shortID

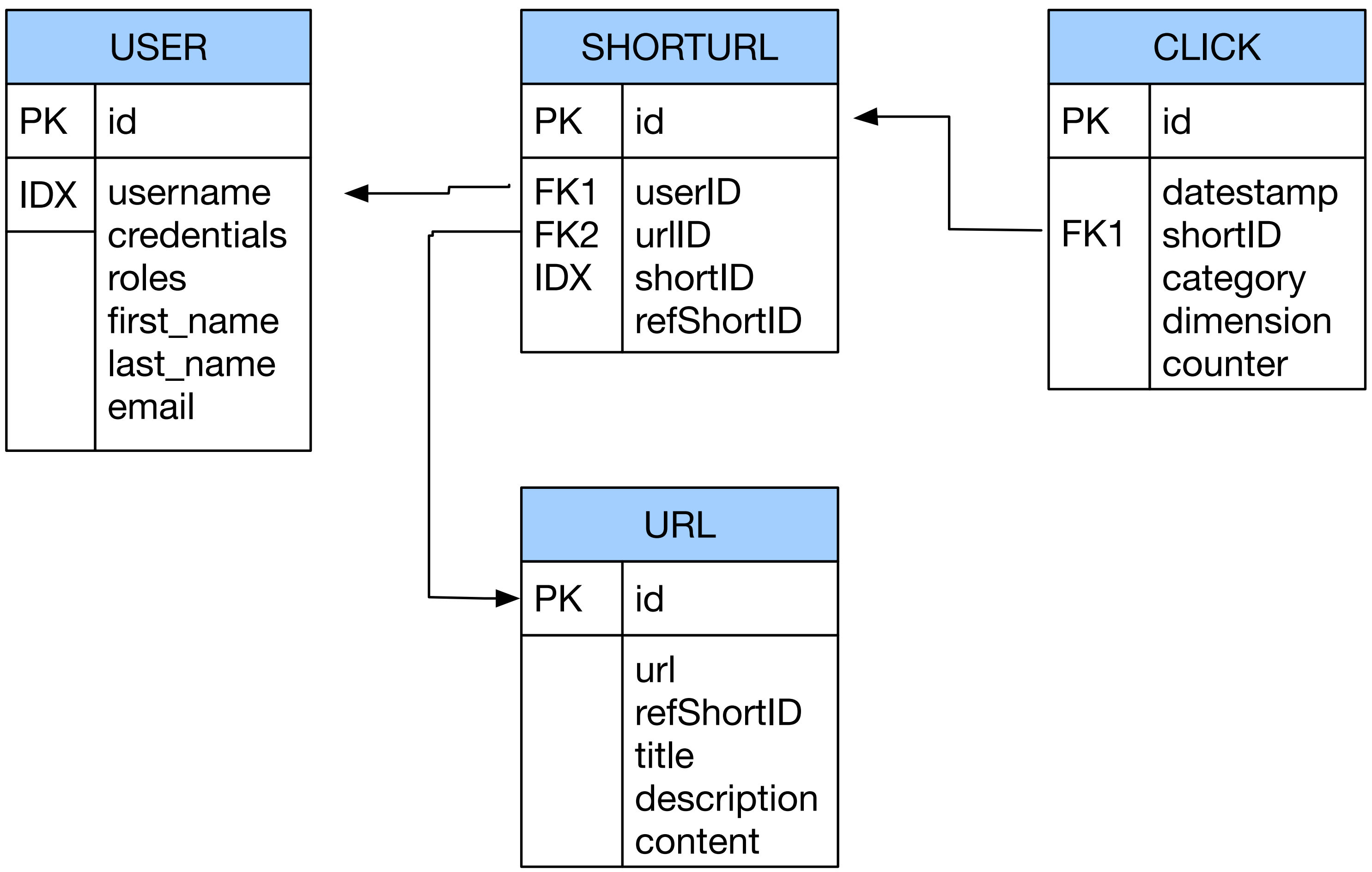
# Data at Very Large Scale

- Purpose: maps long URLs to short URLs



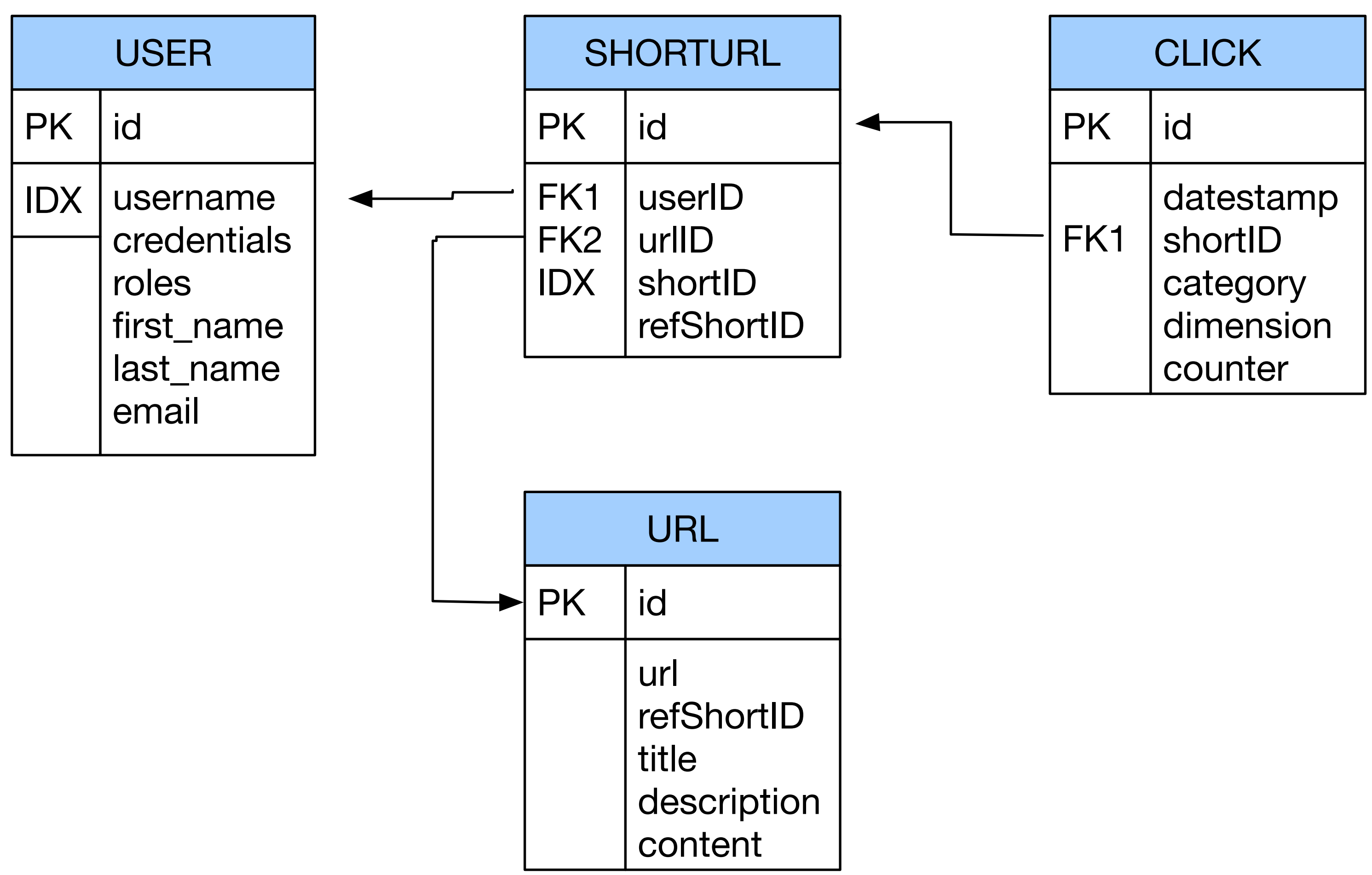
# Data at Very Large Scale

- Short URL can be given to others
- This is translated to the full URL



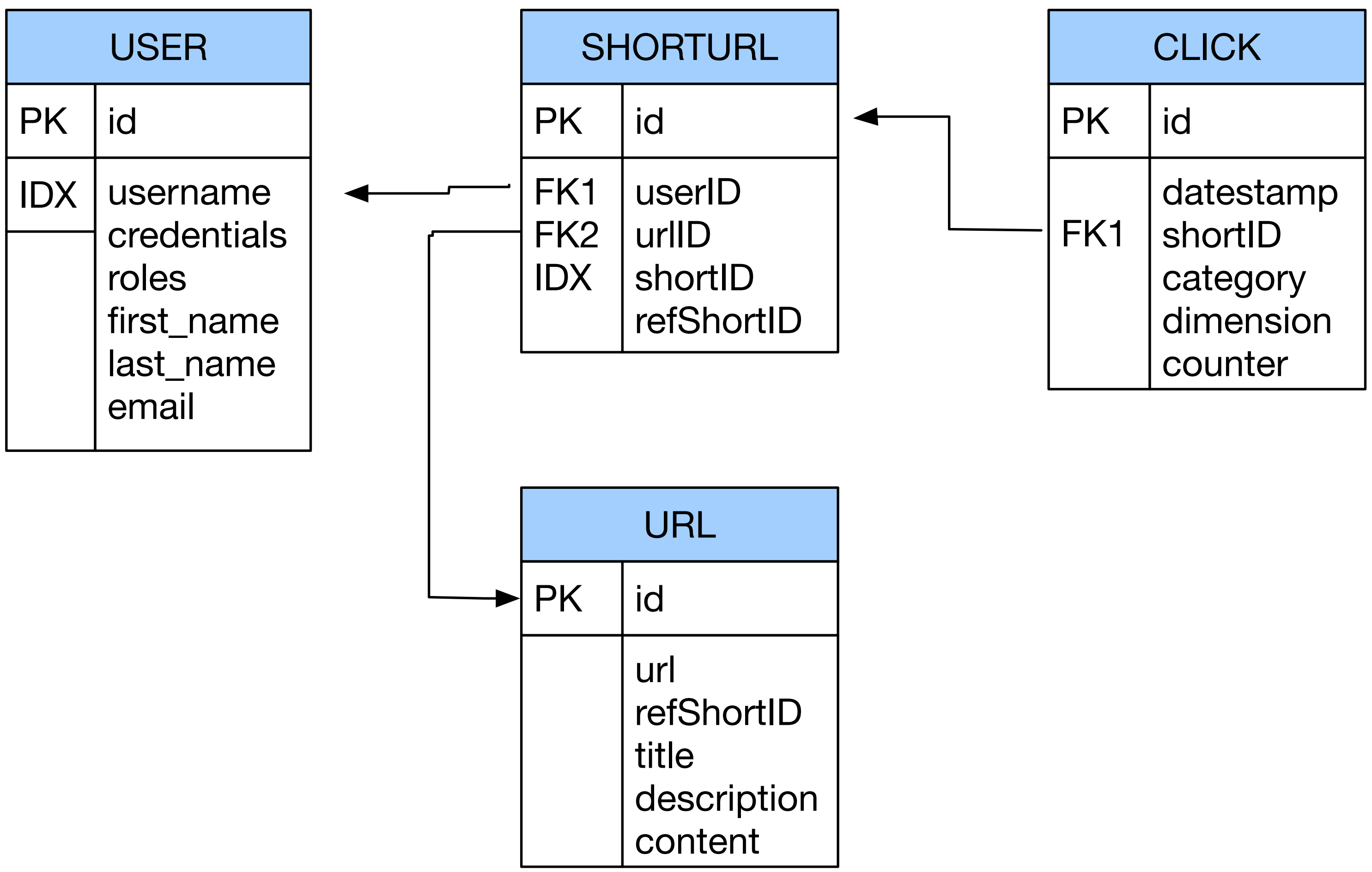
# Data at Very Large Scale

- Each click is tracked, which aggregates to weekly usage numbers



# Data at Very Large Scale

- All these operations require joins



# Data at Very Large Scale

- Bandwidth problem
  - Especially for joins
  - Need to store data in joins together, not look them up separately
  - But can relax on the consistency model:
    - No need to serialize short URL creation or URL translations or have atomic updates
  - Might be able to relax integrity constraints
    - Statistics need to be approximately correct

# Data at Very Large Scale

- Denormalization:
  - Key idea: Store data together that is likely to be joined
  - Means:
    - massive duplication of data
    - relaxed consistency needed
    - but faster reads / writes

# Data at Very Large Scale

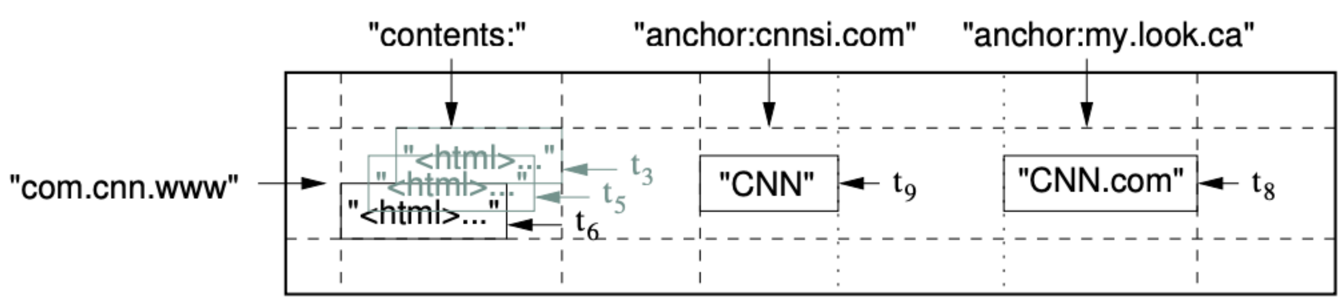
- Key-value database
  - Every record is a key-value pair
    - Extremely simple lookup: very high performance
    - Limited complexity
  - A large set of tools predating no-sql databases in general
  - Redis, Amazon DynamoDB, Microsoft Azure CosmosDB, Memcached

