OrientDB
Multi-Model Database

Live Global Class

July 2017

Multi-Model Class Definition

In this collection of exercises, you'll gain hands-on experience creating classes, extending them, and establishing relationships among them.

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

```
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 a 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"}}'
```

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');
```

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"}
```

Create a vertex using OrientDB SQL

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

Create a vertex instance

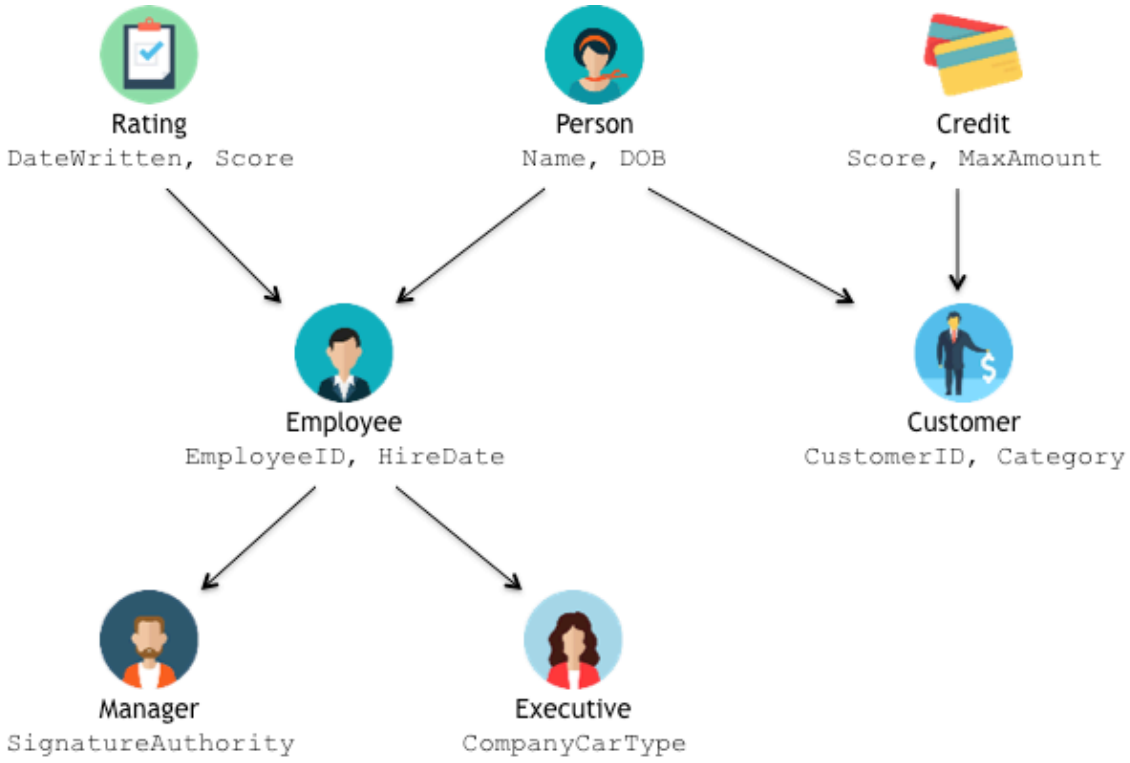
```
CREATE VERTEX Flight SET Id = 1000,
FlightNumber = 'EI 1923', FlightDate =
'2017-09-07', FlightFrom = 'DUB',
FlightTo = 'LHR'
```

Create a vertex using JSON content

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

Create a document class hierarchy

Using the following diagram as a guide, create the classes and properties necessary to implement this class hierarchy (first row are abstract classes). Don't extend base class V at this time; that will come later.



Add data to the class hierarchy

Spend a few minutes creating some meaningful data to populate the class hierarchy. You can use your own ideas, or create records using what's in the tables below. As you build out your data, try running some simple queries that demonstrate the class hierarchy (hint: query the abstract classes, followed by those that extend them).

Employee

Name	DOB	HireDate
Sidney Falco	1957-10-20	2010-06-10
Carl Showalter	1965-11-01	2012-03-27
Bill Gannon	1955-08-18	2016-06-07
Alfred Bellows	1971-09-07	2000-05-02
Dave Crabtree	1981-09-12	2017-11-05

Manager

Name	DOB	HireDate	SignatureAuthority
Larry Fine	1975-07-17	2009-02-25	20000
Wilbur Post	1967-12-25	1999-08-11	40000

Executive

Name	DOB	HireDate	CompanyCarType
JJ Hunsecker	1952-12-30	2005-01-30	Porsche

Customer

Name	DOB	Score	MaxAmount
Wade Gustafson	1930-09-21	802	19999
Audrey Horne	1990-05-17	678	99
Rust Cohle	1968-07-04	777	4999
Frank Pembleton	1969-01-03	765	599

Extend document classes to vertices

It's very easy to augment existing classes to take advantage of graph capabilities: all that's necessary is to alter the class and set its superclass to V.

For each of the classes you just created and populated, use the following command (don't include the < or > symbols):

```
ALTER CLASS <class name> SUPERCLASS V
```

After you have finished, view the class descriptions in the OrientDB Studio's schema section. They should all now have a superclass of V.

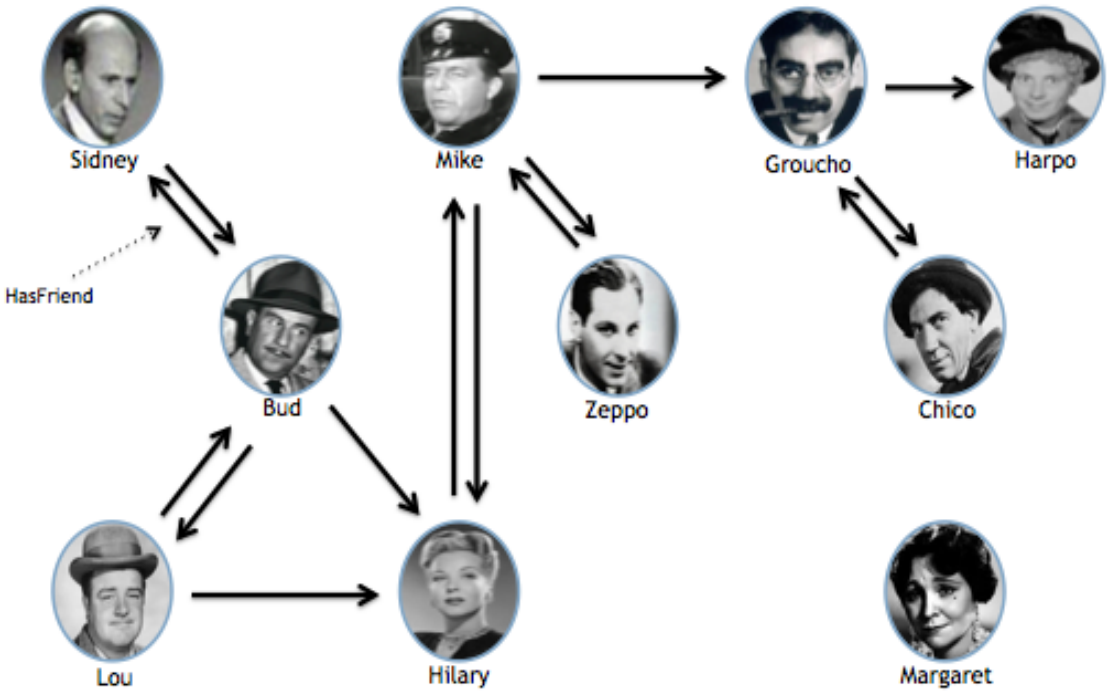
Create edges between vertices

Now that you've converted your documents to vertices, the next step is to establish relationships among them. Use the `CREATE EDGE` statement to accomplish this task.

