



OrientDB Academy Live Global Class

July 2017

This publication is protected by copyright. No part of this publication may be reproduced in any form by any means without prior written authorization by WiseClouds, LLC.

This publication is provided “as is” without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose or non-infringement.

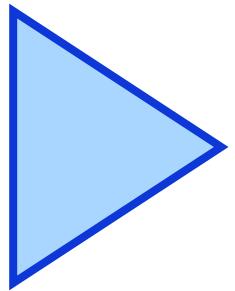
This publication is provided for educational purposes only.

Any product specifications are subject to change without notice.

WiseClouds and the WiseClouds logo are trademarks of WiseClouds, LLC in the United States, other countries, or both. OrientDB, its logo, and the products listed below are registered trademarks of OrientDB Ltd in the United States, other countries, or both. All other company or product names are registered trademarks or trademarks of their respective companies.

OrientDB
OrientDB Studio
Teleporter
Neo4J Importer

© 2017 WiseClouds, LLC. All rights reserved.



Introduction

- Class Structure & Goals
- About WiseClouds
- About OrientDB
- About the Use Case

Course Goals

- Learn why NoSQL solutions are now working alongside traditional RDBMS
- Understand the potential offered by multi-model databases
- Help determine scenarios where a multi-model database makes sense

Course Goals

- Showcase OrientDB's technical capabilities
- Model and design a database using a realistic example
- Interact with data using Java, SQL & graph extensions
- Learn how to administer an OrientDB database

Intended Audiences

- Application developer
- Architect
- Database administrator
- Database designer
- Operations
- Business user

Course Structure

Day one: Multi-model Database Modeling, Design, Interaction

Day two: Working with the OrientDB Java and HTTP APIs

Day three: Administering an OrientDB Database

About WiseClouds

- Highly focused consulting & training company
 - OrientDB reseller
- Key practices
 - Big Data/NoSQL
 - API & SOA
 - Virtualization



T-Mobile



About Me



Blog: rdschneider.com

- 20+ years experience in distributed computing and databases
- Author of eight books and numerous articles on highly complex technical topics
- Speaker and organizer at many conferences

About the Use Case

- Expands on OrientDB demonstration database
 - Social Travel Network
- Easily relatable example
 - Frequent flyer
 - Flight reservation
 - Credit card application

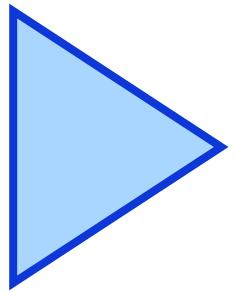
Class Introduction

About the Use Case

- Designed to be simple but comprehensive
 - Basic application code
 - Straightforward SQL
- We'll primarily use OrientDB Studio

Technical Requirements

- Your computer must:
 - Be able to communicate with a remote server
 - No client-side firewall restrictions that will prevent access
 - Be equipped with JDK 1.8
 - Have a Java development environment such as Eclipse or IntelliJ
 - Have the ability to transmit HTTP commands such as GET, POST, PUT, and DELETE
 - Free tools such as Postman and SoapUI offer this capability; you may also install the free ‘REST Easy’ extension for Firefox



Introducing Graph Databases

- State of the Database Landscape
- Understanding the NoSQL Movement
- What are Graph Databases?
- When Should Graph Databases Be Used?

Unit Goals

- Understand the NoSQL movement
- Learn about different NoSQL technologies
- Explain the advantages of graph databases in contrast with other NoSQL products
- Provide a high level overview of what they are, and how they work

Unit Goals

- Point out similarities and differences with relational databases
- Discuss situations where graph databases are especially useful

State of the Database Landscape

- Relational databases (RDBMS) have been the de facto standard for information storage for decades
 - They will be a core part of the IT portfolio for many years to come
- Many applications continue to be developed using RDBMS as the storage repository

State of the Database Landscape

- Despite the dominance of the RDBMS, things have changed greatly in the past five years
- There are now far more options for storing information
- Collectively, many of these have become known as ‘NoSQL’

Database Landscape Changes

- Big Data
 - By far the most significant contributor
- Many definitions, but these three attributes are key:
 - Volume: there's much more data than ever
 - Variety: many new types of data being stored
 - Velocity: it's generated much more quickly than before

RDBMS Challenges

- RDBMS have been experiencing difficulty keeping up with Big Data
- While RDBMS storage capacity is essentially unlimited, their join architecture won't scale
 - This is one of the biggest reasons why so many new data storage technologies have arisen

RDBMS Challenges

- Joins constitute the ‘relations’ in a relational database
 - They’re fine for traditional applications
 - With predictable data and usage patterns
- But join complexity can quickly get out of hand in a dynamic, modern application
 - Particularly in a ‘many-to-many’ relationship scenario
 - Such as found social networks

RDBMS Challenges

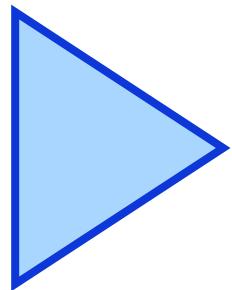
- Joins are computationally expensive
 - This doesn't have a major effect in the simplest cases
 - But things get dramatically slower as the number - and complexity - of many-to-many relationships grows
- Indexing is one long-proven strategy to speed up queries that use joins

RDBMS Challenges

- But excess indexing comes at a steep price:
overhead
 - Processing
 - Data storage
- Although indexes can speed up search, they
slow the performance of
 - Inserts
 - Updates
 - Deletes

RDBMS Challenges

- NoSQL can best be thought of as a logical response to
 - The applications that have become massively popular in the past five years
 - The drawbacks of the RDBMS for these applications
 - The new types of information that are now so prevalent
 - The ways that users want to interact with all this new information



NoSQL

- What Does NoSQL Mean?
- Major NoSQL Technologies
- What's Driving the Move To NoSQL?
- NoSQL Technology Shortcomings

What is NoSQL?

- Common misperception
 - NoSQL means “No to SQL”
- In fact, NoSQL is not anti-SQL
 - Most NoSQL advocates understand that SQL continues to have a vital role to play
- It's better to think of it as ‘Not Only SQL’

What is NoSQL?

- It's a movement that encourages selecting the right tool for the job
 - And recognizes that RDBMS isn't the optimal choice in many cases
- This has led to a flourishing number of new information management technologies
- Most organizations will deploy a variety of solutions

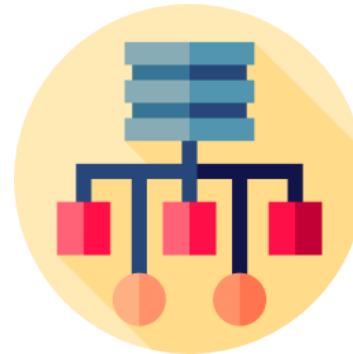
What's Driving the Move to NoSQL?

- Traditional applications were designed to support hundreds of concurrent users
- Today, it's not uncommon for applications to need to scale to hundreds of thousands of active users
- This has forced database designers to seek out alternatives

What's Driving the Move to NoSQL?



Performance



Scalability



Lightweight



Productivity



Flexibility

NoSQL Technologies

- Four primary categories for these databases
 - Key/value
 - Columnar
 - Document
 - Graph
- Certain vendors - such as OrientDB - offer solutions that incorporate cross-category capabilities

Key/Value Databases

- Stores data using simple hash table
 - Made up of key/value pairs
- Commonly distributed across multiple servers
- Examples include
 - Dynamo
 - Redis
 - Riak

Key/Value Data

Key	Value
...	...
1743	191.21, Sobchak, 12/30/1993, 0x212
1744	Kerabatsos, 73, 12.55, 0xEAF
1745	Quintana, 6/10/1996
1746	53.09
...	...

Columnar Databases

- Physical storage is handled via columns, rather than rows
- This improves speed and flexibility
- Examples include
 - Apache Cassandra
 - Google BigTable
 - SAP HANA

Columnar Data

Column	Contents
...	...
EMP_ID	8930 8931 8932 ...
LAST_NAME	Sobchak Kerabatsos Quintana ...
HIRE_DATE	6/10/1996 9/7/2002 1/30/2016 ...
...	...

Document Databases

- Stores collections of individual documents
- Each document is independently structured
 - Often without requiring a schema
 - Greatly enhances flexibility
- Examples
 - OrientDB
 - MongoDB
 - CouchDB
 - Amazon DynamoDB

Document Data

```
{  
  "booking": {  
    "id": "JJKEL9",  
    "flight": "WN 1832",  
    "request": "aisle seat"  
  }  
}
```

```
{  
  "booking": {  
    "id": "NMU324",  
    "flight": "KE 809",  
    "vip": true,  
    "request": "Vegetarian",  
    "special_instructions": {  
      "transfer": "Meet pax at ramp"  
    }  
  }  
}
```

Document Data

```
{  
  "booking": {  
    "id": "JJKEL9",  
    "flight": "WN 1832",  
    "request": "aisle seat"  
  }  
}
```

```
{  
  "documentation": {  
    "DocID": "89392191",  
    "type": "Passport"  
  }  
}
```

```
{  
  "booking": {  
    "id": "NMU324",  
    "flight": "KE 809",  
    "vip": true,  
    "request": "Vegetarian",  
    "special_instructions": {  
      "transfer": "Meet pax at ramp"  
    }  
  }  
}
```

```
{  
  "documentation": {  
    "DocID": "90299112-CA",  
    "type": "Passport",  
    "DocID": "822755_QK3",  
    "type": "Visa"  
  }  
}
```

Document Data

```
{  
  "booking": {  
    "id": "JJKEL9",  
    "flight": "WN 1832",  
    "request": "aisle seat"  
  }  
}
```



```
{  
  "documentation": {  
    "DocID": "89392191",  
    "type": "Passport"  
  }  
}
```

```
{  
  "booking": {  
    "id": "NMU324",  
    "flight": "KE 809",  
    "vip": true,  
    "request": "Vegetarian",  
    "special_instructions": {  
      "transfer": "Meet pax at ramp"  
    }  
  }  
}
```



```
{  
  "documentation": {  
    "DocID": "90299112-CA",  
    "type": "Passport",  
    "DocID": "822755_QK3",  
    "type": "Visa"  
  }  
}
```

Graph Databases

- Collections of vertices linked to each others by edges
 - Vertices equate to records
 - Edges equate to relationships
- Examples
 - OrientDB
 - Neo4J

Introducing Graph Databases

Graph Data



Chico



Groucho

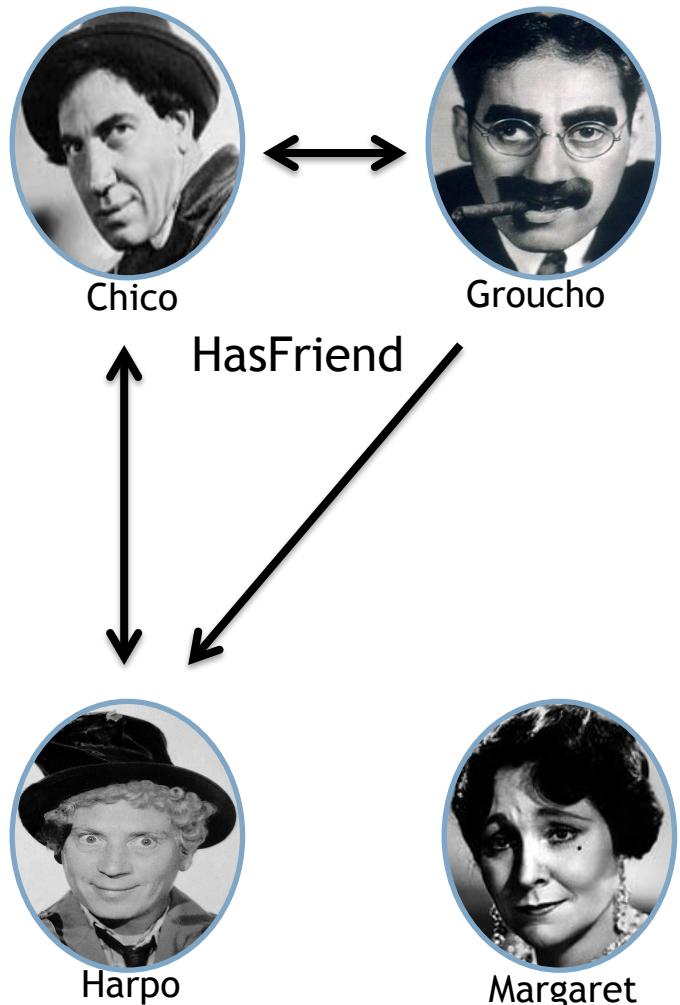


Harpo

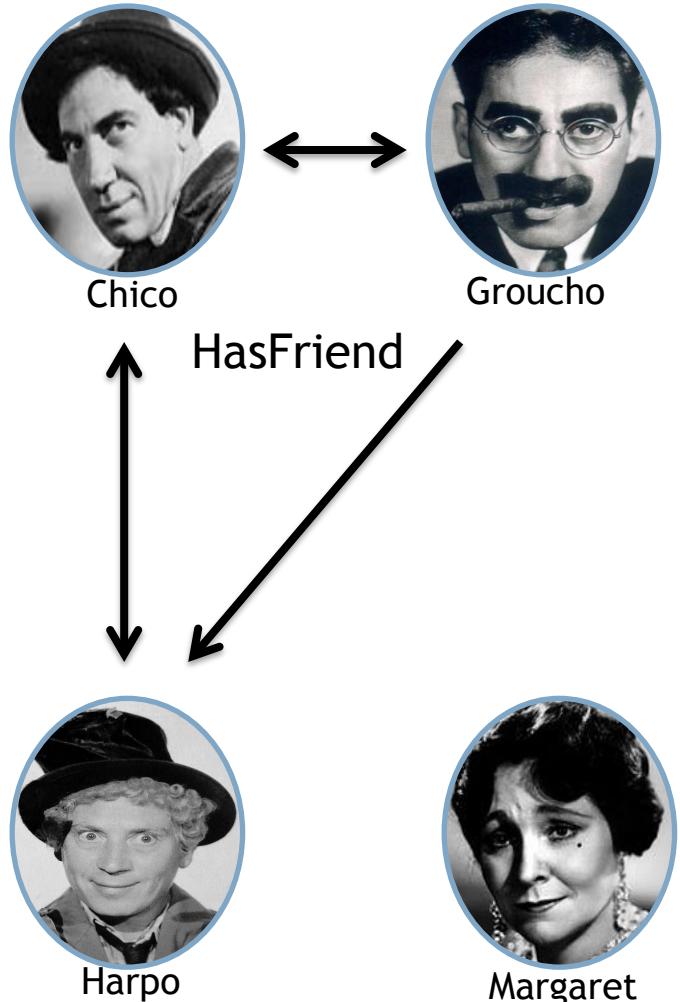


Margaret

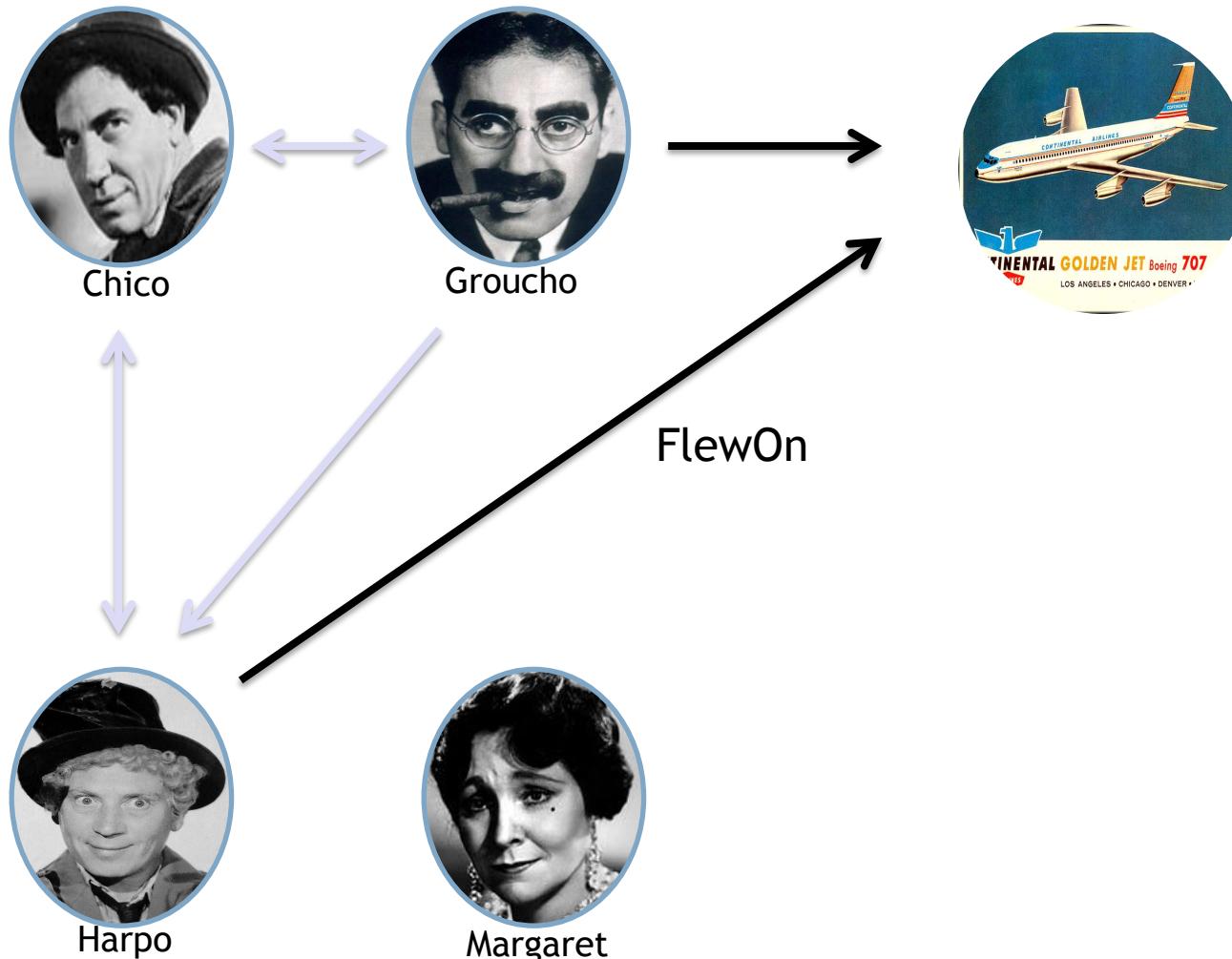
Graph Data



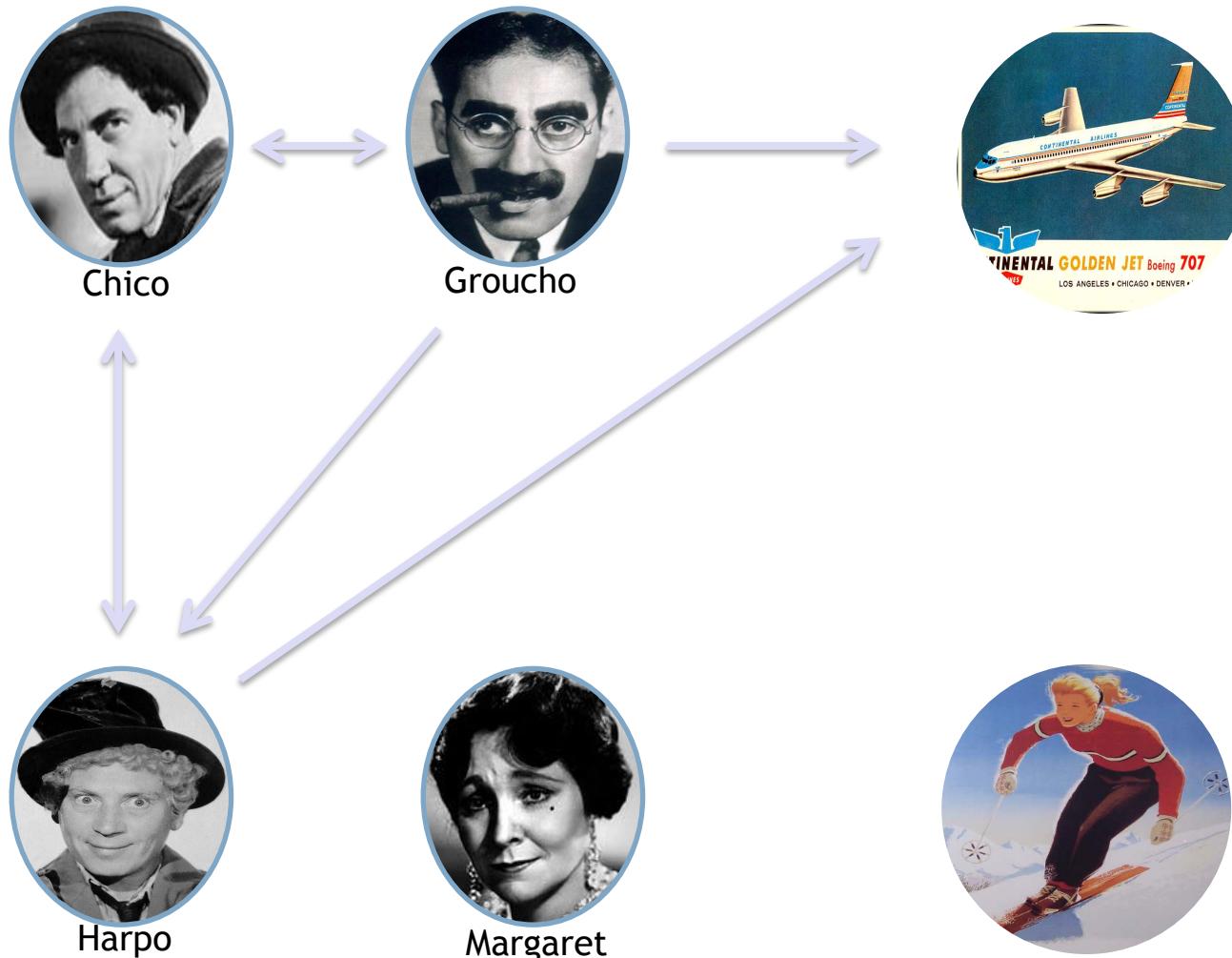
Graph Data



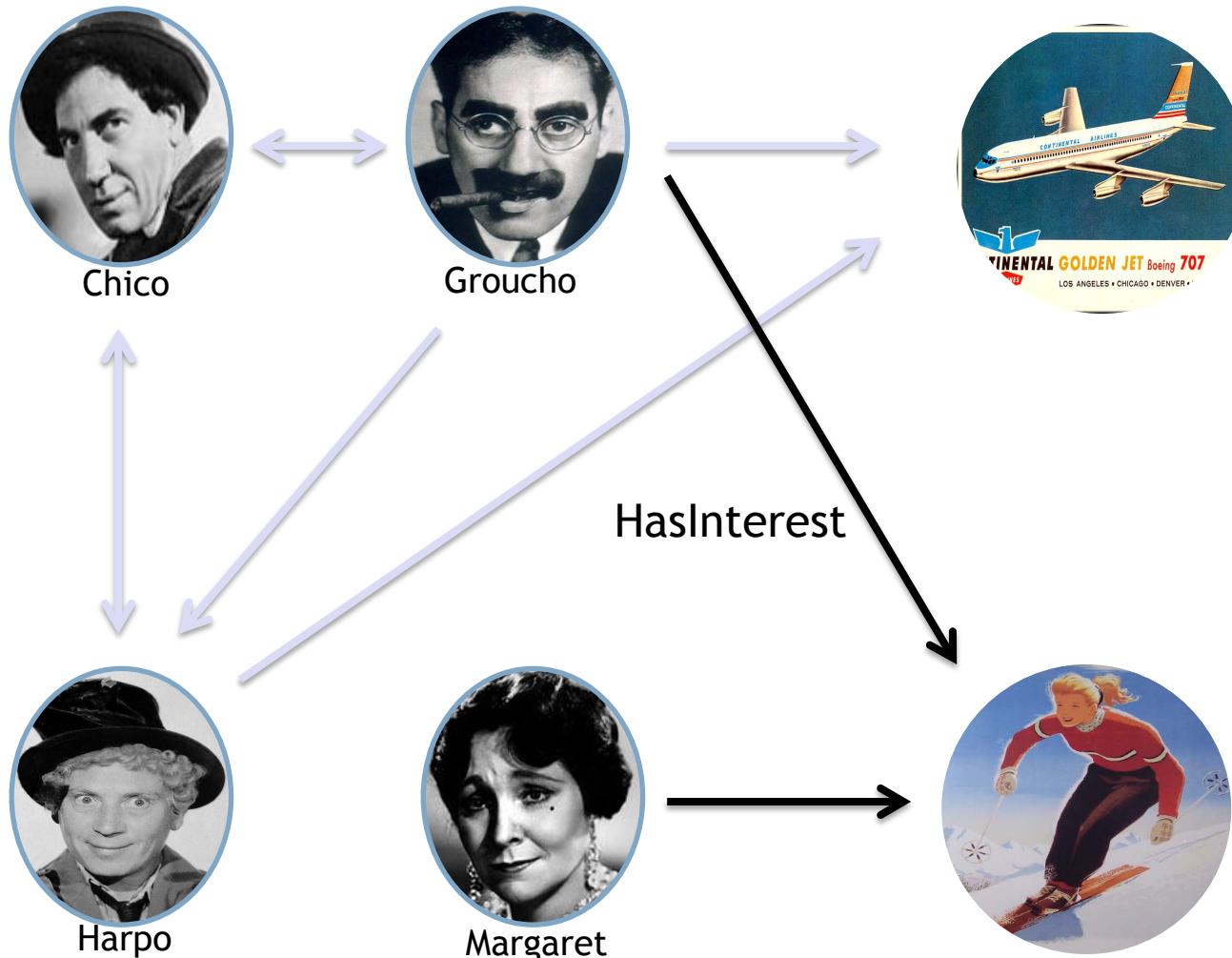
Graph Data



Graph Data



Graph Data



Introducing Graph Databases

Graph Data



Chico



Groucho



CONTINENTAL GOLDEN JET Boeing 707
LOS ANGELES • CHICAGO • DENVER



Harpo



Margaret



Graph Data



Chico



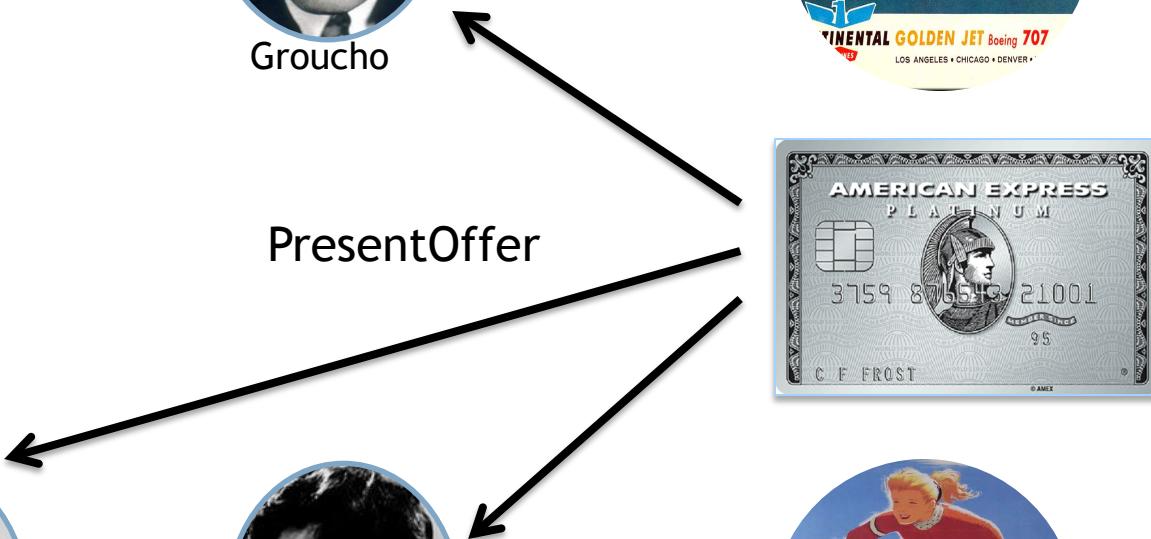
Groucho



Harpo



Margaret

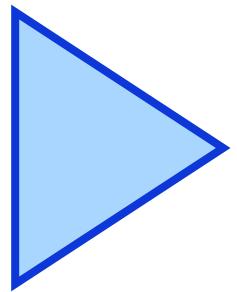


NoSQL Shortcomings

- Although they provide massive scalability and productivity advantages, many NoSQL technologies have some drawbacks
- These gaps include
 - Referential integrity
 - Relationships
 - Transactions
 - Developer and user-friendly access methods

NoSQL Shortcomings

- Lack of easily navigated relationships is one conspicuous deficiency
 - Greatly diminishes the potential for deriving maximum value from the organization's entire data set
- Graph databases are designed to meet this need



Graph Databases

- How They Work
- Communicating With a Graph Database
- Ideal Use Cases
- Comparing RDBMS with Graph Databases

What is a Graph Database?

“A graph database is any storage system that provides index-free adjacency”

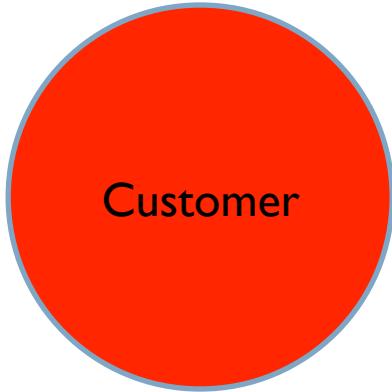
Marko Rodriguez (author of TinkerPop Blueprints)

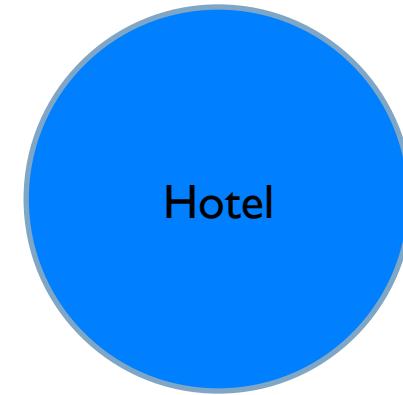
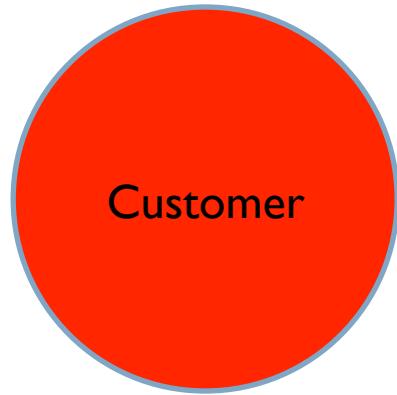
How Does a Graph Database Work?

- Composed of collections - often very large - of vertices and edges
- Vertices
 - Key/value pairs holding properties
 - Can be linked to other vertices
 - Also known as “nodes”

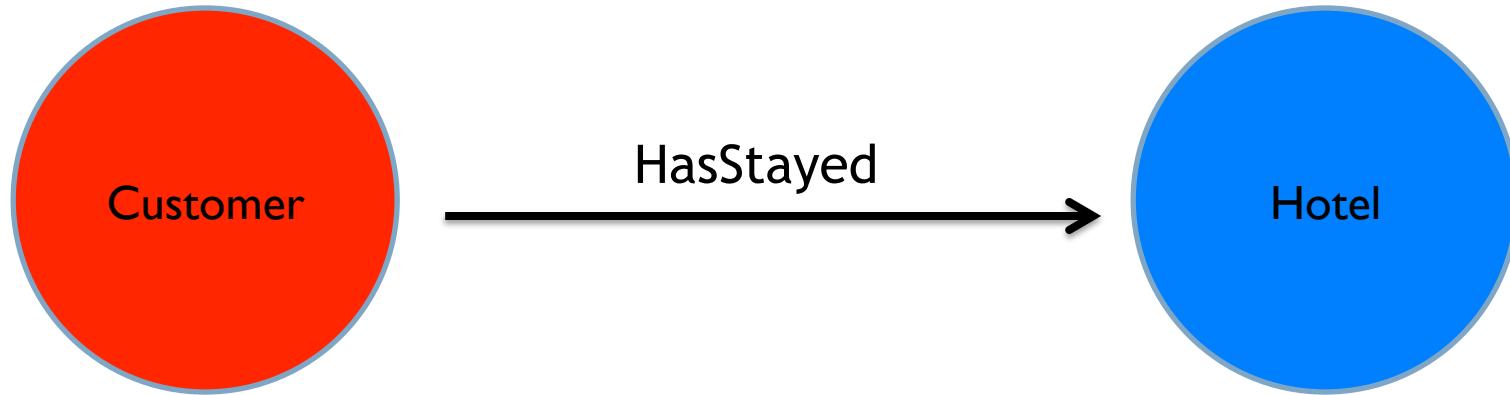
How Does a Graph Database Work?

- Edges
 - Provides linkage between vertices
 - Typically labeled to document the relationship
 - Can have their own properties
 - Also known as “arcs”