# Data at Very Large Scale

- Wide column stores
  - names and format of columns can vary from row to row
    - with potentially millions of different attributes
  - Two-dimensional key-value store:
    - Access via row and attributes
    - Schema-free
  - Google's BigTable is the original
  - Apache Cassandra, Microsoft Azure Cosmos DB, Apache HBase

# Data at Very Large Scale

- Example:



# Data at Very Large Scale

- Columnar:
  - Families of columns are stored together
  - Good for aggregation questions

# Data at Very Large Scale

- Columnar:
  - Good for high write performance
  - Good for colocated data access
- Examples: Cassandra, Apache HBase

# Data at Very Large Scale

- Document stores
  - Schema-free:
    - Different records can have different  columns
    - Types of values can be different
    - Columns can have more than one value
    - Records can have a nested structure
  - XML, JSON databases
  - MongoDB, Couchbase, Amazon DynamoDB, Databricks, Microsoft Azure Cosmos DB

# Data at Very Large Scale

- Graph databases
  - Navigational database successor:
    - information about data interconnectivity or topology as important as data itself
- Triple:
  - A single fact represented by
    - **subject**
    - **property  / relationship**
    - **value**

Adam   likes    cheese

`Subject Property Value`

# Alternatives to Relational Schemes: XML

- Data is often structured hierarchically

```
Invoice = {
  date : "2008-05-24"
  invoiceNumber : 421

  InvoiceItems : {
    Item : {
      description : "Wool Paddock Shet Ret Double Bound Yellow 4'0"
      quantity : 1
      unitPrice : 105.00
    }
    Item : {
      description : "Wool Race Roller and Breastplate Red Double"
      quantity : 1
      unitPrice : 75.00
    }
    Item : {
      description : "Paddock Jacket Red Size Medium Inc Embroidery"
      quantity : 2
      unitPrice : 67.50
    }
  }
}
```

# Alternatives to Relational Schemes: XML

- As an XML document

```xml
<invoice>

 <number>421</number>
 <date>2008-05-24</date>
 <items>
  <item>
   <description>Wool Paddock Shet Ret Double Bound Yellow 4'0"</description>
   <quantity>1</quantity>
   <unitPrice>105.00</unitPrice>
  </item>
  <item>
   <description>Wool Race Roller and Breastplate Red Double</description>
   <quantity>1</quantity>
   <unitPrice>75.00</unitPrice>
  </item>
  <item>
   <description>Paddock Jacket Red Size Medium Inc Embroidery</description>
   <quantity>2</quantity>
   <unitPrice>67.50</unitPrice>
  </item>
 </items>
</invoice>
```

# Alternatives to Relational Schemes: XML

- Advantage of XML
  - Faster to scan all data
  - No joins
- Disadvantages of XML
  - Each record contains the full or an abbreviated scheme
  - Each query needs to select from big chunks of data

# Alternatives to Relational Schemes: JSON

- JSON — JavaScript Object Notation
  - Human-readable
  - Organized as key-value pairs