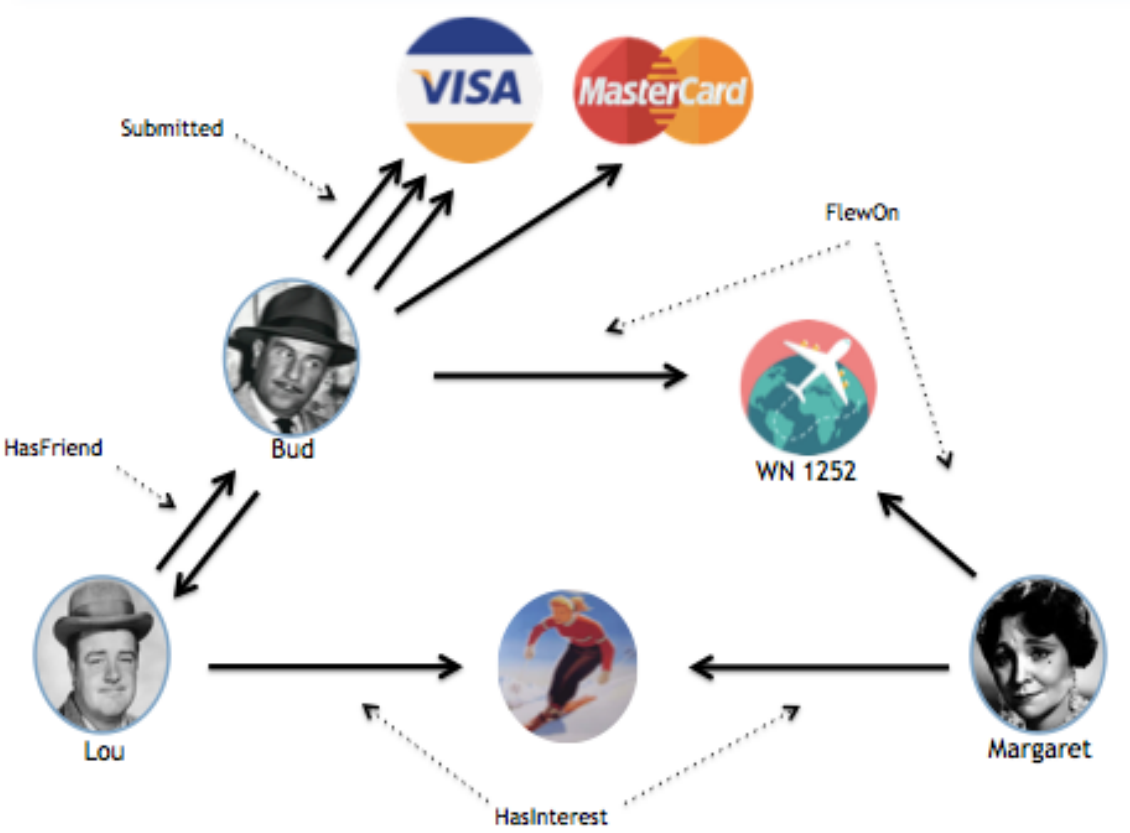
You may use record ids or SQL statements; your instructor will demonstrate both techniques.

MATCH Statement Exercises

The MATCH statement offers unparalleled capabilities for easily navigating your graph. In this section, you'll carry out a series of hands-on exercises to learn more about MATCH. To help you visualize the data, here's a snapshot of a collection of Profiles that are linked by the HasFriend edge:



And here are more vertices and edges. You'll see examples of these relationships in the coming exercises, so be sure to refer back to this diagram to see how everything is linked together.



Return all `Profiles` (limit of 5)

```
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 `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         |WN 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 `Interests` 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 2 `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_FlewO|in_HasFri|Bio      |out_HasFr|
Email    |out_HasIntere|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     |
+----+------------+------------------------------+----------+
```

Java API Labs

In the following section, you'll write code that uses the OrientDB Multi-Model API to perform numerous types of activity with your database.

Establish a connection using connection pool

Use the code below to establish a connection via a connection pool. Be sure to replace the IP address, database name, user name, and password with values provided by your instructor.

```
// Establish a connection using a connection pool
import com.orientechnologies.orient.core.db.ODatabasePool;
import com.orientechnologies.orient.core.db.OrientDB;
import com.orientechnologies.orient.core.db.OrientDBConfig;
import com.orientechnologies.orient.core.db.document.ODatabaseDocument;

public class Connection {

    public static void main(String[] args) throws Exception
    {
        try
        {
            Connection conn = new Connection();
        }
        catch(Exception ex)
        {
            System.out.println("Main error: " + ex);
        }
    }

    private Connection()
    {
        OrientDB orientDB = new OrientDB("remote:<ipaddress>","<username>","<password>", OrientDBConfig.defaultConfig());
        ODatabasePool pool = new ODatabasePool(orientDB,"<database name>","<username>","<password>");
        try (ODatabaseDocument db = pool.acquire()) {
            System.out.println("Connected database name = " + db.getName().toString());
            System.out.println("User = " + db.getUser().toString());
        }
        catch(Exception ex)
        {
            System.out.println("Connection error: " + ex);
        }
        pool.close();
        orientDB.close();
    }
}
```

Create two simple documents, establish a link between them

Use the code below to create two simple documents in your database, and then establish a link between them. Be sure to replace the IP address, database name, user name, and password with values provided by your instructor.

```
// Create two simple documents, establish a link between them
import com.orientechnologies.orient.core.db.ODatabasePool;
import com.orientechnologies.orient.core.db.OrientDB;
import com.orientechnologies.orient.core.db.OrientDBConfig;
import com.orientechnologies.orient.core.db.document.ODatabaseDocument;
import com.orientechnologies.orient.core.record.OElement;

public class CreateDocument {

    public static void main(String[] args) throws Exception
    {
        try
        {
            CreateDocument createdoc = new CreateDocument();
        }
        catch(Exception ex)
        {
            System.out.println("Main error: " + ex);
        }
    }

    private CreateDocument()
    {
        OrientDB orientDB = new OrientDB("remote:<ipaddress>","<username>","<password>", OrientDBConfig.defaultConfig());
        ODatabasePool pool = new ODatabasePool(orientDB,"<database name>","<username>","<password>");
        try (ODatabaseDocument db = pool.acquire()) {
            OElement e1 = db.newInstance("Airline");
            e1.setProperty("Name", "United");
            e1.save();

            OElement e2 = db.newInstance("Airline");
            e2.setProperty("Name", "Skywest");
            e2.save();

            e1.setProperty("Affiliate", e2);
            e1.save();

        }
        catch(Exception ex)
        {
            System.out.println("Connection error: " + ex);
        }
        pool.close();
        orientDB.close();
    }
}
```