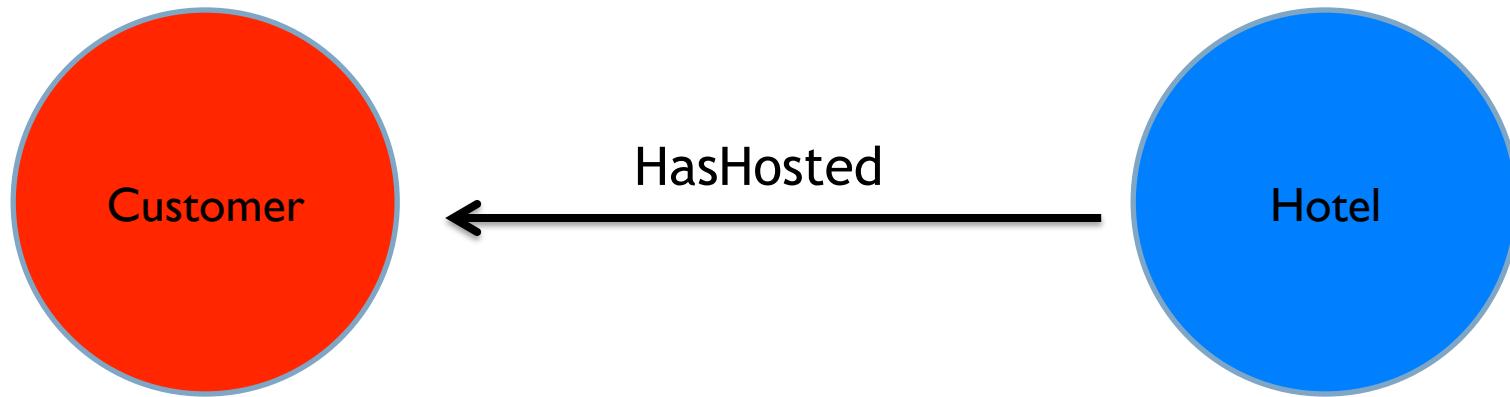




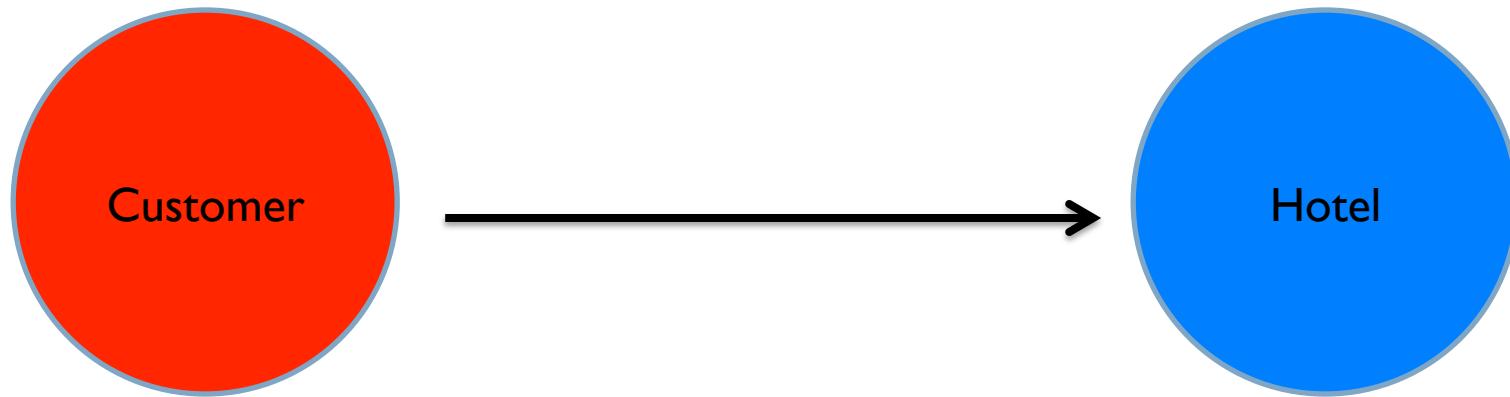
Introducing Graph Databases



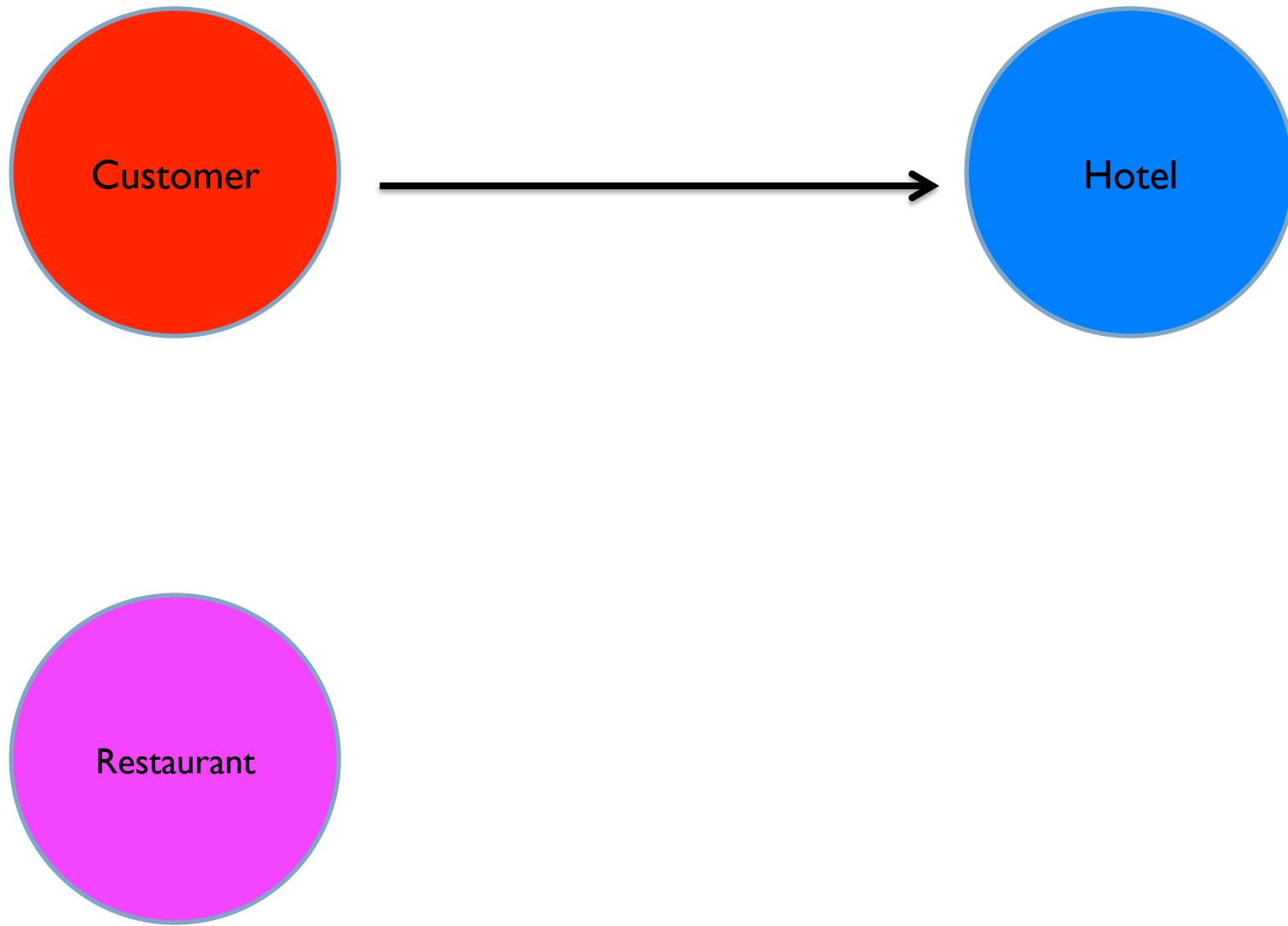
Introducing Graph Databases

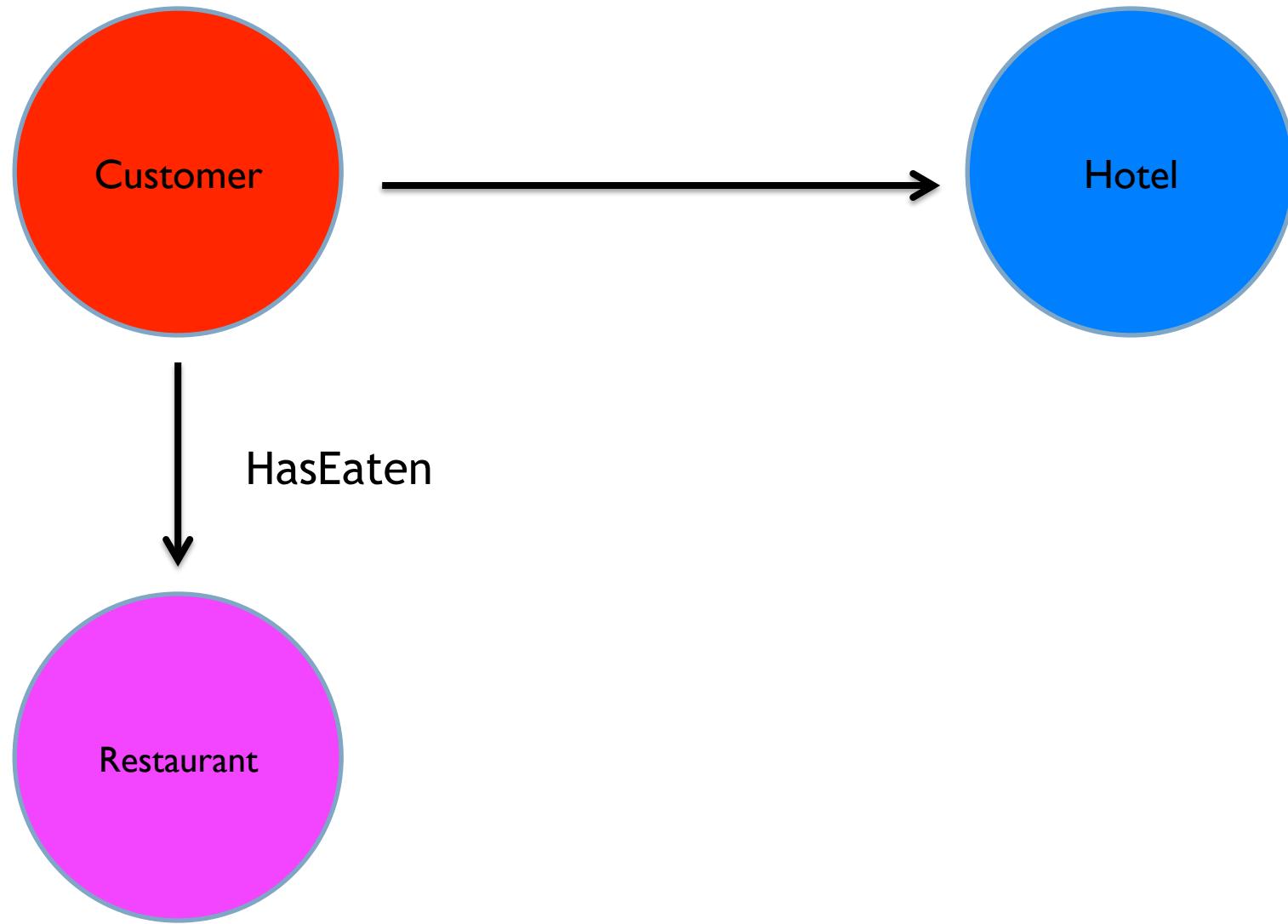


Introducing Graph Databases

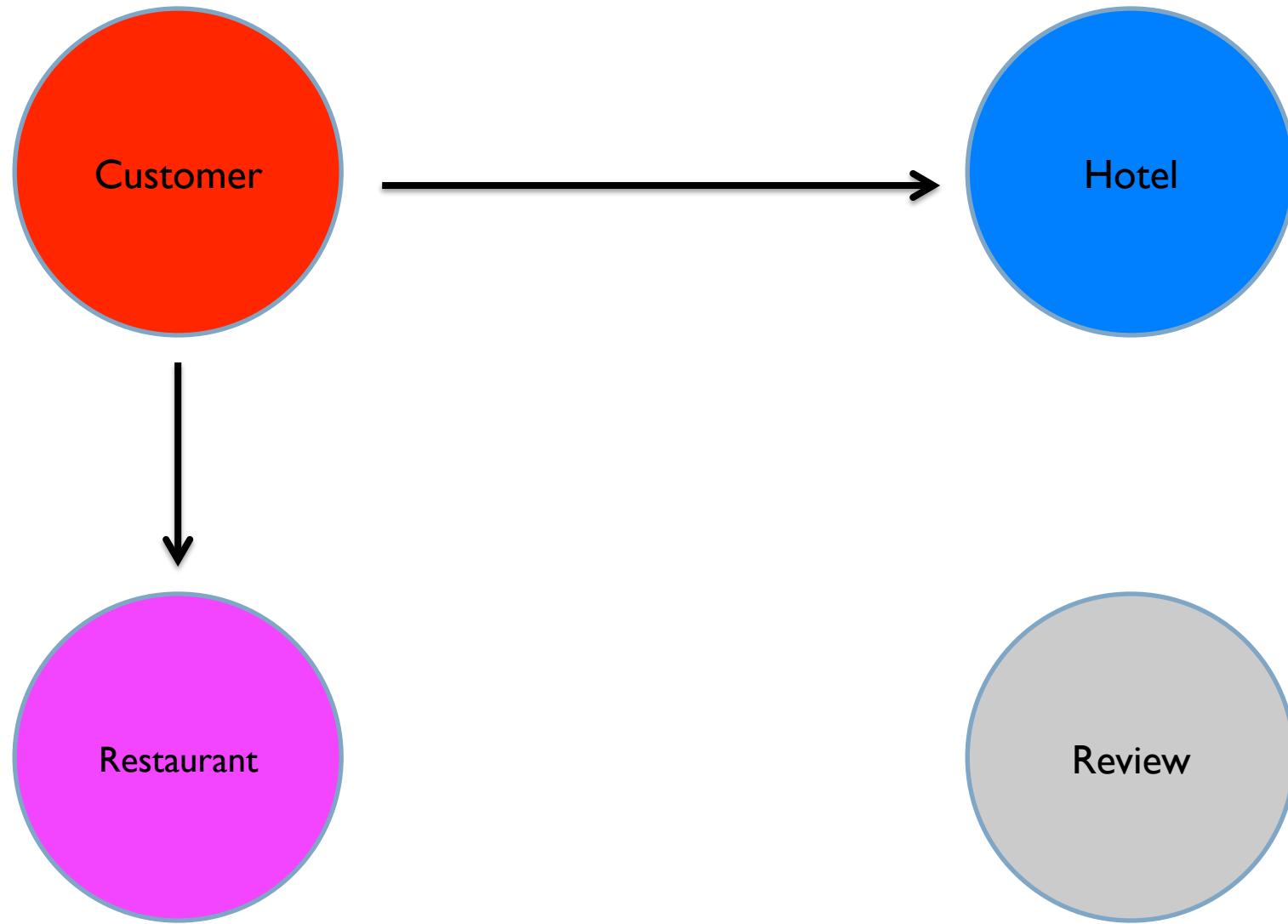


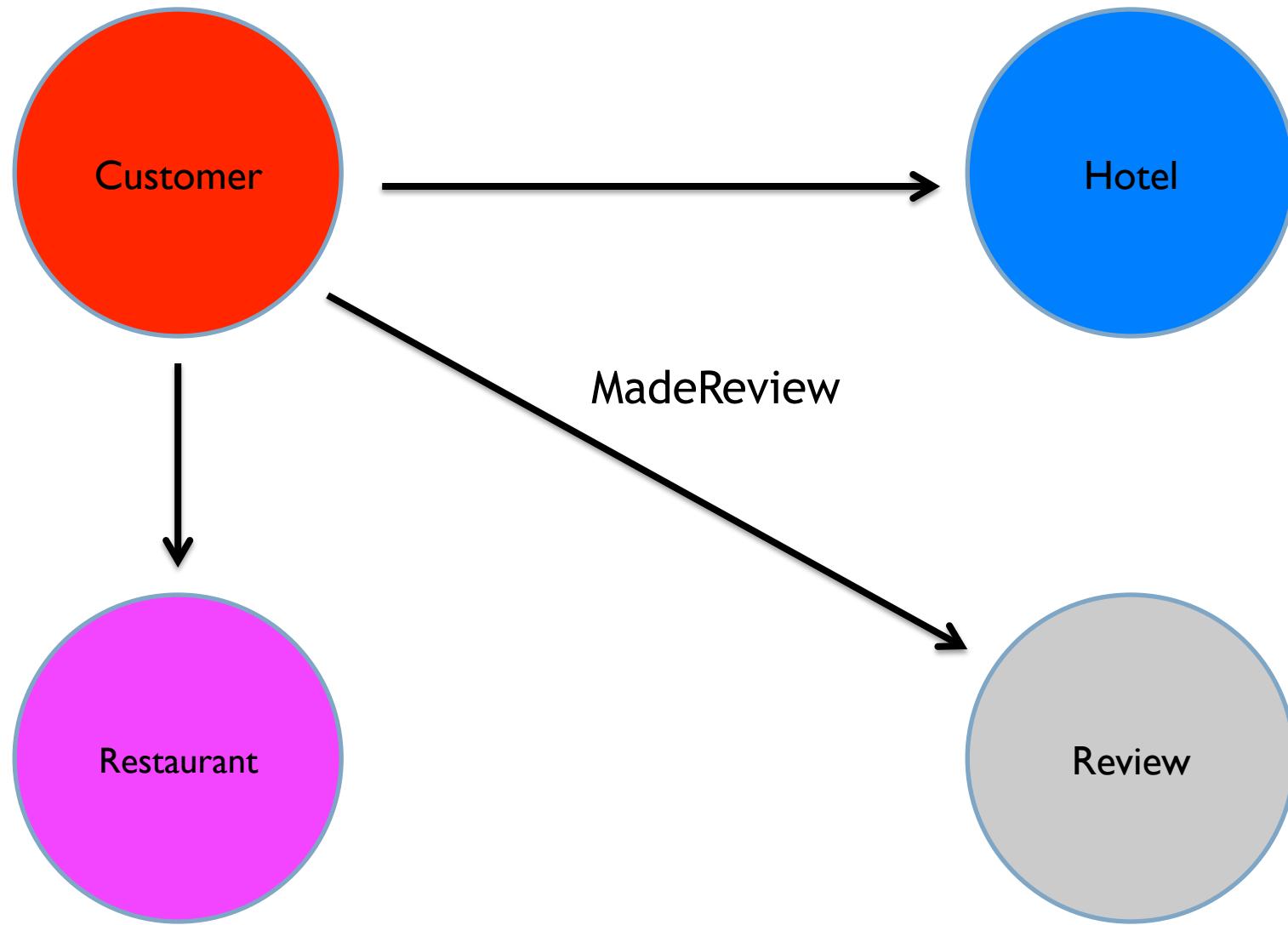
Introducing Graph Databases



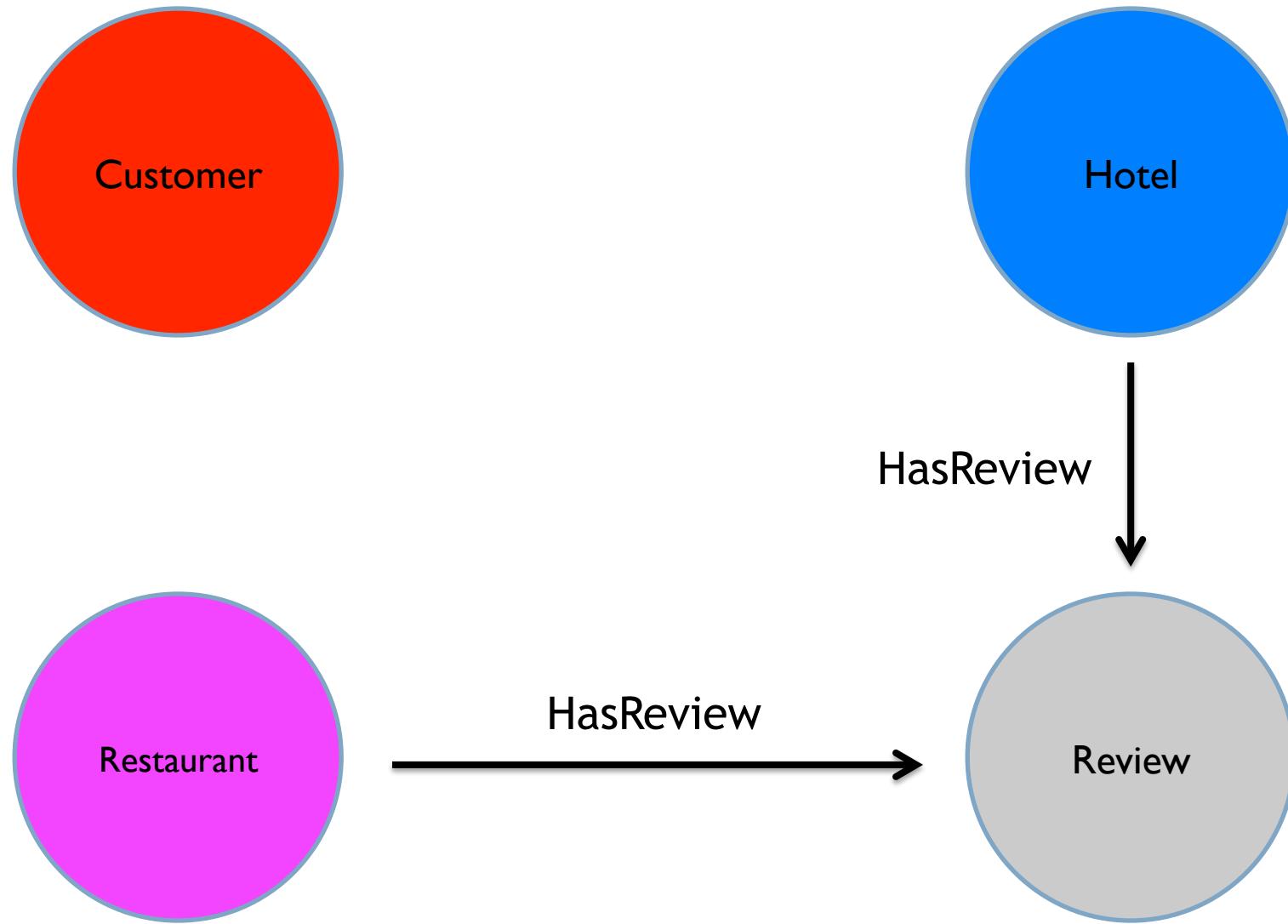


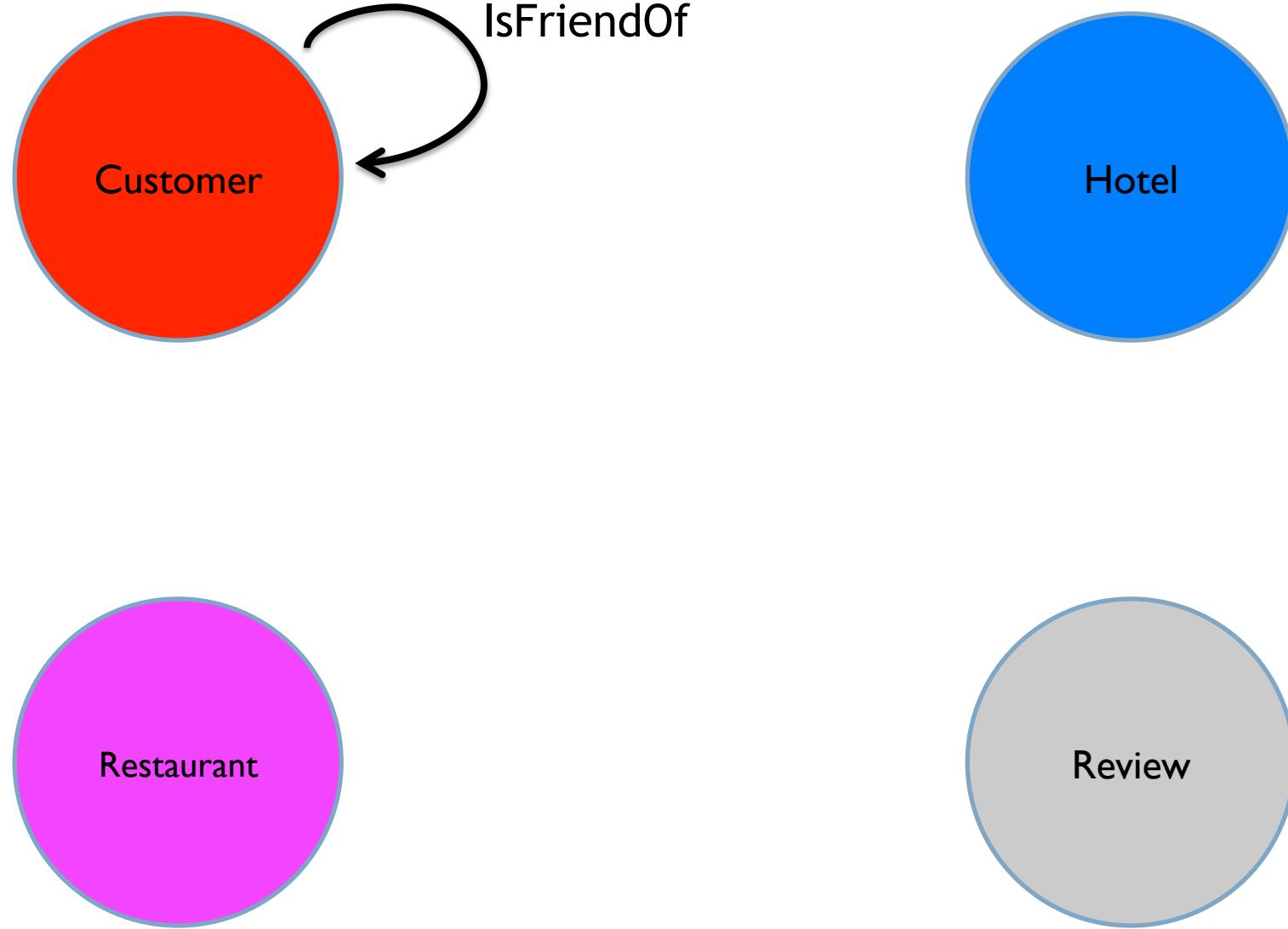
Introducing Graph Databases





Introducing Graph Databases





How Does a Graph Database Work?

- Vertex and edge approach provides much greater scalability
- Also makes it possible to ask all sorts of interesting questions
 - Search for the initial response to a query
 - And then navigate the edges to related vertices
- This is ideal for Big Data

How Does a Graph Database Work?

- Graph databases come with an impressive assortment of technical advantages
- Beyond these benefits, there's one even greater reason to use them:
 - They model the real world much more easily - and accurately - than other database technologies

Communicating with a Graph Database

- Just as with RDBMS, there are multiple ways to communicate with a graph database
 - Programmatic
 - Interactive
- Three primary approaches
 - APIs
 - Query languages
 - Tools

API Communication

- Two major alternatives
 - Platform-specific
 - Cross-platform frameworks
- Platform-specific
 - Each vendor supplies APIs in multiple languages
 - JDBC drivers are yet another way
- Cross-platform frameworks
 - Apache TinkerPop

Query Language Communication

- Two major alternatives
 - Specialized graph languages
 - SQL
- Specialized graph languages
 - Gremlin
 - SPARQL
 - Cypher

Query Language Communication

- SQL
 - Offered by OrientDB
 - Based on standards such as ANSI-92
 - Augmented with extensions for graph database concepts
- As we'll see later, this approach makes it much easier for RDBMS professionals to quickly get productive

Tools

- Available from vendors and third parties
- Command line
 - Quick and easy way to interact with data
- Visual
 - OrientDB Studio is an example
 - Third party business tools are another option

What Can You Ask of a Graph Database?

- One of the most compelling attributes of a graph database is how easy it lets you probe your data in new and innovative ways
- Prior to graph databases, these types of queries required extensive programming
 - And commonly demanded changing the RDBMS database structure to support more many-to-many relationships

What Can You Ask of a Graph Database?

“Which frequent flyers on flights during last Christmas season responded to our credit card promotion?”

“How much impact did our marketing campaign have on the average order size from inflight shopping when families were traveling together?”

What Can You Ask of a Graph Database?

“Did holding a special promotion on vacation packages have an impact for people who traveled with co-workers?”

“Does reaching out to opinion leaders with special catalog offers have any effect with their friends and families?”

RDBMS vs. Graph Databases

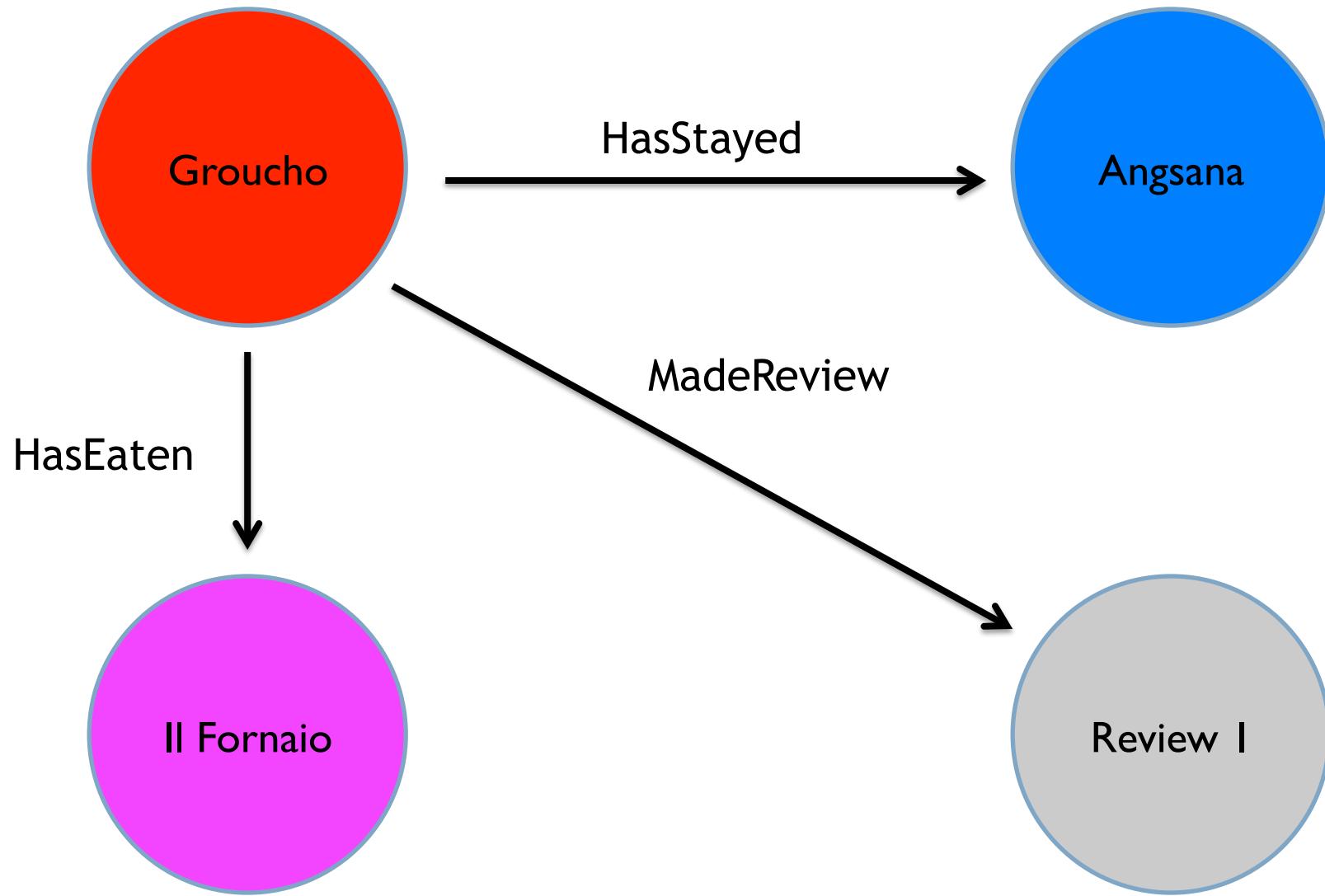
RDBMS	Graph
Table	Vertices & edges set
Row	Vertex
Column	Key/value pairs
Joins	Edges

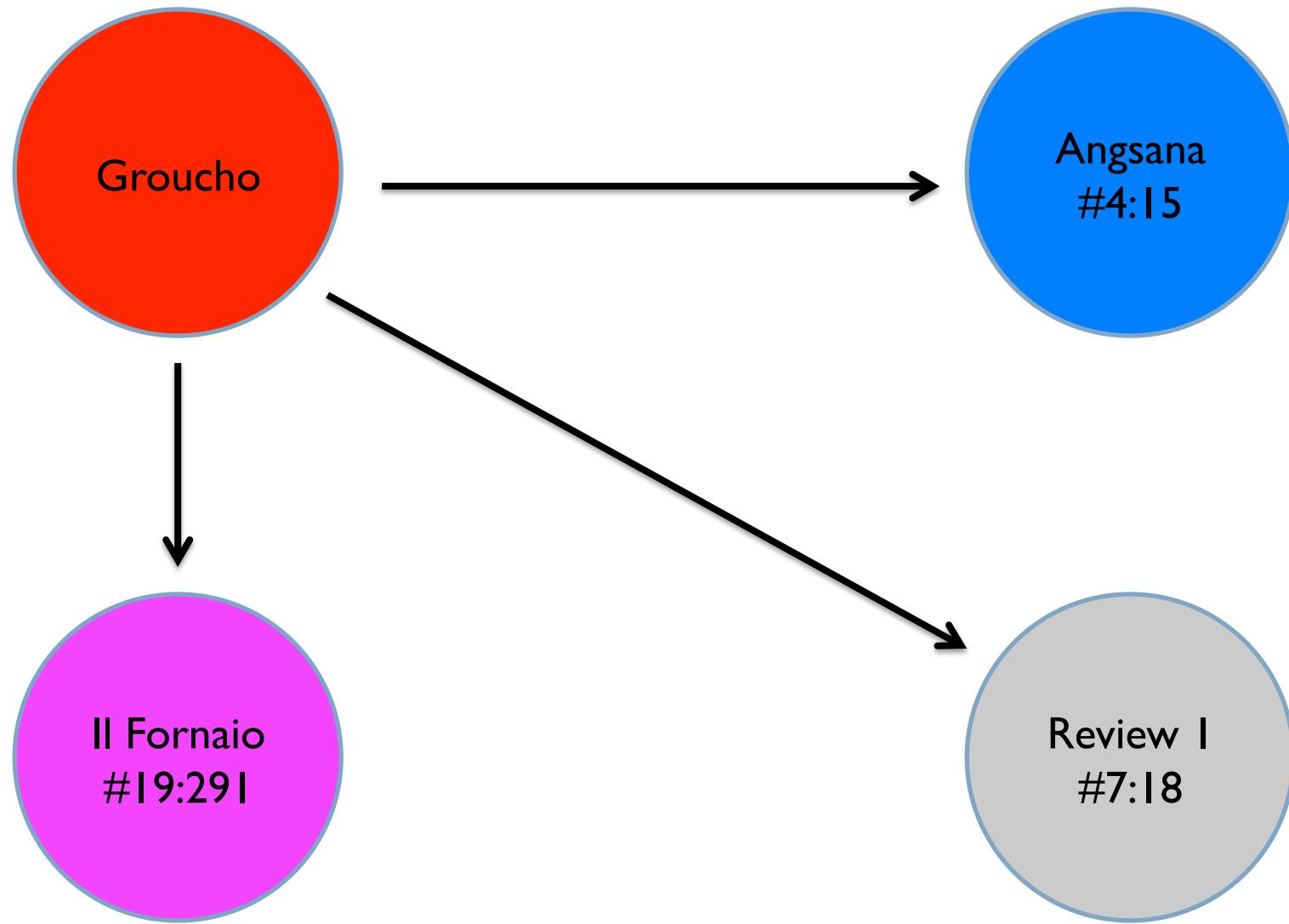
RDBMS Relationships

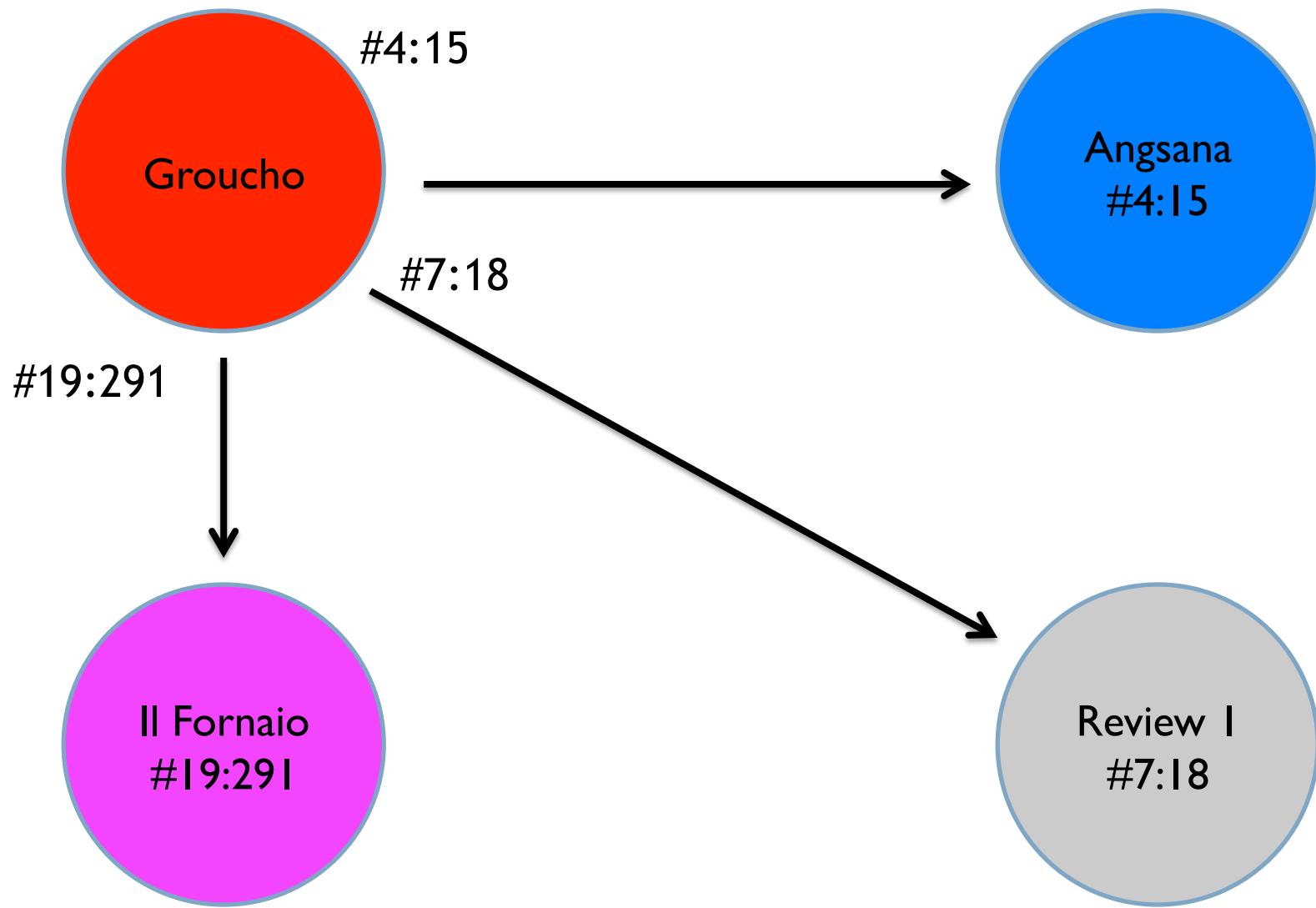
- Keys
 - Primary: Preserves uniqueness
 - Foreign: Provides linkage to records in other tables
- One-to-one
- One-to-many
- Many-to-many

Graph Relationships

- Multiple edges to represent 1-N and N-M relationships
- No joins, but direct links instead
 - Linked record ids are stored
- Multiple relationships are expressed as a List, Set, or Map of links







Introducing Graph Databases



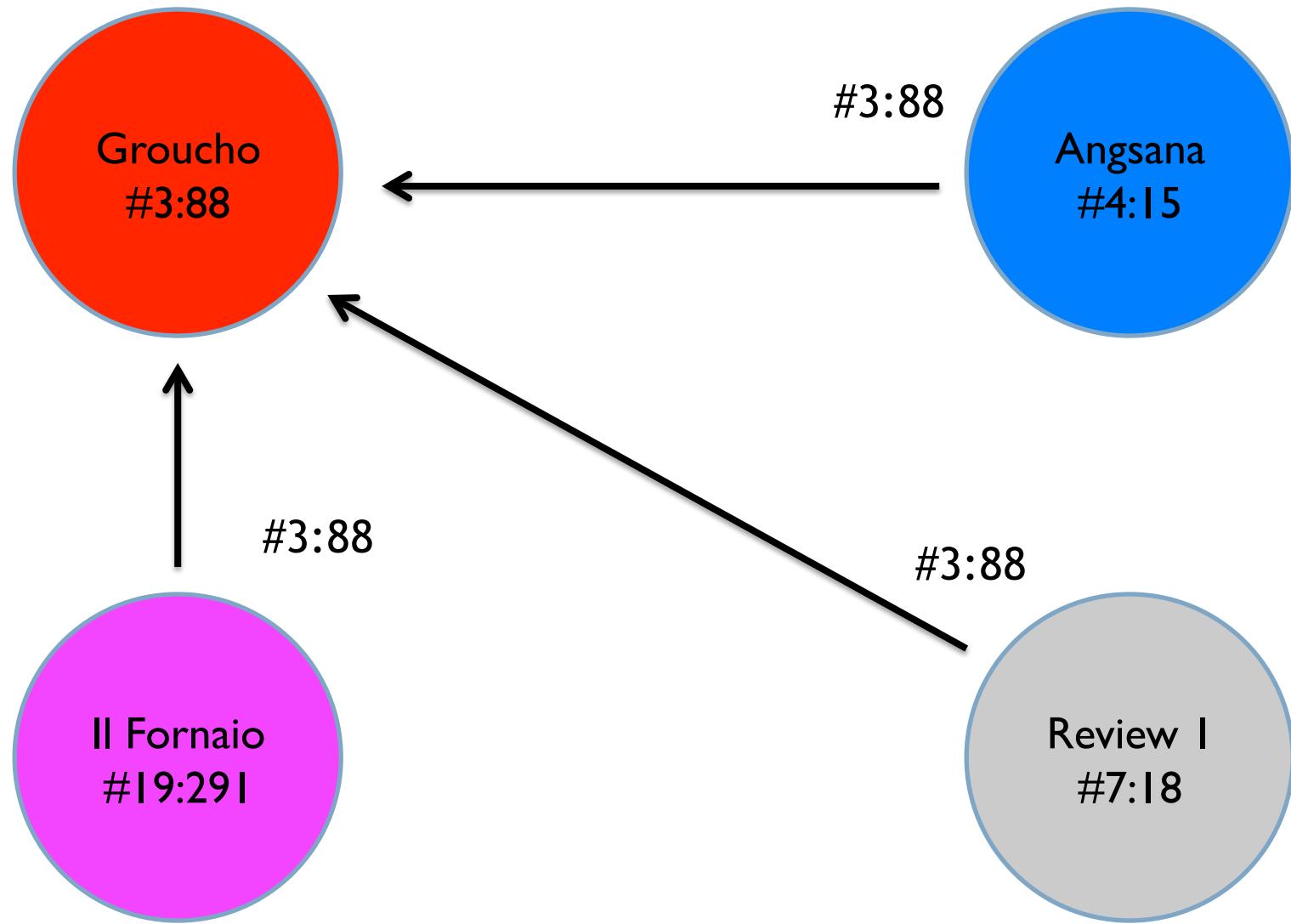
#3:88



#3:88

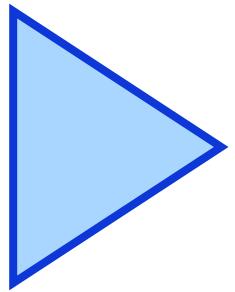
#3:88





Ideal Graph Database Use Cases

- Big Data
 - High performance requirements
- Business intelligence
 - Open-ended and/or unplanned queries
- Very dynamic environments
 - Rapidly evolving data models
 - Or situations where there isn't a fixed data model



Introducing OrientDB

- Product Background
- Multi-Model Architecture
- Storage and Connections
- Supporting and Embedded Technologies

Unit Goals

- Discuss the origins and evolution of OrientDB
- Describe its technical architecture
- Examine the major components and technologies that it includes
- Set the stage for modeling a database

Four Pillars of OrientDB



Performance



APIs



Operations



Security

OrientDB Editions

- Written in 100% Java
- Community edition
 - Download source from <https://github.com/orientechnologies/orientdb>
- Enterprise edition
 - A scalable, robust, and secure multi-model database

Enterprise Edition

- Enhanced security and RDBMS synchronization
- Incremental nonstop backup
- Expanded support for replication to multiple data centers
- 24/7 support with SLA plus quick priority issue resolution

Enterprise Edition

- Commercial licenses for test and production
- Query profiler
- Distributed clustering
- Live monitoring with alerts & metrics recording
- Direct private chat channel to development

OrientDB Advantages

- From the beginning, OrientDB was architected to support the most demanding environments
- This also includes addressing some of the most prominent shortcomings of existing NoSQL technologies

OrientDB Advantages

- Multi-Model architecture
- Flexible schema options
- Numerous ways to access data
- High performance
- Comprehensive security

Multi-Model Architecture

- Built into core engine
 - Rather than needing to use an API translation layer
- Better speed and scalability
- Linkage to RDBMS

Multi-Model Architecture

- Model options
 - Graph
 - Document
- Multi-model capabilities are combined into a single user and developer experience

Flexible Schema Options

- Schema-less
- Schema-full
- Schema-mixed

Numerous Ways to Access Data

- Multi-Model API
 - Provides Document and Graph capabilities in a unified API
- SQL
 - With necessary extensions for graph data architecture
- Apache TinkerPop & Gremlin

Numerous Ways to Access Data

- JSON
 - Direct insertion to database
 - Updates (merges) to current record
- Embedded functions, methods, and hooks

High Performance

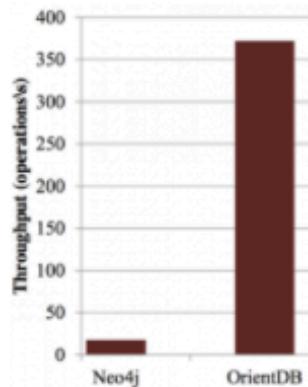
- Caching
- Clustering
- Sharding
- Replication

Introducing OrientDB

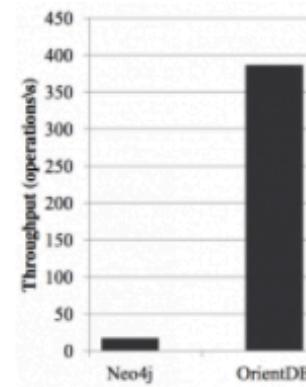
Benchmark

Workload A – Update heavy:

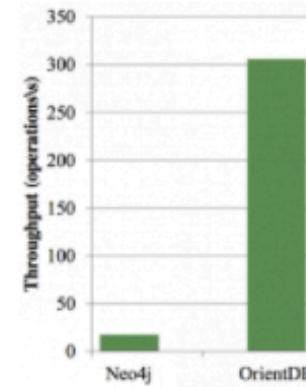
A mix of 50/50 read/update workload. Read operations query a vertex V and reads all its attributes. Update operation changes the last login time.


Workload B – Read mostly:

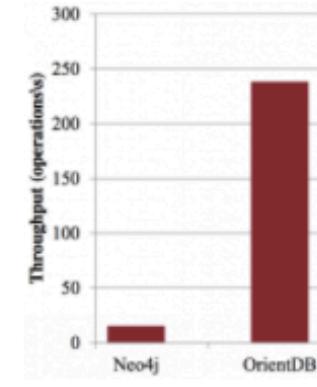
The following graph represents a mix of 95/5 read/update workload. Those operations are similar to Workload A.


Workload D – Read latest:

Inserts new vertices to the graph. The inserts are made in such a way that the power law relations of the original graph are preserved.

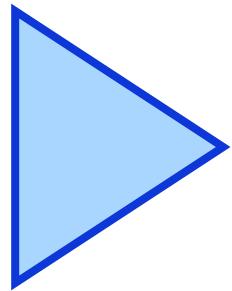

Workload E – Short ranges:

Reads all the neighbouring vertices and their Vertex attributes. For Example loading the closest friend to a person in a social graph.