# Alternatives to Relational Schemes: JSON

- JSON record example


```json
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

# Alternatives to Relational Schemes: JSON

- JSON can use a schema (type definition)
- JSON was first used for data transmission as a data serialization format

# Alternatives to Relational Schemes: JSON

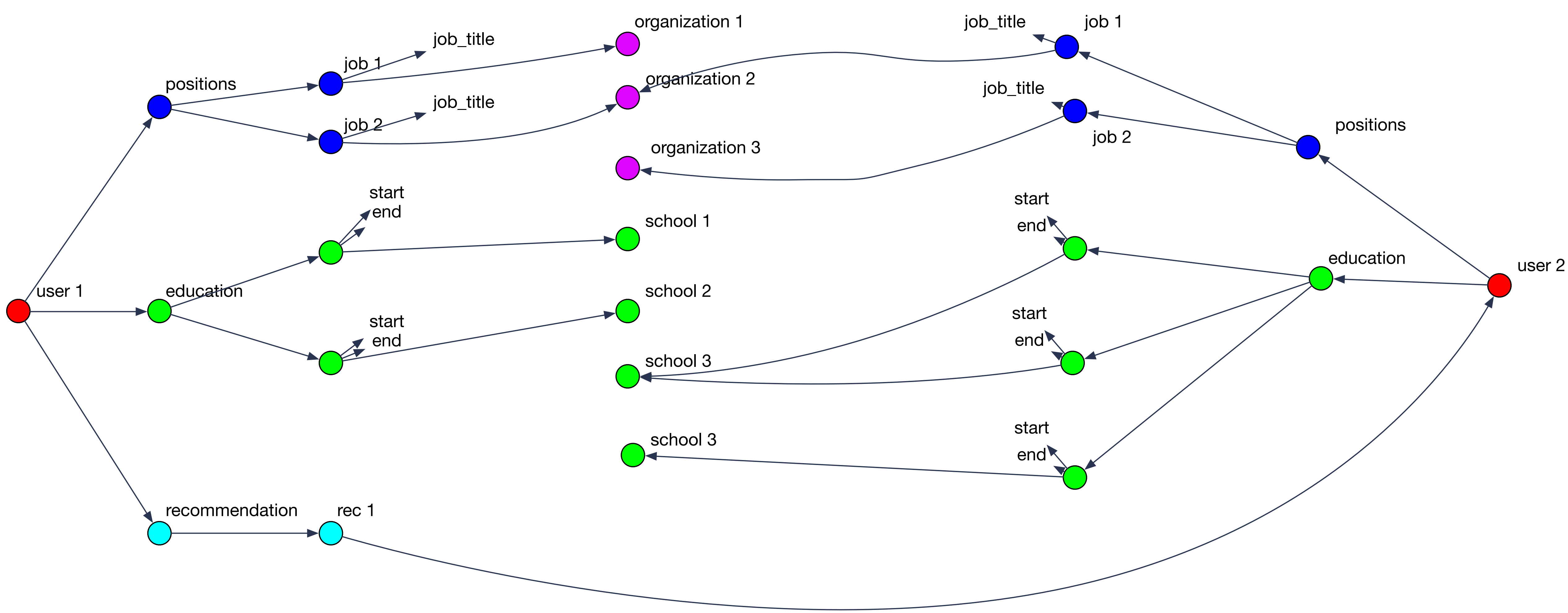
- Many-to-One and Many-to-Many Relationships
  - Modeled by the same value for the same key
    - Problem:  Need to standardize / internationalize these values
    - Using id-s instead of plain text to avoid problems
    - Table of id-s reintroduce a relational scheme through a backdoor

# Alternatives to Relational Schemes: JSON

- Resumé
  - Users present people
  - People have jobs, education, and recommenders
- But they share jobs, companies, degrees, schools, recommenders
  - Should they stay text strings or become entities?
    - Latter allows to add information to all resumés
- If recommenders get a photo, then all resumés should be updated with this photo, so better to make recommenders entities

# Alternatives to Relational Schemes: JSON

- Data has a tendency to become less-join free



# Document Databases

- Records are documents
  - Encode in
    - XML
    - YAML
    - JSON
    - BSON (Mongo DB)
  - CRUD operations: create, read, update, delete

# Document Databases

- Enforcing schema
  - Most document databases do not enforce schema
    - —> “Schemaless”
    - In reality: “Schema on Read”
    - RDBMS would then use “Schema on Write”
- Allows schema updates in simple form

# Document Databases

- Schema on Read:
  - Advantages:
    - Data might come from external sources
  - Disadvantages:
    - No data checking

# Document Databases

- Document database support
  - Most commercial database systems now support XML databases

# Query Languages

- Documents lend themselves to object-oriented querying
  - Imperative code
- SQL is declarative:
  - Programmer explains a solution
  - System figures out the best way to find the solution
- Use declarative query languages for document databases