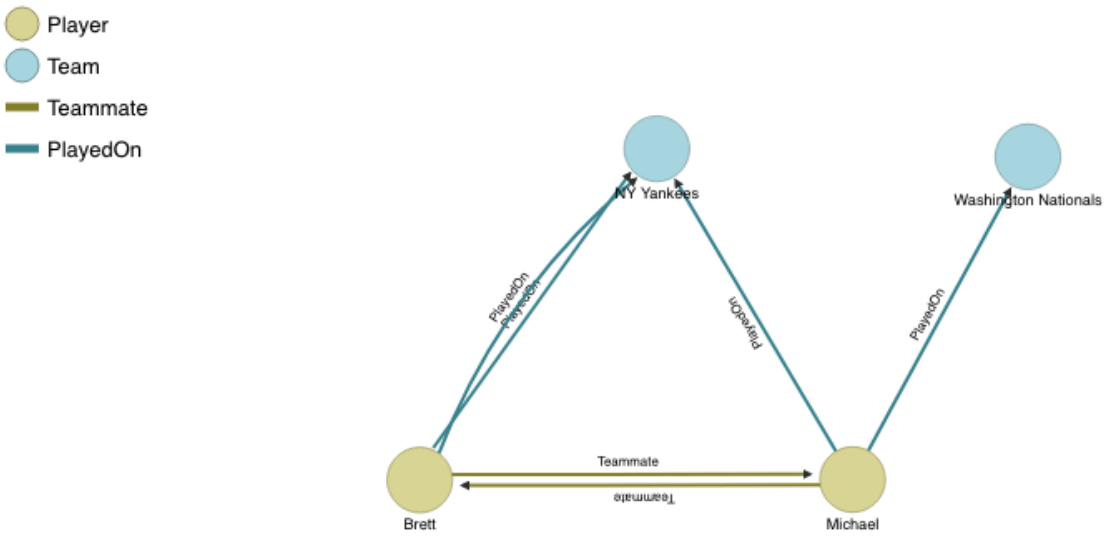
Create vertices and edges

Use the code below to create several simple vertices in your database, and then establish relationships among them using edges. Be sure to replace the IP address, database name, user name, and password with values provided by your instructor.


```java
// Create vertices and edges
import com.orientechnologies.orient.core.db.ODatabasePool;
import com.orientechnologies.orient.core.db.OrientDB;
import com.orientechnologies.orient.core.db.OrientDBConfig;
import com.orientechnologies.orient.core.db.document.ODatabaseDocument;
import com.orientechnologies.orient.core.record.*;

public class Create {

    public static void main(String[] args) throws Exception
    {
        try
        {
            Create cr = new Create();
        }
        catch(Exception ex)
        {
            System.out.println("Main error: " + ex);
        }
    }

    private Create()
    {
        OrientDB orientDB = new OrientDB("remote:<ipaddress>","<username>","<password>", OrientDBConfig.defaultConfig());
        ODatabasePool pool = new ODatabasePool(orientDB,"<database name>","<username>","<password>");
        try (ODatabaseDocument db = pool.acquire()) {
            // Create first vertex of type 'Player'
            OVertex v = db.newVertex("Player");
            v.setProperty("Name", "Brett");
            v.setProperty("Age", 27);
            v.save();

            // Samples of what's available from newly-created vertex
            System.out.println(v.toString());
            System.out.println(v.toJSON(null));
            System.out.println(v.getVersion());
            System.out.println(v.getIdentity().toString());
            System.out.println(v.getSchemaType());
            System.out.println(v.getDatabase().toString());

            // Create second vertex of type 'Player'
            OVertex v1 = db.newVertex("Player");
            v1.setProperty("Name", "Michael");
            v1.setProperty("Age", 23);
            v1.save();

            // Create an edge from Brett to Michael via the vertex
            v.addEdge(v1, "Teammate");
            v.save();

            // Create an edge from Brett to Michael via the vertex
            v1.addEdge(v, "Teammate");
            v1.save();

            // Create a new type of vertex 'Team'
            OVertex v100 = db.newVertex("Team");
            v100.setProperty("Name", "NY Yankees");
            v100.save();

            // Create a second 'Team' vertex
            OVertex v101 = db.newVertex("Team");
            v101.setProperty("Name", "Washington Nationals");
            v101.save();

            // Create an edge from Brett to NY Yankees via Edge
            OEdge e = db.newEdge(v, v100, "PlayedOn");
            e.setProperty("Year", "2017");
            e.save();

            // Create an edge from Michael to NY Yankees via Edge, set edge properties
            OEdge e1 = db.newEdge(v1, v100, "PlayedOn");
            e1.setProperty("Year", "2017");
            e1.save();
```

```
            // Create another edge from Brett to NY Yankees via Edge, set edge properties
            OEdge e2 = db.newEdge(v, v100, "PlayedOn");
            e2.setProperty("Year", "2016");
            e2.save();

            // Create an edge from Michael to Washington via Edge, set edge properties
            OEdge e3 = db.newEdge(v1, v101, "PlayedOn");
            e3.setProperty("Year", "2016");
            e3.save();

        }
        catch(Exception ex)
        {
            System.out.println("Connection error: " + ex);
        }
        pool.close();
        orientDB.close();
    }

}
```

The resulting graph (`SELECT FROM Player`) should look like this:

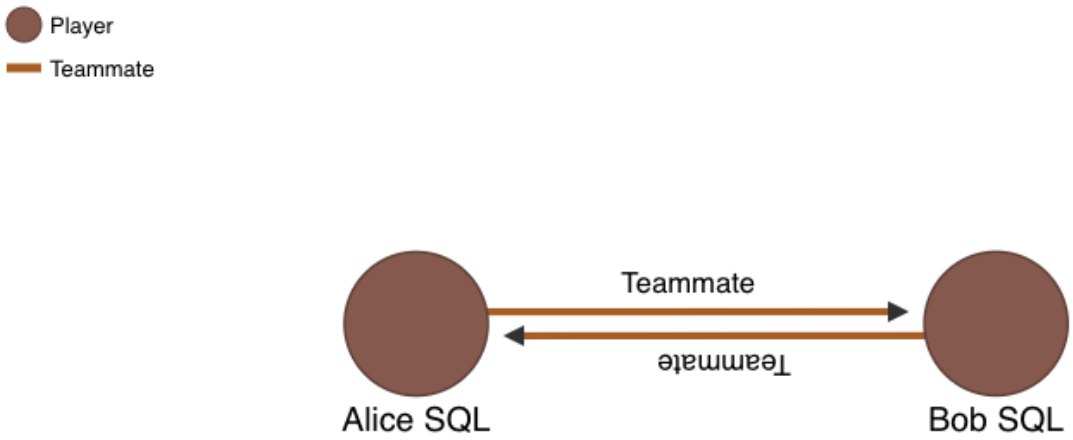


Create vertices and edges using SQL

Use the code below to create two simple vertices in your database, and then establish relationships between them using edges – all through SQL. Be sure to replace the IP address, database name, user name, and password with values provided by your instructor.

```
// Create vertices and edges using SQL

import com.orientechnologies.orient.core.db.ODatabasePool;
import com.orientechnologies.orient.core.db.OrientDB;
import com.orientechnologies.orient.core.db.OrientDBConfig;
import com.orientechnologies.orient.core.db.document.ODatabaseDocument;
import com.orientechnologies.orient.core.record.*;
import com.orientechnologies.orient.core.sql.executor.OResultSet;

public class Create {

    public static void main(String[] args) throws Exception
    {
        try
        {
            Create cr = new Create();
        }
        catch(Exception ex)
        {
            System.out.println("Main error: " + ex);
        }
    }

    private Create()
    {
        OrientDB orientDB = new OrientDB("remote:<ipaddress>","<username>","<password>", OrientDBConfig.defaultConfig());
        ODatabasePool pool = new ODatabasePool(orientDB,"<database name>","<username>","<password>");
        try (ODatabaseDocument db = pool.acquire()) {
            String sql_statement = "CREATE VERTEX Player SET Name = 'Alice SQL', Age = 34";
            OResultSet res = db.execute("SQL", sql_statement);
            sql_statement = "CREATE VERTEX Player SET Name = 'Bob SQL', Age = 19";
            res = db.execute("SQL", sql_statement);
            sql_statement = "CREATE EDGE Teammate FROM (SELECT FROM Player WHERE Name = 'Alice SQL') TO (SELECT FROM Player WHERE Name = 'Bob SQL')";
            res = db.execute("SQL", sql_statement);
            sql_statement = "CREATE EDGE Teammate FROM (SELECT FROM Player WHERE Name = 'Bob SQL') TO (SELECT FROM Player WHERE Name = 'Alice SQL')";
            res = db.execute("SQL", sql_statement);
        }
        catch(Exception ex)
        {
            System.out.println("Connection error: " + ex);
        }
        pool.close();
        orientDB.close();
    }

}
```

After you've run your code, graphing
`SELECT FROM Player WHERE Name = 'Alice SQL'`
should produce results like this:



Create new vertex and edge classes

Use the code below to define new vertex and edge classes. Be sure to replace the IP address, database name, user name, and password with values provided by your instructor.

```
// Create vertex and edge classes

import com.orientechnologies.orient.core.db.ODatabasePool;
import com.orientechnologies.orient.core.db.OrientDB;
import com.orientechnologies.orient.core.db.OrientDBConfig;
import com.orientechnologies.orient.core.db.document.ODatabaseDocument;
import com.orientechnologies.orient.core.metadata.schema.OClass;
import com.orientechnologies.orient.core.metadata.schema.OProperty;
import com.orientechnologies.orient.core.metadata.schema.OType;

public class Create {

    public static void main(String[] args) throws Exception
    {
        try
        {
            Create cr = new Create();
        }
        catch(Exception ex)
        {
            System.out.println("Main error: " + ex);
        }
    }

    private Create()
    {
        OrientDB orientDB = new OrientDB("remote:<ipaddress>","<username>","<password>", OrientDBConfig.defaultConfig());
        ODatabasePool pool = new ODatabasePool(orientDB,"<database name>","<username>","<password>");
        try (ODatabaseDocument db = pool.acquire()) {
            OClass myVertex = db.createVertexClass("Shipment");
            OProperty CustomerName = myVertex.createProperty("CustomerName", OType.STRING);
            CustomerName.setMandatory(true);
            myVertex.createProperty("ShipmentDate", OType.DATE);
            OClass myEdge = db.createEdgeClass("PartOf");
            OProperty Description = myEdge.createProperty("Description", OType.STRING);
        }
        catch(Exception ex)
        {
            System.out.println("Connection error: " + ex);
        }
        pool.close();
        orientDB.close();
    }

}
```

Work with transactions

Use the code below to create two transactions: one that is committed; one that is rolled back. Be sure to replace the IP address, database name, user name, and password with values provided by your instructor.

```
// Example of committing and rolling back transactions

import com.orientechnologies.orient.core.db.ODatabasePool;
import com.orientechnologies.orient.core.db.OrientDB;
import com.orientechnologies.orient.core.db.OrientDBConfig;
import com.orientechnologies.orient.core.db.document.ODatabaseDocument;
import com.orientechnologies.orient.core.record.*;

public class Create {

    public static void main(String[] args) throws Exception
    {
        try
        {
            Create cr = new Create();
        }
        catch(Exception ex)
        {
            System.out.println("Main error: " + ex);
        }
    }

    private Create()
    {
        OrientDB orientDB = new OrientDB("remote:<ipaddress>","<username>","<password>", OrientDBConfig.defaultConfig());
        ODatabasePool pool = new ODatabasePool(orientDB,"<database name>","<username>","<password>");
        try (ODatabaseDocument db = pool.acquire()) {
            db.begin();
            OElement doc1 = db.newInstance("Tx_Sample");
            doc1.setProperty("Name", "Bob");
            doc1.save();
            db.commit();

            db.begin();
            OElement doc2 = db.newInstance("Tx_Sample");
            doc2.setProperty("Name", "Alice");
            doc2.save();
            db.rollback();
        }
        catch(Exception ex)
        {
            System.out.println("Connection error: " + ex);
        }
        pool.close();
        orientDB.close();
    }

}
```

Run a simple query

Use the code below to run a very simple SQL query (with no parameters) against your database. Be sure to replace the IP address, database name, user name, and password with values provided by your instructor.

```
// Connect to a database, run a query with no parameters, print results

import com.orientechnologies.orient.core.db.ODatabasePool;
import com.orientechnologies.orient.core.db.OrientDB;
import com.orientechnologies.orient.core.db.OrientDBConfig;
import com.orientechnologies.orient.core.db.document.ODatabaseDocument;
import com.orientechnologies.orient.core.sql.executor.OResult;
import com.orientechnologies.orient.core.sql.executor.OResultSet;

public class Simple {

    public static void main(String[] args) throws Exception
    {
        try
        {
            Simple s = new Simple();
        }
        catch(Exception ex)
        {
            System.out.println("Main error: " + ex);
        }
    }

    private Simple()
    {
        OrientDB orientDB = new OrientDB("remote:<ipaddress>","<username>","<password>", OrientDBConfig.defaultConfig());
        ODatabasePool pool = new ODatabasePool(orientDB,"<database name>","<username>","<password>");
        try (ODatabaseDocument db = pool.acquire()) {
            String sql_statement = "SELECT FROM Player";
            OResultSet res = db.query(sql_statement);
            while(res.hasNext()){
              OResult row = res.next();
              String Name = row.getProperty("Name");
              System.out.println(Name);
            }
            res.close();
        }
        catch(Exception ex)
        {
            System.out.println("Connection error: " + ex);
        }
        pool.close();
        orientDB.close();
    }
}
```