Comprehensive Security

- Authentication
- Auditing
- Encryption
- Fine-grained control



Multi-Model Architecture

- Architecture
- Storage Models
- Classes
- Document
- Graph

Multi-Model Architecture

- NoSQL means choosing the right tool for the job
- OrientDB was built around a multi-model architecture
- This provides maximum flexibility when designing a database

Multi-Model Architecture

- OrientDB uses a consistent internal storage mechanism for graph and document data
- Data is persisted using key/value pairs
 - Also known as fields or properties
- Key provides access to the value

Multi-Model Architecture

- Values may be
 - Primitive data types
 - Embedded documents
 - Arrays of other values
 - List, sets, maps of any kind

- Advantages
 - Flexibility
 - Speed

Multi-Model Architecture

- Classes are the logical units that underpin everything in OrientDB
- They're the foundation of graph & document capabilities
- OrientDB's class architecture offers object oriented benefits

Multi-Model: Document

- Often encoded in JSON
- Generally don't require schemas
- Offers better adaptability to change
- Easier to modify

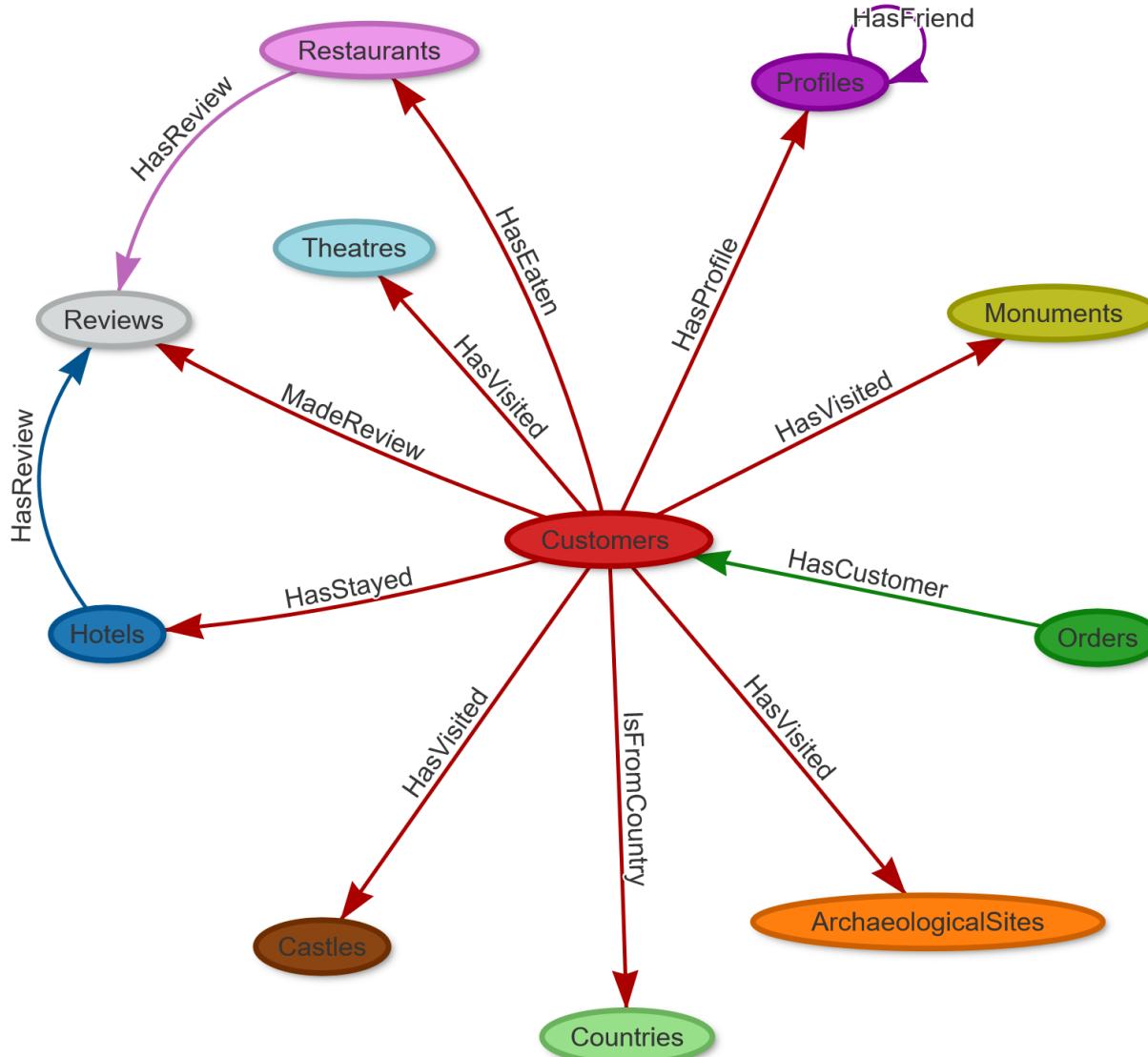
RDBMS vs. Document vs. OrientDB

RDBMS	Document	OrientDB
Table	Collection	Class or Cluster
Row	Document	Document
Column	Key/value pairs	Document field
Relationship	N/A	Link

Multi-Model: Graph

- Network-like structure
- Properties can make graph databases seem similar to document model
- Key constructs are vertices and edges
- Underlying storage is key/value

Introducing OrientDB



Multi-Model: Graph

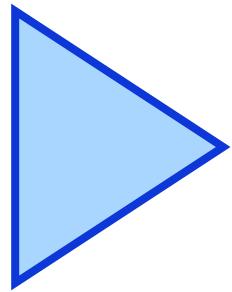
- Vertex
 - Unique identifier
 - Set of incoming edges
 - Set of outgoing edges
 - Custom (user-defined) properties
- Examples
 - *Passenger*
 - *Flight*
 - *Order*

Multi-Model: Graph

- Edge
 - Unique identifier
 - Link to incoming vertex (“Head”)
 - Link to outgoing vertex (“Tail”)
 - Label that defines the relationship
 - Optional custom (user-defined) properties
- Examples
 - *Related To*
 - *Was Transported By*
 - *Applied For*

RDBMS vs. Graph vs. OrientDB

RDBMS	Graph	OrientDB
Table	Vertices & Edge set	Vertices & Edge set + subtype sets
Row	Vertex	Vertex
Column	Vertex/Edge property	Vertex/Edge property
Relationship	Edge	Edge



Connections & Storage

- APIs & Drivers
- Connections
- Storage Models
- Clusters
- Record Identifiers
- Indexes
- Caching
- Transactions

OrientDB Interfaces

Type	Description
Native binary remote	Communicates via the binary protocol over TCP/IP
HTTP/REST	Communicates via HTTP protocol over TCP/IP
Java-wrapped	Any language capable of using the Java virtual machine to link in the native OrientDB driver

OrientDB Languages & APIs

- OrientDB supports application development using a variety of languages and related APIs
- The drivers for interacting with OrientDB are either Java-wrapped or binary

Language	Driver Type	Source
Java	Binary	Built-in to OrientDB
JavaScript	Binary	node.js
PHP	Binary	PhpOrient
Python	Binary	PyOrient
C#	Binary	Innov8tiveSoftware
Scala	Java-wrapped	
Groovy	Java-wrapped	OrientDB Groovy
JRuby	Java-wrapped	OrientDB JRuby

Official APIs

Type	Source
Multi-model	OrientDB (as of 3.0)
Graph	OrientDB
Document	OrientDB
TinkerPop2 Blueprints	OrientDB
TinkerPop3 Gremlin	orientdb-gremlin
JDBC	orientdb-jdbc
.NET	Innov8tiveSoftware
Spring Data	spring-data-orientdb

JSON Support

- OrientDB provides native JSON support
- JSON capabilities are found throughout the OrientDB environment, including:
 - ETL (import/export)
 - Console interaction
 - REST API
 - SQL

Connecting to OrientDB

- Two connection modes
 - Embedded
 - Client/server
- Appropriate connection mode will depend on planned workload and number of users

Embedded Connection

- Runs in the same Java VM as the application itself
 - Uses fewer resources
 - No TCP/IP transport
- Sample connection string:
`plocal:/tmp/db/flights`

Embedded Connection

- Don't overlook the potential benefits of embedded connections
- They may be very helpful for certain transient activities
 - Automated testing
 - Mass data imports

Client/Server Connection

- Client and server run in separate JVMs
- Choice of binary or HTTP TCP/IP transport
- Syntax: remote:<server>:[<port>]/db-name
`remote:localhost:2424/flights`
- TCP/IP ports
 - HTTP on 2480, binary on 2424

Storage Models

- OrientDB offers two primary ways of persisting information
 - Disk
 - Memory
- In addition to user-created databases, OrientDB always offers a temporary, in-memory database

plocal

- Stands for ‘paginated local storage’
 - Uses file system on local computer
 - Known as ‘local’ in earlier versions
- OrientDB maintains a log of changes for crashes
 - Helps restore data that had not yet been persisted
- Sample connection string:
`plocal:/tmp/db/flights`

Memory

- All information stored in RAM
 - No disk involved
 - High performance
- Highly volatile option
 - Shutting down server destroys data
- Sample connection string:
`memory:sort_buffer`

Clusters

- All data in OrientDB is saved to a cluster
 - Similar concept to a relational database table
- Clusters are groupings of records
 - By default, OrientDB creates multiple clusters per class
 - This takes advantage of parallel processing
- Clusters may be physical or in memory

Clusters

- Quantity of CPUs has an impact on the number of clusters
- By default, queries span all clusters
 - Unless specifically targeted in the query syntax

Clusters

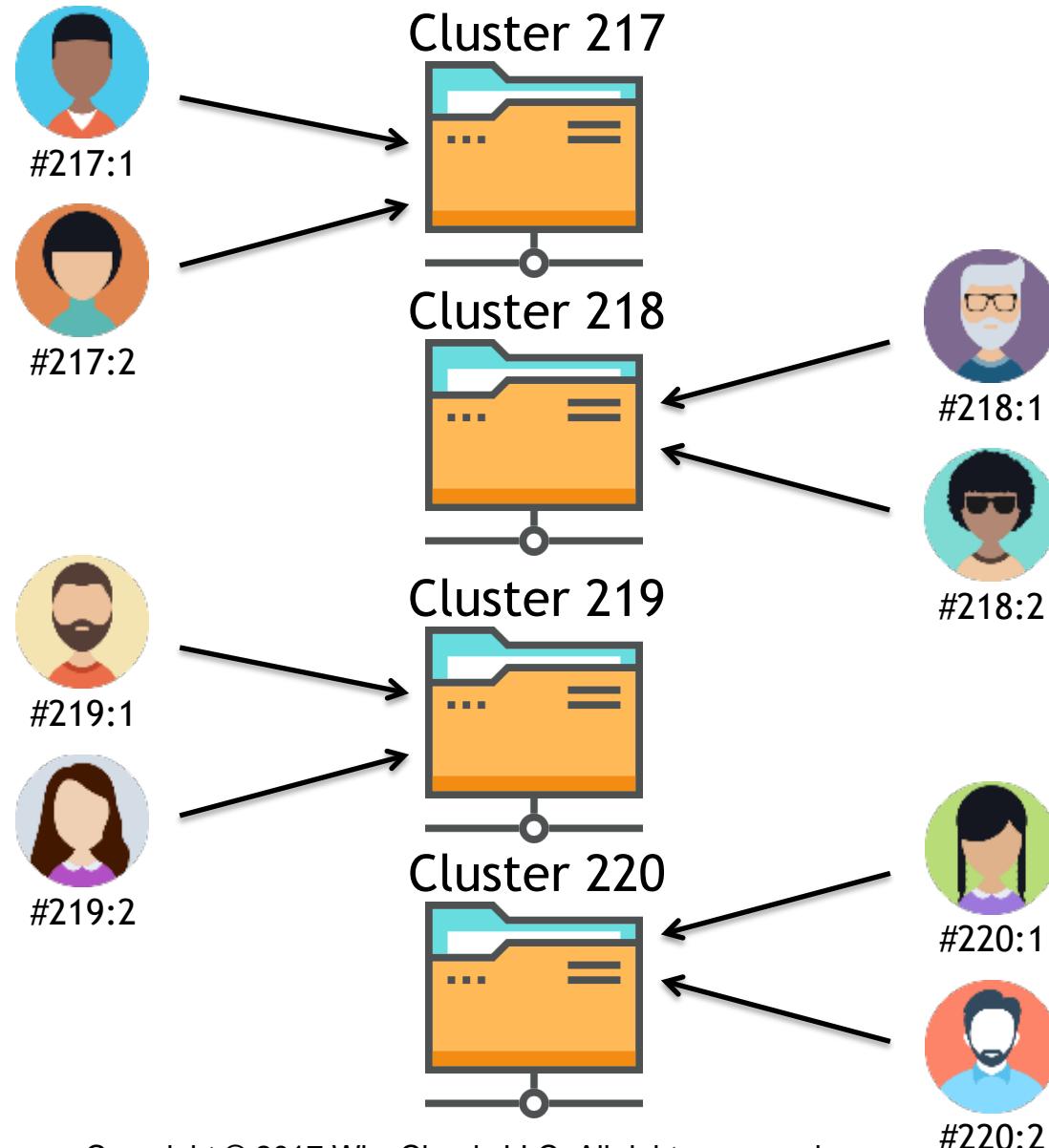
- Physical cluster
 - Uses traditional storage
 - Creates two files on file system
 - .pcl files store the actual data
 - .cpm files store the mapping between a record's cluster position and its real physical position
- In-memory cluster
 - Can be used to help physical databases with temporary storage
 - Cleared when database shuts down

Clusters

- Clusters provide vast storage capabilities
 - A single cluster can hold up to 9,223,372,036,854,775,808 records (2^{63})
 - A single database can have up to 32,768 clusters
- Overall record capacity
302,231,454,903,657,293,676,544 (2^{78})

Record Identifiers

- Each record - regardless of type - in an OrientDB database has a record identifier
- This is guaranteed to be unique
- Composed of two parts
 - Cluster number
 - Record position within the cluster

Introducing OrientDB

Indexes

- Indexes are indispensable aids to speed up queries
- However, they can slow down insertions and updates
- OrientDB indexes may be either automatic or manual

Automatic Indexes

- Comparable to RDBMS indexes
- Index is bound to one or more schema properties
- Only possible when there's a schema in place
- OrientDB will maintain the resulting index without the need for developer/admin work

Manual Indexes

- These are managed by developers or administrators directly
- Helpful for maintaining key/rid pairs for fast lookup
- They have “dictionary” as the default manual index used for root nodes

Index Types

Type	Summary
UNIQUE	Does not allow duplicate entries. Serves as a constraint to guarantee uniqueness.
NOT UNIQUE	Enables storing duplicate values
Composite	Concatenates multiple properties into a single index
FULLTEXT	Utilizes Lucene library to index each word in a schema element

Indexes

- OrientDB offers these index algorithms
 - SBTree
 - Hash-index
 - Auto Sharding
 - Lucene Full Text Index
 - Lucene Spatial Index
- These may be selected as part of the global OrientDB configuration

Index Attributes

Attribute	Meaning
Durable	Changes to base data are immediately reflected in the index
Range queries	Permit a lower and upper bound search on runtime-provided values
Transactional	Index data changes that are made as part of a transaction are visible

Index Algorithm Behaviors

Index	Durable	Trans- actional	Range Queries	Notes
SBTree	Yes	Yes	Yes	Default algorithm
Hash- Index	Yes	Yes	No	Very fast searches; doesn't consume excessive disk resources
Auto sharding	Yes	Yes	No	Supplies distributed hash table capabilities
Lucene Full Text	Yes	Yes	Yes	Only works for full-text data
Lucene Spatial	Yes	Yes	Yes	Only works for spatial data

Caching

- Well-proven technique to improve performance
- Retrieves data already present in memory to reduce costly disk reads
- Behavior is dependent on whether running in local or client/server mode

Caching - Client Side

- Local session cache
 - One cache per `ODatabaseDocumentTx` (transaction)
- If a transaction has started:
 - Transaction is searched for changed, relevant records
- Otherwise:
 - The local session cache is searched for data before contacting server

Caching - Server Side (Disk)

- One per storage device
- Consists of separate caches for reads & writes
- Shared across all threads on the server
- Consulted for queries that the server processes
- Made up of byte arrays that equate to pages

Caching - Server Side (Command)

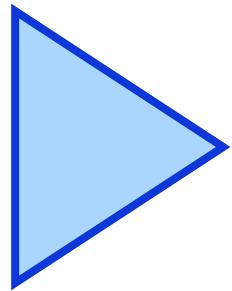
- Meant to maintain recent command results in memory
- Requires sufficient memory to support cache
- Best results for read-intensive database
- Not enabled by default

Transactions

- OrientDB is a fully transactional database
 - Essential for supporting mission-critical applications
- Complete ACID support
 - Atomicity
 - Consistency
 - Isolation
 - Durability

Transactions

- Any multi-user database runs the risk of concurrency problems
 - Phantom records, for example
- OrientDB uses a collection of techniques to prevent these events from occurring
 - Serialization
 - Version control
 - Memory management



Supporting & Embedded Technologies

- JavaScript Engine
- TinkerPop
- Replication
- Integration
- Security
- OrientDB Studio

JavaScript Engine

- OrientDB ships with Nashorn
- Nashorn replaces Mozilla Rhino
 - Beginning with Java 8 and onward
- Provides better compliance with the ECMA normalized JavaScript specification
- Offers better runtime performance

TinkerPop

- For application portability, OrientDB supports the Apache TinkerPop stack
- TinkerPop is a graph computing framework for both graph databases (OLTP) and graph analytic systems (OLAP)
 - Open source
 - Vendor agnostic

TinkerPop

- Collection of interrelated components and features to enable portability
 - Query support (synchronous & asynchronous)
 - Interfaces and APIs
 - Gremlin language
 - Transactions
 - Implementations
 - “Outplementations”
 - Test suites

Replication

- Data distribution onto multiple servers
- Improves performance
- Safeguards data
- Servers can be internal, AWS, or both

Data Distribution

- Distributes data evenly among the nodes in a computing cluster
- Enables horizontal scalability
- Implements multi-master replication
- Transparent to users

Integration

- OrientDB provides several integration solutions
 - ETL
 - Teleporter
 - Neo4J Importer
- It's also possible to use the OrientDB Multi-Model Java API to create customized integration applications

ETL

- Bi-directional data transport
- Can extract from
 - Row
 - CSV
 - JDBC
 - JSON
 - XML
- Transformation flow is a pipeline

Teleporter

- Synchronizes OrientDB with RDBMS
 - Uses JDBC to integrate with major RDBMS offerings
- Creates graph data based on RDBMS tables
- Can be used for a one-time import
 - Can also be used for ongoing synchronization

Neo4J Importer

- Makes it possible to migrate from Neo4J to OrientDB
- Uses Neo4J's Java API to read its graph database
- Uses OrientDB's Java API to import the database

Security

- OrientDB offers comprehensive, yet flexible security
 - Authentication
 - Auditing
 - Record level security
 - At rest data encryption
 - In transit data encryption

Security

- Roles
 - Can be associated with specific classes
 - Roles may be inherited and extended
- Default roles
 - reader
 - writer
 - admin

Security

- New users are assigned to a role
- Default database users
 - reader
 - writer
 - admin
- Server has its own set of users

Security

- OrientDB offers fine-grained, record-level security
- Available permissions
 - Create
 - Read
 - Update
 - Delete

OrientDB Studio

- As described earlier, OrientDB offers two editions of its multi-model database
 - Community
 - Enterprise
- Community includes OrientDB Studio
 - Comprehensive web interface for administering an OrientDB database
 - Also permits full data interaction

OrientDB Studio

- OrientDB Enterprise automatically extends OrientDB Studio with additional capabilities
- These components are designed to streamline the job of supporting a mission-critical deployment

OrientDB Studio Enterprise Extensions

- Dashboard
- Server management
- Backup management
- Query profiler
- Studio auditing
- Data centers
- Cluster management

OrientDB Studio Directory

- [Query Editor](#)
- [Schema Manager](#)
- [Security Manager](#)
- [Graph Editor](#)
- [Database Management](#)
- [Create Record](#)
- [Dashboard](#)
- [Servers Management](#)
- [Servers - Monitoring](#)
- [Servers - Metrics](#)
- [Database List](#)
- [Logs](#)
- [Global Configuration](#)
- [Server Configuration](#)
- [Backup Management](#)
- [Query Profiler](#)
- [Security](#)
- [Teleporter](#)
- [Alerts Management](#)

[Skip to last example](#)

Query Editor

[Return to directory](#)

Screenshot of the OrientDB Studio Query Editor interface.

Toolbar: BROWSE (selected), SCHEMA, SECURITY, GRAPH, FUNCTIONS, DB. User: demodb (root).

Query Input:

```
1 SELECT FROM Profiles WHERE Name = 'Bud'
```

Buttons: RUN, EXPLAIN.

Search: Search in history.

Storage: Local Storage Size: 142.71 KB.

COMMAND: SELECT FROM Profiles WHERE Name = 'Bud'

METADATA			PROPERTIES						IN		OUT		
@rid	@version	@class	Name	Id	Surname	Email	Bio	Gender	HasFriend	Submitted	HasFriend	FlewOn	
#42:125	9	Profiles	Bud	100001	Abbott	bud@example.com	Abbott & Costello	Male	#217:203 #219:203	#258:0 #259:0	#218:202 #220:202 #218:203	#244:0	

Table Options: 10, 25, 50, 100, 1000, 5000.

Message: Query executed in 0.075 sec. Returned 1 record(s). Limit: 20. (CHANGE IT)

View Options: Table (selected), Raw.

Page Footer: javascript:void(0) Copyright © 2017 WiseClouds LLC. All rights reserved. Page 162

Introducing OrientDB

Schema Manager

[Return to directory](#)

Screenshot of the OrientDB Schema Manager interface:

The URL in the browser is <http://localhost:2480/studio/>

The top navigation bar includes links for BROWSE, SCHEMA (which is selected), SECURITY, GRAPH, FUNCTIONS, DB, and a user dropdown for 'demodb (root)'.

The main title is "Schema Manager".

Below the title are buttons for "SAVE COLORS CONFIG", "ALL INDEXES", and "REBUILD ALL INDEXES".

A search bar contains the placeholder "Search classes" and a magnifying glass icon.

Two tabs are visible: "User Classes" (selected) and "System Classes".

Vertex Classes section:

Name	Color	SuperClasses	Alias	Abstract	Clusters	Default Cluster	Cluster Selection	Records	Actions
A	Orange	V		<input type="checkbox"/>	[337,338,339,340,341,342,343,344]	337	round-robin	1	RENAME QUERY ALL + NEW RECORD DROP
ArchaeologicalSites	Orange	V, Attractions		<input type="checkbox"/>	[113,114,115,116,117,118,119,120]	113	round-robin	55	RENAME QUERY ALL + NEW RECORD DROP
Attractions	Green	V, Locations		<input type="checkbox"/>	[81,82,83,84,85,86,87,88]	81	round-robin	436	RENAME QUERY ALL + NEW RECORD DROP
B	Green	V		<input type="checkbox"/>	[345,346,347,348,349,350,351,352]	345	round-robin	1	RENAME QUERY ALL + NEW RECORD DROP
Castles	Red	V, Attractions		<input type="checkbox"/>	[97,98,99,100,101,102,103,104]	97	round-robin	127	RENAME QUERY ALL + NEW RECORD DROP

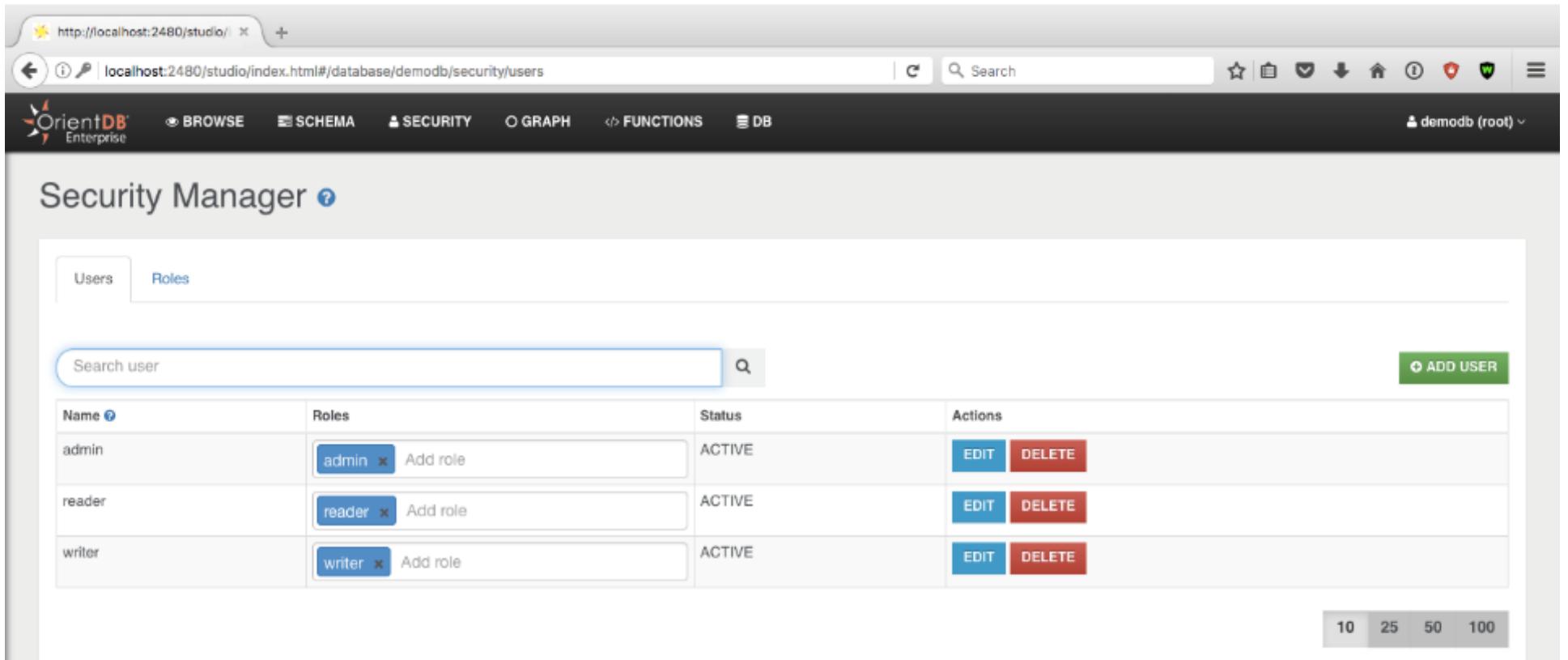
Pagination controls: 1 2 3 4 5

Edge Classes section:

Name	Color	SuperClasses	Alias	Abstract	Clusters	Default Cluster	Cluster Selection	Records	Actions
-	Grey								

Copyright © 2017 WiseClouds LLC. All rights reserved.

Security Manager

[Return to directory](#)

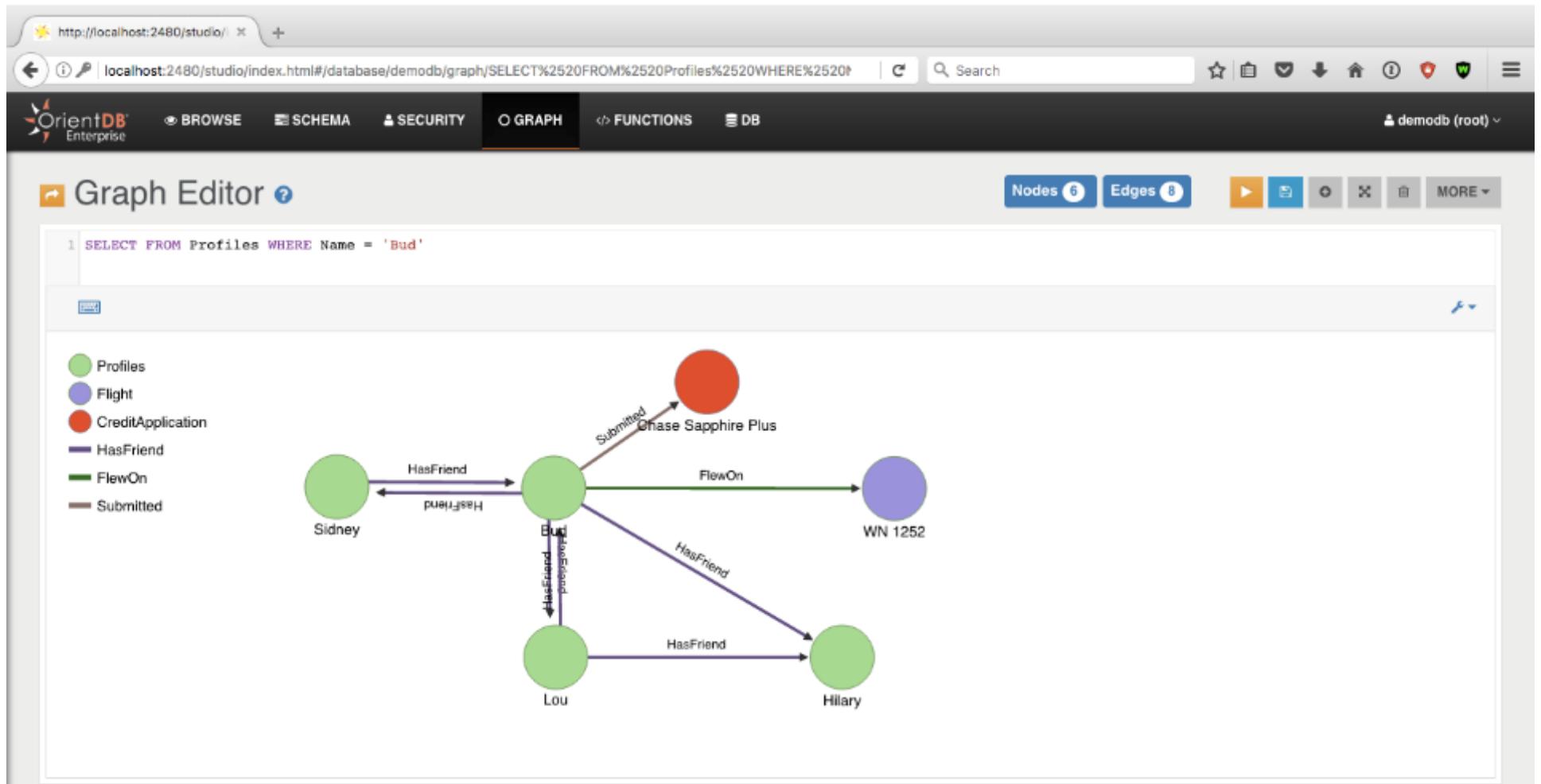
The screenshot shows the OrientDB Studio interface for managing security users. The top navigation bar includes links for BROWSE, SCHEMA, SECURITY, GRAPH, FUNCTIONS, DB, and a dropdown for the current database (demodb) and user (root). The main title is "Security Manager". Below it, there are tabs for "Users" (selected) and "Roles". A search bar labeled "Search user" is present. A green button labeled "ADD USER" is located on the right. The main content area displays a table of users:

Name	Roles	Status	Actions
admin	admin x Add role	ACTIVE	EDIT DELETE
reader	reader x Add role	ACTIVE	EDIT DELETE
writer	writer x Add role	ACTIVE	EDIT DELETE

Pagination controls at the bottom right allow for page sizes of 10, 25, 50, or 100.

Introducing OrientDB

Graph Editor

[Return to directory](#)

Database Management

[Return to directory](#)

Screenshot of the OrientDB Studio interface showing the Database Management screen.

The URL in the browser is <http://localhost:2480/studio/>.

The top navigation bar includes links for BROWSE, SCHEMA, SECURITY, GRAPH, FUNCTIONS, DB, and a user dropdown for demodb (root).

Database Management

Clusters are sets of records grouped by a mean. Any mean you want to assign. You can use a Cluster like a Table of the Relational DBMS world, namely to group records of the same type. Or you could want to group records by different logics.

There are two kinds of clusters: **Physical** and **In-memory**. The first one is persistent, the second one is volatile.

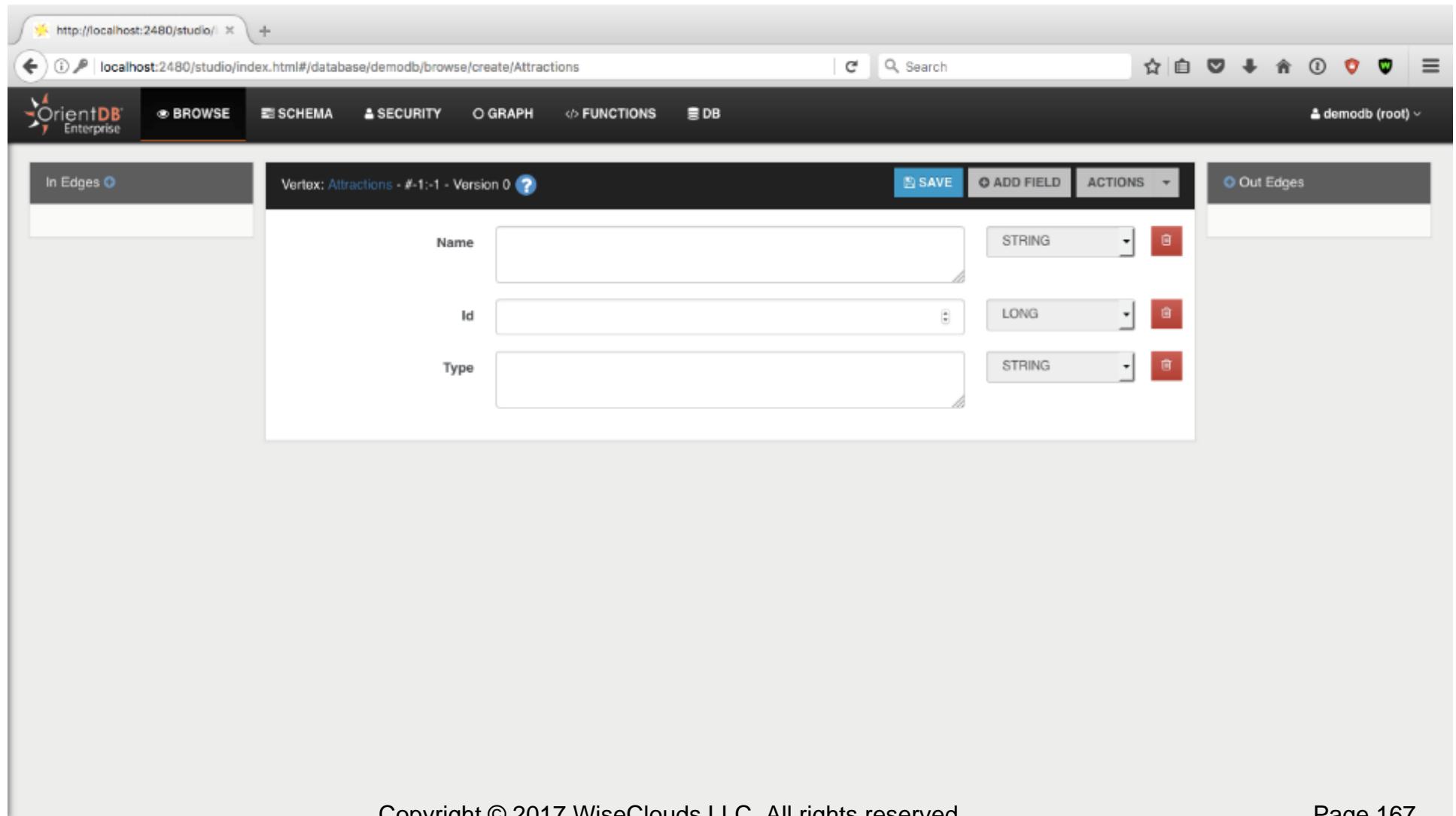
ID	Name	Records	Conflict Strategy
225	_studio	9	version
226	_studio_1	5	content
227	_studio_2	5	automerge
228	_studio_3	5	
229	_studio_4	5	
230	_studio_5	5	
231	_studio_6	5	
232	_studio_7	5	
337	a	1	
361	a1	1	
362	a1_1	1	
363	a1_2	1	
364	a1_3	1	

Copyright © 2017 WiseClouds LLC. All rights reserved.

Page 166

Introducing OrientDB

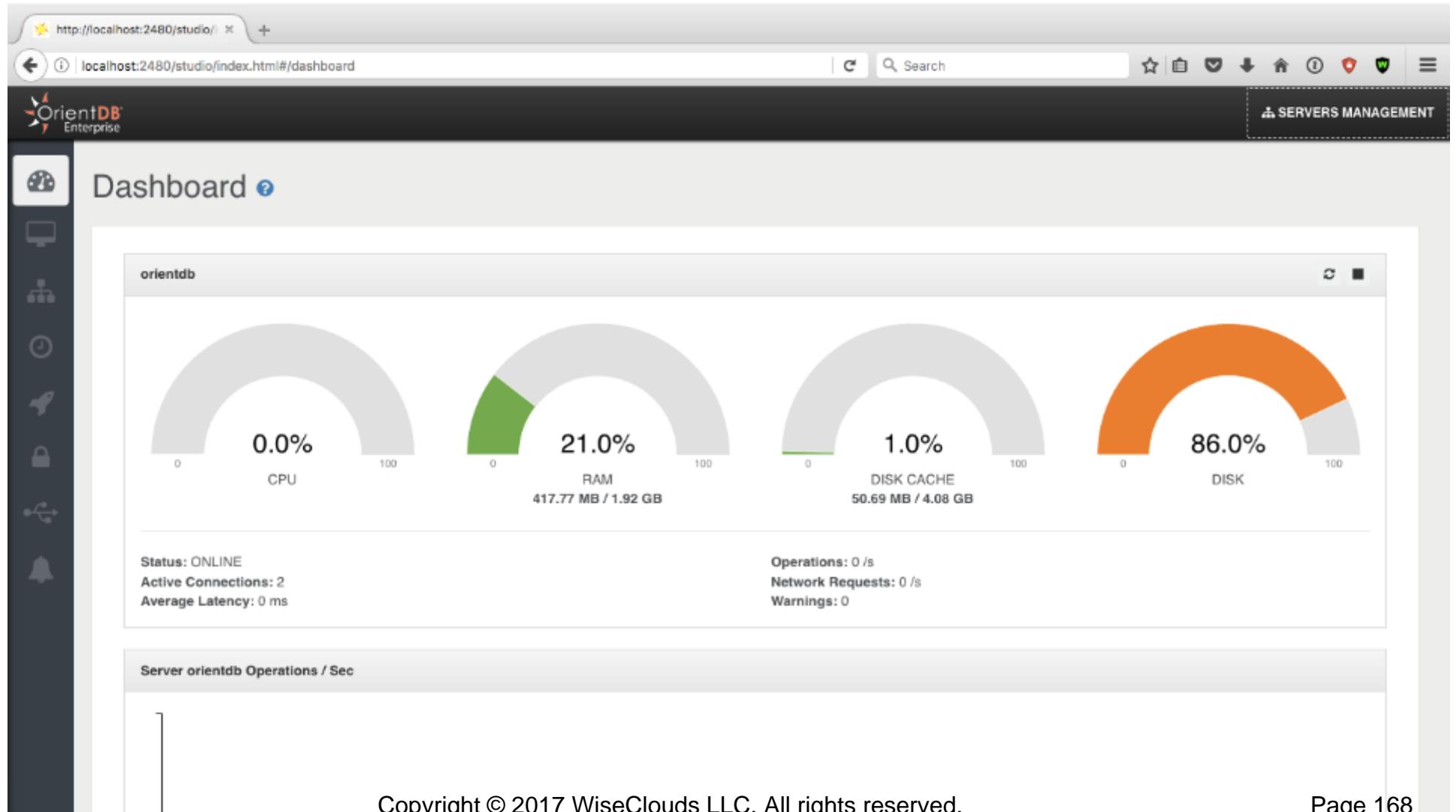
Create Record

[Return to directory](#)

The screenshot shows the OrientDB Studio interface at <http://localhost:2480/studio/>. The top navigation bar includes links for BROWSE, SCHEMA, SECURITY, GRAPH, FUNCTIONS, DB, and a user dropdown for 'demodb (root)'. The main area is titled 'Vertex: Attractions - #1-1 - Version 0'. It features three input fields: 'Name' (String type), 'Id' (Long type), and 'Type' (String type). Each field has a red delete icon to its right. On the left, there's a 'In Edges' panel and on the right, an 'Out Edges' panel. A 'SAVE' button is located at the top right of the form.

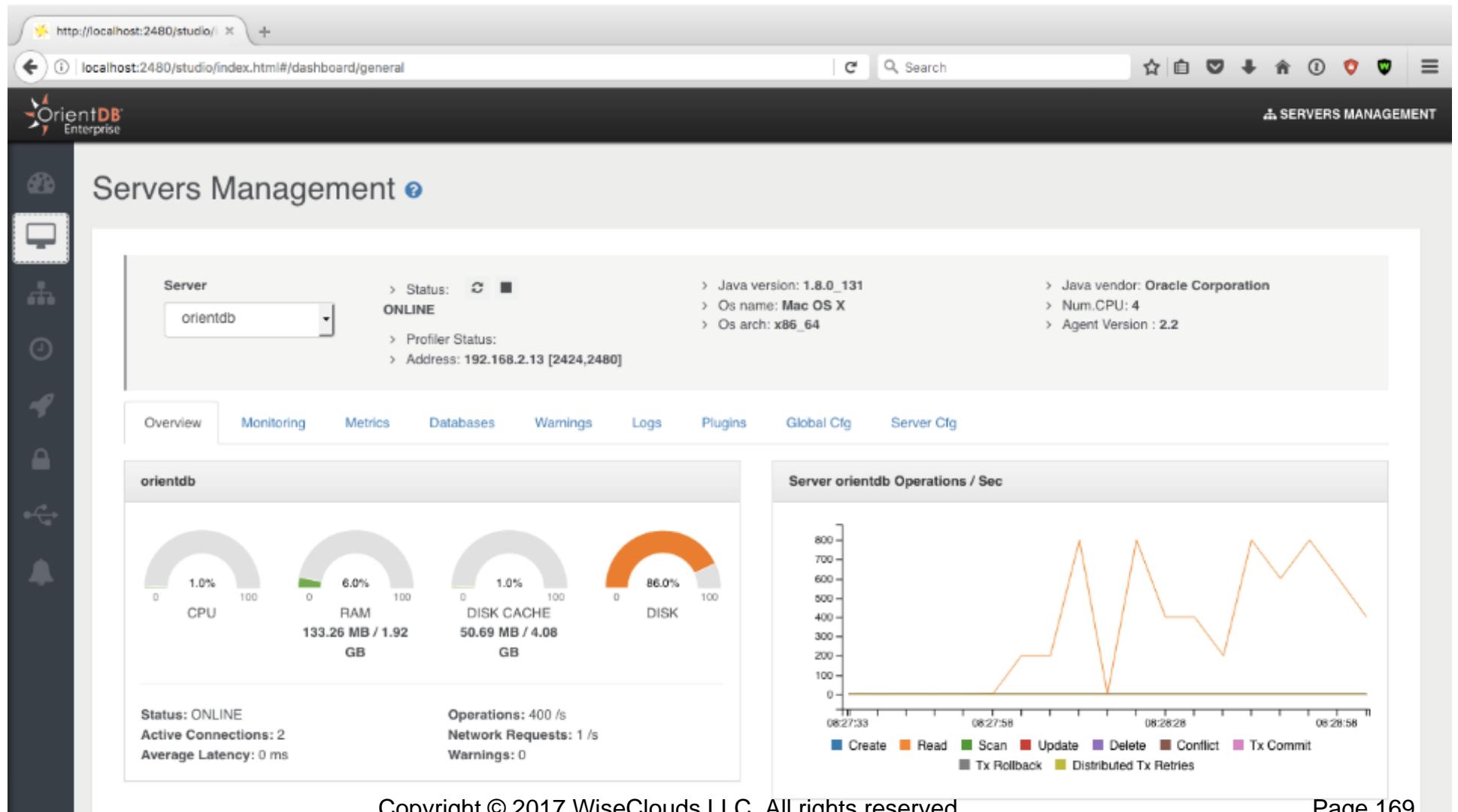
Introducing OrientDB

Dashboard

[Return to directory](#)

Introducing OrientDB

Servers Management

[Return to directory](#)


Servers - Monitoring

[Return to directory](#)

http://localhost:2480/studio/

localhost:2480/studio/index.html#/dashboard/general

Search

OrientDB Enterprise SERVERS MANAGEMENT

Servers Management

Server: orientdb Status: ONLINE

- > Profiler Status:
- > Address: 192.168.2.13 [2424,2480]
- > Java version: 1.8.0_131
- > Os name: Mac OS X
- > Os arch: x86_64
- > Java vendor: Oracle Corporation
- > Num.CPU: 4
- > Agent Version : 2.2

Overview Monitoring Metrics Warnings Logs Plugins Global Cfg Server Cfg

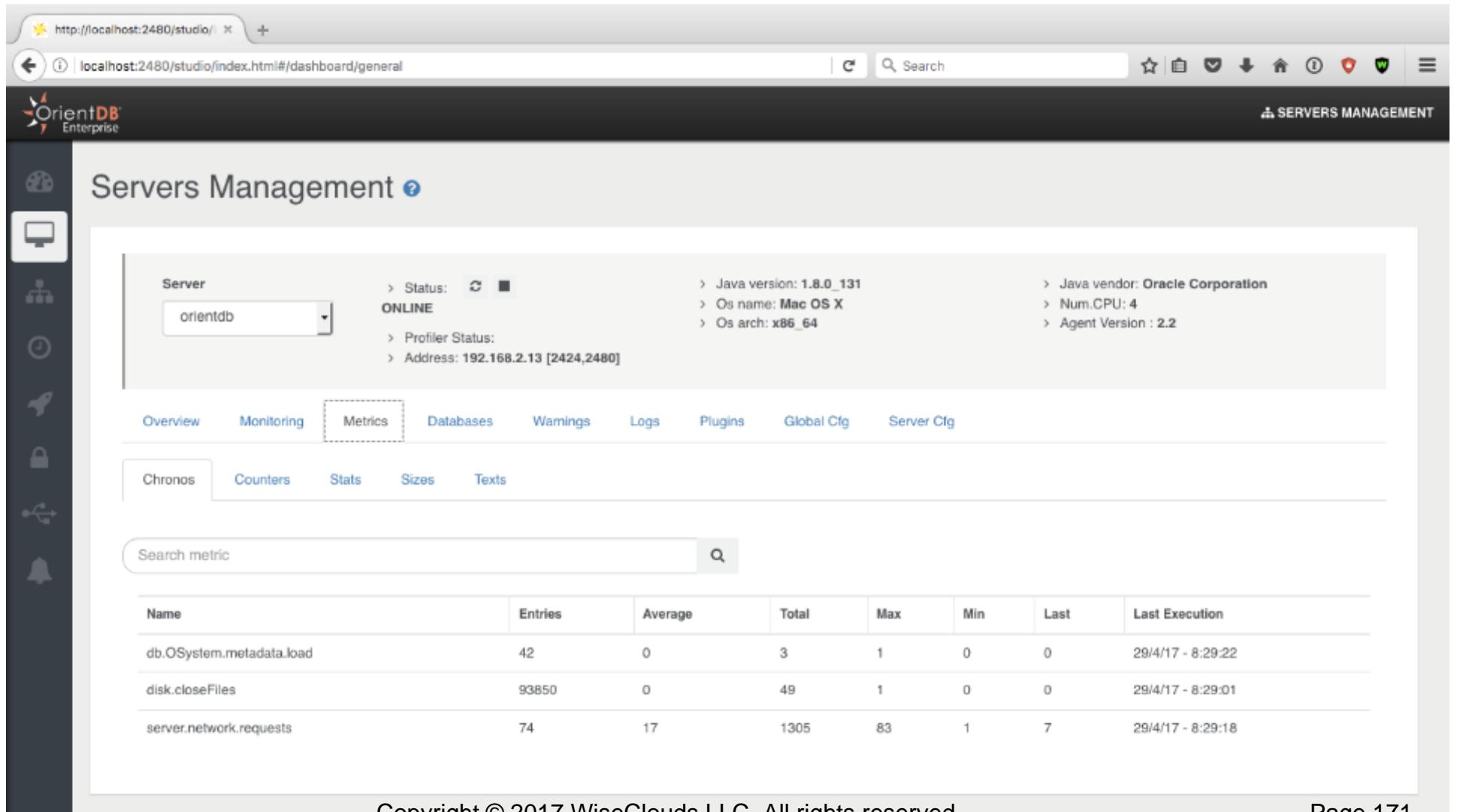
Connections Threads

Below all the active connections. Keep in mind that HTTP connections are stateless, so no database is retained and usually after a short timeout (seconds) they are killed. Binary connections, instead, remain in life until the connection client closes it. Binary connections are used by the [Orient Console](#) tool and by any Orient Java applications that uses the native [Java API](#)

Search connections

Session ID	Client	Address	Database	User	Total Requests	Command Info	Command Detail	Last Command On	Last Command Info	Last Command Detail	Last Execution Time	Total Working Time	Connected Since	Protocol	Client ID	Driver	Command
-2	23	/127.0.0.1:55193	-	-	34	Listening	-	2017-04-29 08:29:00	Execute remote command	select FROM Profiles LIMIT 1000	25	1028	2017-04-29 08:19:05	binary	-	OrientDB Java v2.2.18 Protocol v36	Interrupt Kill

Servers - Metrics

[Return to directory](#)

The screenshot shows the OrientDB Studio interface for 'Servers Management'. The top navigation bar includes links for Overview, Monitoring, Metrics (which is selected), Databases, Warnings, Logs, Plugins, Global Cfg, and Server Cfg. Below this, there are tabs for Chronos, Counters, Stats, Sizes, and Texts. A search bar at the bottom left allows searching for metrics.