# Query Languages

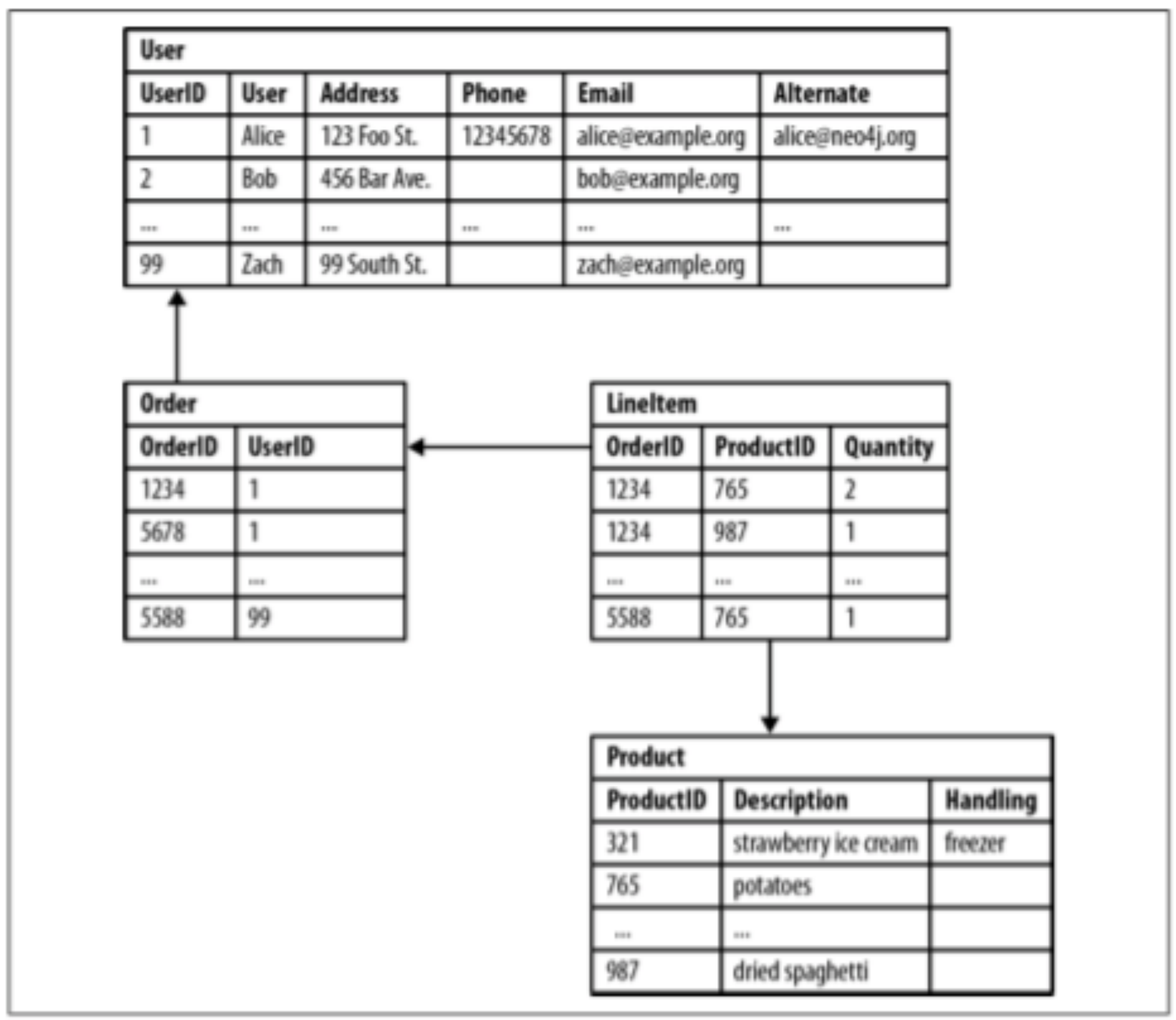
- Map-Reduce (neither declarative nor imperative):
  - Consists of only two pieces of code
    - Mapping:  Selecting from Documents
    - Reducing: Take selection elements and operate on them

# Alternatives to Relational Schemes: Graph Models

- Graphs consists of vertices and edges
  - Example:
    - Social graphs: vertices are people and edges are relationships such “knows”
    - Web graph: vertices are pages and edges are links
    - Road networks: vertices are places and edges are connections

# Alternatives to Relational Schemes: Graph Models

- Relational Database hides semantic relationships



**User**

| UserID | User | Address | Phone | Email | Alternate |
|---|---|---|---|---|---|
| 1 | Alice | 123 Foo St. | 12345678 | alice@example.org | alice@neo4j.org |
| 2 | Bob | 456 Bar Ave. | | bob@example.org | |
| ... | ... | ... | ... | ... | ... |
| 99 | Zach | 99 South St. | | zach@example.org | |

**Order**

| OrderID | UserID |
|---|---|
| 1234 | 1 |
| 5678 | 1 |
| ... | ... |
| 5588 | 99 |

**LineItem**

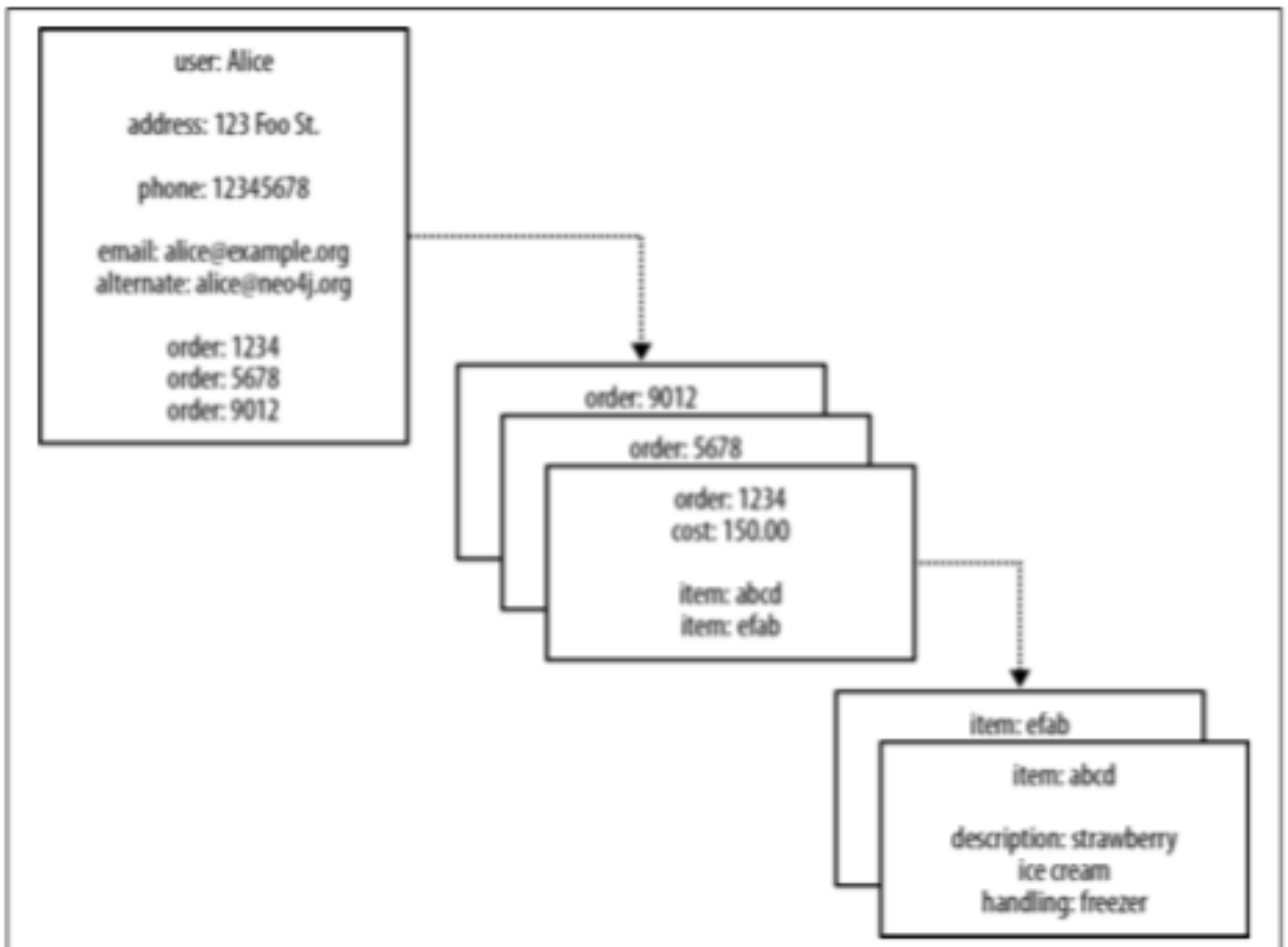
| OrderID | ProductID | Quantity |
|---|---|---|
| 1234 | 765 | 2 |
| 1234 | 987 | 1 |
| ... | ... | ... |
| 5588 | 765 | 1 |

**Product**

| ProductID | Description | Handling |
|---|---|---|
| 321 | strawberry ice cream | freezer |
| 765 | potatoes | |
| ... | ... | |
| 987 | dried spaghetti | |

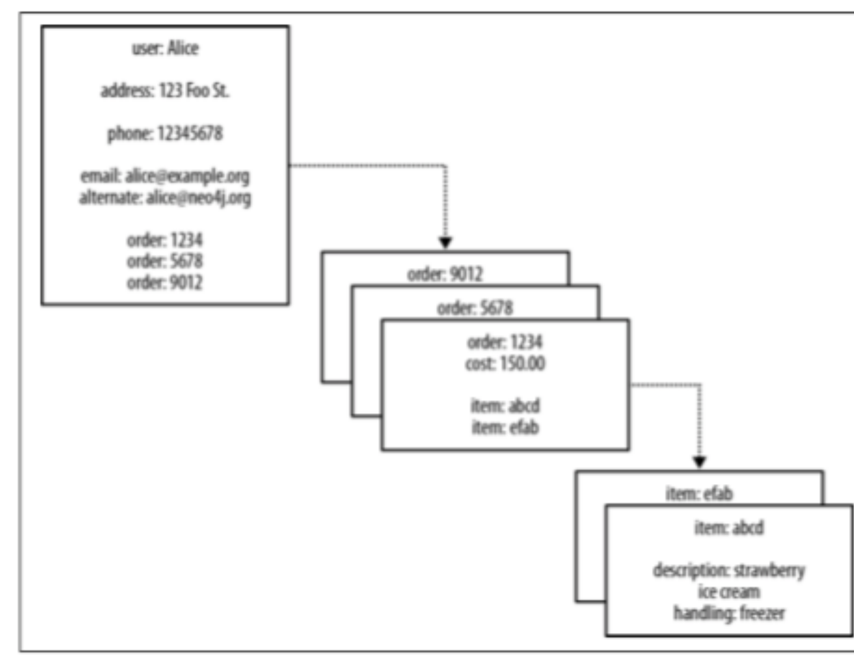
# Alternatives to Relational Schemes: Graph Models

- Document model hides semantic relationships



# Alternatives to Relational Schemes: Graph Models

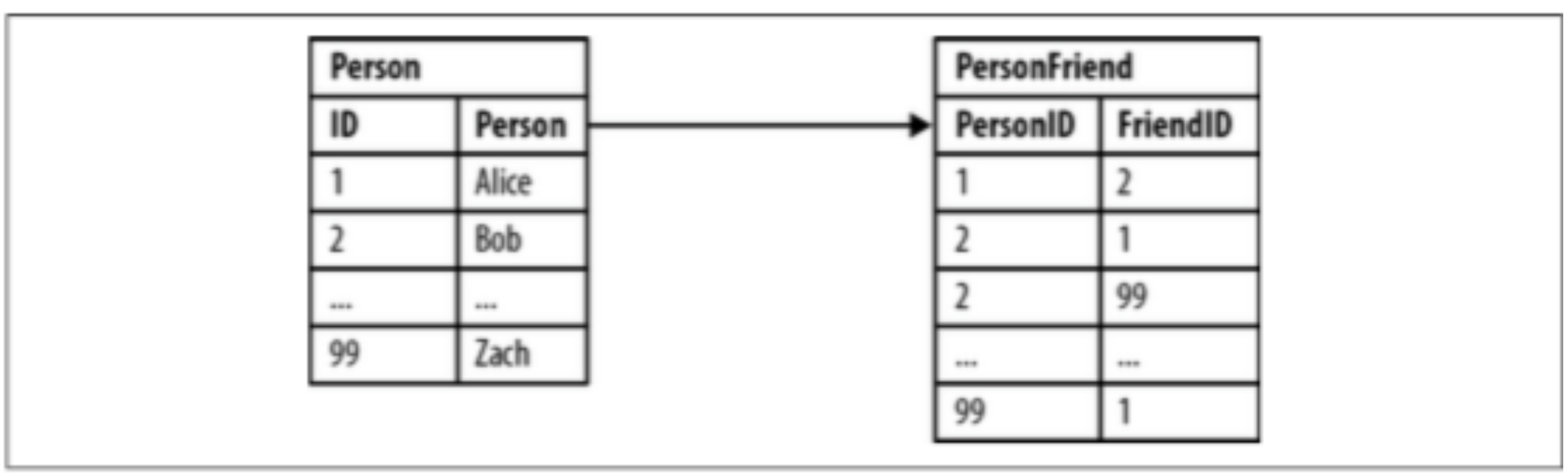
- Some property values are really references to foreign aggregates
  - Aggregate's identifier is a foreign key
- Relationships between them are not explicitly accessible
  - Joining aggregates becomes expensive



# Alternatives to Relational Schemes: Graph Models

- Relational Database
  - Some queries are simple:

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
ON PersonFriend.FriendID = p1.ID JOIN Person p2
ON PersonFriend.PersonID = p2.ID WHERE p2.Person = 'Bob'
```