Run a simple query with parameters

Use the code below to run a very simple SQL query with a single parameter against your database. Be sure to replace the IP address, database name, user name, and password with values provided by your instructor.

```
// Connect to a database, run a query with a single parameter, print results

import com.orientechnologies.orient.core.db.ODatabasePool;
import com.orientechnologies.orient.core.db.OrientDB;
import com.orientechnologies.orient.core.db.OrientDBConfig;
import com.orientechnologies.orient.core.db.document.ODatabaseDocument;
import com.orientechnologies.orient.core.sql.executor.OResult;
import com.orientechnologies.orient.core.sql.executor.OResultSet;

public class Simple {

    public static void main(String[] args) throws Exception
    {
        try
        {
            Simple s = new Simple();
        }
        catch(Exception ex)
        {
            System.out.println("Main error: " + ex);
        }
    }

    private Simple()
    {
        OrientDB orientDB = new OrientDB("remote:<ipaddress>","<username>","<password>", OrientDBConfig.defaultConfig());
        ODatabasePool pool = new ODatabasePool(orientDB,"<database name>","<username>","<password>");
        try (ODatabaseDocument db = pool.acquire()) {
            String sql_statement = "SELECT FROM Player WHERE Name = ?";
            OResultSet res = db.query(sql_statement, "Brett");
            while(res.hasNext()){
              OResult row = res.next();
              String Name = row.getProperty("Name");
              System.out.println(Name);
            }
            res.close();
        }
        catch(Exception ex)
        {
            System.out.println("Connection error: " + ex);
        }
        pool.close();
        orientDB.close();
    }
}
```

Navigate the graph using the `MATCH` statement

Use the code below to navigate your graph using the MATCH statement. Be sure to replace the IP address, database name, user name, and password with values provided by your instructor.

```
// Navigate graph using MATCH statement, print results

import com.orientechnologies.orient.core.db.ODatabasePool;
import com.orientechnologies.orient.core.db.OrientDB;
import com.orientechnologies.orient.core.db.OrientDBConfig;
import com.orientechnologies.orient.core.db.document.ODatabaseDocument;
import com.orientechnologies.orient.core.sql.executor.OResult;
import com.orientechnologies.orient.core.sql.executor.OResultSet;

public class Navigate {

    public static void main(String[] args) throws Exception
    {
        try
        {
            Navigate n = new Navigate();
        }
        catch(Exception ex)
        {
            System.out.println("Main error: " + ex);
        }
    }

    private Navigate()
    {
        OrientDB orientDB = new OrientDB("remote:<ipaddress>","<username>","<password>", OrientDBConfig.defaultConfig());
        ODatabasePool pool = new ODatabasePool(orientDB,"<database name>","<username>","<password>");
        try (ODatabaseDocument db = pool.acquire()) {
            String sql_statement = "MATCH {class: Profiles, as: profile, where: (Name = 'Bud')} -FlewOn-> {as:flight} RETURN profile.Name, flight.FlightNumber";

            OResultSet res = db.query(sql_statement);
            while(res.hasNext()){
              OResult row = res.next();
              String Name = row.getProperty("profile.Name");
              String FlightNumber = row.getProperty("flight.FlightNumber");
              System.out.println(Name);
              System.out.println(FlightNumber);
            }
            res.close();
        }
        catch(Exception ex)
        {
            System.out.println("Connection error: " + ex);
        }
        pool.close();
        orientDB.close();
    }
}
```

Navigate the graph using the `MATCH` statement with parameters

Use the code below to navigate your graph using the MATCH statement with parameters. Be sure to replace the IP address, database name, user name, and password with values provided by your instructor.

```
// Navigate graph using MATCH statement and parameter, print results

import com.orientechnologies.orient.core.db.ODatabasePool;
import com.orientechnologies.orient.core.db.OrientDB;
import com.orientechnologies.orient.core.db.OrientDBConfig;
import com.orientechnologies.orient.core.db.document.ODatabaseDocument;
import com.orientechnologies.orient.core.sql.executor.OResult;
import com.orientechnologies.orient.core.sql.executor.OResultSet;

public class Navigate {

    public static void main(String[] args) throws Exception
    {
        try
        {
            Navigate n = new Navigate();
        }
        catch(Exception ex)
        {
            System.out.println("Main error: " + ex);
        }
    }

    private Navigate()
    {
        OrientDB orientDB = new OrientDB("remote:<ipaddress>","<username>","<password>", OrientDBConfig.defaultConfig());
        ODatabasePool pool = new ODatabasePool(orientDB,"<database name>","<username>","<password>");
        try (ODatabaseDocument db = pool.acquire()) {
            String sql_statement = "MATCH {class: Profiles, as: profile, where: (Name = ?)} -FlewOn-> {as:flight} RETURN profile.Name, flight.FlightNumber";

            OResultSet res = db.query(sql_statement, "Bud");
            while(res.hasNext()){
              OResult row = res.next();
              String Name = row.getProperty("profile.Name");
              String FlightNumber = row.getProperty("flight.FlightNumber");
              System.out.println(Name);
              System.out.println(FlightNumber);
            }
            res.close();
        }
        catch(Exception ex)
        {
            System.out.println("Connection error: " + ex);
        }
        pool.close();
        orientDB.close();
    }
}
```

Update a vertex using API

Use the code below to first create a vertex then update it. Be sure to replace the IP address, database name, user name, and password with values provided by your instructor.

```
// Create a vertex, save it, then update it

import com.orientechnologies.orient.core.db.ODatabasePool;
import com.orientechnologies.orient.core.db.OrientDB;
import com.orientechnologies.orient.core.db.OrientDBConfig;
import com.orientechnologies.orient.core.db.document.ODatabaseDocument;
import com.orientechnologies.orient.core.id.ORID;
import com.orientechnologies.orient.core.record.OVertex;

public class Update {

    public static void main(String[] args) throws Exception
    {
        try
        {
            Update s = new Update();
        }
        catch(Exception ex)
        {
            System.out.println("Main error: " + ex);
        }
    }

    private Update()
    {
        OrientDB orientDB = new OrientDB("remote:<ipaddress>","<username>","<password>", OrientDBConfig.defaultConfig());
        ODatabasePool pool = new ODatabasePool(orientDB,"<database name>","<username>","<password>");
        try (ODatabaseDocument db = pool.acquire()) {
            OVertex myVertex = db.newVertex("Profiles");
            myVertex.setProperty("Name", "ProfileBeforeUpdate");
            myVertex.save();
            ORID oRecord = myVertex.getIdentity();
            System.out.println("Record before update = " + oRecord.getRecord().toString());
            myVertex.setProperty("Name", "ProfileAfterUpdate");
            myVertex.save();
            System.out.println("Record after update = " + oRecord.getRecord().toString());
        }
        catch(Exception ex)
        {
            System.out.println("Connection error: " + ex);
        }
        pool.close();
        orientDB.close();
    }
}
```