Server Information:

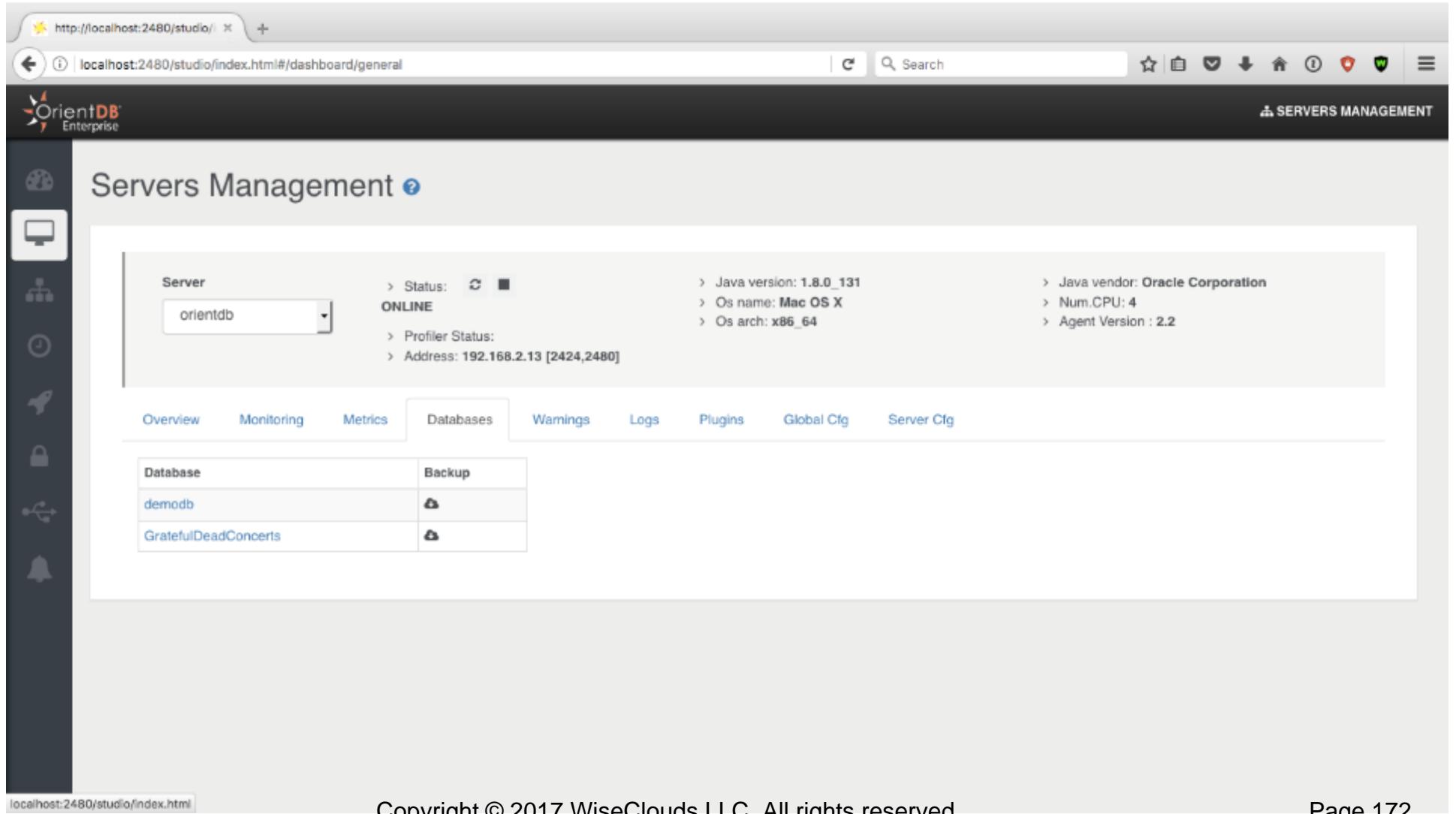
- Status: ONLINE
- Profiler Status: (green)
- Address: 192.168.2.13 [2424,2480]
- Java version: 1.8.0_131
- Os name: Mac OS X
- Os arch: x86_64
- Java vendor: Oracle Corporation
- Num.CPU: 4
- Agent Version : 2.2

Metric Data:

Name	Entries	Average	Total	Max	Min	Last	Last Execution
db.OSystem.metadata.load	42	0	3	1	0	0	29/4/17 - 8:29:22
disk.closeFiles	93850	0	49	1	0	0	29/4/17 - 8:29:01
server.network.requests	74	17	1305	83	1	7	29/4/17 - 8:29:18

Introducing OrientDB

Database List

[Return to directory](#)

The screenshot shows the OrientDB Studio interface at the URL <http://localhost:2480/studio/>. The main title is "Servers Management". On the left, there is a vertical sidebar with icons for Overview, Monitoring, Metrics, Databases, Warnings, Logs, Plugins, Global Cfg, and Server Cfg. The "Databases" icon is highlighted. The main content area displays the "Server" configuration for "orientdb", which is listed as "ONLINE". It shows system information: Java version 1.8.0_131, Os name Mac OS X, Os arch x86_64, Java vendor Oracle Corporation, Num.CPU 4, and Agent Version 2.2. Below this, the "Address" is given as 192.168.2.13 [2424,2480]. A "Database" table lists two databases: "demodb" and "GratefulDeadConcerts", each with a "Backup" button. The "Databases" tab is currently selected.

Introducing OrientDB

Logs

[Return to directory](#)

Screenshot of the OrientDB Studio interface showing the Servers Management screen for the 'orientdb' server.

Server Overview:

- Status: ONLINE
- Profiler Status:
- Address: 192.168.2.13 [2424,2480]
- Java version: 1.8.0_131
- Os name: Mac OS X
- Os arch: x86_64
- Java vendor: Oracle Corporation
- Num.CPU: 4
- Agent Version : 2.2

Logs Tab:

Search in: LAST Type: Search in Logs SEARCH

From: To:

Day	Hour	Type	File	Info
2017-04-28	10:43:51:564	INFO	orient-server.log.0	Loading configuration from: /Users/robertdschneider/Desktop/orientdb-enterprise-2.2.18/config/orientdb-server-config.xml... [OServerConfigurationLoaderXml]
2017-04-28	10:43:52:044	INFO	orient-server.log.0	OrientDB Server v2.2.18 (build 3e8d46e73aa087fce245fa1125ab7d984a247f6e) is starting up... [OServer]
2017-04-28	10:43:52:058	INFO	orient-server.log.0	Databases directory: /Users/robertdschneider/Desktop/orientdb-enterprise-2.2.18/databases [OServer]

Global Configuration

[Return to directory](#)

Screenshot of the OrientDB Studio interface showing the Global Configuration page.

The URL in the browser is <http://localhost:2480/studio/>.

The page title is [localhost:2480/studio/index.html#/dashboard/general](#).

The navigation bar includes links for Overview, Monitoring, Metrics, Databases, Warnings, Logs, Plugins, Global Cfg (selected), and Server Cfg.

The left sidebar contains icons for Overview, Monitoring, Metrics, Databases, Warnings, Logs, Plugins, Global Configuration, and Server Configuration.

Properties

Name	Value
db.pool.min	1
db.pool.max	50
profiler.enabled	false

Global Configuration

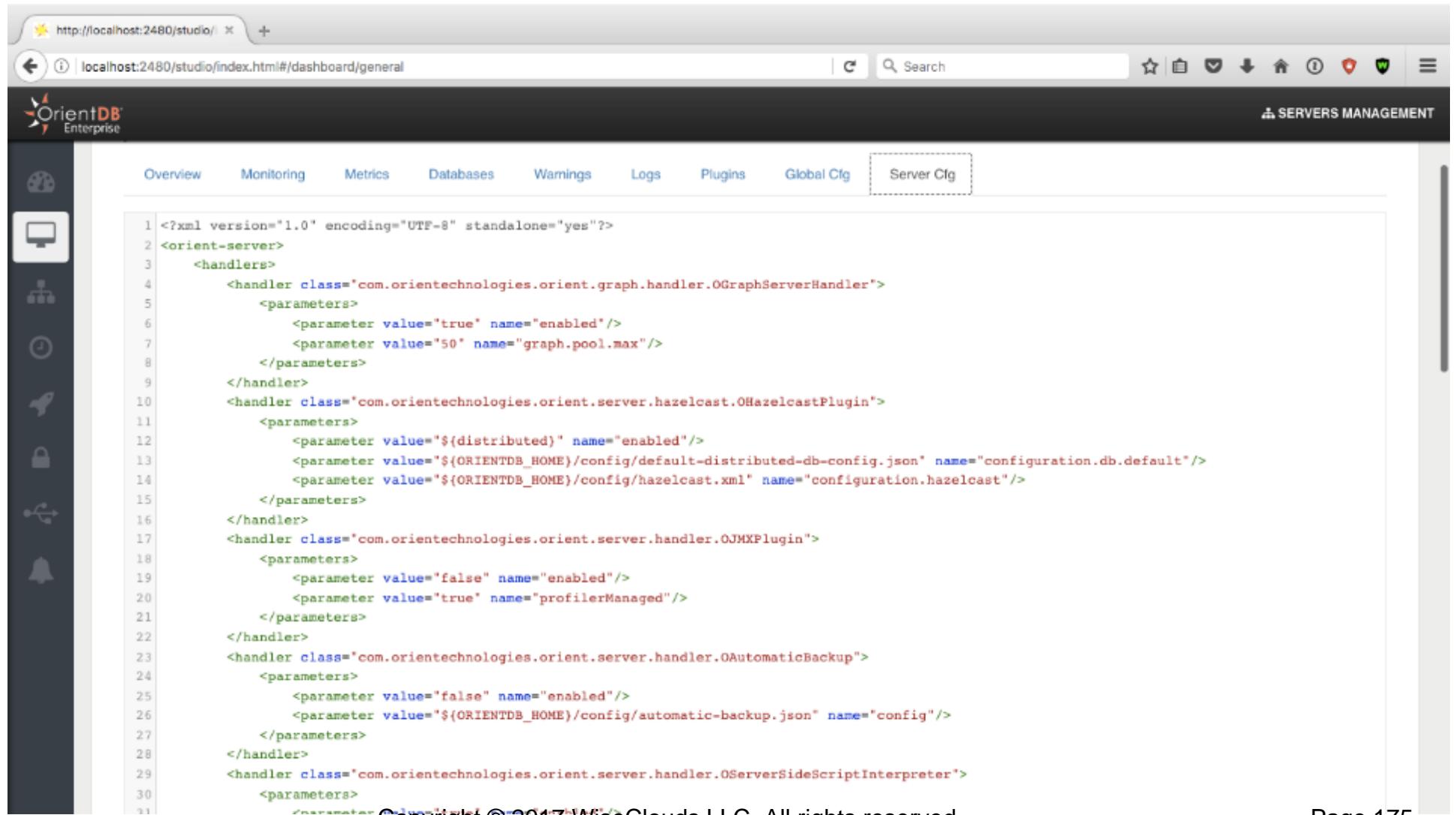
SAVE

Name	Description	Default Value	Value
environment.allowJVMShutdown	Allows the shutdown of the JVM, if needed/requested	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
environment.concurrent	Specifies if running in multi-thread environment. Setting this to false turns off the internal lock management	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
environment.dumpCfgAtStartup	Dumps the configuration during application startup	<input type="checkbox"/>	<input type="checkbox"/>
environment.lockManager.concurrency.level	Concurrency level of lock manager	32	32
memory.chunk.size	Size of single memory chunk (in bytes) which will be preallocated by OrientDB	2147483647	2147483647
memory.directMemory.onlyAlignedMemoryAccess	Some architectures do not allow unaligned memory access or may suffer from speed degradation. For such platforms, this flag should be set to true	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
memory.directMemory.safeMode	Indicates whether to perform a range check before each direct memory update. It is true by default, but usually it can be safely set to false. It should only be to true after dramatic changes have been made in the storage structures	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Search config

Copyright © 2017 WiseClouds LLC. All rights reserved.

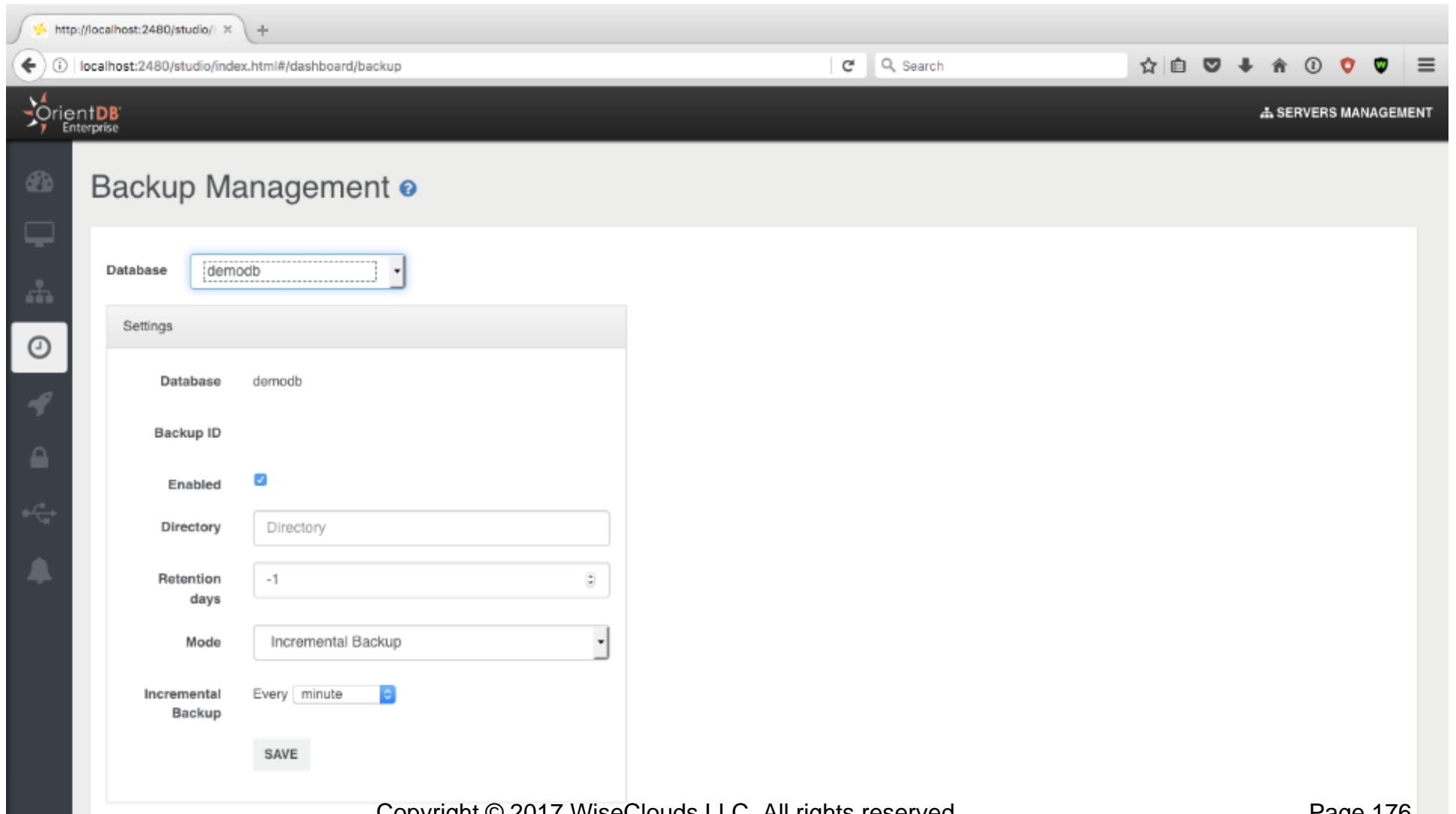
Server Configuration

[Return to directory](#)

The screenshot shows the OrientDB Studio interface with the "Server Cfg" tab selected. The main content area displays the following XML configuration code:

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <orient-server>
3   <handlers>
4     <handler class="com.orientechnologies.orient.graph.handler.OGraphServerHandler">
5       <parameters>
6         <parameter value="true" name="enabled"/>
7         <parameter value="50" name="graph.pool.max"/>
8       </parameters>
9     </handler>
10    <handler class="com.orientechnologies.orient.server.hazelcast.OHazelcastPlugin">
11      <parameters>
12        <parameter value="${distributed}" name="enabled"/>
13        <parameter value="${ORIENTDB_HOME}/config/default-distributed-db-config.json" name="configuration.db.default"/>
14        <parameter value="${ORIENTDB_HOME}/config/hazelcast.xml" name="configuration.hazelcast"/>
15      </parameters>
16    </handler>
17    <handler class="com.orientechnologies.orient.server.handler.OJMXPlugin">
18      <parameters>
19        <parameter value="false" name="enabled"/>
20        <parameter value="true" name="profilerManaged"/>
21      </parameters>
22    </handler>
23    <handler class="com.orientechnologies.orient.server.handler.OAutomaticBackup">
24      <parameters>
25        <parameter value="false" name="enabled"/>
26        <parameter value="${ORIENTDB_HOME}/config/automatic-backup.json" name="config"/>
27      </parameters>
28    </handler>
29    <handler class="com.orientechnologies.orient.server.handler.OServerSideScriptInterpreter">
30      <parameters>
31        <parameter value="true" name="enabled"/>
32      </parameters>
```

Backup Management

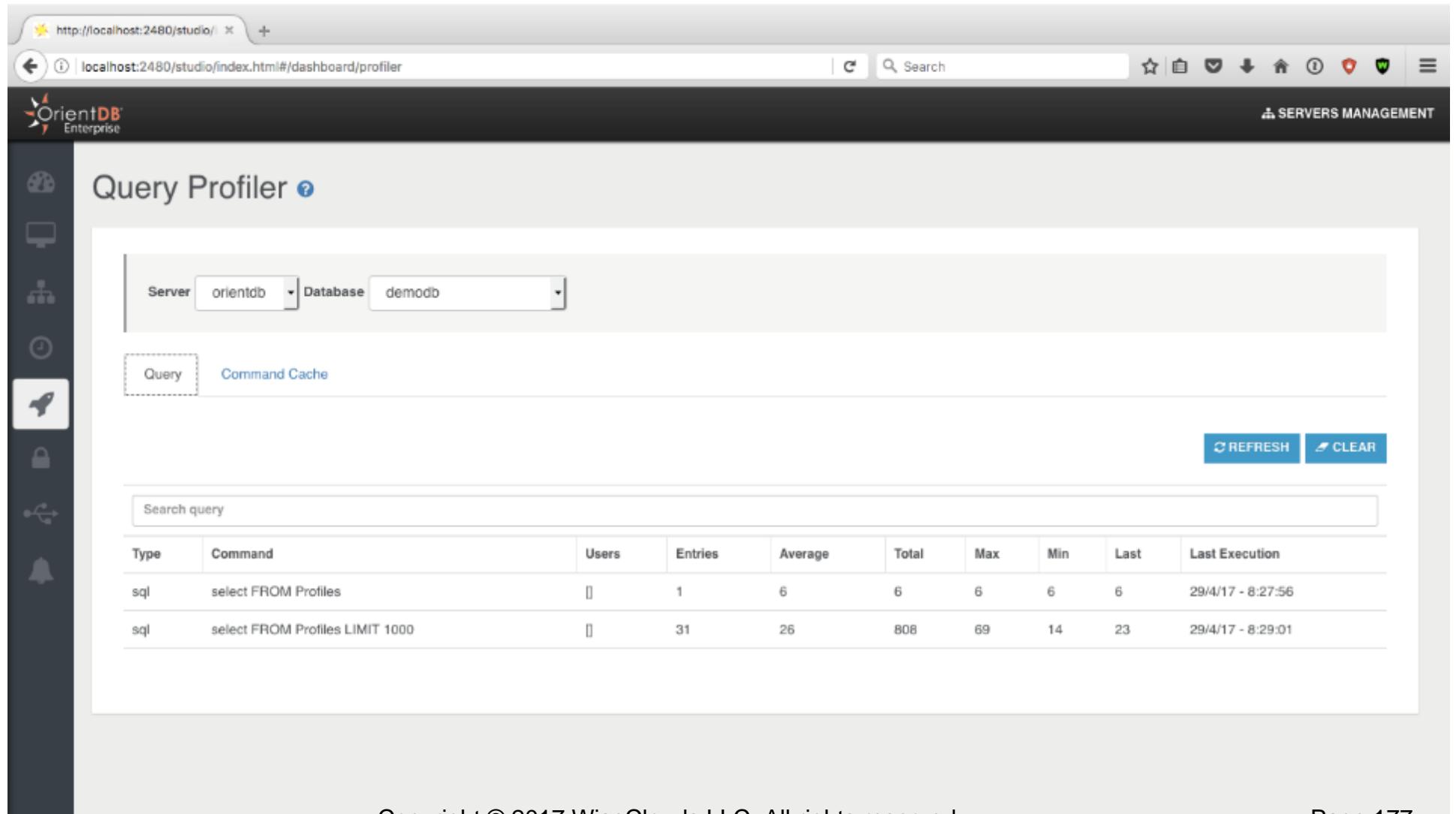
[Return to directory](#)

The screenshot shows the OrientDB Studio interface for 'Backup Management'. The left sidebar has icons for Home, Databases, Servers, Configuration, Scripts, Security, and Notifications. The main area title is 'Backup Management'. A dropdown menu shows 'demodb' selected. The 'Settings' section contains the following configuration:

Setting	Value
Database	demodb
Backup ID	(empty)
Enabled	<input checked="" type="checkbox"/>
Directory	Directory
Retention days	-1
Mode	Incremental Backup
Incremental Backup	Every minute

At the bottom is a 'SAVE' button.

Query Profiler

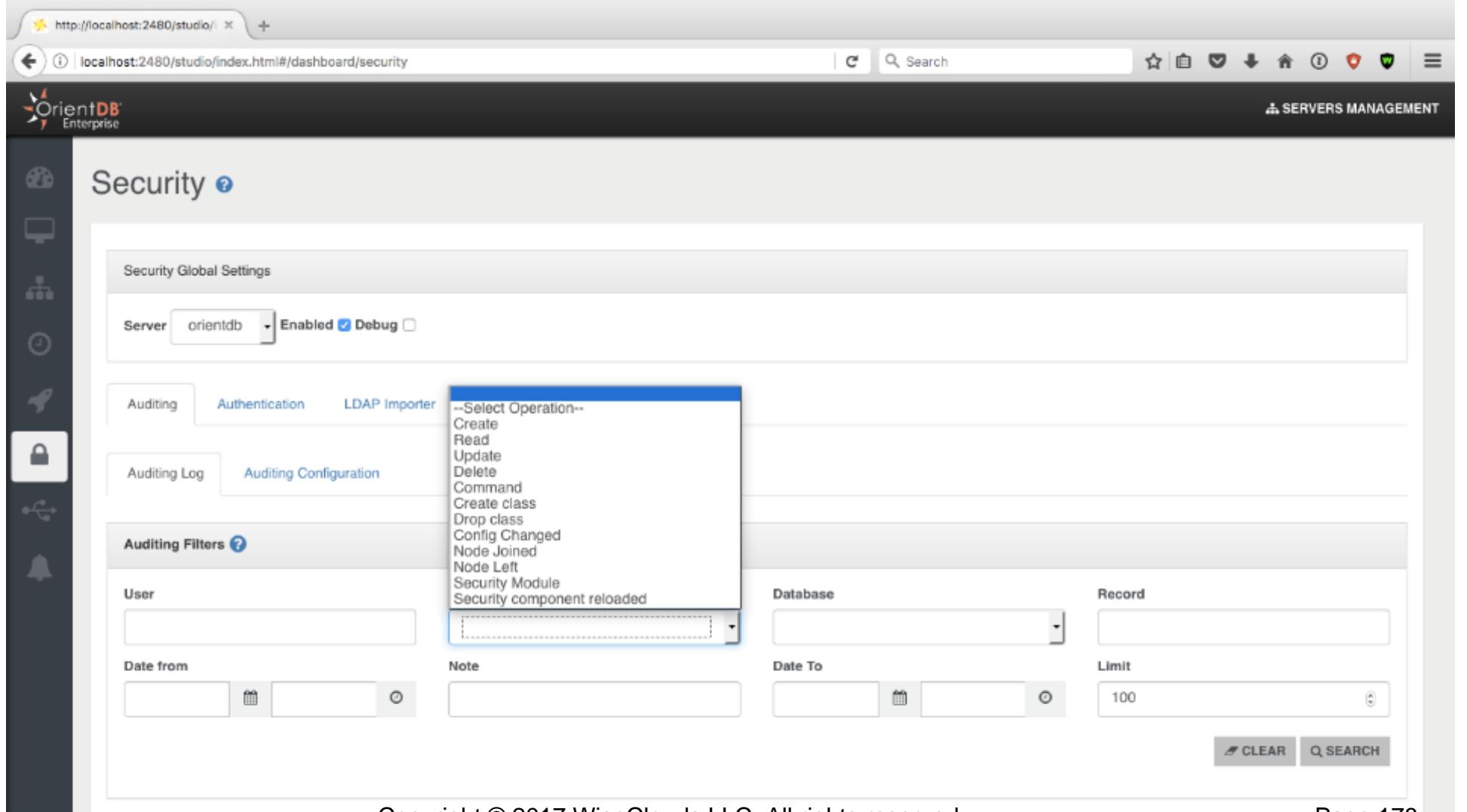
[Return to directory](#)

The screenshot shows the OrientDB Query Profiler interface. At the top, there are dropdown menus for 'Server' (set to 'orientdb') and 'Database' (set to 'demodb'). Below these are tabs for 'Query' (selected) and 'Command Cache'. On the right, there are 'REFRESH' and 'CLEAR' buttons. A search bar labeled 'Search query' is present. The main area displays a table of query execution statistics:

Type	Command	Users	Entries	Average	Total	Max	Min	Last	Last Execution
sql	select FROM Profiles	1	1	6	6	6	6	6	29/4/17 - 8:27:56
sql	select FROM Profiles LIMIT 1000	31	26	808	69	14	23	23	29/4/17 - 8:29:01

Introducing OrientDB

Security

[Return to directory](#)

The screenshot shows the OrientDB Studio interface for managing security settings. On the left, there's a vertical sidebar with icons for Home, Servers, Databases, Schemas, Transactions, Lock, and Notifications.

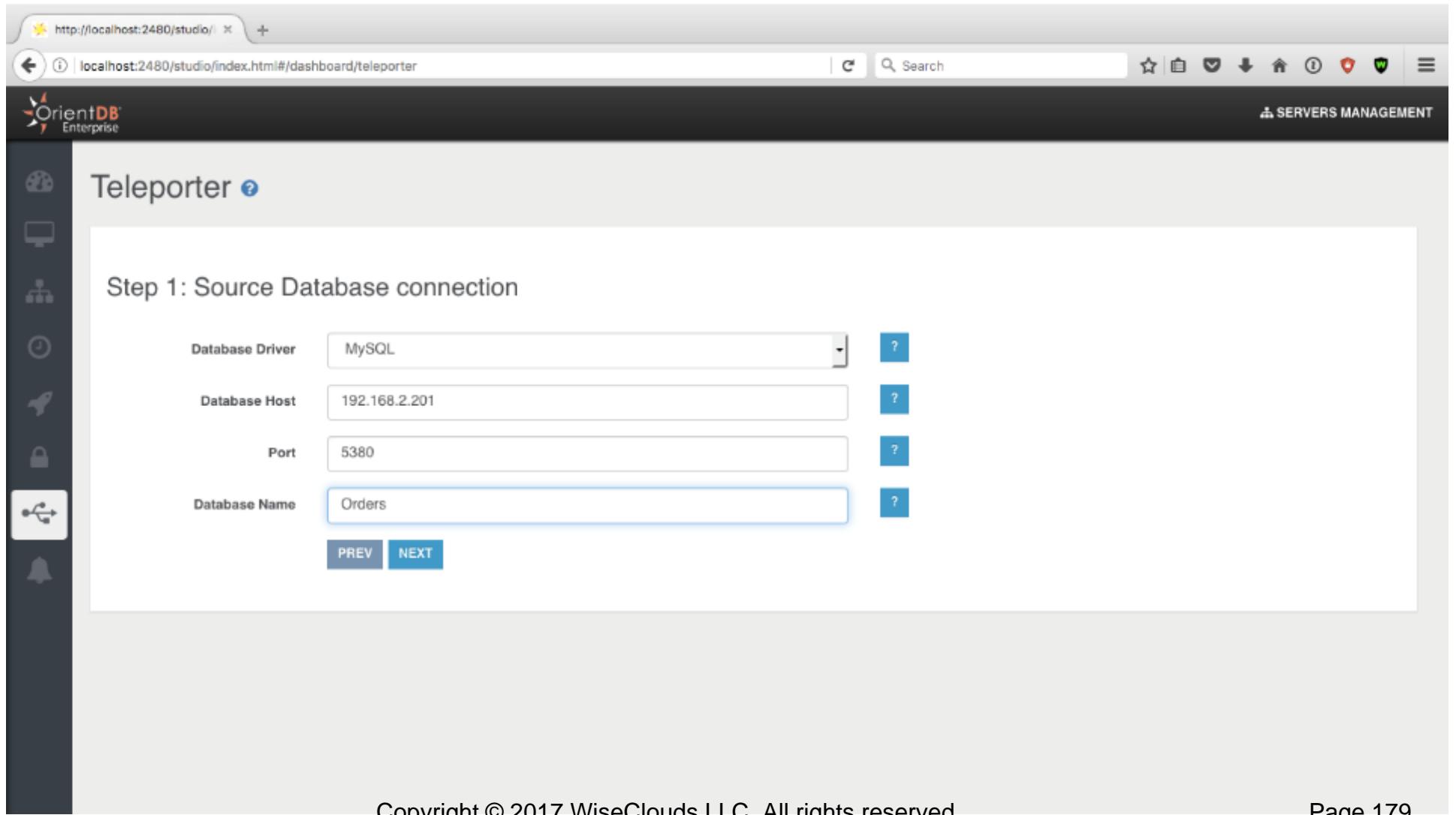
The main area is titled "Security". It includes a "Security Global Settings" section with a dropdown for "Server" set to "orientdb", and checkboxes for "Enabled" (checked) and "Debug" (unchecked). Below this are tabs for "Auditing", "Authentication", and "LDAP Importer", with "Auditing" currently selected. Under "Auditing", there are two sub-tabs: "Auditing Log" (selected) and "Auditing Configuration".

A modal dialog box is open over the "Auditing Log" tab, titled "--Select Operation--". It lists several audit operations:

- Create
- Read
- Update
- Delete
- Command
- Create class
- Drop class
- Config Changed
- Node Joined
- Node Left
- Security Module
- Security component reloaded

Below the modal, there are sections for "Auditing Filters" (User, Date from, Note), "Database" (dropdown), "Record" (text input), "Date To" (date picker), "Limit" (text input set to 100), and buttons for "CLEAR" and "SEARCH".

Teleporter

[Return to directory](#)

The screenshot shows the OrientDB Studio interface with the 'Teleporter' tab selected. On the left, there's a vertical sidebar with icons for Home, Servers Management, Databases, Schemas, Triggers, Indexes, Locks, and Notifications.

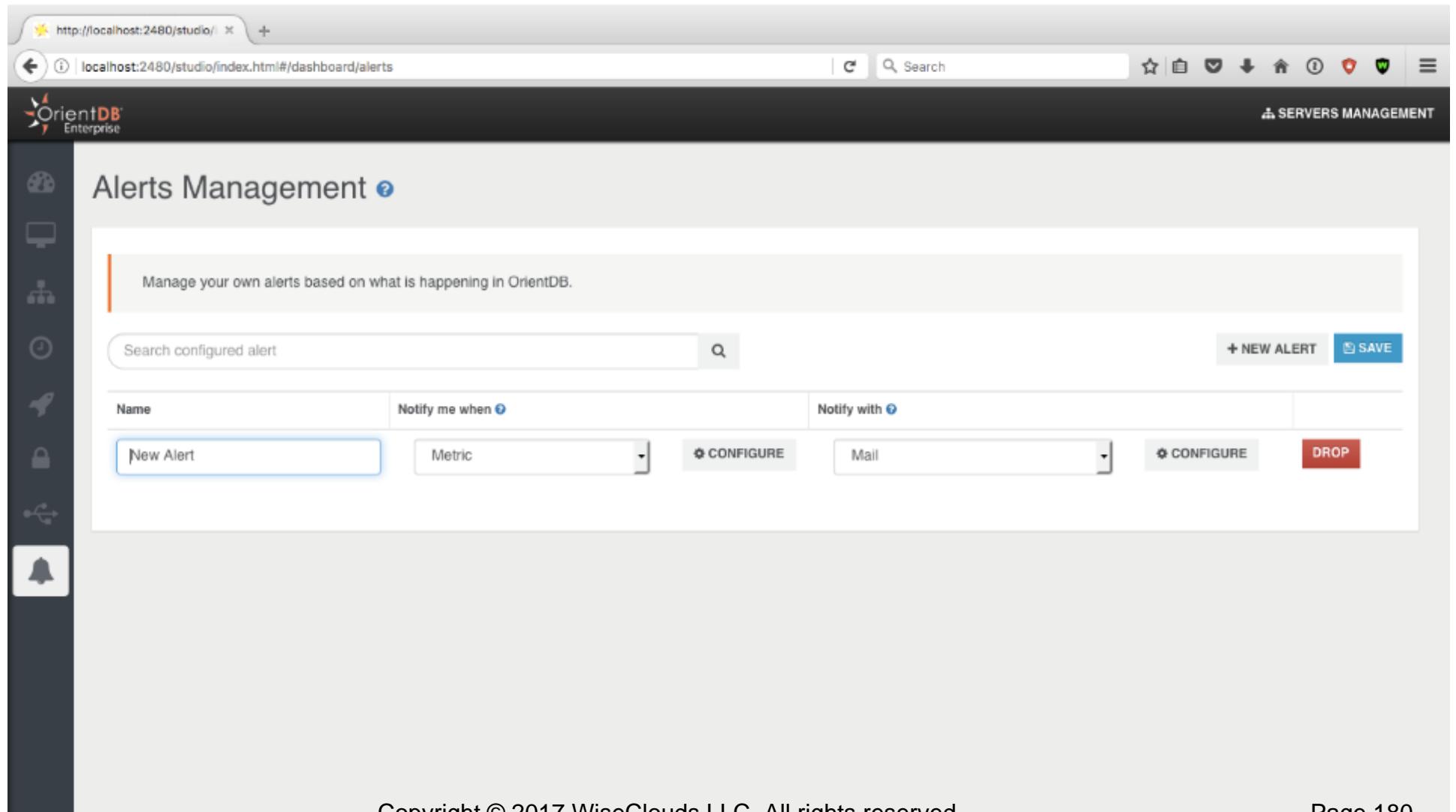
The main area displays the 'Teleporter' configuration screen. The title 'Teleporter' is at the top, followed by a subtitle 'Step 1: Source Database connection'. The configuration fields are as follows:

- Database Driver: MySQL
- Database Host: 192.168.2.201
- Port: 5380
- Database Name: Orders

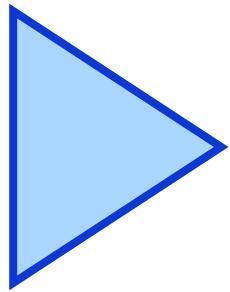
At the bottom of the configuration area are 'PREV' and 'NEXT' buttons.

At the very bottom of the page, the footer reads: Copyright © 2017 WiseClouds LLC. All rights reserved.

Alerts Management

[Return to directory](#)

The screenshot shows the 'Alerts Management' page of the OrientDB Studio. The URL in the browser is <http://localhost:2480/studio/>. The page title is 'Alerts Management'. On the left, there is a vertical sidebar with icons for Home, Servers, Databases, Metrics, Configuration, and Notifications. The main content area has a heading 'Manage your own alerts based on what is happening in OrientDB.' Below this is a search bar labeled 'Search configured alert' with a magnifying glass icon. To the right of the search bar are two buttons: '+ NEW ALERT' and 'SAVE'. A table below the search bar has columns for 'Name', 'Notify me when', and 'Notify with'. The first row in the table is for a 'New Alert' metric, which is currently selected. It includes dropdown menus for 'Metric' and 'Notify with', and buttons for 'CONFIGURE' and 'DROP'. The rest of the table is empty.



Modeling a Database

- Taking Advantage of Multi-Model Capabilities
- Classes
- Relationships
- Documents
- Graph
- A Workflow for Modeling
- Evolving the Model

Unit Goals

- Describe major OrientDB artifacts
- Explain the role that classes play
- Demonstrate the process of designing a multi model database using OrientDB

Unit Goals

- Show the evolution of a model
- Set the stage for interacting with data

NoSQL and Data Models

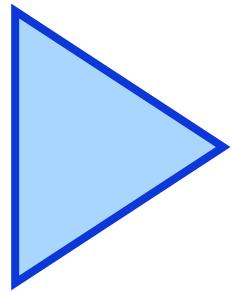
- Many NoSQL technologies impose a single data model on their users
 - Key/value
 - Document
 - Object
 - Graph
- These restrictions can make it necessary to deploy multiple NoSQL solutions to cover all information needs

Multi-Model Capabilities

- OrientDB's multi-model features mean that it can address numerous data storage requirements
- Database designers may select one of the following four record types
 - Vertex
 - Edge
 - Document
 - Binary Large Object (Blob)

Multi-Model Capabilities

- These four record types more closely model the real-world scenarios that database designers and application developers face
- Before moving on to discuss each storage model, it's important to understand the concept of a class in OrientDB



Classes

- Role of Classes in OrientDB
- Abstract Classes
- Inheritance
- Class Attributes & Properties
- Record IDs

Classes

- An essential OrientDB concept & artifact
- The baseline for document and graph capabilities
 - Also can be used to contain key/value information
- Relevant both interactively (via SQL, for example) and via all APIs

Classes

- One aspect of modern applications is that they require flexibility when defining their schema
- Unlike in RDBMS-backed systems, contents of individual records often vary
- OrientDB supports three ways to create a class
 - Schema-full
 - Schema-less
 - Schema-mixed (AKA schema-hybrid)

Schema-Full Classes

- Incorporates constraints on fields
- Strictly validates classes, makes all fields mandatory
- Useful in scenarios where record structure is well-understood
 - And unlikely to change in the short term

Schema-Less Classes

- The default option
- A “relaxed” model
 - Enables classes with no properties
 - Records can have arbitrary fields
- Makes it possible to collect heterogeneous documents together

Schema-Mixed Classes

- Schema has mandatory and optional fields
 - And validation rules
- Enables classes with some fields, but also lets records define custom fields
 - Including constraints where necessary
- A good compromise for situations where certain aspects of the data model are fixed, and others are variable

Abstract Classes

- Baseline for creating child classes
- Can have properties
- Can't have instances
- No cluster is set up when they're created
- Similar to object oriented programming

Creating a Class

- Two primary techniques
 - Interactively
 - Programmatically
- Interactively (SQL)
 - In OrientDB Studio
 - Via the console
- Programmatically (API)
 - Multiple languages supported

Class Attributes

Attribute	Purpose
NAME	Name of the class
SHORTNAME	Alias
SUPERCLASS	Parent class (if any)
OVERSIZE	Use more disk space during create
STRICTMODE	Can't have undefined properties
ADDCLUSTER	Add a cluster
REMOVECLUSTER	Remove a cluster
ABSTRACT	Is the class abstract? (default = false)
CUSTOM	Custom attribute (key/value)

Class Properties

Attribute	Purpose
NAME	Name of the property
TYPE	Data type (e.g. STRING, INTEGER...)
LINKEDTYPE	Linked type (e.g. STRING, INTEGER..)
LINKEDCLASS	Linked persistent class
MIN	Minimum value constraint
MAX	Maximum value constraint
MANDATORY	Is validation rule mandatory?
NOTNULL	Validation rule – can't be null
REGEXP	Regular expression validation

Class Properties (continued)

Attribute	Purpose
CUSTOM	Custom attribute (key/value)
READONLY	Validation rule can't be changed
COLLATE	Case sensitive or not

Dropping Class Properties

- When a property is dropped, it's removed from the schema
- However, existing records that contain the dropped property are unaltered
- This helps preserve data integrity

Inheritance

- Classes may inherit from other classes
 - Extends the parent class
 - Inherits all the attributes
- It's essential for database designers and developers to understand this concept

Inheritance

- Queries are polymorphic
 - Default is for OrientDB to treat all queries this way
- Subclasses may be part of result set
- To create a subclass, just use EXTENDS