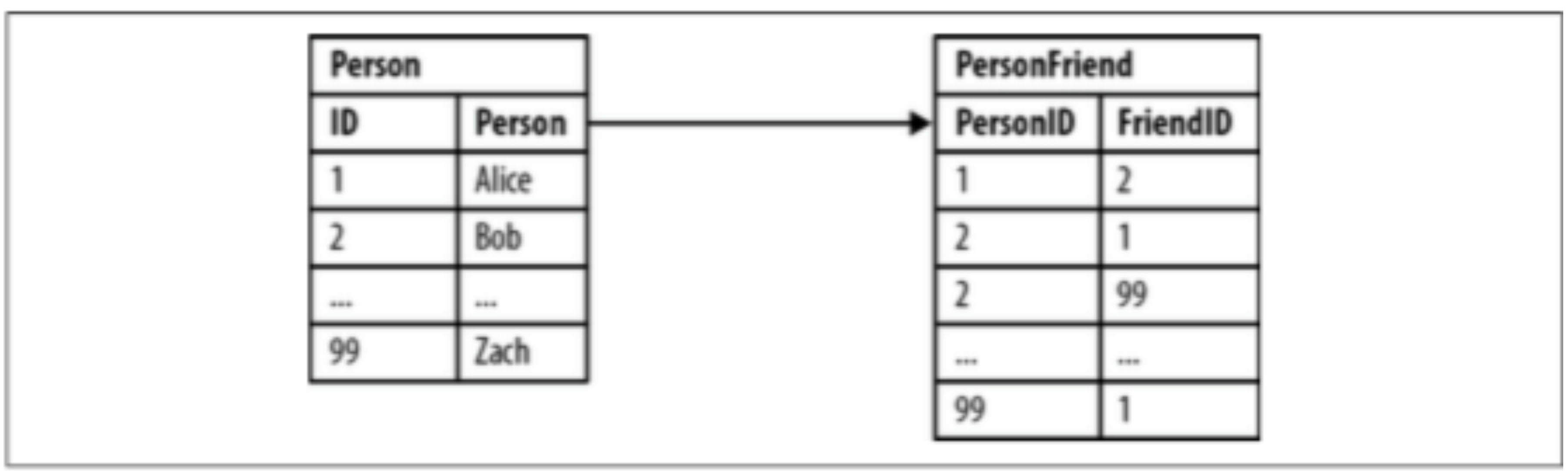
| Person | |
|---|---|
| **ID** | **Person** |
| 1 | Alice |
| 2 | Bob |
| ... | ... |
| 99 | Zach |