Update a vertex using SQL

Use the code below to first create a vertex then update it using SQL. Be sure to replace the IP address, database name, user name, and password with values provided by your instructor.

```
// Create a vertex, save it, then update it using SQL

import com.orientechnologies.orient.core.db.ODatabasePool;
import com.orientechnologies.orient.core.db.OrientDB;
import com.orientechnologies.orient.core.db.OrientDBConfig;
import com.orientechnologies.orient.core.db.document.ODatabaseDocument;
import com.orientechnologies.orient.core.id.ORID;
import com.orientechnologies.orient.core.record.OVertex;
import com.orientechnologies.orient.core.sql.executor.OResult;
import com.orientechnologies.orient.core.sql.executor.OResultSet;

public class UpdateSQL {

    public static void main(String[] args) throws Exception
    {
        try
        {
            UpdateSQL s = new UpdateSQL();
        }
        catch(Exception ex)
        {
            System.out.println("Main error: " + ex);
        }
    }

    private UpdateSQL()
    {
        OrientDB orientDB = new OrientDB("remote:<ipaddress>","<username>","<password>", OrientDBConfig.defaultConfig());
        ODatabasePool pool = new ODatabasePool(orientDB,"<database name>","<username>","<password>");
        try (ODatabaseDocument db = pool.acquire()) {
            String sql_statement = "CREATE VERTEX Player SET Name = 'Fred SQL Before'";
            OResultSet res = db.execute("SQL", sql_statement);
            sql_statement = "SELECT FROM Player WHERE Name = 'Fred SQL Before'";
            res = db.query(sql_statement);
            while(res.hasNext()){
                  OResult row = res.next();
                  String Name = row.getProperty("Name");
                  System.out.println("Record before update: " + Name);
            }
            sql_statement = "UPDATE Player SET Name = 'Fred SQL After' WHERE Name = 'Fred SQL Before'";
            res = db.execute("SQL", sql_statement);
            sql_statement = "SELECT FROM Player WHERE Name = 'Fred SQL After'";
            res = db.query(sql_statement);
            while(res.hasNext()){
                  OResult row = res.next();
                  String Name = row.getProperty("Name");
                  System.out.println("Record after update: " + Name);
            }
        }
        catch(Exception ex)
        {
            System.out.println("Connection error: " + ex);
        }
        pool.close();
        orientDB.close();
    }
}
```

Delete a vertex

Use the code below to first create a vertex then delete it. Be sure to replace the IP address, database name, user name, and password with values provided by your instructor.

```
// Create a vertex, save it, then delete it

import com.orientechnologies.orient.core.db.ODatabasePool;
import com.orientechnologies.orient.core.db.OrientDB;
import com.orientechnologies.orient.core.db.OrientDBConfig;
import com.orientechnologies.orient.core.db.document.ODatabaseDocument;
import com.orientechnologies.orient.core.id.ORID;
import com.orientechnologies.orient.core.record.OVertex;

public class Delete {

    public static void main(String[] args) throws Exception
    {
        try
        {
            Delete d = new Delete();
        }
        catch(Exception ex)
        {
            System.out.println("Main error: " + ex);
        }
    }

    private Delete()
    {
        OrientDB orientDB = new OrientDB("remote:<ipaddress>","<username>","<password>", OrientDBConfig.defaultConfig());
        ODatabasePool pool = new ODatabasePool(orientDB,"<database name>","<username>","<password>");
        try (ODatabaseDocument db = pool.acquire()) {
            OVertex myVertex = db.newVertex("Profiles");
            myVertex.setProperty("Name", "ProfileToDelete");
            myVertex.save();
            ORID oRecord = myVertex.getIdentity();
            System.out.println("Record before delete = " + oRecord.getRecord().toString());
            myVertex.delete();
        }
        catch(Exception ex)
        {
            System.out.println("Connection error: " + ex);
        }
        pool.close();
        orientDB.close();
    }
}
```

REST API Labs

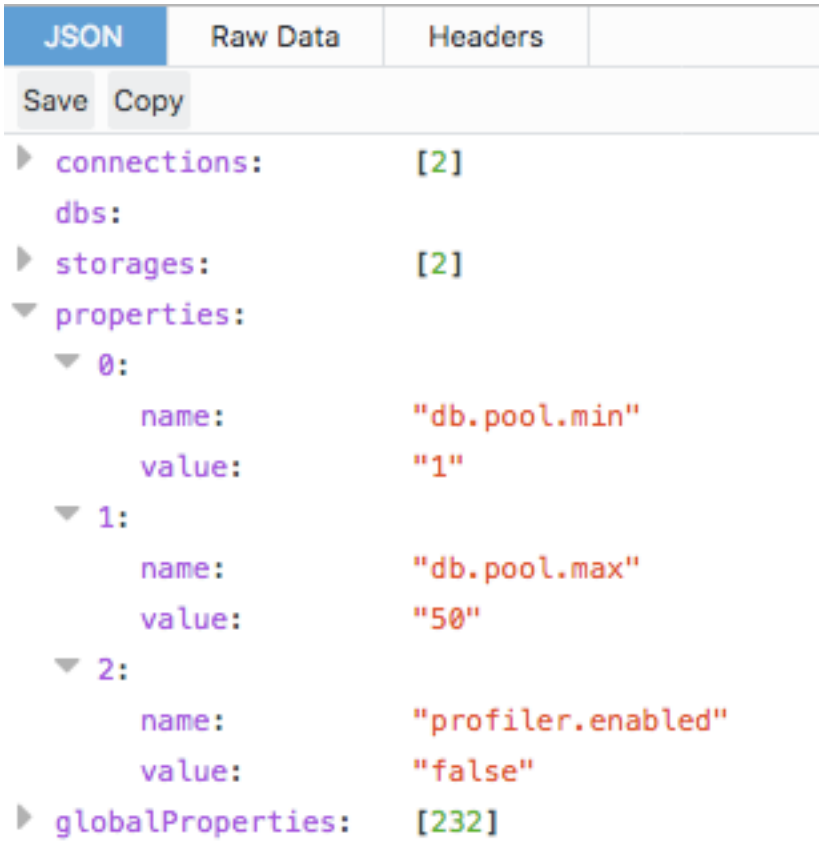
The OrientDB REST API provides a simple but very powerful way to interact with your database. In these labs, you'll learn how to use REST to learn details about your database server, transmit commands, navigate the graph, and more.

Retrieve server information & metadata

Using a browser or API testing tool such as SoapUI or Postman, use the following command to learn details about your database. Be sure to replace the IP address, database name, user name, and password (when requested) with values provided by your instructor. The HTTP command used for all of these requests is `GET`.