```
CREATE CLASS Flight EXTENDS V;
```

```
CREATE CLASS Employee EXTENDS Person;
```

Setting Superclasses

- After creating a class, it's possible to set a parent class for it later
- This is accomplished by altering the class by using SUPERCLASS

```
CREATE CLASS Person;
```

```
CREATE CLASS Employee;
```

```
ALTER CLASS Employee SUPERCLASS PERSON;
```



Person

Name, DOB

Modeling a Database



Rating

DateWritten, Score



Person

Name, DOB

Modeling a Database



Rating

DateWritten, Score



Person

Name, DOB



Credit

Score, MaxAmount

Modeling a Database



Rating

DateWritten, Score



Person

Name, DOB



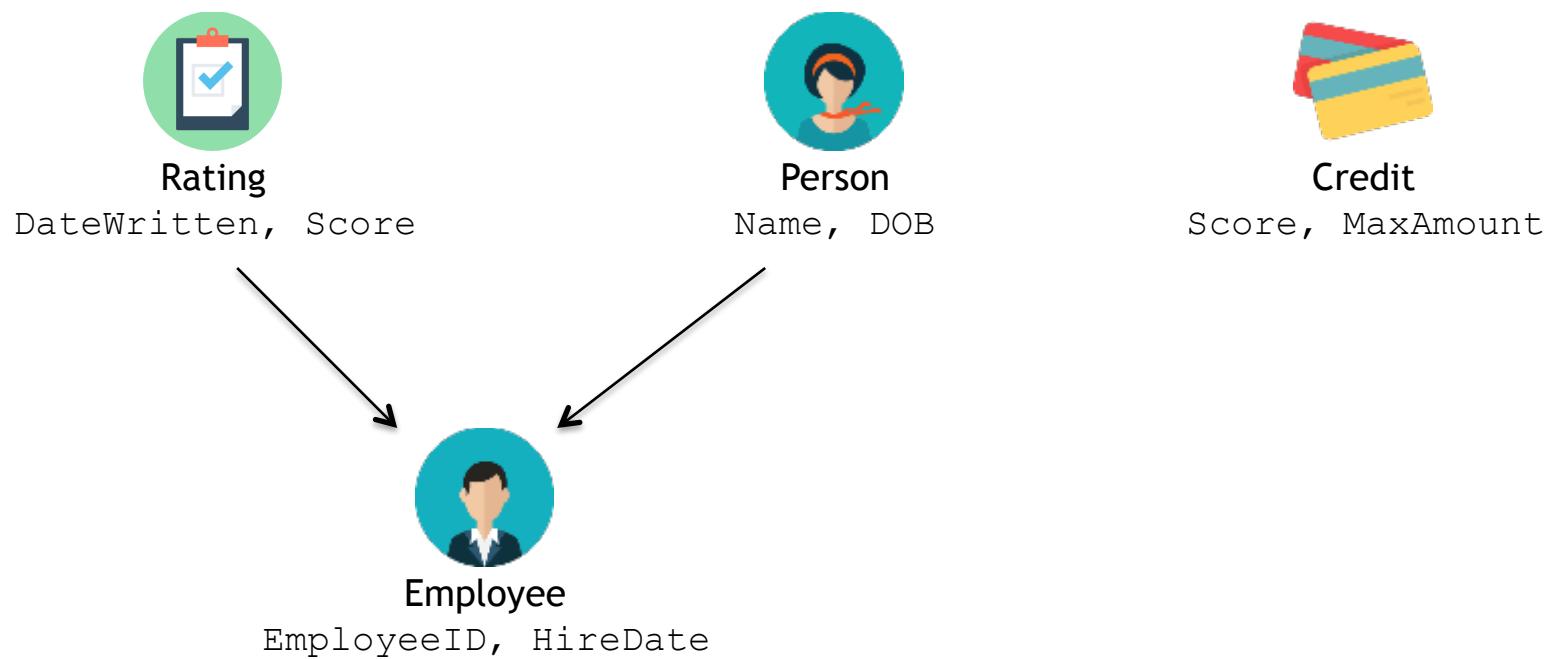
Credit

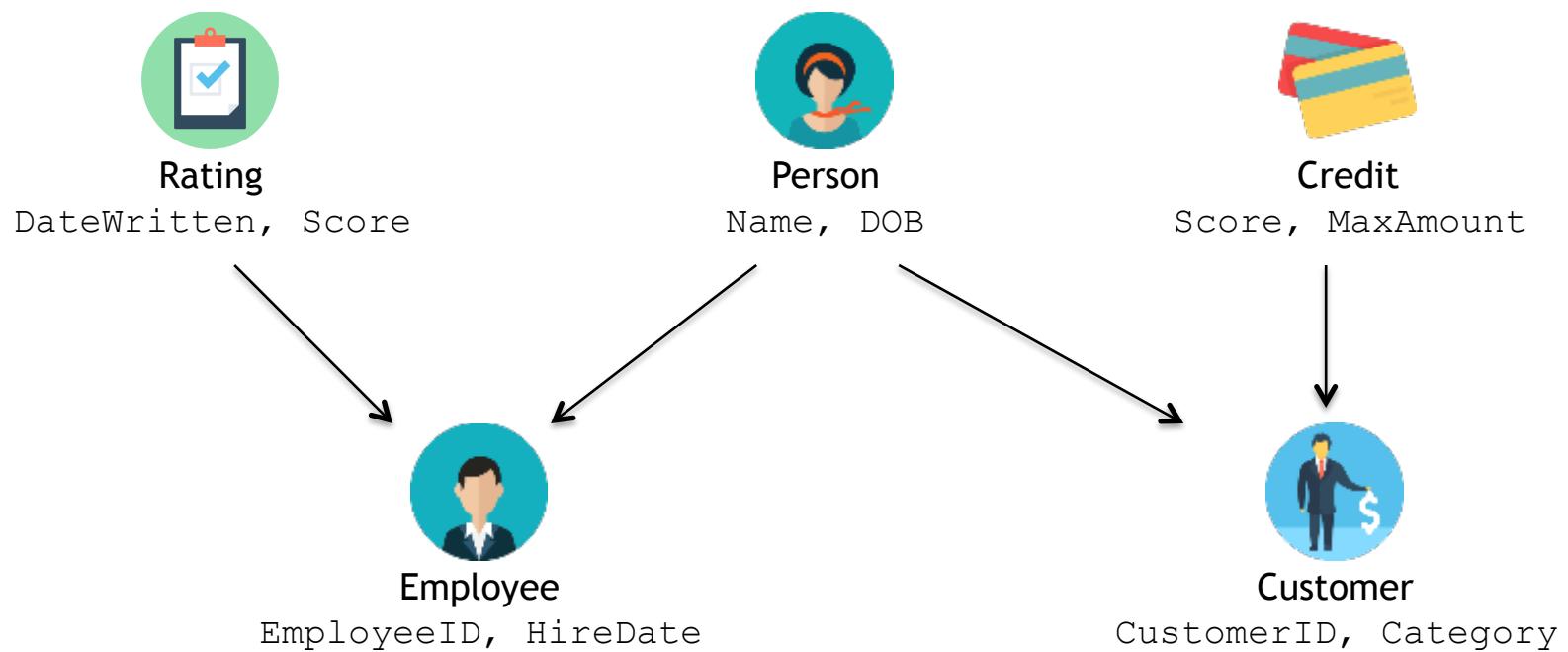
Score, MaxAmount

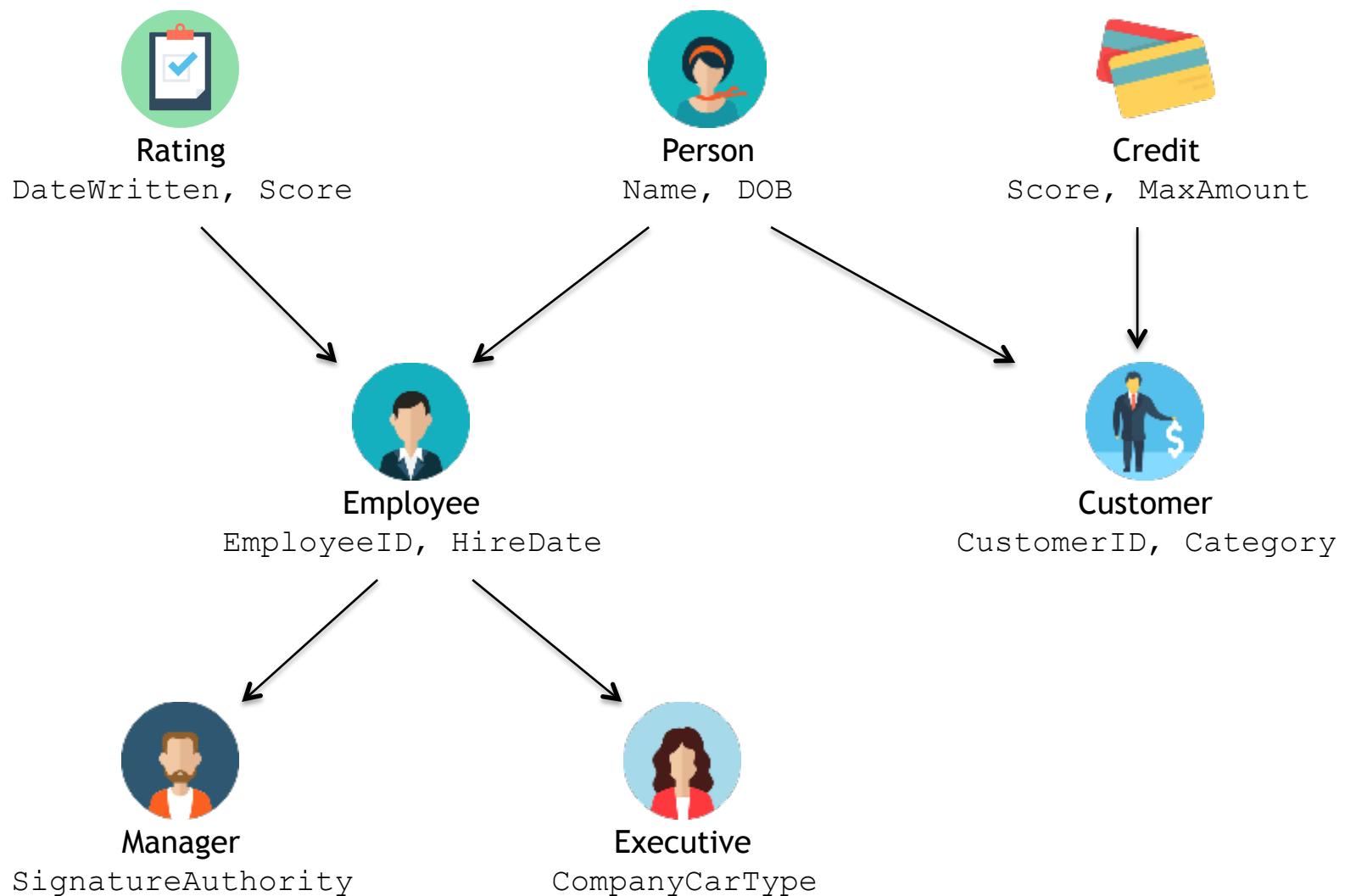


Employee

EmployeeID, HireDate



Modeling a Database

Modeling a Database

DESCRIBE Person

```
CLASS 'Person'

Records.....: 0
Default cluster....: null (id=-1)
Supported clusters...: null(-1)
Cluster selection....: round-robin
Oversize.....: 0.0
Subclasses.....: Employee, Customer
```

PROPERTIES

#	NAME	TYPE	LINKED-TYPE/CLASS	MANDATORY	READONLY	NOT-NULL	MIN	MAX	COLLATE	DEFAULT	
0	DOB	DATE		false	false	false			default		
1	Name	STRING		false	false	false			default		

DESCRIBE Employee

```
CLASS 'Employee'

Records.....: 2
Super classes....: [Rating, Person]
Default cluster....: employee (id=289)
Supported clusters...: employee(289), employee_1(290), employee_2(291), employee_3(292), employee_4(293), employee_5(294), employee_6(295), employee_7(296)
Cluster selection....: round-robin
Oversize.....: 0.0
Subclasses.....: Executive, Manager
```

PROPERTIES

#	NAME	TYPE	LINKED-TYPE/CLASS	MANDATORY	READONLY	NOT-NULL	MIN	MAX	COLLATE	DEFAULT
0	DOB	DATE		false	false	false			default	
1	Name	STRING		false	false	false			default	
2	Score	BYTE		false	false	false			default	
3	DateWritten	DATE		false	false	false			default	
4	EmployeeID	SHORT		false	false	false			default	
5	HireDate	DATE		false	false	false			default	

DESCRIBE Executive

```
CLASS 'Executive'

Records.....: 1
Super classes....: [Employee]
Default cluster...: executive (id=369)
Supported clusters...: executive(369), executive_1(370), executive_2(371), executive_3(372), executive_4(373), executive_5(374), executive_6(375), executive_7(376)
Cluster selection...: round-robin
Oversize.....: 0.0

PROPERTIES
+-----+-----+-----+-----+-----+-----+-----+-----+
| # | NAME      | TYPE    | LINKED-TYPE/CLASS | MANDATORY | READONLY | NOT-NULL | MIN | MAX | COLLATE | DEFAULT |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 0 | DOB        | DATE   |           | false     | false     | false    |    |    | default |
| 1 | Name       | STRING  |           | false     | false     | false    |    |    | default |
| 2 | Score      | BYTE   |           | false     | false     | false    |    |    | default |
| 3 | DateWritten | DATE   |           | false     | false     | false    |    |    | default |
| 4 | EmployeeID | SHORT  |           | false     | false     | false    |    |    | default |
| 5 | HireDate   | DATE   |           | false     | false     | false    |    |    | default |
| 6 | CompanyCarType | STRING |           | false     | false     | false    |    |    | default |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

Modeling a Database

SELECT FROM Person

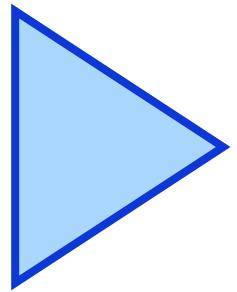
O ★ 1

METADATA			PROPERTIES						
@rid	@version	@class	HireDate	DOB	Name	SignatureAuthority	CompanyCarType	Score	MaxAmount
#290:0	1	Employee	2010-06-10 00:00:00	1957-10-20 00:00:00	Sidney Falco				
#291:0	1	Employee	2012-03-27 00:00:00	1965-11-01 00:00:00	Carl Showalter				
#361:0	1	Manager	2009-02-25 00:00:00	1975-07-17 00:00:00	Larry Fine	20000			
#369:0	1	Executive	2005-01-30 00:00:00	1952-12-30 00:00:00	JJ Hunsecker		Porsche		
#377:0	1	Customer		1930-09-21 00:00:00	Wade Gustafson			802	19999
#378:0	1	Customer		1990-05-17 00:00:00	Audrey Horne			678	99

Modeling a Database

SELECT FROM Employee

METADATA			PROPERTIES				
@rid	@version	@class	HireDate	DOB	Name	SignatureAuthority	CompanyCarType
#290:0	1	Employee	2010-06-10 00:00:00	1957-10-20 00:00:00	Sidney Falco		
#291:0	1	Employee	2012-03-27 00:00:00	1965-11-01 00:00:00	Carl Showalter		
#361:0	1	Manager	2009-02-25 00:00:00	1975-07-17 00:00:00	Larry Fine	20000	
#369:0	1	Executive	2005-01-30 00:00:00	1952-12-30 00:00:00	JJ Hunsecker		Porsche



Relationships

- Purpose
- Referenced Relationships
- Embedded Relationships

Relationships

- Unlike RDBMS platforms, OrientDB doesn't rely on processing-intensive joins to associate different records
- Instead, it uses relationships - in the form of links or edges - to establish these connections and facilitate navigation
- This approach is at the heart of OrientDB's tremendous speed advantages

Relationships

- Two types of relationships available in OrientDB
 - Embedded
 - Referenced
- Both types of relationship may be used for graph as well as document data

Relationships

- It's up to the database designer to decide whether embedded or referenced is more appropriate for the situation at hand
 - Largely driven by anticipated traversal needs
- Embedding data combines two or more records
- Referencing data preserves object independence

Embedded Relationships

- Data from the embedded record is stored within the container record
- The embedded record doesn't have its own record id
 - Not possible to reference it from outside
 - Instead, it's only accessible from within the container
- If the container record is deleted, so is the embedded record

Options for Embedded

Type(s)	Data Type	Usage
I-N, N-I	EMBEDDED	For single record
I-N, N-M	EMBEDDEDLIST	Ordered list of records
I-N, N-M	EMBEDDEDSET	Unordered set of records. Duplicates not permitted.
I-N, N-M	EMBEDDEDMAP	Ordered map of records as values; string as key. Duplicate keys not permitted.

Referenced Relationships

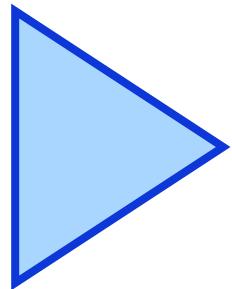
- The primary record contains links to one or more external records
 - Source class --> Destination class
 - Source property --> Destination property
- These references permit very fast cross-record access
- Helpful when importing data from an RDBMS

Referenced Relationships

- Default link direction depends on the type of object in question
- Links between documents are one-way
 - Although you can create a second link in the other direction, which mimics bi-directionality
- Edges between vertices offer bi-directional linkage

Options for Linked

Type(s)	Data Type	Usage
I-N, N-I	LINK	For single record
I-N, N-M	LINKEDLIST	Ordered list of links
I-N, N-M	LINKSET	Unordered set of links. Duplicates not permitted.
I-N, N-M	LINKMAP	Ordered map of links as values; string as key. Duplicate keys not permitted.



Documents

- Structure
- Identifying a Document
- Documents and Schema
- Documents and Relationships

Document Structure

- Collections of key/value pairs
 - Also known as fields or properties
- Key allows access to its value
- Values may contain
 - Primitive data type
 - Embedded document
 - Array of other values

Document Structure

- Similar in concept to document databases
- Documents may belong to a single class
 - Which may in turn inherit from a superclass
- Documents are stored in collections
 - Which enables grouping of different types of data

Identifying a Document

- Each record has its own identity
- This identity is unique in the database
- Format is
 - <cluster-id>:<cluster-position>

Identifying a Document

- When a record is deleted, its identity is recycled
 - Applications are responsible for keeping references valid
- Default RID for new records
 - -1:-1
 - Negative number indicates not yet persisted

Identifying a Document

- As long as the record itself still exists, record ids are fine for identifying related records
- However, this is no substitute for a context-specific primary key
- Application developers should consider creating and maintaining meaningful primary keys

Documents and Schema

- In general, most documents are schema-less or schema-mixed
- This makes it much easier to adjust applications as information requirements evolve

Documents and Relationships

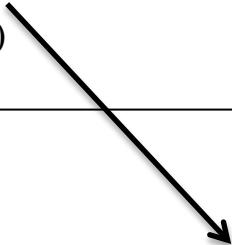
- Links are the mechanisms that establish relationships
- These are one-directional
- Requires developers to maintain integrity

Embedding a Single Document

```
{  
  "booking": {  
    "id": "JJKEL9",  
    "e_documentation": <..... EMBEDDED type  
    {  
      "documentation": {  
        "DocID": "89392191",  
        "type": "Passport"  
      }  
    }  
  }  
}
```

Linking to a Single Document

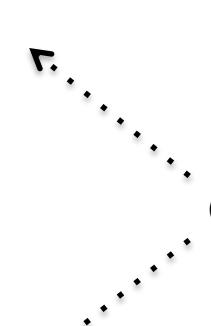
```
{  
  "booking": {  
    "id": "JJKEL9",  
    "l_documentation": <.....> LINK type  
  }  
}  
(Record Identifier)
```



```
{  
  "documentation": {  
    "DocID": "89392191",  
    "type": "Passport"  
  }  
}
```

Embedding a List of Documents

```
{  
  "booking": {  
    "id": "JJKEL9",  
    "e_documentation": <..... EMBEDDEDLIST type  
      {  
        "documentation": {  
          "DocID": "89392191",  
          "type": "Passport"  
        }  
      }  
      {  
        "documentation": {  
          "DocID": "902212",  
          "type": "Visa"  
        }  
      }  
  }  
}
```



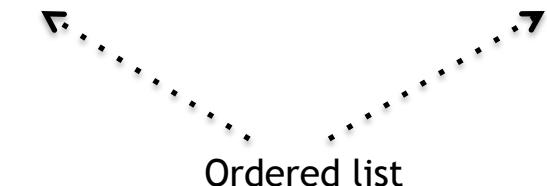
Ordered list

Linking to Multiple Documents

```
{  
  "booking": {  
    "id": "JJKEL9",  
    "l_documentation": <..... LINKEDLIST type  
  }  
}  
(Record Identifiers)
```

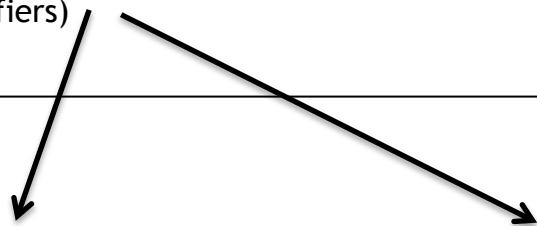
```
{  
  "documentation": {  
    "DocID": "89392191",  
    "type": "Passport"  
  }  
}
```

```
{  
  "documentation": {  
    "DocID": "902212",  
    "type": "Visa"  
  }  
}
```



Linking to a Set of Documents

```
{  
  "booking": {  
    "id": "JJKEL9",  
    "l_documentation": <.....>      LINKSET type  
  }          (Record Identifiers)  
}
```



```
{  
  "documentation": {  
    "DocID": "902212",  
    "type": "Visa"  
  }  
}
```

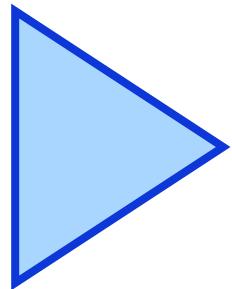
```
{  
  "documentation": {  
    "DocID": "89392191",  
    "type": "Passport"  
  }  
}
```

Embedding a Set of Documents

```
{  
  "booking": {  
    "id": "JJKEL9",  
    "e_documentation": <..... EMBEDDEDSET type  
    {  
      "documentation": {  
        "DocID": "902212",  
        "type": "Visa"  
      }  
    }  
    {  
      "documentation": {  
        "DocID": "89392191",  
        "type": "Passport"  
      }  
    }  
  }  
}
```

RDBMS vs. Document vs. OrientDB

RDBMS	Document	OrientDB
Table	Collection	Class or Cluster
Row	Document	Document
Column	Key/value pairs	Document field
Foreign key / Join table	N/A	Link



Graphs

- Base Classes
- Vertex
- Edge
- Relationships
- Sample OrientDB Data Model

Base Classes

- Vertex
 - Inherits from V class
- Edge
 - Inherits from E class

Vertex

- Primary storage object for graph data
- Can extend another class
- Can be assigned to a cluster
- Can be associated with a different class after creation
- Can be linked to other vertices via edges

Mandatory Vertex Properties

- Unique identifier
 - Composed of
 - Cluster number
 - Record identifier
- Set of incoming edges (record ids)
- Set of outgoing edges (record ids)
- One or more key/value pairs (optional)

Creating Vertices

- Vertices may be created with or without data
 - Properties may be set later
- Vertices may contain properties, JSON content, or both
- Vertices may be placed on specific clusters

Edge

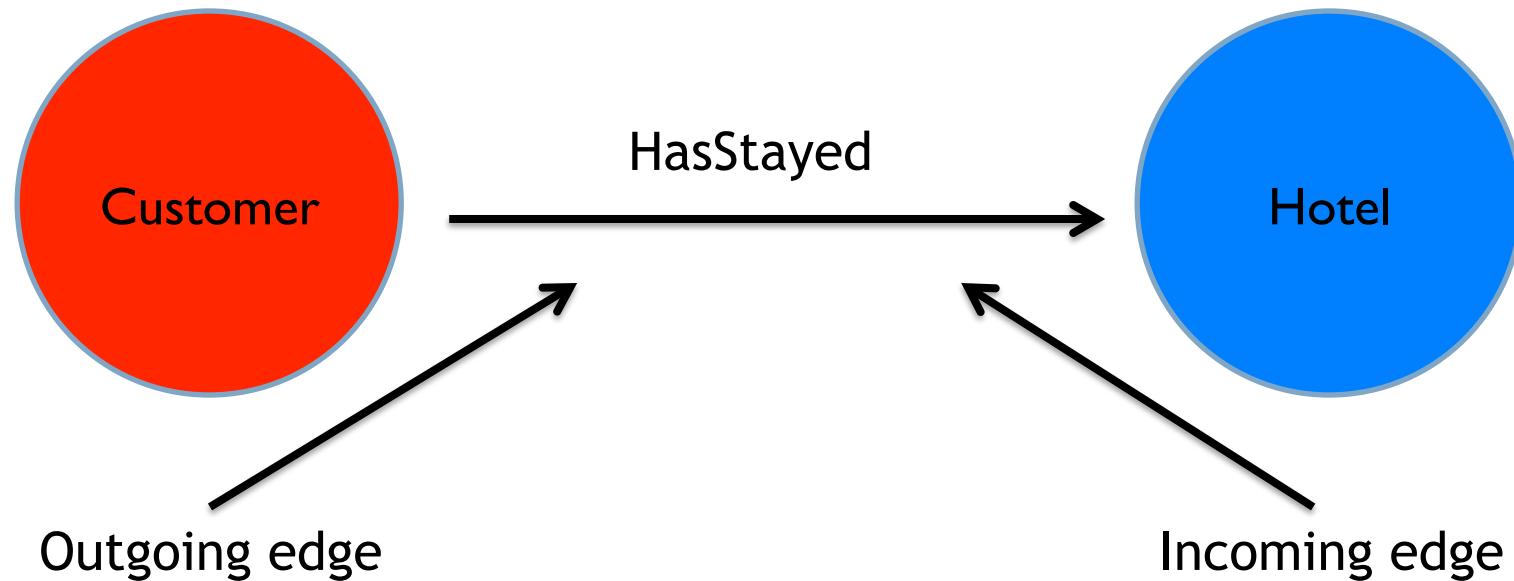
- Meant to link two vertices
- The base edge class (E) inherits from links
- Edges typically - but not always - represent a 1 to many relationship

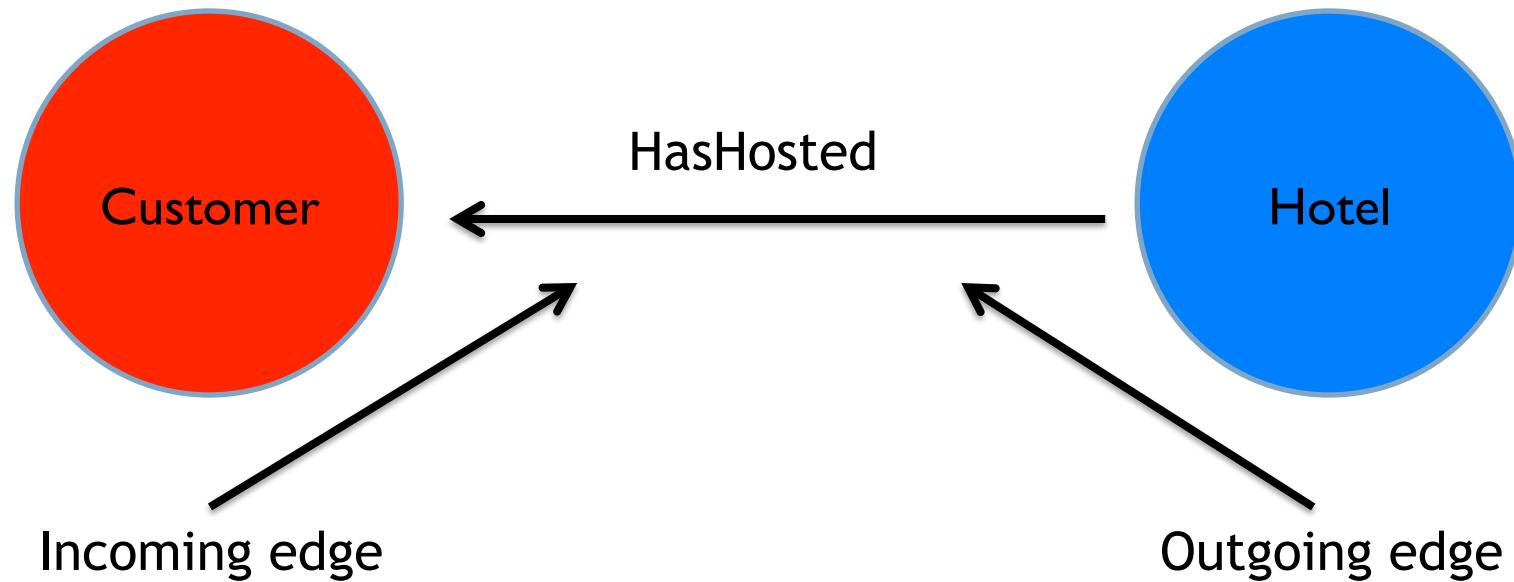
Mandatory Edge Properties

- Unique identifier
 - Composed of
 - Cluster number
 - Record identifier
- Link to incoming vertex
- Link to outgoing vertex
- Label which defines the type of connection/relationship

Creating Edges

- Edges may inherit from other edges
- Edges may contain properties, JSON content, or both
 - Can help enforce database integrity
- Edges may be placed on specific clusters
- Edges may be defined with constraints, such as unique indexes (which make them mandatory)





Lightweight Edges

- If an edge has no properties, it's stored as a link inside its related vertex
 - This is known as a 'lightweight edge'
- Lightweight edges help improve performance and reduce storage requirements
- Default database behavior is regular edges
 - Setting a database flag enables creating lightweight edges

Graph and Relationships

- Graph uses edges to link vertices
- Bi-directional
- Once established, OrientDB automatically maintains these relationships