| PersonFriend | |
|---|---|
| **PersonID** | **FriendID** |
| 1 | 2 |
| 2 | 1 |
| 2 | 99 |
| ... | ... |
| 99 | 1 |

# Alternatives to Relational Schemes: Graph Models

- Relational Database
  - Some queries are more involved: Friends of Bob

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
   ON PersonFriend.PersonID = p1.ID JOIN Person p2
   ON PersonFriend.FriendID = p2.ID
WHERE p2.Person = 'Bob'
```



| Person | |
|---|---|
| ID | Person |
| 1 | Alice |
| 2 | Bob |
| ... | ... |
| 99 | Zach |

| PersonFriend | |
|---|---|
| PersonID | FriendID |
| 1 | 2 |
| 2 | 1 |
| 2 | 99 |
| ... | ... |
| 99 | 1 |

# Alternatives to Relational Schemes: Graph Models

- Relational Database
  - Some queries others are difficult: Alice’s friends of friends

```
SELECT p1.Person AS PERSON, p2.Person AS FRIEND_OF_FRIEND FROM
PersonFriend pf1 JOIN Person p1
ON pf1.PersonID = p1.ID JOIN PersonFriend pf2
ON pf2.PersonID = pf1.FriendID JOIN Person p2
ON pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
```

**Person**

| ID | Person |
|---|---|
| 1 | Alice |
| 2 | Bob |
| ... | ... |
| 99 | Zach |

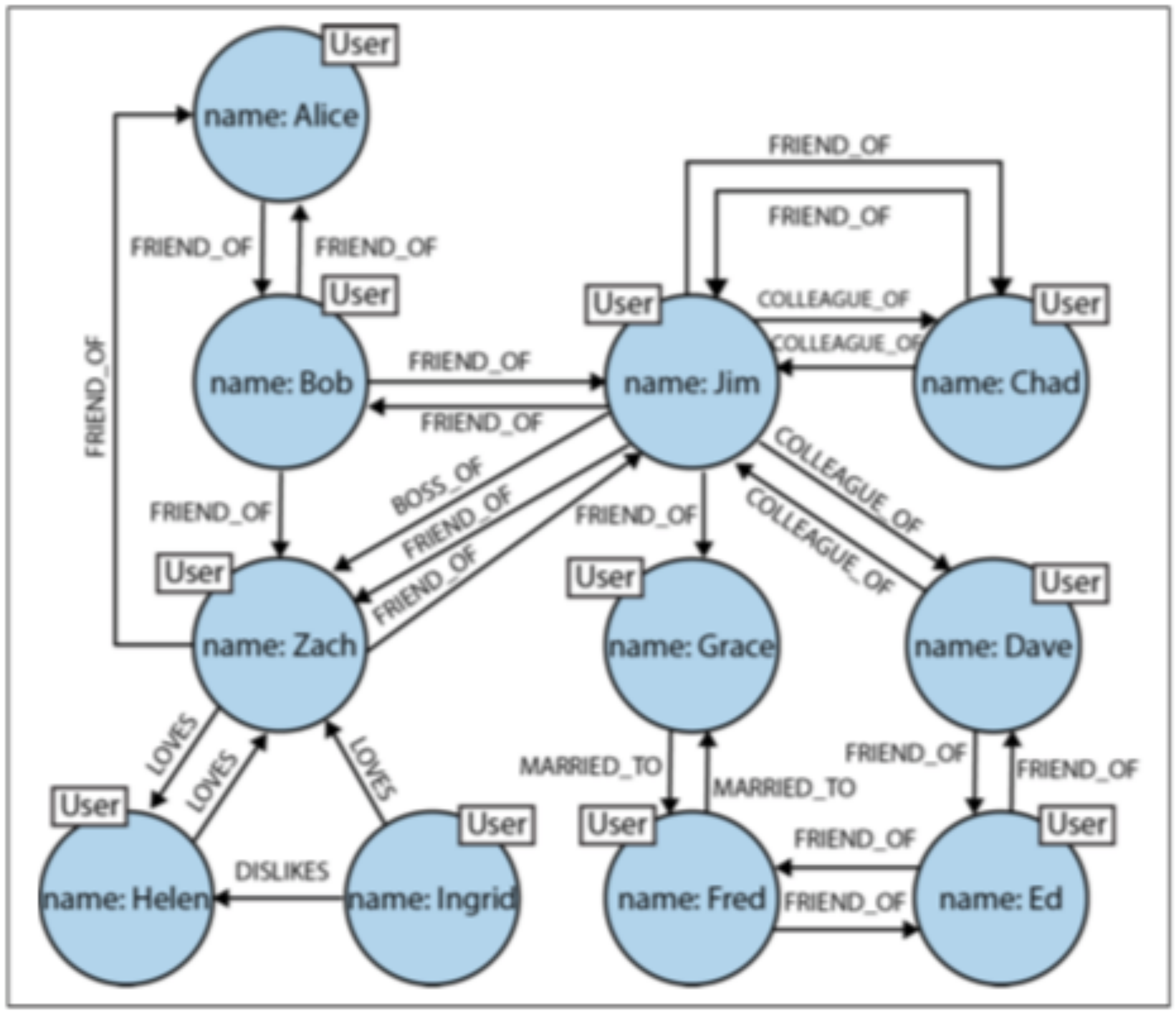
**PersonFriend**

| PersonID | FriendID |
|---|---|
| 1 | 2 |
| 2 | 1 |
| 2 | 99 |
| ... | ... |
| 99 | 1 |

# Alternatives to Relational Schemes: Graph Models

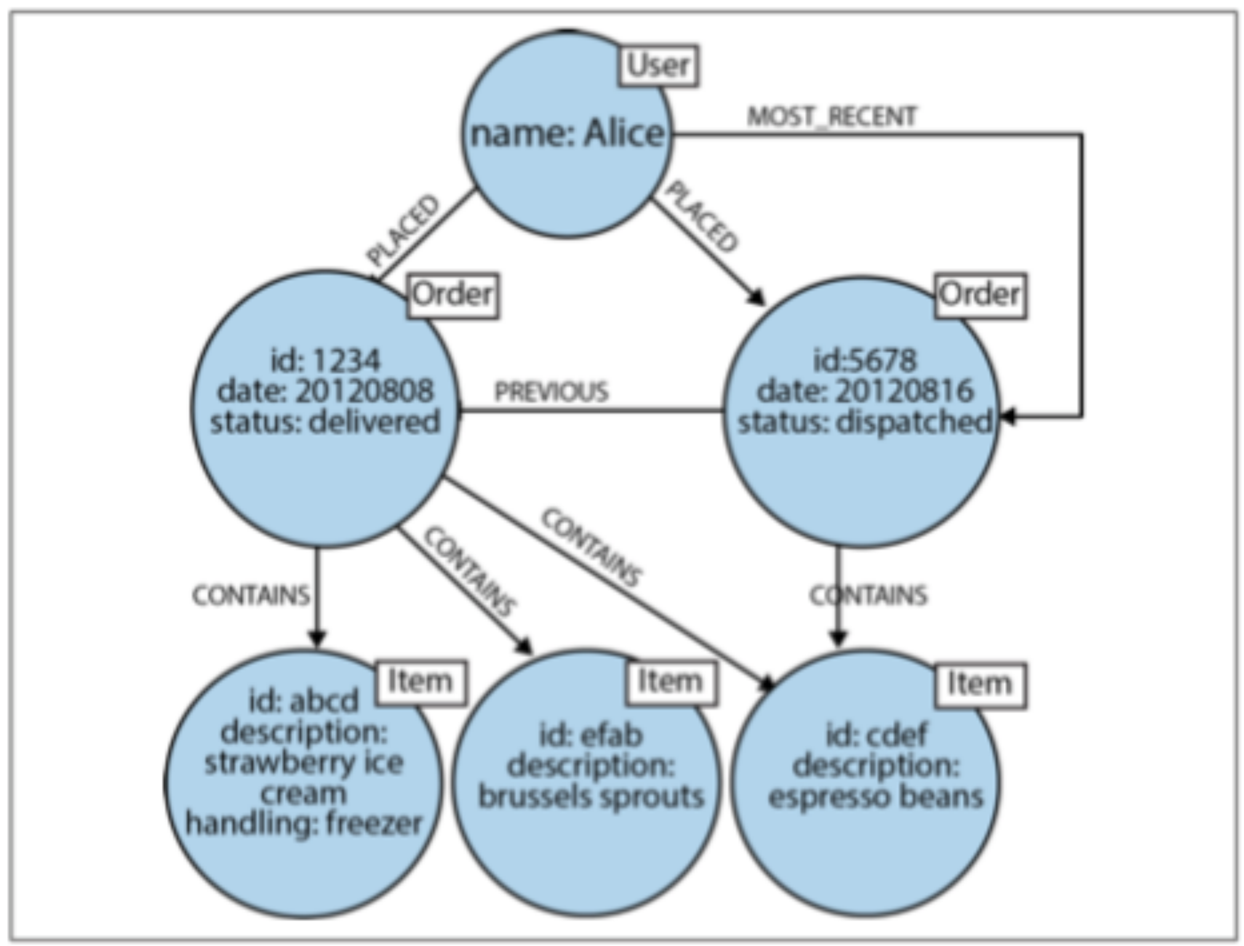
- Property graph model by Neon
  - Each vertex consists of
    - A unique identifier
    - A set of outgoing edges
    - A set of incoming edges
    - A collection of properties — key-value pairs
  - Each edge consists of
    - A unique identifier
    - The tail vertex
    - The head vertex
    - A label to describe the relationship
    - A collection of properties — key-value pairs

# Alternatives to Relational Schemes: Graph Models



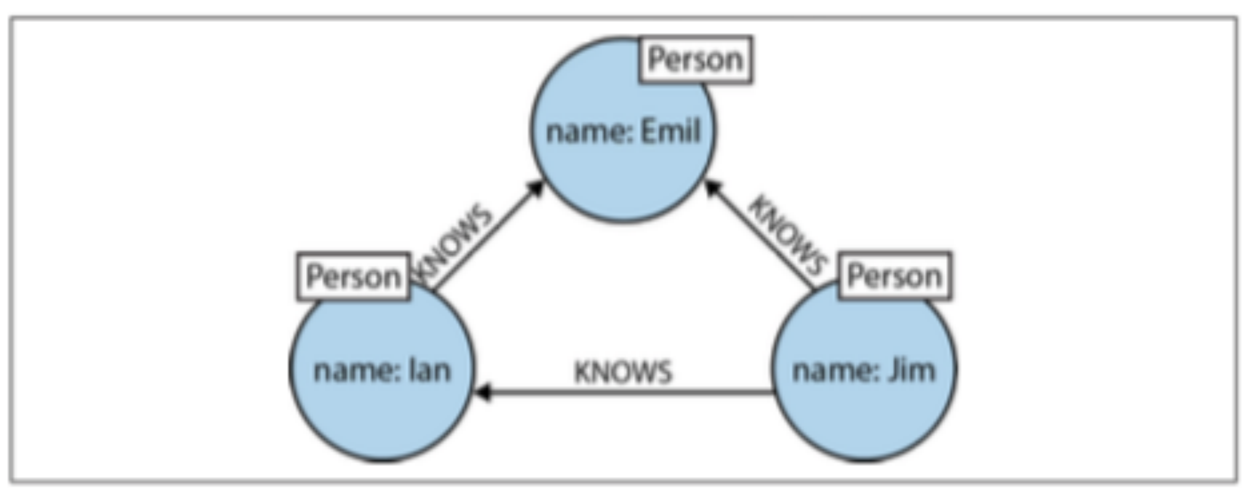
# Alternatives to Relational Schemes: Graph Models

- Order history as a property graph



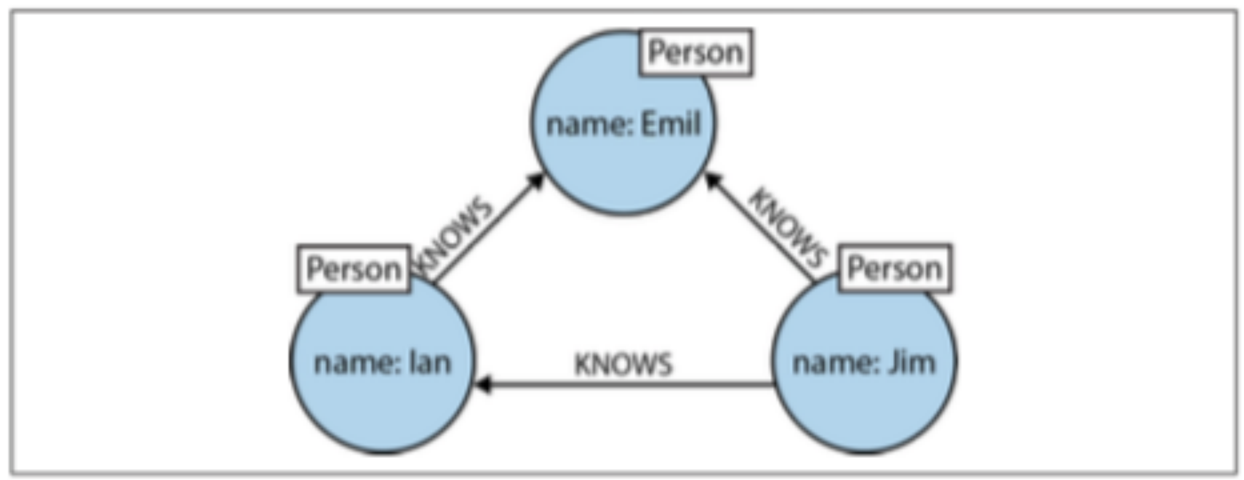
# Alternatives to Relational Schemes: Graph Models

- Processing queries in Neo4j
  - Use Cypher (from “The matrix”)
  - Can describe a path