First, you can learn details about your database server:

```
http://<server_ip_address>:2480/server
```



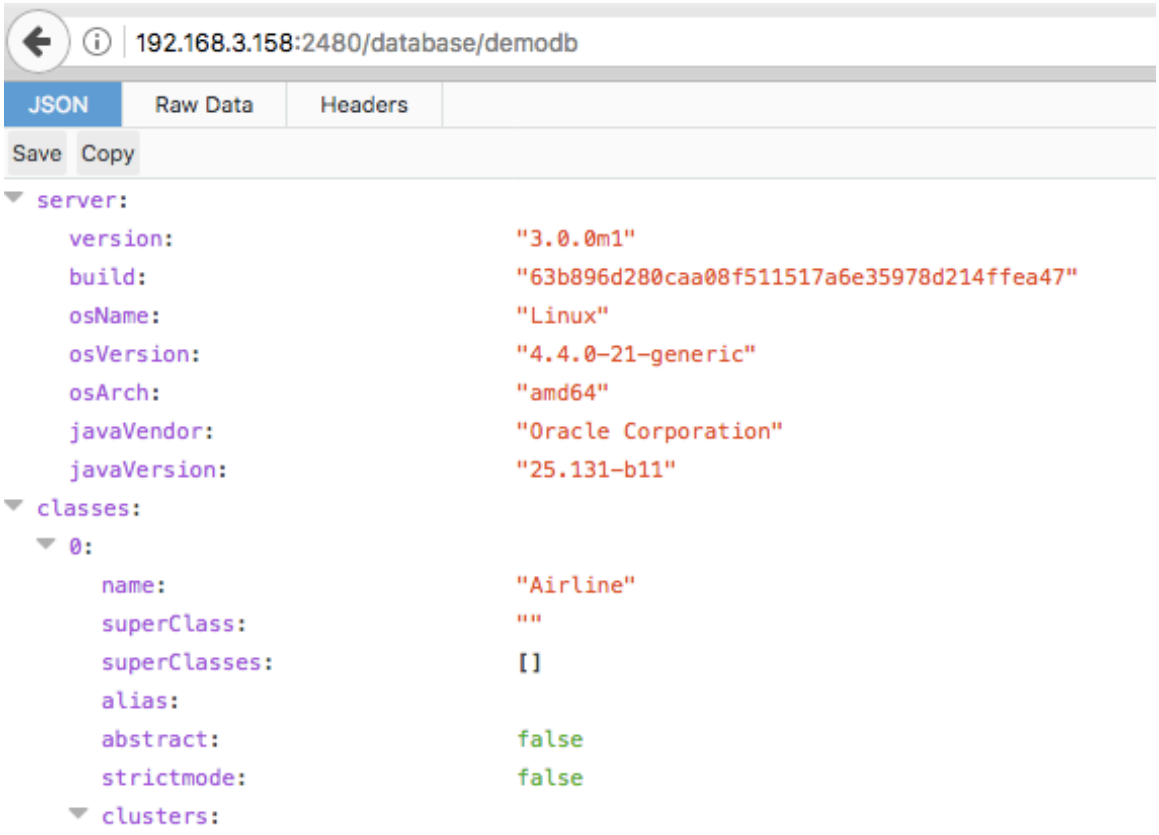
Spend some time exploring each of the major sections in the response.

Next, you can retrieve a list of all the databases on the server:

```
http://<server_ip_address>:2480/listDatabases
```

Next, drill into a specific database to learn more about it:

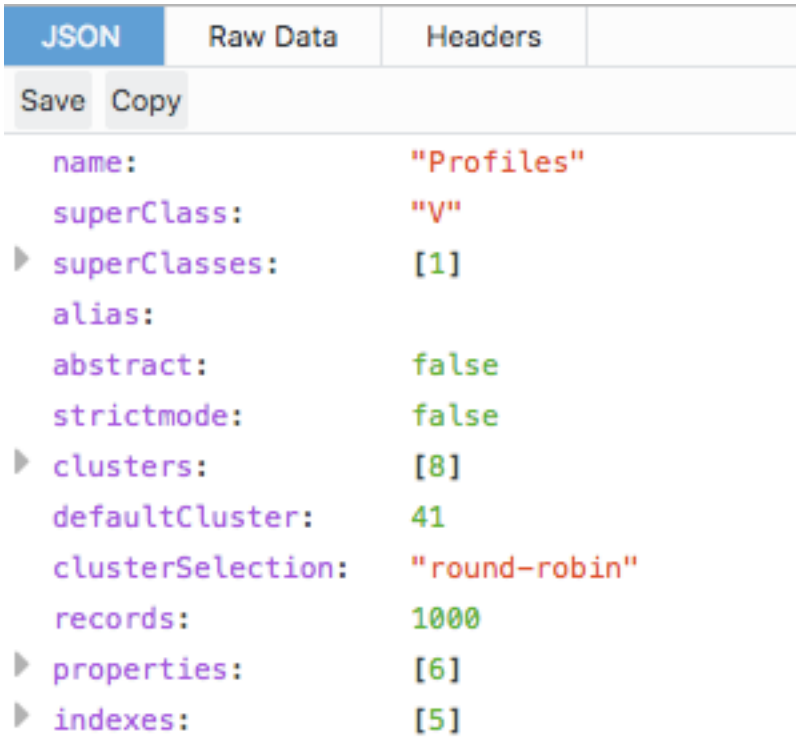
```
http://<server_ip_address>:2480/database/<database_name>
```



Take a few minutes to review the extensive information provided in the response.

Next, drill down to learn more details about the `Profiles` class:

```
http://<server_ip_address>:2480/class/<database_name>/Profiles
```



Spend a few minutes exploring details about this class. Bonus: run this command for other classes that you've used.

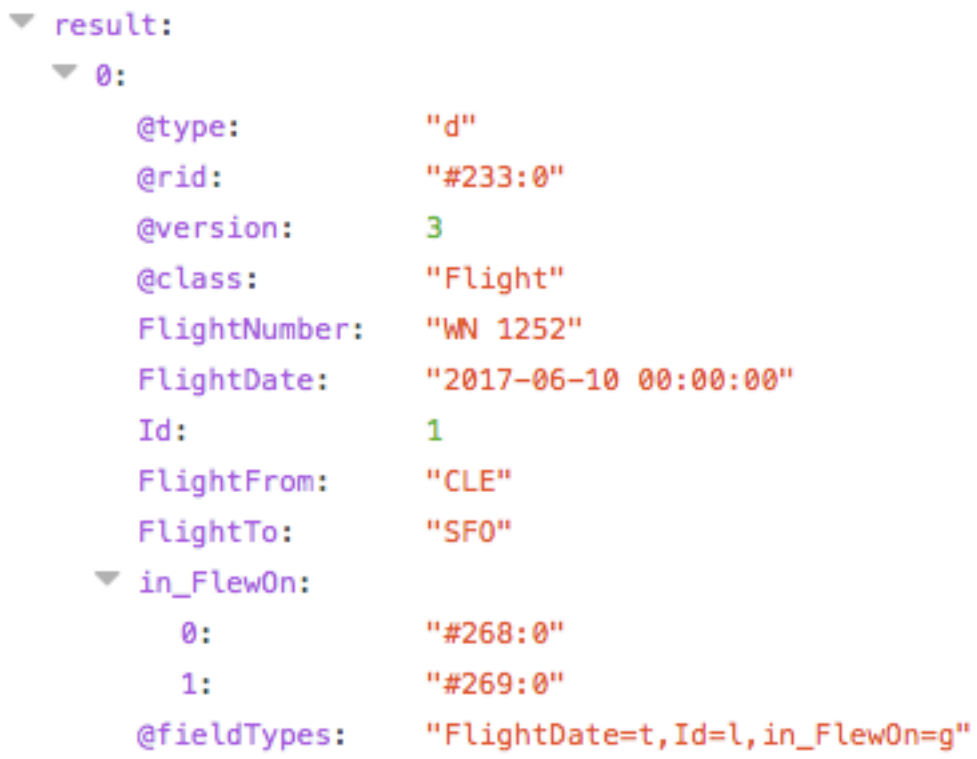
Transmit commands

One nice feature of the REST API is its ability to directly transmit SQL statements and other server commands. In this lab you'll see several examples of this feature in action.