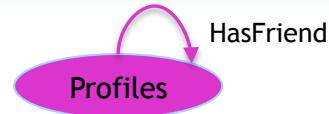


Modeling a Database

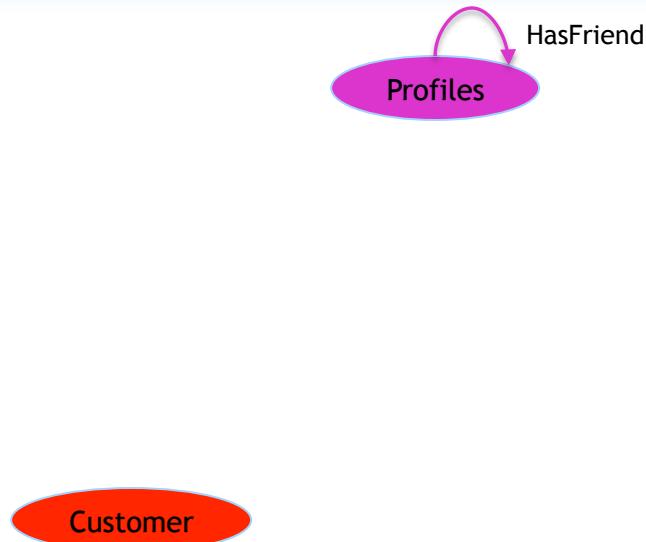


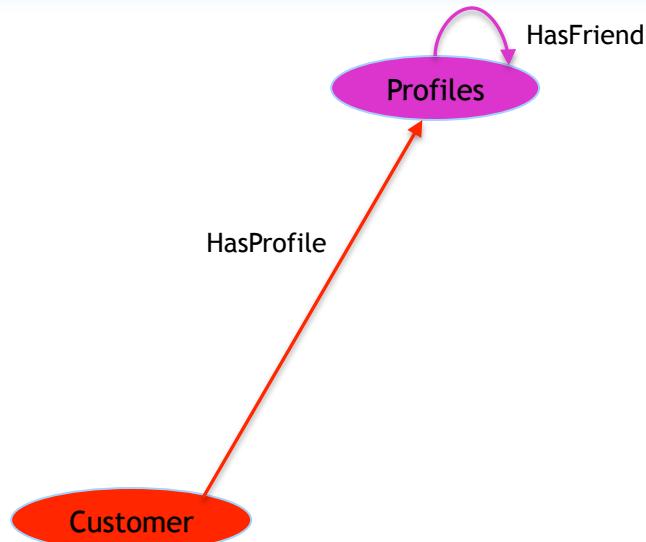
Profiles

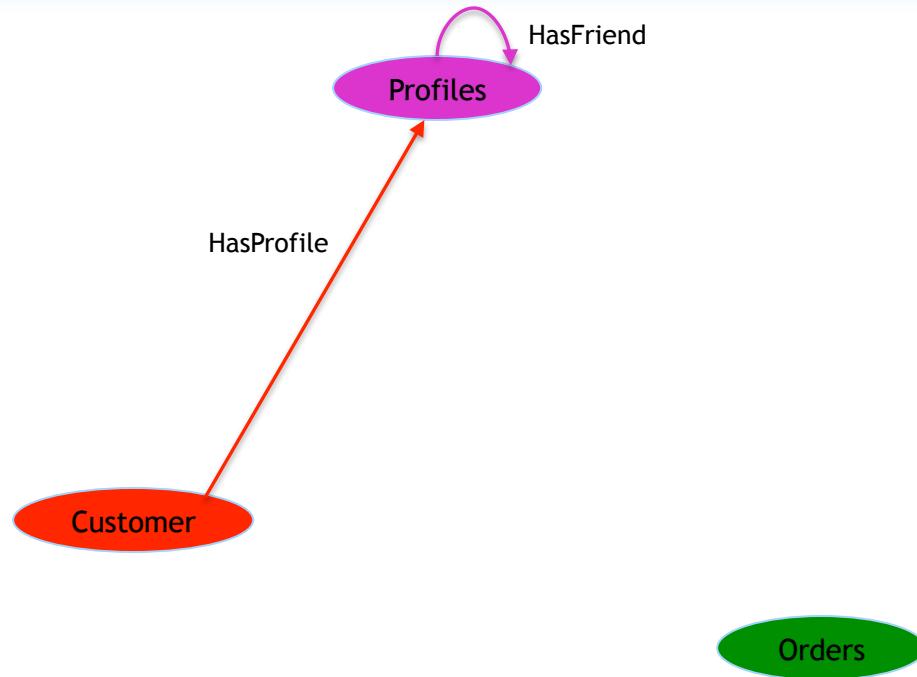
Modeling a Database

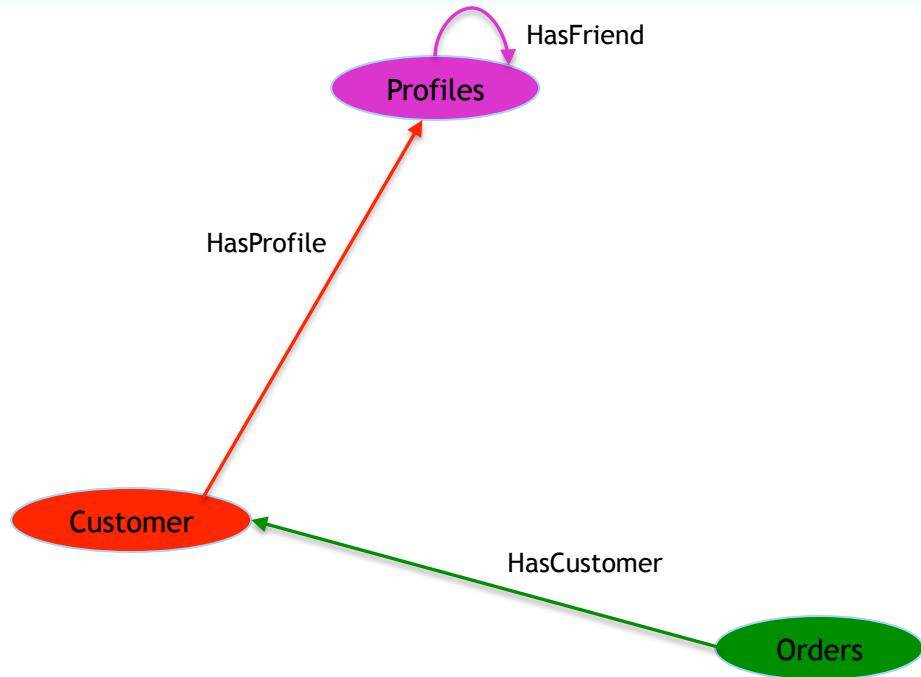


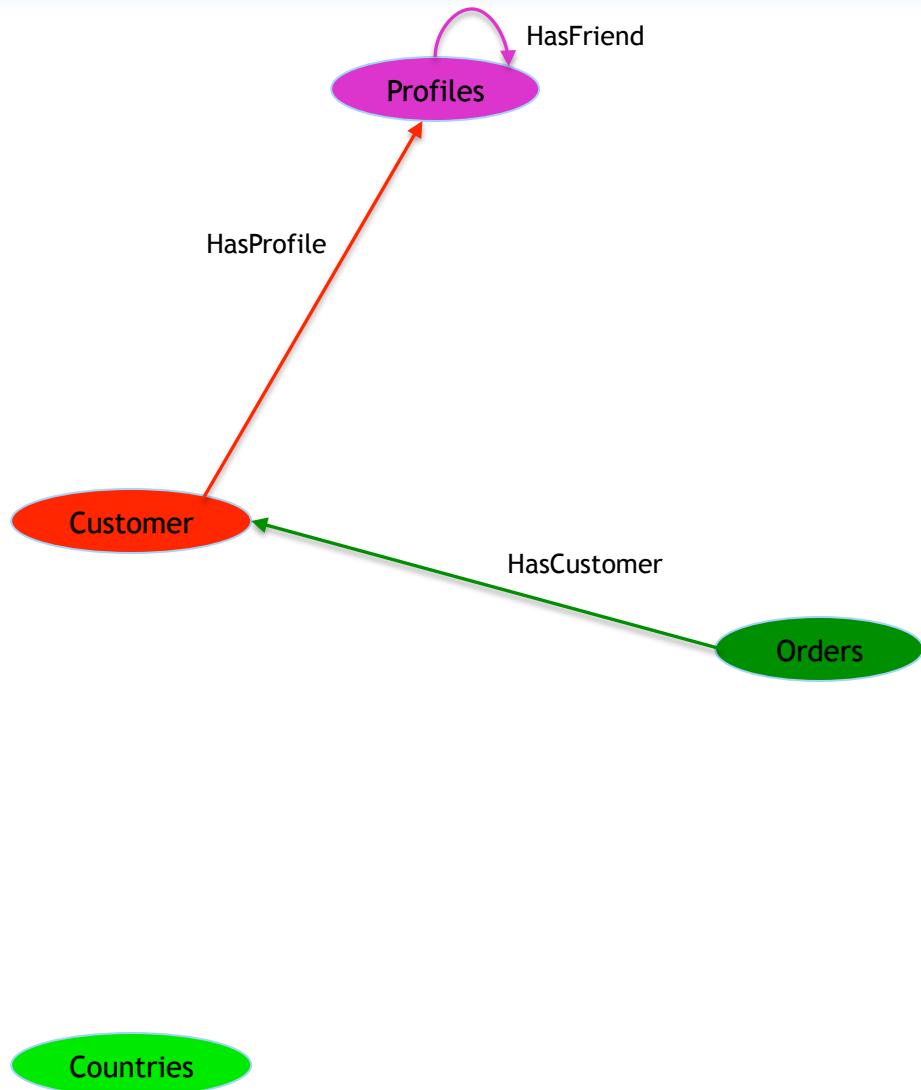
Modeling a Database

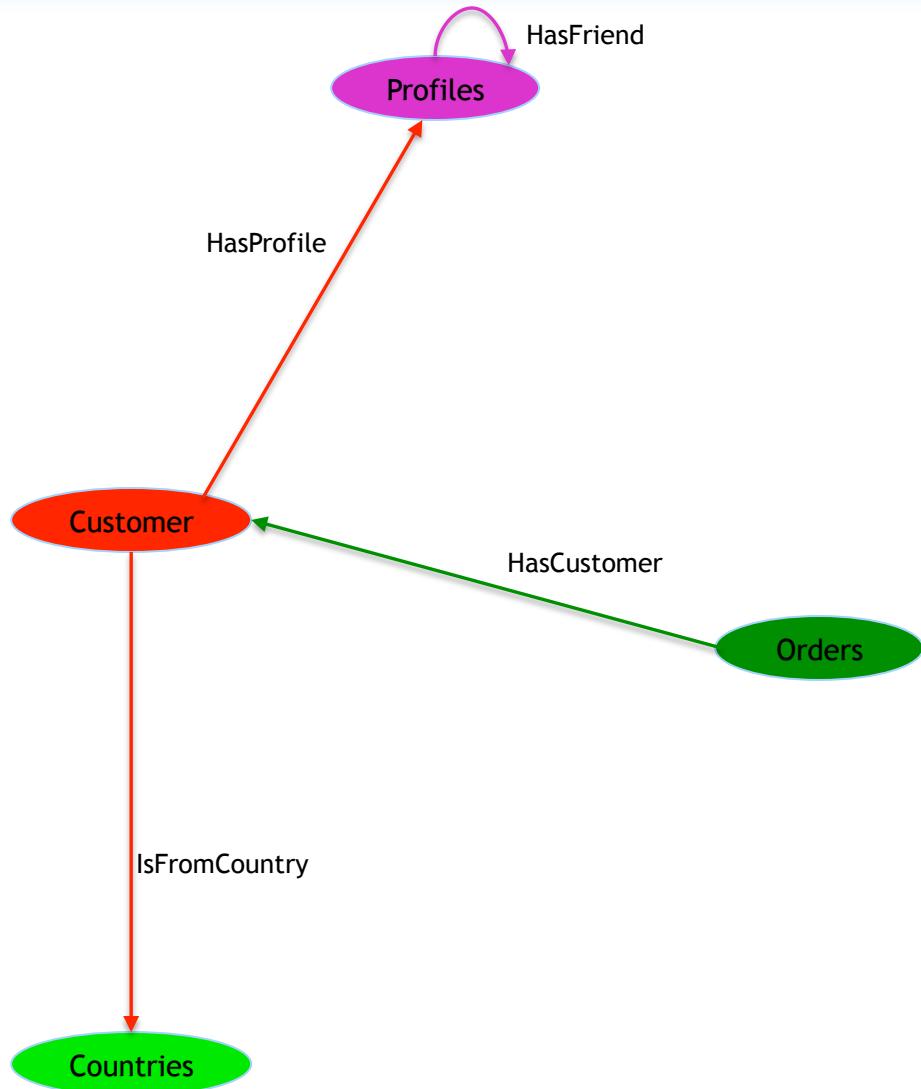


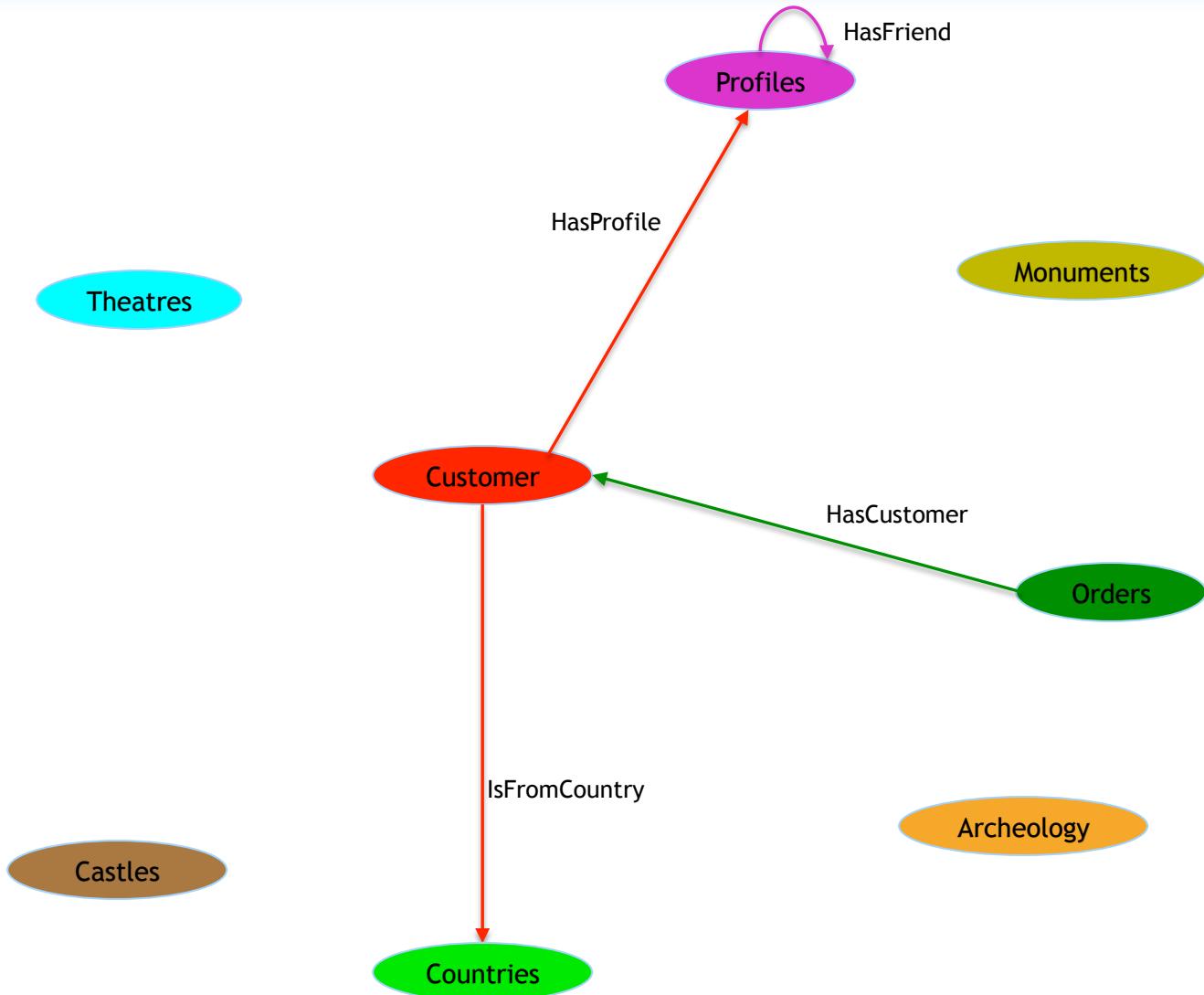
Modeling a Database

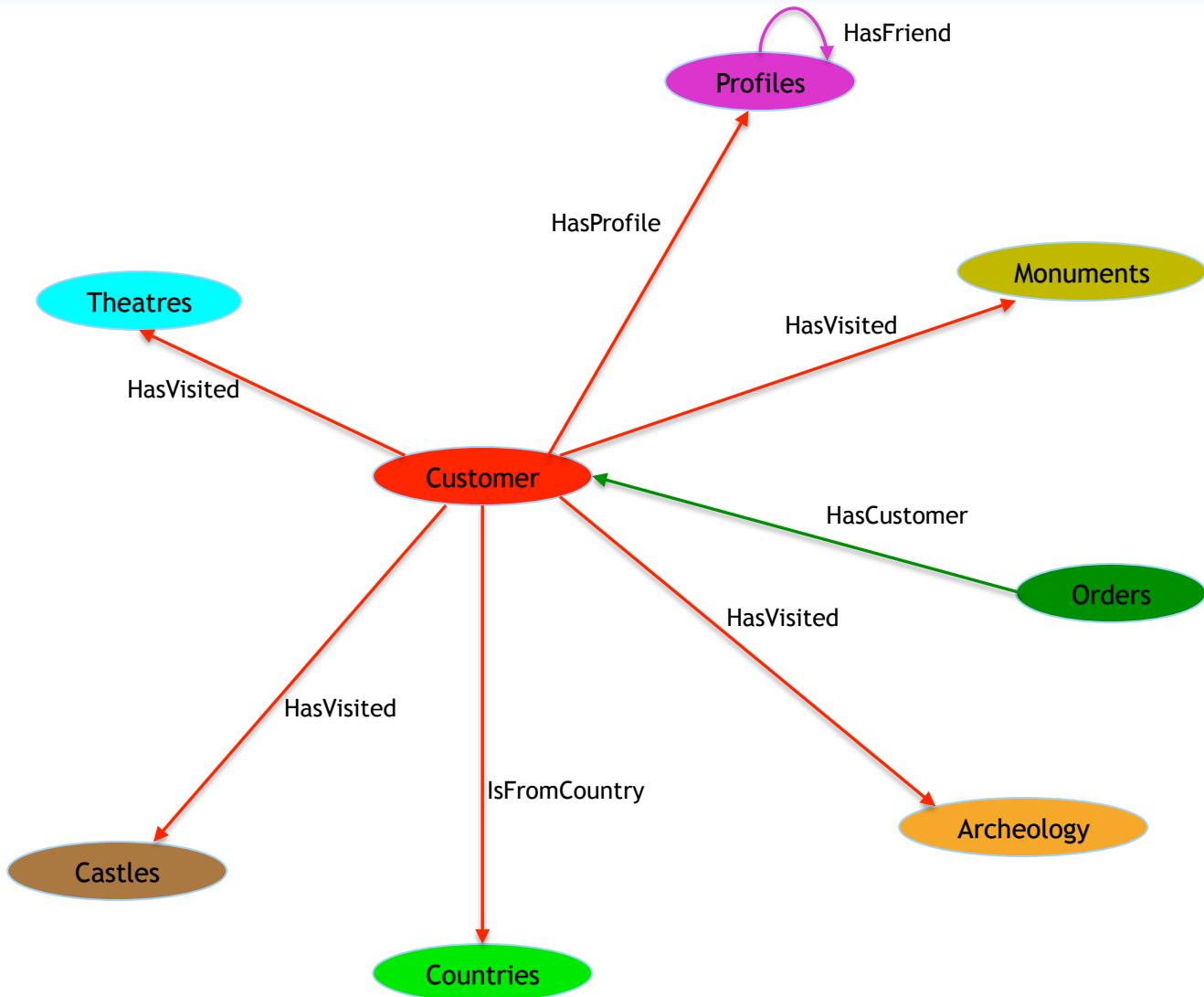
Modeling a Database

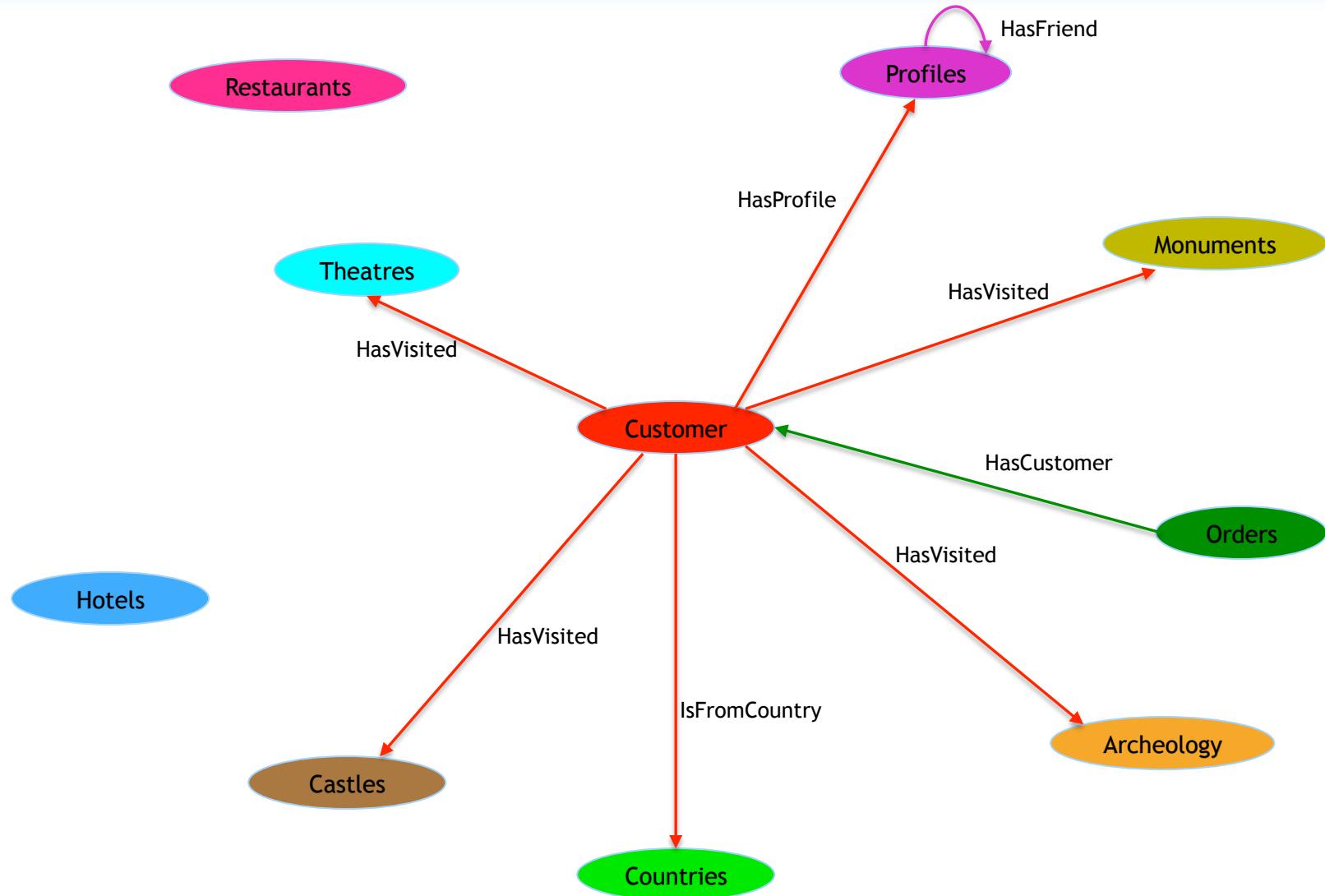
Modeling a Database

Modeling a Database

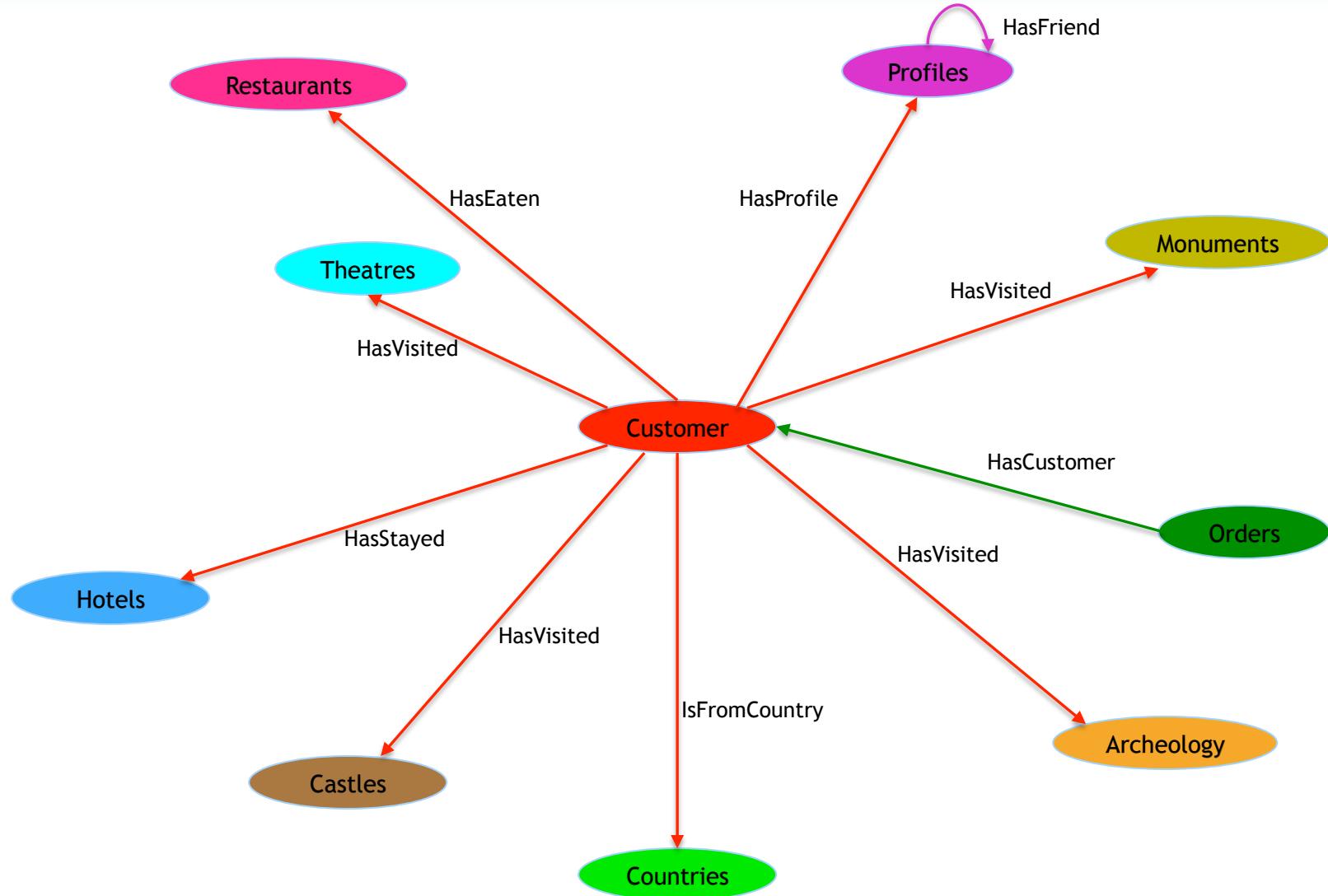
Modeling a Database

Modeling a Database

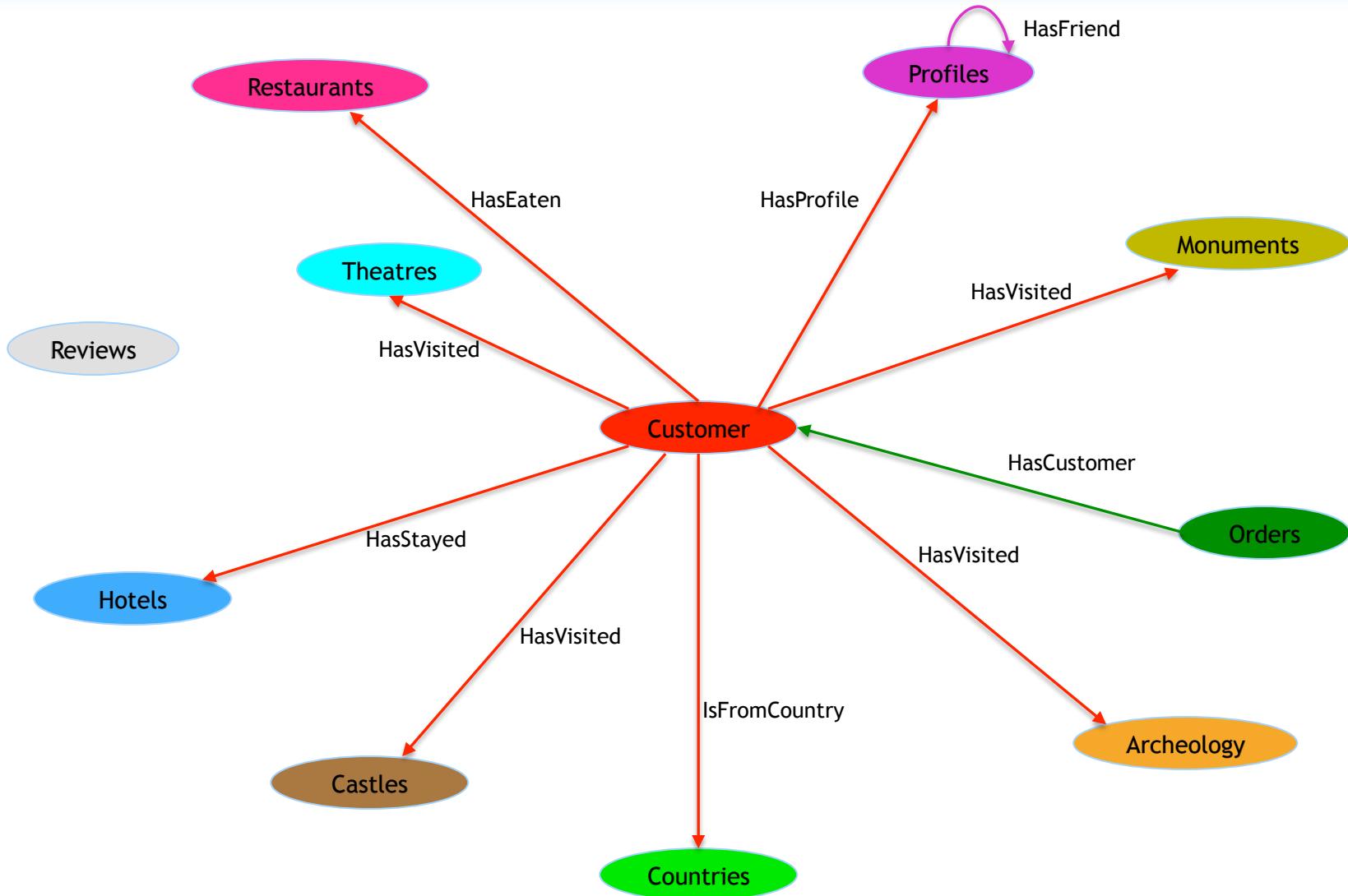
Modeling a Database

Modeling a Database

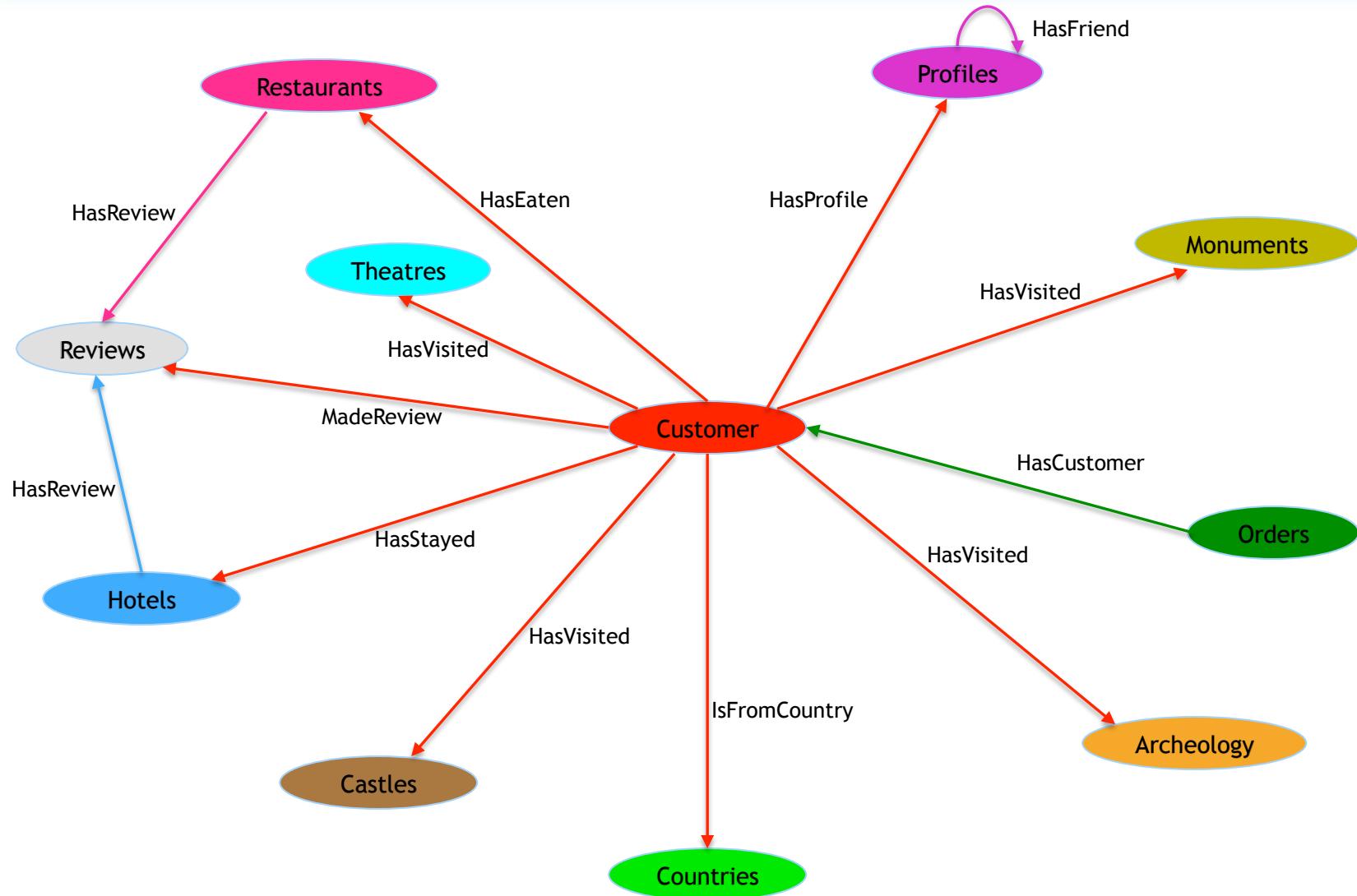
Modeling a Database



Modeling a Database

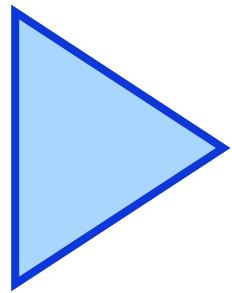


Modeling a Database



RDBMS vs. Graph vs. OrientDB

RDBMS	Key/Value	OrientDB
Table	Vertices & Edge set	Vertices & Edge set + subtype sets
Row	Vertex	Vertex
Column	Vertex/Edge property	Vertex/Edge property
Relationship	Edge	Edge



A Workflow for Modeling

- Comparison with RDBMS Approach
- Discovering User Stories
- Defining Major Artifacts
- Testing Model with Sample Queries
- Refining the Model

Modeling

- Designing a multi-model database is quite straightforward
- This is particularly notable when compared with how most relational databases are modeled

RDBMS Modeling

- Although each design process is unique, most relational modeling initiatives follow this flow:
 - Model entities and their relationships
 - Create the logical model
 - Create the physical model
 - Adjust the physical model for performance or other operational reasons
- The outcome is often a model - and application logic - that do not cope well with change

RDBMS Modeling

- Business requirement evolution can have serious impacts on a relational model that's already in production
- In turn, relational model changes can easily break application code
- This often requires making major - and risky - revisions to the database and associated software

Graph & Document Modeling

- In general, things are easier with multi-model databases
- They more closely match how real world situations are depicted
- The resulting physical data model commonly closely matches the logical data model

Graph & Document Modeling

- There's no need for
 - Tables
 - Primary keys
 - Foreign keys
 - Joins
 - Normalization
 - Denormalization

Modeling Steps

- Begin with user stories
- Define major artifacts
- Consider linkage to a relational database
- Test model with sample queries
- Refine the model

User Stories

- Include domain experts
 - Much more important than in relational modeling
- Use natural language
 - Critical for defining relationships
- Diagram interaction among key business objects

User Stories

- Refer to existing database and/or systems for guidance
 - RDBMS or otherwise
- Consider importing RDBMS data into a sample OrientDB database to help visualize relationships
 - OrientDB Teleporter makes this easy

User Stories

- Don't underestimate the importance of expected queries
- These are essential ingredients in coming up with an accurate data model

Define Major Artifacts

- Common nouns often equate to vertices
- Identify vertex properties
- These properties are usually represented as proper nouns
 - Simple concepts that describe the rest of the domain
 - That don't have their own identity

Define Major Artifacts

- Adjectives also relevant for properties
- Common verbs often equate to edges
- Remember that when using lightweight edges, OrientDB stores record ids within the vertex
 - For performance advantages

RDBMS Linkage

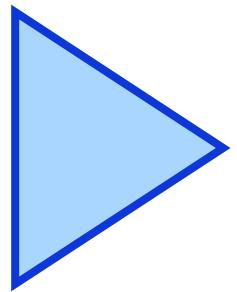
- Some of the required data may already be resident in a relational database
- Rather than importing all of this information, it may be possible to simply link to it

Sample Queries

- Sample queries are a great way to see if the model will meet real world requirements
 - It's also important to provide some sample data
- Use an index to speed the initial lookup
- To conduct graph queries, don't forget to consider MATCH and/or TRAVERSE

Refine the Model

- Sample queries against sample data may uncover areas for improvement
- It's easy to make these alterations to the model and then re-try the queries



Evolving the Model

- Comparison with RDBMS Approach
- Steps to Follow

RDBMS Model Evolution

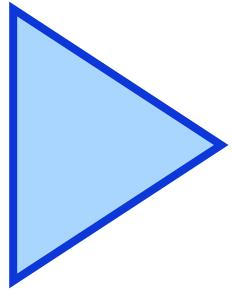
- Modeling a relational database usually entails sacrificing model accuracy for performance
 - Denormalization, for example
- For a relational database, model changes can be daunting
 - Table/column/index changes
 - Application code alterations

Graph & Document Evolution

- Designing multi-model databases assumes that alterations will take place
 - Much more in line with modern agile approaches
- In general, this is fairly straightforward
 - Add new vertices, edges, or properties as needed

Graph & Document Evolution

- Existing application code should continue to function
- As time permits, new logic can be added to make use of enhanced data model



Interacting with an OrientDB Database

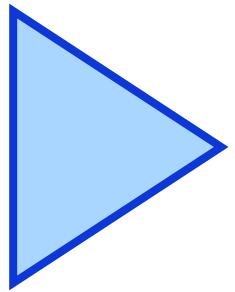
- Interaction Options
- Creating a Database
- Creating Data
- Retrieving Data
- Updating & Deleting Data
- Functions & Hooks

Unit Goals

- Explore different techniques to interact with data
- Learn how to use SQL for graph & document data
- Use functions to simplify database interaction

Unit Goals

- Use hooks to capture and take action on events
- See how concurrency can prevent data damage
- Understand how transactions can safeguard data



Interaction Options

- Console
- OrientDB Studio
- HTTP Tools
- APIs
- SQL Overview

Interactive Options

- OrientDB supplies several components for interacting with a database
- Console
 - Run from a terminal or Windows command console
 - Provides access to an array of useful commands
- OrientDB Studio
 - Rich, graphical browser based application

Interactive Options

- HTTP tools
 - Using URL syntax
 - cURL
 - Postman
 - SoapUI
- This class focuses on SQL interaction through OrientDB Studio and Console

APIs

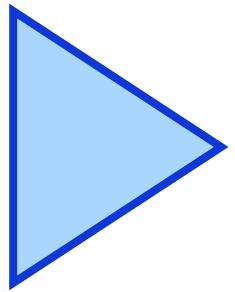
- OrientDB offers APIs for numerous languages, such as
 - Java
 - Python
 - Groovy
 - And many others
- There's also an HTTP/REST API

SQL Overview

- OrientDB purposely chose SQL as the primary interactive database language
 - Unlike other graph database technologies that require learning an often-proprietary language
- Experienced RDBMS users will be able to learn OrientDB SQL quickly
- For graph database access, OrientDB extended standard SQL

SQL Overview

- Syntax is derived from JDO/JPA standards
 - JDO = Java Data Objects
 - JPA = Java Persistence API
- OrientDB SQL is able to work with
 - Properties (names are case sensitive)
 - Variables (user defined as well as contextual)
 - Multi-values
 - Relationships
 - Schema-less data



Creating a Database

- Requirements
- Database Attributes

Creating a Database

- Database name must be unique
- Database may be stored in memory, or on disk
- Database may be created and then immediately restored from an incremental backup

Major Database Attributes

Attribute	Purpose
database-url	Path to the database
user	Default username
password	Default password
storage-type	PLOCAL or MEMORY
db-type	Document or Graph (default)

Data Types

- OrientDB supports all major data types
- See table in workbook
- Learn more at
 - <http://orientdb.com/docs/master/Types.html>

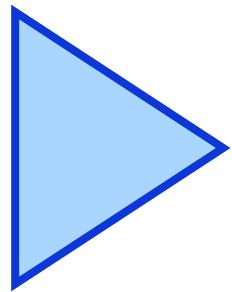
OrientDB & Dates

- Internally, dates are stored in UNIX time format
 - Long
 - Counts milliseconds since Jan 1 1970
- OrientDB offers a collection of functions to work with dates
 - <http://orientdb.com/docs/last/Managing-Dates.html>

OrientDB & Binary Data

Three options for storing binary data in an OrientDB database:

Strategy	Stored As	Details
Path	String	Stores a reference to the file system or a blob object
Embedded byte[]	Binary	Encoded as BASE64, which consumes 33% more space
ORecordBytes	Link	Most efficient approach, but requires more complex management



Creating Data

- About the Examples
- Core Concepts
- Documents
- Vertices & Edges - Graph
- Transactions

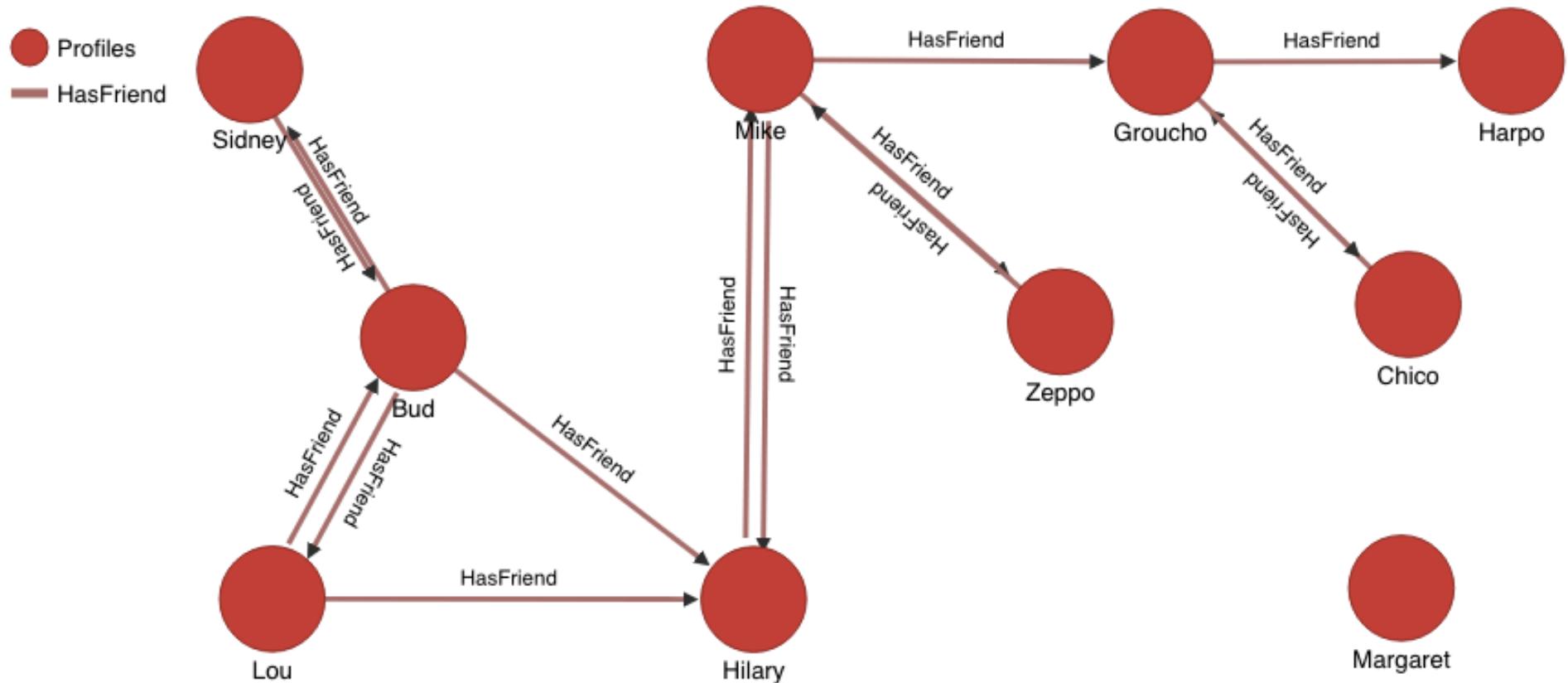
About the Examples

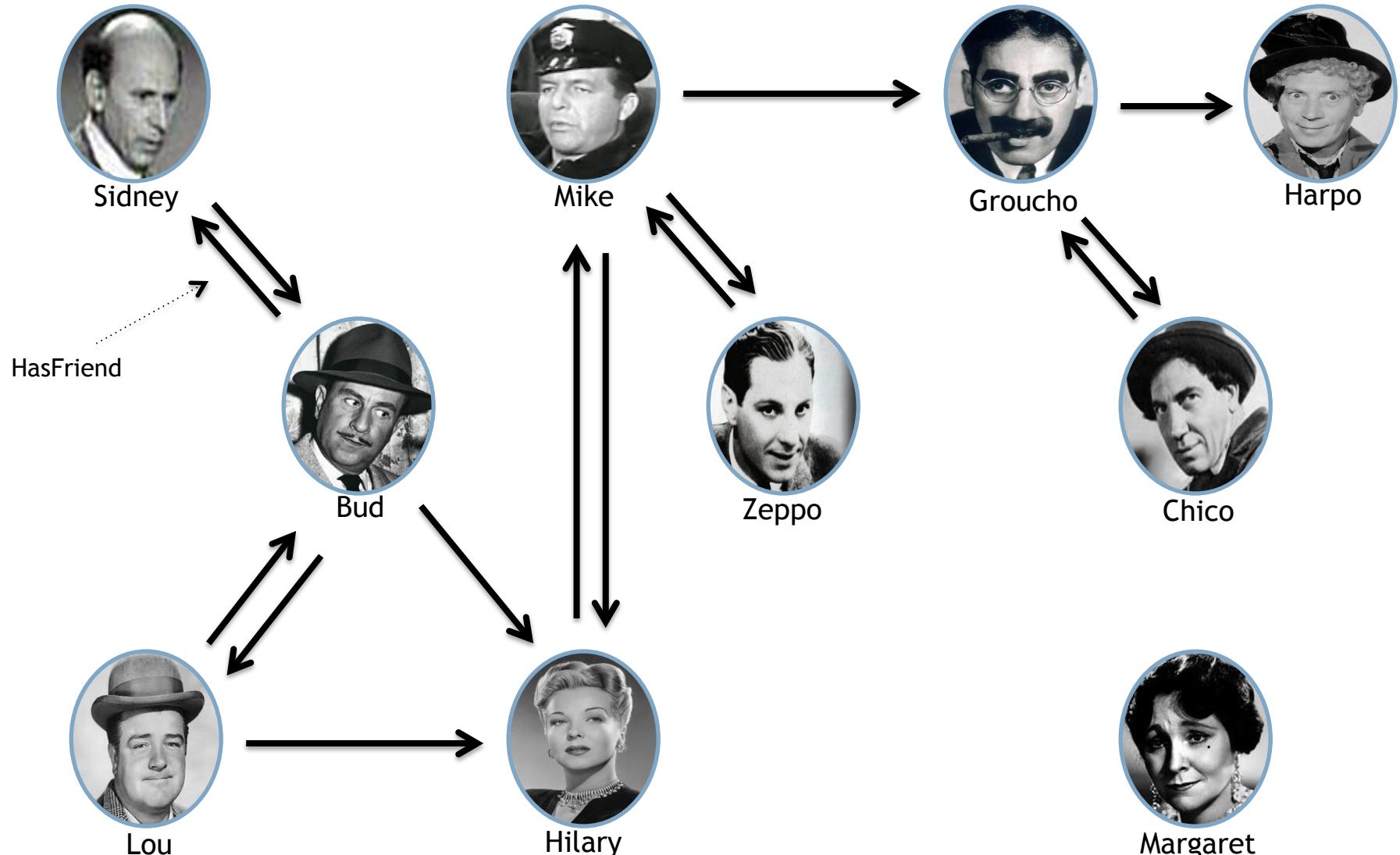
- Throughout this unit, we'll provide demonstrations of interacting with OrientDB
- They'll all build on the data model described in the previous unit
 - Primarily focused on a social graph of 10 people
 - Plus related flights and credit card applications

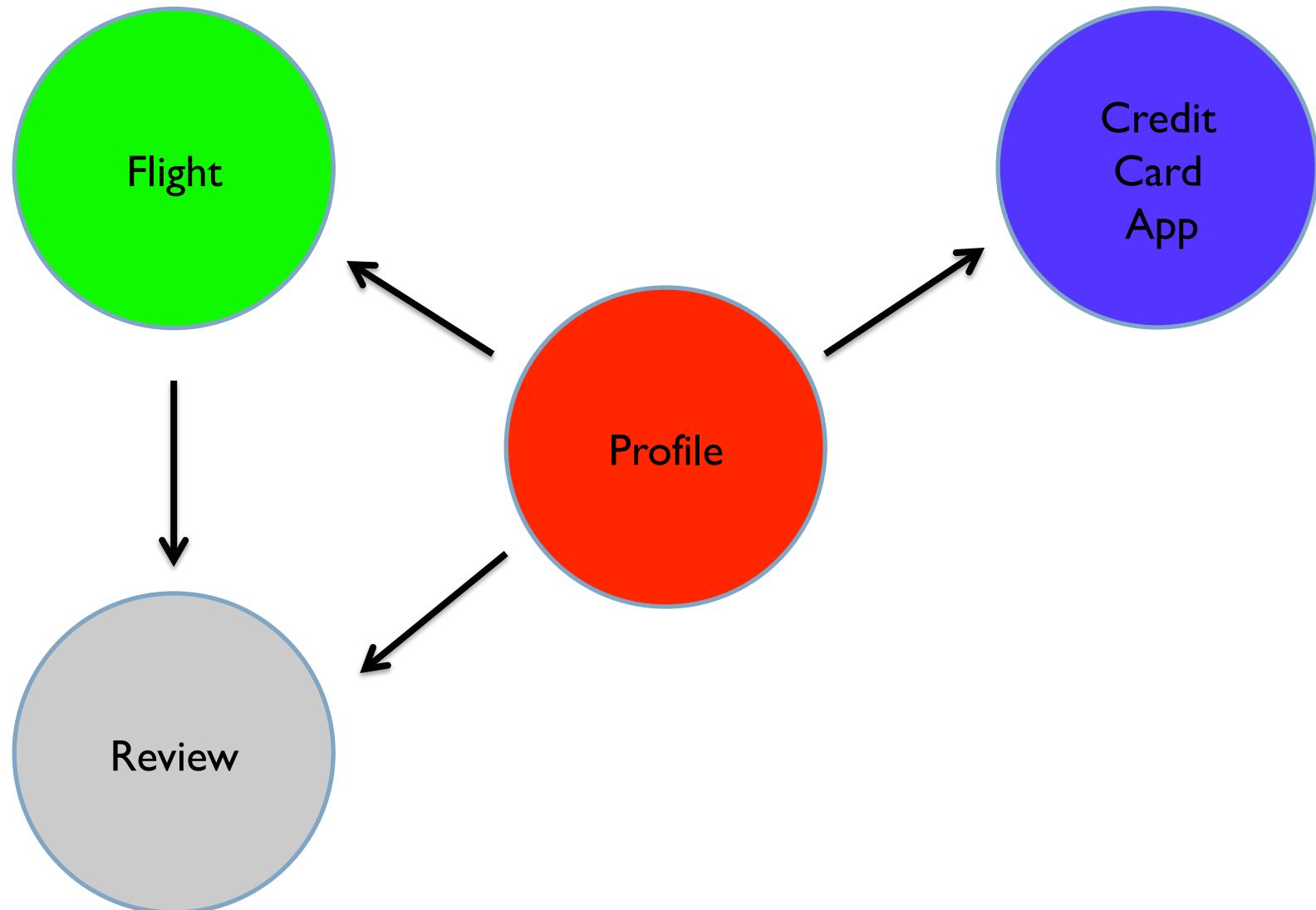
```
// Note that there is currently no
// record with an ID of 'LMN876'
UPDATE Booking SET ID = 'XYZ987'
UPSERT WHERE ID = 'LMN876';

SELECT * FROM Booking;
```

#	@RID	@CLASS	ID
0	#297:0	Booking	XYZ987

Interacting with an OrientDB Database

Interacting with an OrientDB Database



Core Data Creation Concepts

- Under the covers, document and graph data are essentially the same
- Primary difference is that graph data extends base classes (V and E)
- There are also a number of helpful graph-oriented functions

Creating New Records

- Before adding data to an OrientDB database, it's important to understand record ids (RID)
- An RID indicates the physical position of a record within a database
- Consists of the cluster number paired with the record position within the cluster

Creating New Records

- There are multiple choices for adding one or more records to a database
 - INSERT
 - CREATE VERTEX
 - CREATE EDGE

INSERT INTO

- Works for document and graph data
- Standard SQL, OrientDB, and JSON syntax supported
- Relationships can be specified
- Class name may be implicit or explicit

INSERT INTO

- Can specify a particular cluster
- Can insert a single record or multiple records
- Can insert from a subquery
 - INSERT or SELECT syntax supported