```
(emil)<-[:KNOWS]-(jim)-[:KNOWS]->(ian)-[:KNOWS]->(emil)
```

# Alternatives to Relational Schemes: Graph Models




```
(emil:Person {name:'Emil'})
      <-[:KNOWS]-(jim:Person {name:'Jim'})
      -[:KNOWS]->(ian:Person {name:'Ian'})
      -[:KNOWS]->(emil)
```

# Alternatives to Relational Schemes: Graph Models

- Finding the mutual friends of Jim:

```
MATCH (a:Person {name:'Jim'})-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-
[:KNOWS]->(c)
RETURN b, c
```

# Alternatives to Relational Schemes: Graph Models

- Triple Stores a.k.a. Resource Description Framework

- Information is stored as (subject, predicate, object)
  - Subjects correspond to vertices
  - Objects are
    - A value in a primitive data type — ( jim : age : **64**)
    - Another vertex — (jim : friend_of : thomas)

# Alternatives to Relational Schemes: Graph Models

```
@prefix : </example>
_:lucy    a            :Person
_:lucy    :name        “Lucy”
_:lucy    :born_in     _:idaho
_:idaho   a            :Location
_:idaho   :name        “Idaho”
_:idaho   :type        “State”
_:idaho   :within      _:usa
```

# Alternatives to Relational Schemes: Graph Models

- Triple stores are the language of the semantic web
- Semantic web:
  - Machine readable description of type of links
    - e.g. image, text, …
  - Creates web of data — a database of everything
- Stored in Resource Description Framework (RDF)
- SPARQL — query language for triple stores