Data Creation Enforcement via STRICTMODE

```
CREATE CLASS Booking;  
  
ALTER CLASS Booking STRICTMODE TRUE;  
  
// STRICTMODE prevents this record from  
// being added - no properties defined  
INSERT INTO Booking SET ID = '3RWV43',  
Meal = 'Gluten free';
```

```
Error: com.orientechnologies.orient.core.exception.OValidationException: Found additional  
field 'ID'. It cannot be added because the schema class 'Booking' is defined as STRICT  
DB name="demodb"  
DB name="demodb"
```

```
// Adding properties to the class  
// will make it possible to create data  
CREATE PROPERTY Booking.ID STRING;  
CREATE PROPERTY Booking.Meal STRING;  
  
INSERT INTO Booking SET ID = '3RWV43',  
Meal = 'Gluten free';
```

```
Inserted record 'Booking#297:0{ID:3RWV43,  
Meal:Gluten free} v1' in 0.002000 sec(s).
```

Data Creation Enforcement via MIN, MAX

```
CREATE CLASS Phone EXTENDS V;
```

```
CREATE PROPERTY Phone.AreaCode STRING  
(MANDATORY TRUE, MIN 3, MAX 3);
```

```
// MIN, MAX prevents these records  
// from being created
```

```
INSERT INTO Phone SET AreaCode = '9'
```

```
Error: com.orientechnologies.orient.core.exception.OValidationException:  
The field 'Phone.AreaCode' contains fewer characters than 3 requested
```

```
INSERT INTO Phone SET AreaCode = '9999'
```

```
Error: com.orientechnologies.orient.core.exception.OValidationException:  
The field 'Phone.AreaCode' contains more characters than 3 requested
```

```
CREATE CLASS Payment EXTENDS V;

CREATE PROPERTY Payment.Amount FLOAT
(MIN .01, MAX 1000);

// MIN, MAX prevents these records
// from being created
INSERT INTO Payment SET Amount = -1
Error: com.orientechnologies.orient.core.exception.OValidationException:
The field 'Payment.Amount' is less than .01

INSERT INTO Payment SET Amount = 9999
Error: com.orientechnologies.orient.core.exception.OValidationException:
The field 'Payment.Amount' is greater than 1000
```

Defining Indexes

- Indexes may be created on vertices, edges, or documents
 - One or more properties must be in place
- If there's a schema in place, you may specify class.property
 - OrientDB treats it as an automatic index
- Indexes may be composite
 - Made up of multiple fields

Transactions

- Transactions are an essential technique for preserving data integrity
- OrientDB provides transactional support
 - Begin
 - All records have a temporary id until committed
 - Commit
 - Rollback

Creating Document Data

- Very easy to create document data
- All that's necessary is to create a class
 - Even without any defined properties
- Data can then be added with consistent or variable properties
- Full inheritance and polymorphism capabilities

Creating Document Examples

[New document using JSON](#)

[New document with an embedded set](#)

[Multiple new documents with different properties](#)

[New document with embedded document list](#)

[New document with link to another document](#)

[New document with links to multiple documents](#)

[New document with an embedded document](#)

[New document with an embedded list](#)

[New document with an embedded map](#)

[Create an index on document data](#)

Create Document Using JSON

```
CREATE CLASS Booking;
```

```
CREATE PROPERTY Booking.ID STRING;
```

```
CREATE PROPERTY Booking.Request STRING;
```

```
INSERT INTO Booking SET ID = 'NMU324',  
Request = '{"booking": {"id":  
"NMU324", "vip":true, "meal": "Vegan"} }'
```

```
SELECT ID, Request FROM Booking;
```

```
+----+-----+-----+
| #   | ID      | Request
+----+-----+-----+
| 0   | NMU324 | {"booking": {"id": "NMU324", "vip": true, "meal": "Vegan"}}
+----+-----+-----+
```

[Return to directory](#)

[Skip examples](#)

Create Multiple Documents with Different Properties

```
CREATE Class Booking;
```

```
CREATE PROPERTY Booking.ID STRING;
```

```
INSERT INTO Booking SET ID = 'NTV987';
```

```
INSERT INTO Booking SET ID = 'QSN822',  
Extra = 'Dynamic data - can vary by  
record';
```

```
INSERT INTO Booking SET ID = 'LWT83Y',  
Extra1 = 'Create even more  
properties';
```

```
SELECT FROM Booking;
```

#	@RID	@CLASS	ID	Extra	Extra1
0	#297:0	Booking	NTV987		
1	#298:0	Booking	QSN822	Dynamic data - can vary by record	
2	#299:0	Booking	LWT83Y		Create even more properties

[Return to directory](#)

[Skip examples](#)

Create a Document and Link To Another Document

```
CREATE CLASS Booking;
```

```
CREATE PROPERTY Booking.ID STRING;
```

```
CREATE PROPERTY  
Booking.L_Documentation LINK;
```

```
CREATE CLASS Documentation;
```

```
CREATE PROPERTY Documentation.DocID  
STRING;
```

```
CREATE PROPERTY Documentation.Type  
STRING;
```

```
INSERT INTO Documentation SET DocID =  
'89392191', Type = 'Passport';  
  
INSERT INTO Booking SET ID = 'SD1KOW',  
L_Documentation = (SELECT FROM  
Documentation WHERE DocID =  
'89392191');  
  
SELECT FROM Booking;
```

#	@RID	@CLASS	L_Documentation	ID
0	#297:0	Booking	#385:0	SD1KOW

```
SELECT EXPAND(L__Documentation)
FROM Booking
```

#	@RID	@CLASS	DocID	Type
0	#385:0	Documentation	89392191	Passport

[Return to directory](#)

[Skip examples](#)

Create a Document and Embed Another Document Within It

```
CREATE CLASS Booking;
```

```
CREATE PROPERTY Booking.ID STRING;
```

```
CREATE PROPERTY Booking.Request  
EMBEDDED;
```

```
INSERT INTO Booking SET ID = 'SD1KOW',  
Request = { "@type": "d", "@version":  
0, "@class": "Request", "ID":  
"SD1KOW", "flight": "KE  
809", "vip": true, "meal": "Vegan" }
```

```
SELECT ID, Request FROM Booking
```

#	ID	Request
0	SD1K0W	Request{ID:SD1K0W, flight:KE 809, vip:true, meal:Vegan}

```
SELECT EXPAND(Request) FROM Booking
```

#	@CLASS	ID	flight	vip	meal
0	Request	SD1K0W	KE 809	true	Vegan

[Return to directory](#)

[Skip examples](#)

Create a Document and
Embed A Map Within It

```
CREATE CLASS Booking;

CREATE PROPERTY Booking.ID STRING;

CREATE PROPERTY Booking.E_Requests
EMBEDDEDMAP;

INSERT INTO Booking
SET ID = 'SD1KOW', E_Requests =
{ 'Seating':'Window', 'Meal':'Vegetarian'
} ;
```

```
SELECT FROM Booking
```

#	@RID	@CLASS	E_Requests	ID
0	#297:0	Booking	{Seating=Window, Meal=Vegetarian}	SD1K0W

```
SELECT EXPAND(E_Requests) FROM Booking
```

#	value
0	Window
1	Vegetarian

[Return to directory](#)

[Skip examples](#)

Create a Document and
Embed A Set Within It

```
CREATE CLASS Booking;  
  
CREATE PROPERTY Booking.ID STRING;  
  
CREATE PROPERTY Booking.E_Requests  
EMBEDDEDSET;  
  
INSERT INTO Booking SET ID = 'SD1KOW',  
E_Requests = { 'Seating':  
{ 'Position': 'Window', 'Cabin': 'Premium  
economy' }, 'Meal':  
{ 'Restriction': 'Vegetarian', 'Breakfast  
': 'Declined' } }
```

```
SELECT FROM Booking
```

#	@RID	@CLASS	E_Requests	ID
0	#297:0	Booking	[{Seating:[2], Meal:[2]}]	SD1K0W

```
SELECT EXPAND(E_Requests) FROM Booking
```

#	Seating	Meal
0	{Cabin=Premium economy, Position=Window}	{Restriction=Vegetarian, Breakfast=Declined}

[Return to directory](#)

[Skip examples](#)

Create a Document and
Embed A Document List Within It

```
CREATE CLASS Booking;

CREATE PROPERTY Booking.ID STRING;

CREATE PROPERTY Booking.E_Requests
EMBEDDEDLIST;

INSERT INTO Booking SET ID = 'SD1KOW',
E_Requests = [ { "@type": "d", "@version": 0, "@class": "Documentation", "Type": "Passport" }, { "@type": "d", "@version": 0, "@class": "Documentation", "Type": "Visa" } ]
```

```
SELECT FROM Booking
```

```
+----+-----+-----+-----+-----+
| #  | @RID   | @CLASS  | E_Requests          | ID      |
+----+-----+-----+-----+
| 0  | #297:0|Booking | [Documentation{Type:Passport}, Documentation{Type:Visa}] | SD1K0W |
+----+-----+-----+-----+
```

```
SELECT EXPAND(E_Requests) FROM Booking
```

```
+----+-----+-----+
| #  | @CLASS      | Type    |
+----+-----+-----+
| 0  | Documentation|Passport|
| 1  | Documentation|Visa    |
+----+-----+-----+
```

[Return to directory](#)

[Skip examples](#)

Create a Document and Link To Multiple Documents

```
CREATE CLASS Booking;
```

```
CREATE PROPERTY Booking.ID STRING;
```

```
CREATE PROPERTY  
Booking.L_Documentation LINKLIST;
```

```
CREATE CLASS Documentation;
```

```
CREATE PROPERTY Documentation.DocID  
STRING;
```

```
CREATE PROPERTY Documentation.Type  
STRING;
```

```
INSERT INTO Documentation  
SET DocID = '89392191', Type =  
'Passport';
```

```
INSERT INTO Documentation  
SET DocID = '902212', Type = 'Visa';
```

```
INSERT INTO Booking SET ID = 'SD1KOW',  
L_Documentation = (SELECT FROM  
Documentation WHERE DocID = '89392191'  
OR DocID = '902212')
```

```
SELECT FROM Booking
```

#	@RID	@CLASS	L_Documentation	ID
0	#297:0	Booking	[#385:0,#386:0]	SD1K0W

```
SELECT EXPAND(L_Documentation)  
FROM Booking
```

#	@RID	@CLASS	DocID	Type
0	#385:0	Documentation	89392191	Passport
1	#386:0	Documentation	902212	Visa

[Return to directory](#)[Skip examples](#)

Create a Document and
Embed A List Within It

```
CREATE CLASS Booking;  
  
CREATE PROPERTY Booking.ID STRING;  
  
CREATE PROPERTY Booking.E_Requests  
EMBEDDEDLIST;  
  
INSERT INTO Booking  
SET ID = 'SD1KOW', E_Requests =  
['Window seat', 'Vegetarian meal'];
```

```
SELECT FROM Booking
```

#	@RID	@CLASS	E_Requests	ID
0	#297:0	Booking	[Window seat, Vegetarian meal]	SD1K0W

```
SELECT EXPAND(E_Requests) FROM Booking
```

#	value
0	Window seat
1	Vegetarian meal

[Return to directory](#)

[Skip examples](#)

Create Indexes on Document Data

```
CREATE CLASS Booking;
```

```
CREATE PROPERTY Booking.ID STRING;
```

```
INSERT INTO Booking SET ID = 'NTV987';
```

```
INSERT INTO Booking SET ID = 'QSN822',  
Extra = 'Dynamic data - can vary by  
record';
```

```
INSERT INTO Booking SET ID = 'LWT83Y',  
Extra1 = 'Create even more properties,  
but they are not yet formally part of  
class definition';
```

```
CREATE INDEX Booking.ID UNIQUE;
```

```
// Following won't work because Extral  
isn't yet in the class definition
```

```
CREATE INDEX Booking.Extral FULLTEXT;
```

```
// Add this property to class, and  
then re-attempt to create the index
```

```
CREATE PROPERTY Booking.Extral STRING;
```

```
CREATE INDEX Booking.Extral FULLTEXT;
```

```
SELECT FROM INDEX:Booking.Extral;
```

#	key	rid
0	Cre	#299:0
1	Crea	#299:0
2	Creat	#299:0
3	Create	#299:0
4	eve	#299:0
5	even	#299:0
6	mor	#299:0
7	more	#299:0
8	pro	#299:0
9	prop	#299:0
10	prope	#299:0
11	proper	#299:0
12	propert	#299:0
13	properti	#299:0
14	propertie	#299:0
15	properties	#299:0

[Return to directory](#)[Skip examples](#)

Creating Graph Data

- Almost identical to creating document data
- Prime difference is that graph data extends base classes V and E
 - V for vertex
 - E for edge
- Inheritance and polymorphism are available for graph data as well

Defining Vertices

- When creating a vertex, it's necessary to first define a class
- Class must extend base class V
 - Or a parent class that already extends V
 - Otherwise, it will be treated as a document

Defining Vertices

- Properties may be defined when creating the class
 - Or later
- CREATE PROPERTY
 - Supports numerous data types
 - Properties may be defined as mandatory
 - String values may be restricted by size (max, min)

CREATE VERTEX

- Meant for graph data
- Vertices may be created directly
 - Added to base class V
- A better approach is to first define a class and then add the vertex instance(s) afterwards

CREATE VERTEX

- Vertex data can be created by
 - Specifying property names and values, or
 - Providing JSON content
- Vertices may be assigned to a specific cluster

Defining Edges

- When creating an edge, it's necessary to first either
 - Define an edge class, or
 - Directly create the edge between two vertices
- An edge class must extend base class E
 - Or a parent class that already extends E

Defining Edges

- As part of creating the edge, establish a relationship between two vertices
- One vertex will be FROM; the other will be TO
- Vertices being connected by edges may be identified by record ids or a query

Placing Constraints on Edges

- Along with connecting vertices, edges can help with maintaining database integrity
- For example, an edge can be defined with properties that restrict which types of vertices they may connect

CREATE EDGE

- Creates a relationship between two vertices
- Edges may be created directly
 - Added to base class E
- A better approach is to first create an edge class and then the edge instance(s) afterwards

CREATE EDGE

- Edge data can be created by
 - Specifying property names and values, or
 - Providing JSON content
- Edges may be assigned to a specific cluster
- Timeout (in milliseconds) can be specified in case of contention

Creating Graph Examples

[Creating a vertex that extends V](#)

[Create a vertex that extends an intermediate class](#)

[Create an edge with constraints on vertices](#)

[Create an edge with constraints and an index](#)

[Insert a vertex using standard SQL](#)

[Insert a vertex using OrientDB SQL](#)

[Insert a vertex into a specific cluster](#)

[Create a vertex instance](#)

[Create a vertex using JSON content](#)

[Create an edge between two vertices](#)

[Create an edge using record ids](#)

[Create a vertex in a transaction](#)

[Rollback a transaction](#)

Create a Vertex Class that
Directly Extends V

```
CREATE CLASS CreditApplication  
EXTENDS V;
```

```
CREATE PROPERTY CreditApplication.Id  
LONG;
```

```
CREATE PROPERTY  
CreditApplication.DateSubmitted DATE;
```

```
CREATE PROPERTY  
CreditApplication.CardType STRING;
```

```
CREATE INDEX CreditApplication.Id ON  
CreditApplication(Id) UNIQUE;
```

DESCRIBE CreditApplication;

```
CLASS 'CreditApplication'

Records.....: 5
Super classes....: [V]
Default cluster....: creditapplication (id=249)
Supported clusters...: creditapplication(249), creditapplication_1(250), creditapplication_2(251), creditapplication_3(252), creditapplication_4(253), creditapplication_5(254), creditapplication_6(255), creditapplication_7(256)
Cluster selection....: round-robin
Oversize.....: 0.0
Subclasses.....: PromotionApplications
```

PROPERTIES

#	NAME	TYPE	LINKED-TYPE/CLASS	MANDATORY	READONLY	NOT-NULL	MIN	MAX	COLLATE	DEFAULT	
0	Id	LONG		false	false	false			default		
1	CardType	STRING		false	false	false			default		
2	DateSubmitted	DATE		false	false	false			default		

INDEXES (1 altogether)

#	NAME	PROPERTIES
0	CreditApplication.Id	[Id]

[Return to directory](#)

[Skip examples](#)

Create a Vertex Class that
Extends An Intermediate Class

```
// Class CreditApplication extends V
CREATE CLASS PromotionApplications
EXTENDS CreditApplication;

CREATE PROPERTY
PromotionApplications.Description
STRING;

INSERT INTO PromotionApplications SET
Id = 101, DateSubmitted =
'2018-10-17', CardType = 'Amex
Starwood Business', DESCRIPTION =
'Mailed to all previous cardholders';
```

DESCRIBE PromotionApplications

```
CLASS 'PromotionApplications'

Records.....: 1
Super classes.....: [CreditApplication]
Default cluster....: promotionapplications (id=489)
Supported clusters...: promotionapplications(489), promotionapplications_1(490), promotionapplications_2(491), promotionapplications_3(492), promotionapplications_4(493), promotionapplications_5(494), promotionapplications_6(495), promotionapplications_7(496)
Cluster selection....: round-robin
Oversize.....: 0.0

PROPERTIES
+-----+-----+-----+-----+-----+-----+-----+-----+
| # | NAME      | TYPE    | LINKED-TYPE/CLASS | MANDATORY | READONLY | NOT-NULL | MIN | MAX | COLLATE | DEFAULT |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 0 | Id        | LONG   |                   | false     | false    | false   |     |     | default  |           |
| 1 | CardType  | STRING |                   | false     | false    | false   |     |     | default  |           |
| 2 | DateSubmitted | DATE |                   | false     | false    | false   |     |     | default  |           |
| 3 | Description | STRING |                   | false     | false    | false   |     |     | default  |           |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
SELECT FROM PromotionApplications
```

#	@RID	@CLASS	Id	DateSubmitted	CardType	Description
0	#489:0	PromotionApplications	101	2018-10-17 00:00:00	Amex Starwood Bus...	Mailed to all previous cardholders

[Return to directory](#)

[Skip examples](#)

Create an Edge with Constraints On Vertices

```
CREATE CLASS HasFriend EXTENDS E;  
  
// Only capable of accepting incoming  
// vertices from Profiles class  
CREATE PROPERTY HasFriend.in  
LINK Profiles;  
  
// Only capable of accepting outgoing  
// vertices to Profiles class  
CREATE PROPERTY HasFriend.out  
LINK Profiles;
```

```
DESCRIBE HasFriend;
```

```
CLASS 'HasFriend'
```

```
Records.....: 1631
```

```
Super classes....: [E]
```

```
Default cluster....: hasfriend (id=217)
```

```
Supported clusters...: hasfriend(217), hasfriend_1(218), hasfriend_2(219), hasfriend_3(220)  
, hasfriend_4(221), hasfriend_5(222), hasfriend_6(223), hasfriend_7(224)
```

```
Cluster selection....: round-robin
```

```
Oversize.....: 0.0
```

```
PROPERTIES
```

#	NAME	TYPE	LINKED-TYPE/CLASS	MANDATORY	READONLY	NOT-NULL	MIN	MAX	COLLATE	DEFAULT
0	in	LINK	Profiles		false	false	false		default	
1	out	LINK	Profiles		false	false	false		default	

[Return to directory](#)

[Skip examples](#)

Creating an Edge With Constraints and an Index

```
CREATE CLASS Submitted EXTENDS E;

CREATE PROPERTY Submitted.DateReceived
DATE;

CREATE PROPERTY Submitted.Method
STRING;

ALTER PROPERTY Submitted.Method
REGEXP "[M|W|P]";

CREATE INDEX Submitted.DateReceived ON
Submitted(DateReceived) NOTUNIQUE;
```

DESCRIBE Submitted;

```
CLASS 'Submitted'
Records.....: 5
Super classes....: [E]
Default cluster....: submitted (id=257)
Supported clusters...: submitted(257), submitted_1(258), submitted_2(259), submitted_3(260), submitted_4(261), submitted_5(262), submitted_6(263), submitted_7(264)
Cluster selection....: round-robin
Oversize.....: 0.0

PROPERTIES
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| # | NAME      | TYPE    | LINKED-TYPE/CLASS | MANDATORY | READONLY | NOT-NULL | MIN | MAX | COLLATE | DEFAULT |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 0 | DateReceived | DATE |           | false     | false     | false   |   |   | default |   |
| 1 | Method      | STRING |           | false     | false     | false   |   |   | default |   |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

INDEXES (1 altogether)

```
+-----+-----+
| # | NAME          | PROPERTIES   |
+-----+-----+
| 0 | Submitted.DateReceived | [DateReceived] |
+-----+-----+
```

Return to directory

Skip examples

Insert a Vertex Using Standard SQL

```
INSERT INTO Flight (Id, FlightNumber,  
FlightDate, FlightFrom, FlightTo)  
VALUES (5, 'DL 1973', '2018-02-25',  
'SJC', 'YVR')
```

```
Inserted record 'Flight#233:1{Id:5,FlightNumber:DL 1973,FlightDate:Sat Feb  
24 15:00:00 PST 2018,FlightFrom:SJC,FlightTo:YVR} v1' in 0.134000 sec(s).
```

[Return to directory](#)

[Skip examples](#)

Insert a Vertex Using OrientDB SQL Syntax

```
INSERT INTO Flight SET Id = 999,  
FlightNumber = 'AF 833', FlightDate =  
'2018-01-25', FlightFrom = 'GVA',  
FlightTo = 'CDG'
```

```
Inserted record 'Flight#234:1{FlightFrom:GVA,FlightTo:CDG,Id:6,FlightDate:  
Wed Jan 24 15:00:00 PST 2018,FlightNumber:AF 833} v1' in 0.028000 sec(s).
```

[Return to directory](#)

[Skip examples](#)

Insert a Vertex Into a Specific Cluster

```
INSERT INTO CLUSTER:Flight_3  
SET Id = 7, FlightNumber = 'WN 1253',  
FlightDate = '2017-08-22', FlightFrom  
= 'SFO', FlightTo = 'CLE'
```

```
Inserted record 'Flight#236:1{FlightFrom:SFO,FlightTo:CLE,Id:7,FlightDate:  
Mon Aug 21 15:00:00 PDT 2017,FlightNumber:WN 1253} v1' in 0.044000 sec(s).
```

[Return to directory](#)

[Skip examples](#)

Create a Vertex Instance

```
CREATE VERTEX Flight SET Id = 1000,  
FlightNumber = 'EI 1923', FlightDate =  
'2017-09-07', FlightFrom = 'DUB',  
FlightTo = 'LHR'
```

```
Created vertex 'Flight#235:1{Id:8,FlightNumber:EI 1923,FlightDate:Wed Sep 06 15:00:  
00 PDT 2017,FlightFrom:DUB,FlightTo:LHR} v1' in 0.020000 sec(s).
```

[Return to directory](#)

[Skip examples](#)

Create a Vertex Using JSON Content

```
CREATE VERTEX Flight  
CONTENT { "Id": "999", "FlightNumber" :  
"BA 8121", "FlightDate" :  
"2017-12-30", "FlightFrom" : "LGW",  
"FlightTo" : "NAP"} ;
```

```
Created vertex 'Flight#238:0{Id:999,FlightNumber:BA 8121,FlightDate:Fri Dec 29 15:00:00 PST  
2017,FlightFrom:LGW,FlightTo:NAP} v1' in 0.010000 sec(s).
```

```
SELECT FROM Flight WHERE Id = '999'
```

```
+----+-----+-----+-----+-----+-----+-----+-----+-----+  
| # |@RID |@CLASS|Id |FlightNumber|FlightDate |FlightFrom|FlightTo|  
+----+-----+-----+-----+-----+-----+-----+-----+  
| 0 |#238:0|Flight|999 |BA 8121 |2017-12-30 00:00:00|LGW |NAP |  
+----+-----+-----+-----+-----+-----+-----+-----+
```

[Return to directory](#)

[Skip examples](#)

Create an Edge Between Two Vertices

```
CREATE EDGE HasFriend FROM (SELECT
FROM Profiles WHERE Id=100004) TO
(SELECT FROM Profiles WHERE Id=100005)
```

```
+-----+-----+-----+-----+-----+
| #    | @RID      | @CLASS     | out      | in      |
+-----+-----+-----+-----+-----+
| 0    | #219:204 | HasFriend | #46:125 | #47:125 |
+-----+-----+-----+-----+-----+
Created '1' edges in 0.011000 sec(s).
```

[Return to directory](#)

[Skip examples](#)

Create an Edge Between
Two Vertices Using Record Id

```
CREATE EDGE HasFriend  
FROM 46:125 TO 47:125
```

```
+----+-----+-----+-----+-----+  
| # | @RID | @CLASS | out | in |  
+----+-----+-----+-----+-----+  
| 0 | #220:204 | HasFriend | #46:125 | #47:125 |  
+----+-----+-----+-----+-----+  
Created '1' edges in 0.031000 sec(s).
```

[Return to directory](#)

[Skip examples](#)

Create a Vertex in a Transaction

```
orientdb {db=demodb}> script sql
[Started multi-line command. Type just 'end' to finish and execute]
orientdb {db=demodb}> BEGIN;
orientdb {db=demodb}> INSERT INTO Payments SET Date = '2018-01-30', Amount = 88.88;
orientdb {db=demodb}> COMMIT;
orientdb {db=demodb}> end

Server side script executed in 0.013000 sec(s). Value returned is: Payments#508:0{Amount:88.88
Date:Mon Jan 29 15:00:00 PST 2018} v1
orientdb {db=demodb}> SELECT FROM Payments;

+----+----+----+----+----+
| # | @RID | @CLASS | Amount | Date           |
+----+----+----+----+----+
| 0 | #508:0 | Payments | 88.88 | 2018-01-30 00:00:00 |
+----+----+----+----+----+

1 item(s) found. Query executed in 0.006 sec(s).
```

[Return to directory](#)

[Skip examples](#)

Roll Back a Transaction

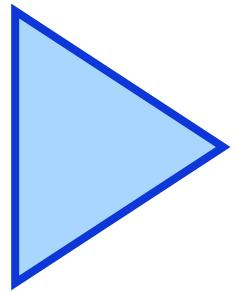
```
orientdb {db=demodb}> script sql
[Started multi-line command. Type just 'end' to finish and execute]
orientdb {db=demodb}> BEGIN;
orientdb {db=demodb}> INSERT INTO Payments SET Date = '2018-05-01', Amount = 129.02;
orientdb {db=demodb}> ROLLBACK;
orientdb {db=demodb}> end

Server side script executed in 0.004000 sec(s). Value returned is: Payments#509:-2{Amount:129.0
2,Date:Mon Apr 30 15:00:00 PDT 2018} v0
orientdb {db=demodb}> SELECT FROM Payments;

0 item(s) found. Query executed in 0.003 sec(s).
```

[Return to directory](#)

[Skip examples](#)



Retrieving Data

- Browsing a Class
- Querying From Indexes
- Working Across the Graph

Retrieving Records

- This next section will show using interactive commands and SQL through the console and OrientDB Studio
- Interactive commands
 - LIST CLASSES
 - BROWSE CLASS
 - LOAD RECORD
 - DISPLAY RECORD

BROWSE CLASS

- Displays all records within a named class
- LIST CLASSES retrieves an inventory of all classes within the database

Browse a Class

```
// Class is a Vertex  
BROWSE CLASS Flight
```

#	@RID	@CLASS	FlightFrom	FlightTo	Id	FlightDate	FlightNumber	in_FlewOn	
0	#233:0 Flight CLE	SFO	1	2017-06-10 00:00:00 WN 1252		[#244:0,#245:0]			
1	#233:1 Flight SJC	YVR	5	2018-02-25 00:00:00 DL 1973					
2	#234:0 Flight LHR	ORD	2	2017-10-17 00:00:00 AA 290		[#241:0,#242:0]			
3	#234:1 Flight GVA	CDG	6	2018-01-25 00:00:00 AF 833					
4	#235:0 Flight ICN	SIN	3	2017-12-30 00:00:00 SQ 2		[#243:0]			
5	#235:1 Flight DUB	LHR	8	2017-09-07 00:00:00 EI 1923					
6	#236:0 Flight FRA	GVA	4	2018-01-17 00:00:00 LH 1190					
7	#236:1 Flight SFO	CLE	7	2017-08-22 00:00:00 WN 1253					
8	#236:2 Flight LGW	NAP	9	2017-12-30 00:00:00 BA 8121					
9	#239:0 Flight FRA	ZRH	900	2018-01-17 00:00:00 LH 230					
10	#240:1 Flight FRA	BCN	901	2018-01-17 00:00:00 LH 598					

```
// Class is an Edge  
BROWSE CLASS FlewOn
```

#	@RID	@CLASS	out	in
0	#241:0	FlewOn	#41:125	#234:0
1	#242:0	FlewOn	#43:125	#234:0
2	#243:0	FlewOn	#43:125	#235:0
3	#244:0	FlewOn	#42:125	#233:0
4	#245:0	FlewOn	#41:125	#233:0

```
// Class is a Document  
BROWSE CLASS Booking
```

#	@RID	@CLASS	L_Documentation	ID
0	#321:0	Booking	#457:0	SD1K0W

LOAD RECORD

- Makes it possible to retrieve a specific record from the database
- Requires the specific record id
- It may be necessary to first run a query to determine the record id

Load a Record

```
// Record is a Document
```

```
LOAD RECORD #321:0
```

```
DOCUMENT @class:Booking @rid:#321:0 @version:1
+-----+-----+-----+
| #   | NAME          | VALUE   |
+-----+-----+-----+
| 0   | L_Documentation| #457:0 |
| 1   | ID             | SD1K0W |
+-----+-----+-----+
```

```
// Record is a Vertex
```

```
LOAD RECORD #233:0
```

```
DOCUMENT @class:Flight @rid:#233:0 @version:3
+-----+-----+-----+
| #   | NAME          | VALUE   |
+-----+-----+-----+
| 0   | FlightFrom    | CLE      |
| 1   | FlightTo       | SFO      |
| 2   | Id              | 1        |
| 3   | FlightDate     | 2017-06-10 00:00:00 |
| 4   | FlightNumber   | WN 1252  |
| 5   | in_FlewOn      | [#244:0,#245:0] |
+-----+-----+-----+
```

```
// Record is an Edge
```

```
LOAD RECORD #321:0
```

```
DOCUMENT @class:FlewOn @rid:#241:0 @version:1
```

#	NAME	VALUE
0	out	#41:125
1	in	#234:0

DISPLAY RECORD

- Presents details about the most recently retrieved result set
- Requires a record number
- Typically used after running a query or otherwise retrieve a list of records

EXPORT RECORD

- Returns the current record in an export-ready format
- Requires using the console, fetching the record first
- Current formats supported
 - CSV
 - JSON

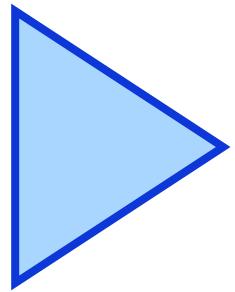
Export a Record in JSON Format

```
SELECT FROM Profiles  
WHERE Name = 'Chico'
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
|# |@RID |@CLASS |Name |Id |Surname|Email |Bio |Gender|in_HasFriend|out_HasFriend|  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
|0 |#48:125|Profiles|Chico|100006|Marx |chico@example.com|Marx Brothers|Male |[#223:202] |[#222:203] |  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
EXPORT RECORD JSON
```

```
{  
  "@type": "d",  
  "@rid": "#48:125",  
  "@version": 3,  
  "@class": "Profiles",  
  "Name": "Chico",  
  "Id": 100006,  
  "Surname": "Marx",  
  "Email": "chico@example.com",  
  "Bio": "Marx Brothers",  
  "Gender": "Male",  
  "in_HasFriend": [  
    "#223:202"  
  ],  
  "out_HasFriend": [  
    "#222:203"  
  ],  
  "@fieldTypes": "Id=l,in_HasFriend=g,out_HasFriend=g"  
}
```



Simple Queries

- SELECT Statement
- Filtering Results
- Sorting and Aggregation

SQL & Data Retrieval

- Just as in standard SQL, the SELECT statement is the primary record retrieval command in OrientDB
- By default, queries are performed against classes

Simple SELECT From Documents

```
// Not necessary to specify property  
// names or * to retrieve all  
// properties
```

```
SELECT FROM Booking
```

#	@RID	@CLASS	FlightNumber	ID
0	#297:0	Booking	AA 2586	NEHJMN
1	#298:0	Booking	SQ 1	AAKEJW

SELECT Specific Properties From Documents

```
SELECT @RID, FlightNumber FROM Booking
```

#	RID	FlightNumber
0	#297:0	AA 2586
1	#298:0	SQ 1

Simple SELECT From Vertices

```
// Not necessary to specify property  
// names or * to retrieve all  
// properties
```

```
SELECT FROM Flight
```

#	@RID	@CLASS	FlightFrom	FlightTo	Id	FlightDate	FlightNumber	in_FlewOn	
0	#233:0	Flight	CLE	SFO	1	2017-06-10 00:00:00	WN 1252	[#244:0, #245:0]	
1	#233:1	Flight	SJC	YVR	5	2018-02-25 00:00:00	DL 1973		
2	#234:0	Flight	LHR	ORD	2	2017-10-17 00:00:00	AA 290	[#241:0, #242:0]	
3	#234:1	Flight	GVA	CDG	6	2018-01-25 00:00:00	AF 833		
4	#235:0	Flight	ICN	SIN	3	2017-12-30 00:00:00	SQ 2	[#243:0]	
5	#235:1	Flight	DUB	LHR	8	2017-09-07 00:00:00	EI 1923		
6	#236:0	Flight	FRA	GVA	4	2018-01-17 00:00:00	LH 1190		
7	#236:1	Flight	SFO	CLE	7	2017-08-22 00:00:00	WN 1253		
8	#236:2	Flight	LGW	NAP	9	2017-12-30 00:00:00	BA 8121		
9	#239:0	Flight	FRA	ZRH	900	2018-01-17 00:00:00	LH 230		
10	#240:1	Flight	FRA	BCN	901	2018-01-17 00:00:00	LH 598		

**SELECT Specific Properties
From Vertices**

```
SELECT FlightDate, FlightFrom,  
FlightTo FROM Flight
```

#	FlightDate	FlightFrom	FlightTo
0	2017-06-10 00:00:00	CLE	SFO
1	2018-02-25 00:00:00	SJC	YVR
2	2017-10-17 00:00:00	LHR	ORD
3	2018-01-25 00:00:00	GVA	CDG
4	2017-12-30 00:00:00	ICN	SIN
5	2017-09-07 00:00:00	DUB	LHR
6	2018-01-17 00:00:00	FRA	GVA
7	2017-08-22 00:00:00	SFO	CLE
8	2017-12-30 00:00:00	LGW	NAP
9	2018-01-17 00:00:00	FRA	ZRH
10	2018-01-17 00:00:00	FRA	BCN

Filtering Results

- The WHERE clause is the most common way of filtering results in SQL
- OrientDB also has numerous record attributes that may be used in filtering

SELECT From Documents Using
WHERE Clause on
Defined Properties

```
SELECT FROM Booking WHERE ID =  
'AAKEJW'
```

#	@RID	@CLASS	FlightNumber	ID
0	#298:0	Booking	SQ 1	AAKEJW

SELECT From Documents Using
WHERE Clause on
Undefined Properties

```
SELECT FROM Booking WHERE ExtraField  
LIKE 'Text provided at runtime%'
```

#	@RID	@CLASS	FlightNumber	ExtraField	ID
0	#299:0	Booking	LH 454	Text provided at runtime without a predefined property 89IUET	89IUET

SELECT From Vertices Using
WHERE Clause on
Defined Properties

```
SELECT FROM Flight WHERE FlightFrom =  
'FRA'
```

#	@RID	@CLASS	FlightFrom	FlightTo	Id	FlightDate	FlightNumber
0	#236:0	Flight	FRA	GVA	4	2018-01-17 00:00:00	LH 1190
1	#239:0	Flight	FRA	ZRH	900	2018-01-17 00:00:00	LH 230
2	#240:1	Flight	FRA	BCN	901	2018-01-17 00:00:00	LH 598

SELECT From Vertices Using
WHERE Clause on
Undefined Properties

```
SELECT FROM Flight WHERE Extra =  
'Runtime-provided data'
```

#	@RID	@CLASS	FlightNumber	FlightDate	Id	Extra	FlightFrom	FlightTo
0	#233:3	Flight	DL 1174	2018-02-09 00:00:00	8765	Runtime-provided data	SJC	JAX

Sorting and Aggregation

- Three major techniques to affect how records are returned
 - ORDER BY
 - GROUP BY
 - Pagination

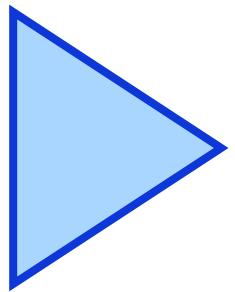
**Retrieve Vertices and Sort
Results Using ORDER BY**

```
SELECT FlightNumber, FlightTo,  
FlightFrom  
FROM Flight ORDER BY FlightNumber
```

#	FlightNumber	FlightTo	FlightFrom
0	AA 290	ORD	LHR
1	AF 833	CDG	GVA
2	BA 8121	NAP	LGW
3	DL 1973	YVR	SJC
4	EI 1923	LHR	DUB
5	LH 1190	GVA	FRA
6	SQ 2	SIN	ICN
7	WN 1252	SFO	CLE
8	WN 1253	CLE	SFO

```
SELECT FlightNumber, FlightTo,  
FlightFrom  
FROM Flight ORDER BY FlightNumber DESC
```

#	FlightNumber	FlightTo	FlightFrom
0	IWN 1253	ICLÉ	ISFO
1	IWN 1252	ISFO	ICLÉ
2	ISQ 2	ISIN	IICN
3	ILH 1190	IGVA	IFRA
4	IEI 1923	ILHR	IDUB
5	IDL 1973	IYVR	ISJC
6	IBA 8121	INAP	ILGW
7	IAF 833	ICDG	IGVA
8	IAA 290	IORD	ILHR



Working Across the Graph

- Specialized Functions
- TRAVERSE
- MATCH

Working Across the Graph

- OrientDB SQL offers several specialized functions for working with graph data
 - `in()`, `inE()`, `inV()`
 - `out()`, `outE()`, `outV()`
 - `both()`, `bothE()`
- These can be used in conjunction with `expand()` to get more information

Incoming

- The `in()` function lists the record ids for adjacent incoming vertices
- The `inE()` function lists the record ids for adjacent incoming edges
- The `inV()` function lists adjacent incoming vertices from an edge

Outgoing

- The `out()` function lists adjacent outgoing vertices
- The `outE()` function lists adjacent outgoing edges
- The `outV()` function lists adjacent outgoing vertices from an edge

Both Directions

- The both () function lists adjacent incoming and outgoing vertices
- The bothE () function lists adjacent incoming and outgoing edges

Get Lists of Incoming and Outgoing Vertices

```
SELECT BOTH() FROM Profiles  
WHERE Name = 'Bud'
```

```
+-----+  
| # | BOTH  
+-----+  
| 0 | [#41:125,#43:125,#43:125,#44:125,#41:125,#233:0,#250:0] |  
+-----+
```

Get Lists of Incoming and Outgoing Edges

```
SELECT BOTHE() FROM Profiles  
WHERE Name = 'Bud'
```

```
+-----+  
| #    | BOTHE  
+-----+  
| 0    | [#217:203,#219:203,#218:202,#220:202,#218:203,#244:0,#258:0] |  
+-----+
```

Get Details About Incoming and Outgoing Vertices

```
SELECT EXPAND(BOTH()) FROM Profiles  
WHERE Name = 'Bud' LIMIT 1
```

#	@RID	@CLASS	out_Submitted	out_HasFriend	out_FlewOn	Name	Id	Surname	Email	Bio	Gender	in HasFriend
1	0	[#41:125]Profiles	[#257:0,#257:1]	[#217:203,#219:202]	[#241:0,#245:0]	Lou	100000	Costello	lou@example.com	Abbott & Costello	Male	[# 218:203]

Expand Incoming and
Outgoing Edges

```
SELECT EXPAND(BOTH( )) FROM Profiles  
WHERE Name = 'Bud'
```

#	@RID	@CLASS	out	in	
0	#217:203	HasFriend	#41:125	#42:125	
1	#219:203	HasFriend	#43:125	#42:125	
2	#218:202	HasFriend	#42:125	#43:125	
3	#220:202	HasFriend	#42:125	#44:125	
4	#218:203	HasFriend	#42:125	#41:125	
5	#244:0	FlewOn	#42:125	#233:0	
6	#258:0	Submitted	#42:125	#250:0	

7 item(s) found. Query executed in 0.133 sec(s).

Get Incoming and Outgoing Vertices for An Edge

```
SELECT BOTHV() FROM Profiles  
WHERE @rid = #217:203
```

#	bOTHV
0	[#42:125, #41:125]

```
1 item(s) found. Query executed in 0.012 sec(s).
```

Expand Incoming and Outgoing Vertices for An Edge

```
SELECT EXPAND(BOTHV()) FROM Profiles  
WHERE @rid = #217:203
```

#	@RID	@CLASS	Name	Id	Gender	Surname	Email	out_FlewOn	out_Submitted	Bio	in_HasFriend	out_HasFrie
10	#42:125	Profiles	Bud	100001	Male	Abbott	bud@example...	[#244:0]	[#258:0]	Abbott & Cos...	[#217:203,#219...]	[#218:202,#220:202,#2...
11	#41:125	Profiles	Lou	100000	Male	Costello	lou@example...	[#241:0,#2...	[#257:0,#2...	Abbott & Cos...	[#218:203]	[#217:203,#219:202]

2 item(s) found. Query executed in 0.007 sec(s).

Querying Across the Graph

- Combining SELECT with other commands makes it easy to navigate graph information
 - Choice of TRAVERSE or MATCH
- This showcases the true power of graph databases

TRAVERSE

- Statement that navigates the graph
 - Works with document data too
- Can run standalone, or in conjunction with SELECT
- Traverse strategy may be specified
 - DEPTH_FIRST
 - BREADTH_FIRST

While TRAVERSE is still an option,
it's wiser to explore using MATCH for graph navigation

Traverse the Graph From a Given Profile

```
SELECT Name, Surname FROM (TRAVERSE
both ("HasFriend")
FROM (SELECT FROM Profiles WHERE Name
= 'Hilary') WHILE $depth < 2)
```

#	Name	Surname
0	Hilary	Brooks
1	Mike	Kelly
2	Lou	Costello
3	Bud	Abbott

```
SELECT Name, Surname FROM (TRAVERSE
both ("HasFriend")
FROM (SELECT FROM Profiles WHERE Name
= 'Hilary') WHILE $depth < 3)
```

#	Name	Surname
0	Hilary	Brooks
1	Mike	Kelly
2	Zeppo	Marx
3	Groucho	Marx
4	Lou	Costello
5	Bud	Abbott

DEPTH_FIRST

- Default strategy
- Using recursion, OrientDB explores each node to its deepest level before moving to the next node
- Depth is configurable

BREADTH_FIRST

- OrientDB explores all nodes on the same level
- Once that's done, it moves on to the next level down
 - And repeats the process of exploring all nodes on that level

TRAVERSE Context Variables

Variable	Purpose
\$depth	Integer indicating nesting depth in tree; first level of tree is 0
\$path	String representation of current position as a sum of the traversed nodes
\$stack	Provides a stack of the traversed nodes
\$history	Provides the entire collection of visited nodes

**View \$path For BREADTH_FIRST
and DEPTH_FIRST Strategies**

```
SELECT $path FROM (TRAVERSE
OUT ('HasFriend')
FROM #42:125 STRATEGY BREADTH_FIRST)
```

```
+-----+
| #   | $path
+-----+
| 0   | (#42:125)
| 1   | (#42:125).out[0] (#43:125)
| 2   | (#42:125).out[1] (#44:125)
| 3   | (#42:125).out[2] (#41:125)
| 4   | (#42:125).out[1] (#44:125).out[0] (#46:125)
| 5   | (#42:125).out[1] (#44:125).out[0] (#46:125).out[0] (#42:126)
| 6   | (#42:125).out[1] (#44:125).out[0] (#46:125).out[2] (#47:125)
| 7   | (#42:125).out[1] (#44:125).out[0] (#46:125).out[2] (#47:125).out[0] (#48:125)
| 8   | (#42:125).out[1] (#44:125).out[0] (#46:125).out[2] (#47:125).out[1] (#41:126)
+-----+
```

```
SELECT $path FROM (TRAVERSE
OUT ('HasFriend')
FROM #42:125 STRATEGY DEPTH_FIRST)
```

```
+-----+
| #   | $path
+-----+
| 0   | (#42:125)
| 1   | (#42:125).out[0] (#43:125)
| 2   | (#42:125).out[1] (#44:125)
| 3   | (#42:125).out[1] (#44:125).out[0] (#46:125)
| 4   | (#42:125).out[1] (#44:125).out[0] (#46:125).out[0] (#42:126)
| 5   | (#42:125).out[1] (#44:125).out[0] (#46:125).out[2] (#47:125)
| 6   | (#42:125).out[1] (#44:125).out[0] (#46:125).out[2] (#47:125).out[0] (#48:125)
| 7   | (#42:125).out[1] (#44:125).out[0] (#46:125).out[2] (#47:125).out[1] (#41:126)
| 8   | (#42:125).out[2] (#41:125)
+-----+
```

MATCH

- OrientDB SQL extension
- Enables declarative queries via pattern matching
- Similar to other graph database query languages
 - Yet still interoperates with SQL

MATCH Arrow Notation

Operator	Notation	Example
out()	-->	-HasFriend->
in()	<--	<-HasFriend-
both()	--	-HasFriend-

MATCH Context Variables

Variable	Purpose
\$match	Provides the current matched record
\$depth	Specifies search depth in conjunction with while statement
\$currentMatch	Provides entire current node in the match

Use MATCH for a Simple
Wildcard Search

```
MATCH {class: Profiles, as: profile,  
where: (Name LIKE 'B%')} RETURN  
profile.Name, profile.Surname LIMIT 10
```

#	profile_Name	profile_Surname
0	Blanche	Hubbard
1	Blanche	Stevenson
2	Bessie	Clarke
3	Brandon	McCormick
4	Bertha	Norman
5	Beatrice	Wagner
6	Bernard	Atkins
7	Brian	Taylor
8	Brian	Lynch
9	Bruce	Henderson

Find All Flights Taken By a Given Profile

```
MATCH {class: Profiles, as: profile,  
where: (Name = 'Bud') } -FlewOn->  
{as:flight}  
RETURN profile.Name,  
flight.FlightNumber
```

#	profile_Name	flight_FlightNumber
0	Bud	WN 1252

Find Names of All Incoming Friends For a Given Profile

```
MATCH {class: Profiles, as: profile,  
where: (Name = 'Bud') }.in('HasFriend')  
{as: friend}  
RETURN profile.Name, friend.Name
```

#	profile_Name	friend_Name
0	Bud	Lou
1	Bud	Sidney

Find All Friends and Friends of Friends For a Given Profile

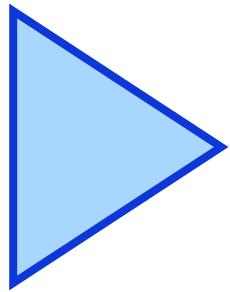
```
MATCH {class: Profiles, as: profile,
where: (Name = 'Hilary') }
-HasFriend- { } -HasFriend-
{as: FriendOfFriend,
where: ($matched.profile != $currentMatch) }
RETURN profile.Name, FriendOfFriend.Name
```

#	profile_Name	FriendOfFriend_Name
0	Hilary	Zeppo
1	Hilary	Groucho
2	Hilary	Bud
3	Hilary	Sidney
4	Hilary	Lou

Find All Credit Card Applications by Friends of a Given Profile

```
MATCH {class: Profiles, as: profile,  
where: (Name = 'Bud')} -HasFriend->  
{as:friend} -Submitted-> {as:submitted}  
RETURN friend.Name, submitted.CardType;
```

#	friend_Name	submitted_CardType
0	Sidney	Wyndham Rewards Visa
1	Lou	American Express Business Platinum
2	Lou	Chase Sapphire Plus
3	Lou	Wyndham Rewards Visa
4	Lou	Shell Preferred
5	Lou	Citibusiness American Airlines



Alternate Ways to Access Data

- Record Identifiers
- Clusters
- Indexes

SQL & Data Retrieval

- Targets other than classes are supported
 - One or more clusters
 - It's necessary to specify the cluster
 - Record id
 - Indexes
 - Needs an 'index:' prefix

Retrieve All Records From a Specific Cluster

```
SELECT FROM CLUSTER:Flight_3
```

#	@RID	@CLASS	FlightFrom	FlightTo	Id	FlightDate	FlightNumber
0	#236:0	Flight	FRA	GVA	4	2018-01-17 00:00:00	LH 1190
1	#236:1	Flight	SFO	CLE	7	2017-08-22 00:00:00	WN 1253

Retrieve Certain Records From a Specific Cluster

```
SELECT FROM CLUSTER:Flight_3  
WHERE FlightFrom = 'FRA'
```

#	@RID	@CLASS	FlightFrom	FlightTo	Id	FlightDate	FlightNumber
0	#236:0	Flight	FRA	GVA	4	2018-01-17 00:00:00	LH 1190

Retrieve a Record Using Record Id

```
SELECT FROM #233:0
```

#	@RID	@CLASS	FlightFrom	FlightTo	Id	FlightDate	FlightNumber	in_FlewOn
0	#233:0	Flight	CLE	SFO	1	2017-06-10 00:00:00	WN 1252	[#244:0, #245:0]

Querying from Indexes

- Indexes are composed of one or more properties
- It's possible to directly query indexes
 - As long as all the information required is present in the index

Retrieve All Data From an Index

```
SELECT FROM INDEX:Flight.id
```

#	key	rid
0	1	#233:0
1	2	#234:0
2	3	#235:0
3	4	#236:0
4	5	#233:1
5	6	#234:1
6	7	#236:1
7	8	#235:1
8	9	#236:2

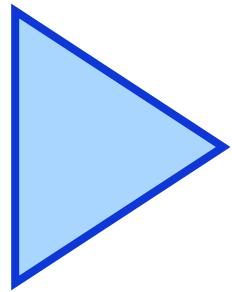
```
9 item(s) found. Query executed in 0.023 sec(s).
```

Retrieve Specific Data From an Index

```
SELECT FROM INDEX:Profiles.Email  
WHERE key = 'bud@example.com'
```

#	key	rid
0	bud@example.com	#42:125

1 item(s) found. Query executed in 0.004 sec(s).



The MATCH Statement

- Drawbacks of `SELECT` for Graph Navigation
- Introducing `MATCH`
- Statement Structure
- Examples

Using SELECT for Graph Queries

SELECT statements typically compose these behaviors:

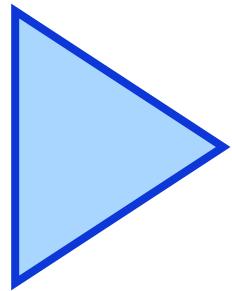
1. Cartesian product (join)
2. Filtering (WHERE clause)
3. Projection (One or more column names)
4. Aggregation (SUM, GROUP BY)
5. Sorting (ORDER BY)

Using SELECT for Graph Queries

- Graph data usually doesn't have the underlying foundations for joins
- Instead, relationships are managed by edges
 - No concept of a many-to-many table
- This can be very challenging to process
- The result is that there's a need for an easily understood graph navigation command

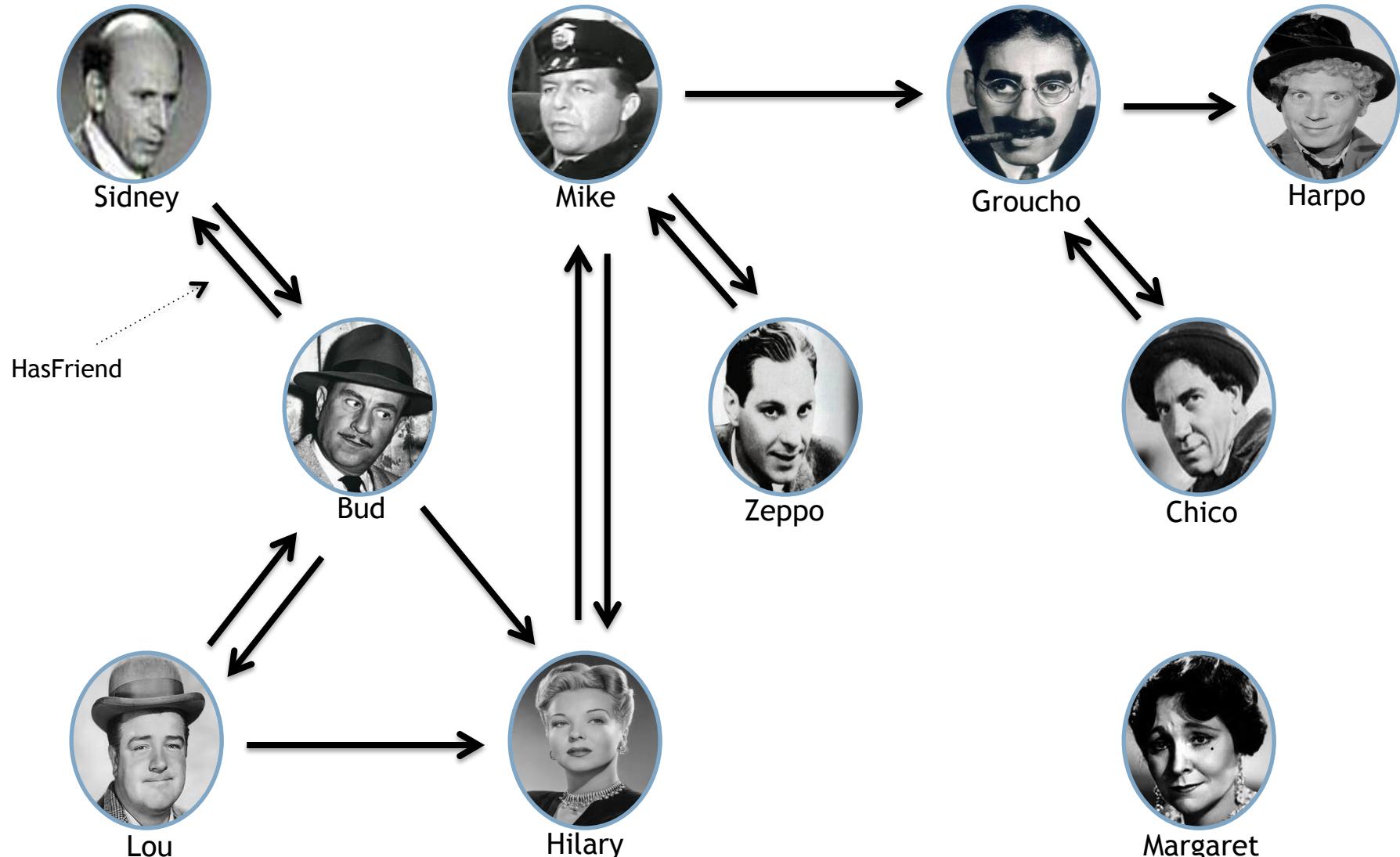
The MATCH Statement

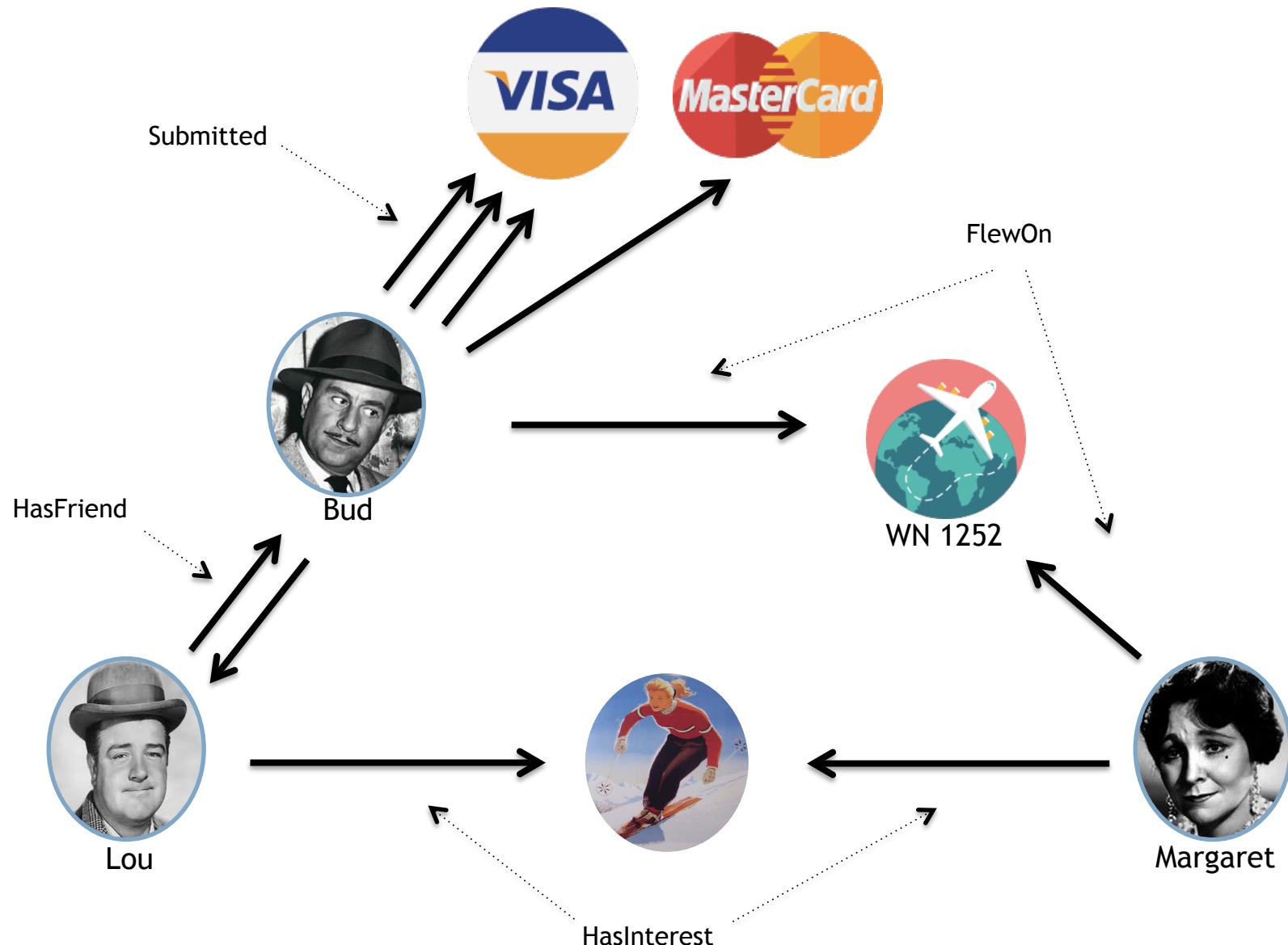
- OrientDB offers the **MATCH statement**
 - Represents vertices and edges as patterns
 - Uses JSON-like syntax
 - Works with OrientDB's multi-model architecture
- Automatically chooses the right starting point
 - Using WHERE clause plus internally-maintained statistics and metrics
- Still requires indexes for optimal performance

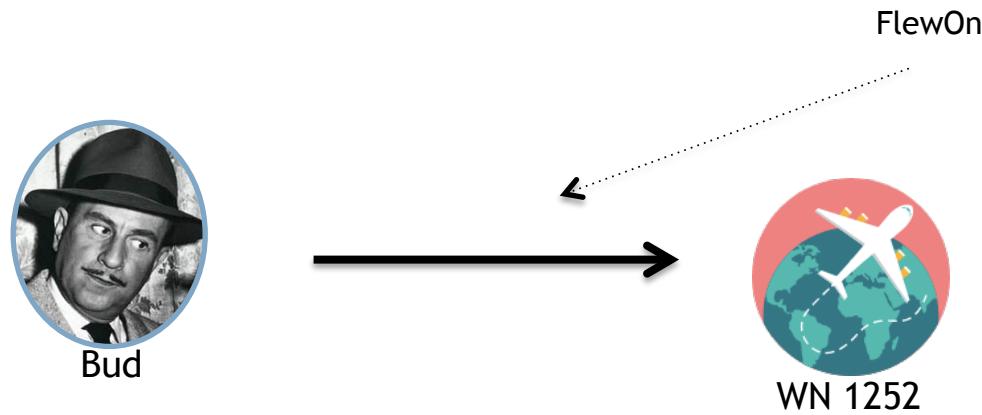


MATCH Statement Structure

- Sample Data Model
- Building a MATCH Statement

The MATCH Statement

The MATCH Statement



```
MATCH {class: Profiles, as: profile,  
where: (Name = 'Bud') }  
-FlewOn-> {as:flight}  
RETURN profile.Name, flight.FlightNumber
```

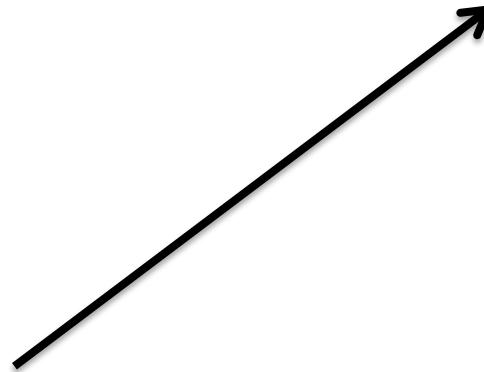
PROPERTIES	
profile_Name	flight_FlightNumber
Bud	WN 1252

```
MATCH {class: Profiles,
```



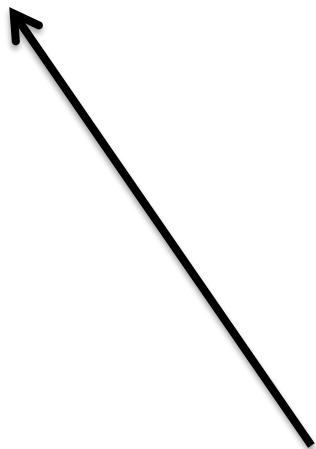
Starting vertex class for the query

```
MATCH {class: Profiles, as: profile,
```



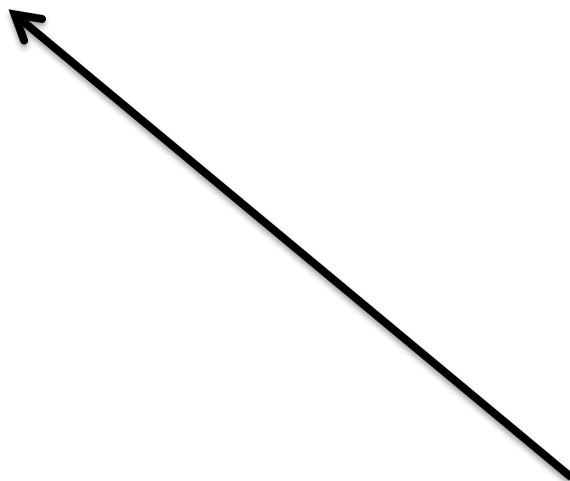
Alias for the class

```
MATCH {class: Profiles, as: profile,  
where: (Name = 'Bud')}
```



Narrowing search to one or more vertices

```
MATCH {class: Profiles, as: profile,  
where: (Name = 'Bud') }  
-FlewOn->
```



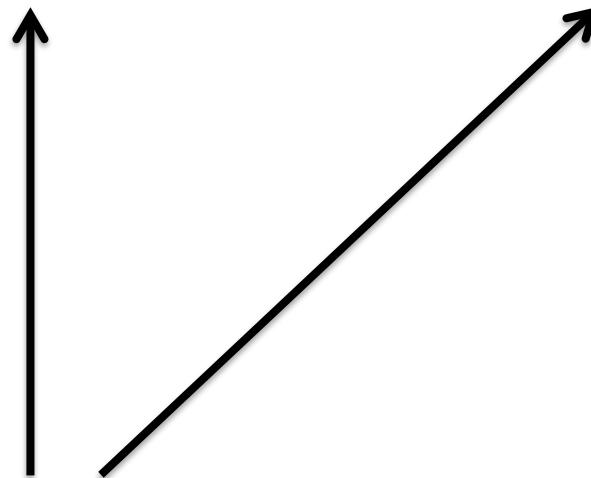
Outgoing edge type (from Profiles)

```
MATCH {class: Profiles, as: profile,  
where: (Name = 'Bud') }  
-FlewOn-> {as:flight}
```



Vertices that have an incoming
FlewOn edge from Bud

```
MATCH {class: Profiles, as: profile,  
where: (Name = 'Bud') }  
-FlewOn-> {as:flight}  
RETURN profile.Name, flight.FlightNumber
```



Properties from returned vertices

Incoming Edge Functions

Function	Purpose
in()	Return the record ids for each vertex that has an edge connected to this vertex
inE()	Return the record ids for each edge connected to this vertex
inV()	Return the record ids for each vertex connected to this edge

Outgoing Edge Functions

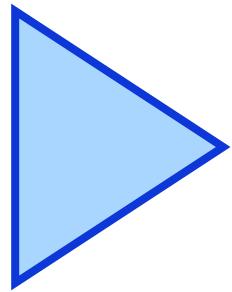
Function	Purpose
<code>out()</code>	Return the record ids for each vertex that this vertex has an edge connected to
<code>outE()</code>	Return the record ids for each edge that this vertex has an edge connected to
<code>outV()</code>	Return the record ids for each vertex that this edge connects to

MATCH Arrow Notation

Operator	Notation	Example
out()	-->	-HasFriend->
in()	<--	<-HasFriend-
both()	--	-HasFriend-

MATCH Result Variables

Variable	Purpose
\$patterns	Return relevant record ids
\$paths	Return all paths
\$elements	Return an expansion and pivoting of \$patterns
\$pathelements	Return an expansion and pivoting of \$paths



MATCH Statement Examples

Return All Profiles

```
MATCH
{class: Profiles, as: profile}
RETURN profile LIMIT 5
```

#	profile
0	#45:66
1	#48:85
2	#41:107
3	#43:7
4	#46:26

**Return All Properties
Within All Profiles**

```
MATCH
{class: Profiles, as: profile}
RETURN $elements LIMIT 5
```

#	@RID	@CLASS	Id	Gender	Name	Surname	in_HasPro	Birthday	Email	in_HasFri	out_HasFr	Bio
0	#45:66	Profiles	533	Male	Connor	Walton		1963-0...	maku@l...	[#219:...]	[#217:...]	Fiwtad...
1	#48:85	Profiles	688	Male	Carolyn	Schultz	[#187:35]	1953-0...	no@hal...	[#223:...]	[#217:...]	Rilgis...
2	#41:107	Profiles	857	Female	Bernard	Atkins	[#190:43]	1968-1...	ufocus...	[#224:82]	[#219:...]	Ziw fi...
3	#43:7	Profiles	59	Male	Eleanor	Lloyd	[#187:3]	1997-0...	ara@mo...	[#218:...]		Izokeg...
4	#46:26	Profiles	214	Male	Ina	Rodriquez	[#190:12]	1980-0...	afaev@...			Siunae...

Find a Specific Profiles Instance

```
MATCH  
{class: Profiles, as: profile,  
 where: (Name = 'Bud') }  
RETURN profile.Name, profile.Surname
```

#	profile_Name	profile_Surname
0	Bud	Abbott

Find a Specific Profile Using Multiple AND Criteria

```
MATCH
{class: Profiles, as: profile,
 where: (Name = 'Bud' AND Surname =
'Abbott') }
RETURN profile.Name, profile.Surname
```

#	profile_Name	profile_Surname
0	Bud	Abbott

Find Profiles Using Multiple OR Criteria

```
MATCH  
{class: Profiles, as: profile, where:  
(Name = 'Bud' OR Surname = 'Abbott') }  
RETURN profile.Name, profile.Surname
```

#	profile_Name	profile_Surname
0	Bud	Abbott
1	Inez	Abbott
2	Daniel	Abbott
3	Jennie	Abbott

Find Profiles Using Not Equals Criteria

```
MATCH
{class: Profiles, as: profile, where:
(Name <> 'Bud' ) }
RETURN profile.Name, profile.Surname
LIMIT 5
```

#	profile_Name	profile_Surname
0	Connor	Walton
1	Carolyn	Schultz
2	Bernard	Atkins
3	Eleanor	Lloyd
4	Ina	Rodriquez

Combine UPDATE with MATCH

```
UPDATE
```

```
(MATCH {class: Profiles, as:  
profile, where: (Name = 'Robert') }  
RETURN $elements)
```

```
SET Bio = 'New bio'
```

```
Updated record(s) '2' in 0.094000 sec(s).
```

Combine DELETE with MATCH

```
DELETE VERTEX FROM (
MATCH {class: Profiles, as: profile,
where: (Name = 'ToDelete') }
RETURN $elements)
```

```
Delete record(s) '1' in 0.028000 sec(s).
```

Find Flights Using an Outgoing Edge from Profiles

```
MATCH
{class: Profiles, as: profile, where:
(Name = 'Bud') }
-FlewOn-> {as:flight}
RETURN profile.Name, flight.FlightNumber
```

#	profile_Name	flight_FlightNumber
0	Bud	IWN 1252

Find Profiles Using an Incoming Edge from Flights

```
MATCH
{class: Flight, as: flight, where:
(FlightNumber = 'WN 1252' )}
<-FlewOn- {as:profile}
RETURN profile.Name, flight.FlightNumber
```

#	profile_Name	flight_FlightNumber
0	Bud	WN 1252
1	Margaret	WN 1252

Find Profiles with
More than 2 Interests

```
MATCH
{class: Profiles, as: profile, where:
(out('HasInterest').size() > 2) }
RETURN profile.Name
```

#	profile_Name
0	Lou
1	Margaret

Find Profiles and Interest
for Profiles with More than 2
Interests

```
MATCH
{class: Profiles, as: profile, where:
(out('HasInterest').size() > 2)}
-HasInterest-> {as: interest}
RETURN profile.Name, interest.Name
```

#	profile_Name	interest_Name
0	Lou	Basketball
1	Lou	Cooking
2	Lou	Golf
3	Lou	Skiing
4	Margaret	Skiing
5	Margaret	Tennis
6	Margaret	Hiking

Find Closest Friends

```
MATCH
{class: Profiles, as: profile, where:
(Name = 'Bud') }
-HasFriend- { }
-HasFriend- {as: FriendOfFriend, while:
($depth < 0) }
RETURN profile.Name, FriendOfFriend.Name
```

#	profile_Name	FriendOfFriend_Name
0	Bud	Sidney
1	Bud	Hilary
2	Bud	Lou

Find Friends of Friends to a
Depth of 1

```
MATCH
{class: Profiles, as: profile, where:
(Name = 'Bud') }
-HasFriend- { }
-HasFriend- {as: FriendOfFriend, while:
($depth < 1) }
RETURN profile.Name, FriendOfFriend.Name
```

#	profile_Name	FriendOfFriend_Name
0	Bud	Sidney
1	Bud	Bud
2	Bud	Hilary
3	Bud	Mike
4	Bud	Lou

Find Friends of Friends to a
Depth of 2

```
MATCH
{class: Profiles, as: profile, where:
(Name = 'Bud' ) }
-HasFriend- { }
-HasFriend- {as: FriendOfFriend, while:
($depth < 2) }
RETURN profile.Name, FriendOfFriend.Name
```

#	profile_Name	FriendOfFriend_Name
0	Bud	Sidney
1	Bud	Bud
2	Bud	Hilary
3	Bud	Lou
4	Bud	Mike
5	Bud	Zeppo
6	Bud	Groucho

Find Friends of Friends to a
Depth of 6

```
MATCH
{class: Profiles, as: profile, where:
(Name = 'Bud' ) }
-HasFriend- { }
-HasFriend- {as: FriendOfFriend, while:
($depth < 6) }
RETURN profile.Name, FriendOfFriend.Name
```

#	profile_Name	FriendOfFriend_Name
0	Bud	Sidney
1	Bud	Bud
2	Bud	Hilary
3	Bud	Lou
4	Bud	Mike
5	Bud	Zeppo
6	Bud	Groucho
7	Bud	Chico
8	Bud	Harpo

**Find Friends of Friends to a Depth of 6,
Remove Self From Results**

```
MATCH
{class: Profiles, as: profile, where:
(Name = 'Bud') }
-HasFriend- { }
-HasFriend- {as: FriendOfFriend, while:
($depth < 6), where:
($matched.profile != $currentMatch) }
RETURN profile.Name, FriendOfFriend.Name
```

#	profile_Name	FriendOfFriend_Name
0	Bud	Sidney
1	Bud	Hilary
2	Bud	Lou
3	Bud	Mike
4	Bud	Zeppo
5	Bud	Groucho
6	Bud	Chico
7	Bud	Harpo

Find Common Friends for Two Profiles

```
MATCH
{class: Profiles, where: (Name = 'Lou'
AND Surname = 'Costello') }
-HasFriend-
{as: common}.both('HasFriend') {class:
Profiles, where: (Name = 'Bud') }
RETURN common.Name, common.Surname
```

#	commonfriend_Name	commonfriend_Surname
0	Hilary	Brooks

Find Profiles With No Friends
that Have Applied for a Credit Card

```
MATCH
{class: Profiles, as: profile, where:
((both('HasFriend').size() = 0)) }
-Submitted-> {as:submitted}
RETURN profile.Name, submitted.CardType
```

#	profile_Name	submitted_CardType
0	Margaret	Citibusiness American Airlines

Find and expand() Profiles
with an Interest in Skiing

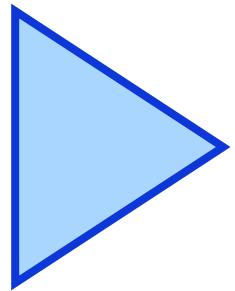
```
SELECT expand(profile) FROM
(MATCH {class: Profiles, as: profile}
-HasInterest-> {as: interest, where:
(Name = 'Skiing')} )
RETURN profile)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|#   |@RID    |@CLASS   |Id      |Gender|Name      |Surname |out_Flew0|in_HasFri|Bio       |out_HasFr|
Email   |out_HasInterest|out_Submitted   |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|0   |#41:125|Profiles|100000|Male   |Lou     |Costello|[#241:0]  |[#218:...]|Abbott...|[#217:...]
lou@ex...|[#473:0,#4...|[#257:0,#257:1,#25...
|1   |#43:126|Profiles|100009|Female|Margaret|Dumont  |[#241:1]  |          |Marx B...|
margar...|[#477:0,#4...|[#258:1,#261:0]   |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

Search on Edge Properties

```
MATCH
{class: Profiles, as: profile, where:
(Name = 'Bud') }
-Submitted-> {as:submitted}.inE() {as:
subE, where: (Bonus = 25000)}.outV()
RETURN profile.Name,
submitted.CardType, subE.Bonus
```

#	profile_Name	submitted_CardType	subE_Bonus
0	Bud	Chase Sapphire Plus	25000
1	Bud	Citibusiness American Airlines	25000



Updating and Deleting Data

- Changing Vertex & Edge Content
- Removing Records

Updating Records

- OrientDB SQL offers a traditional UPDATE command, with several added capabilities
 - Creating new fields in schema-less records
 - Collection-oriented features
 - Map-oriented features
 - Providing for concurrency
 - Merging content from JSON
 - Inserting a new record if it doesn't already exist (UPSERT)
 - Setting a timeout

Update Schema-Less Record with New Fields

```
CREATE CLASS Booking;
CREATE PROPERTY Booking.ID STRING;
INSERT INTO Booking SET ID = 'ABC123';
SELECT FROM Booking;
```

#	@RID	@CLASS	ID
0	#321:0	Booking	ABC123

```
UPDATE Booking SET ID = '123ABC',
NewField = 'Added' WHERE ID =
'ABC123';
SELECT FROM Booking;
```

#	@RID	@CLASS	ID	NewField
0	#321:0	Booking	123ABC	Added

Update Using UPSERT

```
// Note that there is currently no
// record with an ID of 'LMN876'
UPDATE Booking SET ID = 'XYZ987'
UPSERT WHERE ID = 'LMN876';

SELECT * FROM Booking;
```

#	@RID	@CLASS	ID
0	#322:0	Booking	XYZ987

Update with JSON Using MERGE

```
CREATE CLASS Booking;
CREATE PROPERTY Booking.ID STRING;
INSERT INTO Booking SET ID = 'ABC123';
SELECT FROM Booking;
```

#	@RID	@CLASS	ID
0	#297:0	Booking	ABC123

```
UPDATE 297:0 MERGE {"Seat": "Window",
"Baggage_fee": 25};
```

```
SELECT FROM Booking;
```

#	@RID	@CLASS	ID	Seat	Baggage_fee
0	#297:0	Booking	ABC123	Window	25

Update With TIMEOUT Specified

```
UPDATE Booking SET ID = 'TMA432' WHERE  
ID = 'XYZ987' TIMEOUT 1;
```

```
Error: com.orientechnologies.common.concur.OTimeoutException:  
Command execution timeout exceed (1ms)
```

```
UPDATE Booking SET ID = 'TMA432' WHERE  
ID = 'XYZ987' TIMEOUT 100;
```

```
Updated record(s) '1' in 0.010000 sec(s).
```

MOVE VERTEX

- Enables moving a vertex from one class to another
- Destination cluster can be specified
 - Useful for distributed computing
- Will update edges, but not document links

Deleting Records

- The choice of command to delete information will depend on what's being deleted
- Documents = DELETE
- Vertices = DELETE VERTEX
- Edges = DELETE EDGE

Deleting Records

- Other considerations
 - Deleting records from single clusters
 - Deleting records from multiple clusters
 - Removing properties from a class
- Classes may also be dropped
 - Data consistency safeguards may be overridden with UNSAFE

DELETE VERTEX

- Offers standard SQL delete capabilities
- Also provides graph extensions
 - Such as deleting vertices based on edges

DELETE EDGE

- Edges to be removed can be identified by
 - Record ID
 - From/to relationship
- Filtering (via WHERE clause) can be specified

Delete One or More Documents
Using WHERE Clause Or Record Id

```
DELETE FROM Booking WHERE ID =  
'NTV987'
```

```
Delete record(s) '1' in 0.064000 sec(s).
```

```
DELETE FROM Booking WHERE @rid = 324:0
```

```
Delete record(s) '1' in 0.012000 sec(s).
```

```
DELETE FROM Booking WHERE ID =  
'QSN822' OR ID = 'NTV987'
```

```
Delete record(s) '2' in 0.009000 sec(s).
```

```
DELETE FROM Booking WHERE @rid IN  
[322:0, 325:0]
```

```
Delete record(s) '2' in 0.010000 sec(s).
```

Delete One or More Vertices Using
WHERE Clause Or Record Id

```
DELETE VERTEX Flight WHERE Id = '999'  
Delete record(s) '1' in 0.064000 sec(s).  
  
DELETE VERTEX 240:0  
Delete record(s) '1' in 0.012000 sec(s).  
  
DELETE VERTEX Flight WHERE Id = '999'  
OR Id = '1000'  
Delete record(s) '2' in 0.009000 sec(s).  
  
DELETE VERTEX Flight WHERE @rid IN  
[235:2, 236:3]  
Delete record(s) '2' in 0.010000 sec(s).
```

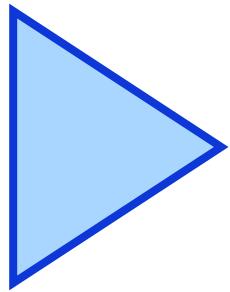
Delete One or More Edges Using
WHERE Clause Or Record Id

```
DELETE EDGE FROM (SELECT FROM Profiles  
WHERE Name = 'Mike' AND Surname =  
'Kelly') TO (SELECT FROM Profiles  
WHERE Name = 'Groucho' AND Surname =  
'Marx')
```

```
Delete record(s) '1' in 0.102000 sec(s).
```

```
// OrientDB will delete as many edges  
from the list as it can find  
DELETE EDGE HasFriend WHERE @rid IN  
[237:0, 238:1]
```

```
Delete record(s) '1' in 0.029000 sec(s).
```



Functions, Methods, and Hooks

- Database Functions
- User-defined Functions
- Methods
- Invoking Functions

Functions

- Functions can help simplify and streamline how users interact with OrientDB
- OrientDB bundles a broad set of helpful functions
 - Graph
 - Math
 - Collections
 - Miscellaneous

Functions

- Users and developers are free to create their own functions
- Lets OrientDB run JavaScript on the server
 - Function code stored in database itself

Functions

- Similar to stored procedures on other platforms
 - But uses standard JavaScript or SQL
 - Ruby, Scala, Java, and other languages are on the product roadmap
- May be run via SQL, REST, Java, console, or OrientDB Studio
- Functions may invoke other functions

Functions

- Capable of receiving parameters, return values
- Support recursion
- Automatically map parameters by position and name
- Plugins can inject new objects to be used by functions

Graph Navigation Functions

Graph Editor ?

Nodes 4 Edges 4 ⏪ ⏹

```
1 SELECT expand(shortestPath(42:125,47:125))
```

Keyboard icon

Legend:

- Profiles (Red circle)
- Interests (Blue circle)
- HasFriend (Black arrow)
- HasInterest (Purple arrow)

```
graph LR; Groucho((Groucho)) -- HasInterest --> Golf((Golf)); Lou((Lou)) -- HasInterest --> Golf; Bud((Bud)) -- HasFriend --> Lou; Bud -- HasFriend --> Groucho;
```

Graph Editor

Nodes 4 Edges 4

```
1 SELECT expand(shortestPath(42:125, 47:125, 'BOTH', 'HasFriend'))
```

Profiles HasFriend

```
graph LR; Bud -- HasFriend --> Hilary; Hilary -- HasFriend --> Mike; Mike -- HasFriend --> Groucho
```

Methods

- Similar in concept to functions
 - Invoke, process information, return something
- Both can accept parameters
- Major difference is that methods are directly launched from a value

Methods

- Grouped by category:
 - Conversions
 - String manipulation
 - Collections
 - Miscellaneous

String Manipulation Methods

```
SELECT FlightFrom,
FlightFrom.toLowerCase(),
FlightFrom.prefix('Departure Airport:'),
' ', FlightFrom.append(' available')
FROM Flight
```

#	FlightFrom FlightFrom2 FlightFrom23	FlightFrom24	
0	CLE	cle	Departure Airport: CLE CLE available
1	SJC	sjc	Departure Airport: SJC SJC available
2	LHR	lhr	Departure Airport: LHR LHR available
3	GVA	gva	Departure Airport: GVA GVA available
4	ICN	icn	Departure Airport: ICN ICN available
5	DUB	dub	Departure Airport: DUB DUB available
6	FRA	fra	Departure Airport: FRA FRA available
7	SFO	sfo	Departure Airport: SFO SFO available
8	LGW	lgw	Departure Airport: LGW LGW available
9	FRA	fra	Departure Airport: FRA FRA available
10	FRA	fra	Departure Airport: FRA FRA available

```
SELECT Name, Surname, Email FROM  
Profiles WHERE Name = 'Bud' AND  
Surname.left(3) = 'Abb'
```

#	Name	Surname	Email
0	Bud	Abbott	bud@example.com

```
SELECT Email.replace('.com', '.org')  
FROM Profiles WHERE Name = 'Bud'
```

#	Email
0	bud@example.org

type() & javaType() Methods

```
SELECT @rid, @rid.type(), Name,  
Name.type(), Id, Id.type() FROM  
Profiles WHERE Name = 'Bud'
```

#	rid	rid2	Name	Name2	Id	Id2
0	#42:125	LINK	Bud	STRING	100001	LONG

```
SELECT @rid, @rid.javaType(), Name,  
Name.javaType(), Id, Id.javaType()  
FROM Profiles WHERE Name = 'Bud'
```

#	rid	rid2	Name	Name2	Id	Id2
0	#42:125	com.orientechnologies.orient.core.id.ORecordId	Bud	java.lang.String	100001	java.lang.Long

toJSON () Method

```
CREATE CLASS Booking;  
CREATE PROPERTY Booking.ID STRING;  
CREATE PROPERTY Booking.Seat STRING;  
CREATE PROPERTY Booking.Baggage_fee  
INTEGER;
```

```
INSERT INTO Booking CONTENT  
{ "ID": "SMCJWS", "Seat": "Window",  
"Baggage_fee": 35 }
```

```
SELECT @this.toJSON() FROM Booking;
```

```
+---+-----+  
| # | this |  
+---+-----+  
| 0 | { "@type": "d", "@rid": "#297:0", "@version": 1, "@class": "Booking", "ID": "SMCJWS", "Seat": "Window", "Bag... |  
+---+-----+
```

Hooks

- Event-based, server-side mechanisms
- Similar to triggers in RDBMS
- Meant to take action on events related to classes, records, databases, transactions

Hooks

- Common usages
 - Implement a custom cascade deletion algorithm
 - Enforce constraints
- Two types of hooks
 - Dynamic
 - Native (Java)

Hooks: Dynamic

- Capable of being set at runtime
 - For a given class or a given record
- Invokes OrientDB functions or Java static methods
- Current function languages:
 - SQL
 - JavaScript
 - Groovy
 - Java

Hooks: Dynamic

Event	Behavior
onBeforeCreate	Called before a new record is created
onAfterCreate	Called after a new record is created
onBeforeRead	Called before a record is read
onAfterRead	Called after a record is read
onBeforeUpdate	Called before a record is updated
onAfterUpdate	Called after a record is updated
onBeforeDelete	Called before a record is deleted
onAfterDelete	Called after a record is deleted

Hooks: Native

- Dynamic hooks are more flexible
 - Native (Java) hooks are faster to develop
- Three basic ways to implement
 1. Create a class that implements the `ORecordHook` interface
 2. Create a class that extends the `ORecordHookAbstract` class
 3. Create a class that extends the `ODocumentHookAbstract` class