Using a browser or API testing tool such as SoapUI or Postman, use the following command to run a simple query against your database. Be sure to replace the IP address, user name, and password (when requested) with values provided by your instructor.

HTTP method: `GET`

```
http://<server_ip_address>:2480/command/<database_name>/sql
/select from Flight
```



Bonus: add a `WHERE` clause to filter results.

Navigate graph using `MATCH`

You can use the REST API for much more than simple requests. In this lab, you'll transmit a command to use the MATCH statement to navigate the graph. Be sure to replace the IP address, user name, and password (when requested) with values provided by your instructor.

HTTP method: `GET`

```
http://<server_ip_address>:2480/document/<database_name>/sql/MATCH  {class: Profiles, as: profile, where: (Name = 'Bud')}  -FlewOn-> {as:flight}  RETURN profile.Name, flight.FlightNumber
```

You should receive a response similar to this:

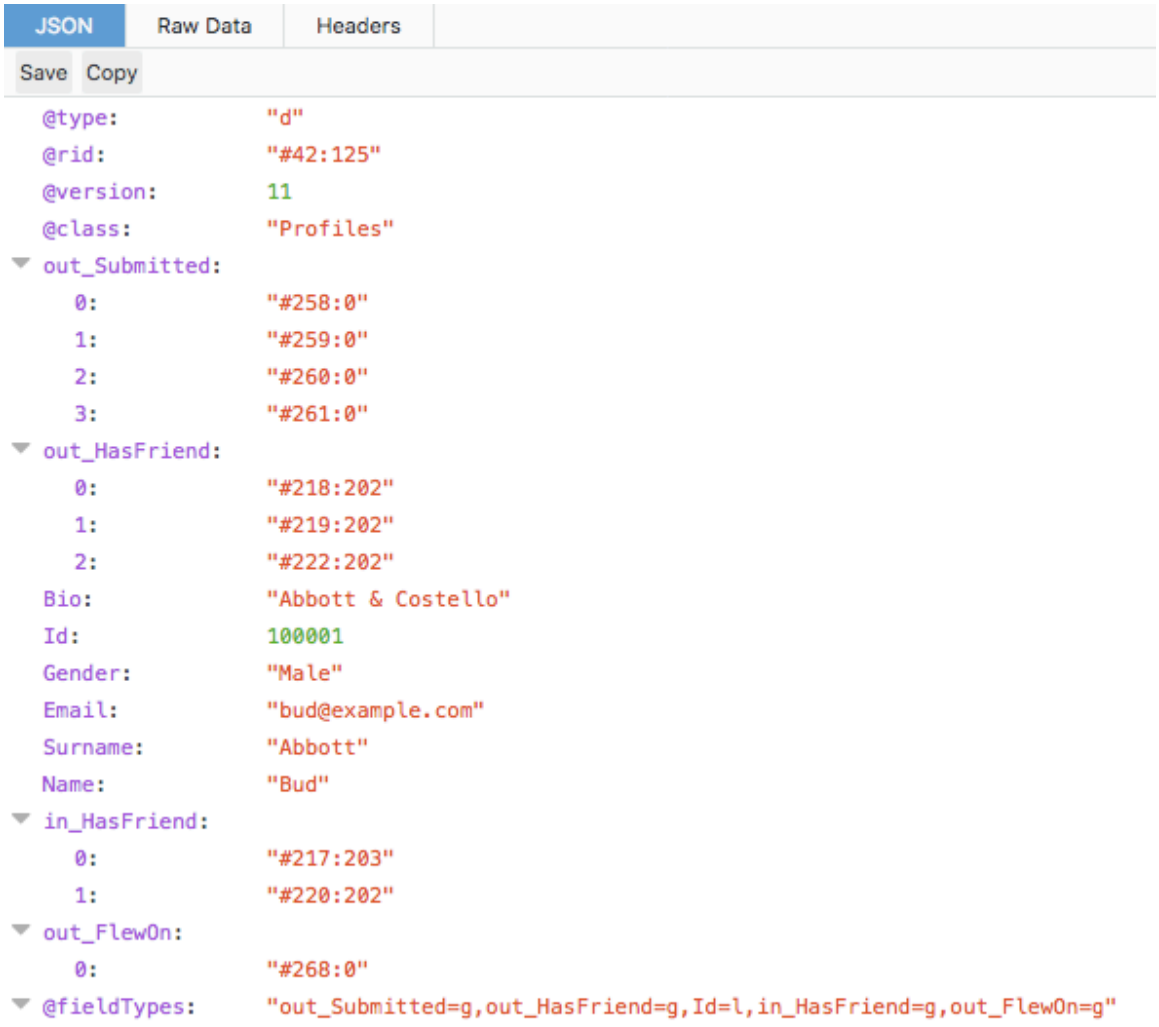
```
{"result":[{"@type":"d","@version":0,"profile_Name":"Bud","flight_FlightNumber":"WN 1252"}]}
```


Use record id to retrieve a single record

You can provide a record id to instruct the REST API to return a single record. Use this syntax to execute the request. Be sure to replace the IP address, database name, user name, and password (when requested) with values provided by your instructor. Use any valid record id (run a separate query in OrientDB Studio or console if necessary to decide on a record id). There's no need to include the '#' symbol with the record id.

HTTP method: `GET`

```
http://<server_ip_address>:2480/document/<database_name>/<record id>
```



Bonus: try the same command using record ids for documents, vertices, and edges.

Create a new record via POST

Along with easily searching for existing data, the REST API lets you create new information. Use this syntax to execute a request to store a new record. Be sure to replace the IP address, database name, user name, and password (when requested) with values provided by your instructor.

HTTP method: `POST`

Mime type: `application/json`

Request URL:
`http://<server_ip_address>:2480/document/<database_name>`

Message body: `{"@class": "Team", "Name": "SF Giants"}`

You should receive a response similar to this:

```
{"@type":"d","@rid":"#290:0","@version":1,"@class":"Team","
Name":"SF Giants"}
```

Update an existing record via `PUT`

Use this syntax to execute a request to update an existing record. Be sure to replace the IP address, database name, user name, and password (when requested) with values provided by your instructor. Use the record id that was generated in the last lab.

HTTP method: `PUT`

Mime type: `application/json`

Request URL:
```
http://<server_ip_address>:2480/document/<database_name>/<record id>
```

Message body:
```
{"@type":"d","@rid":"#289:0","@version":0,"@class":"Team","Name":"SF Giants - updated"}
```

Delete an existing record via `DELETE`

Use this syntax to execute a request to delete an existing record. Be sure to replace the IP address, database name, user name, and password (when requested) with values provided by your instructor. Use the record id from the last lab.

HTTP method: `DELETE`

Mime type: `application/json`

Request URL:
`http://<server_ip_address>:2480/document/<database_name>/<record id>`

Run a query against the database to see if the record was successfully deleted.

Create a new record via POST and then update one property via PATCH

Rather than requiring you to transmit a message containing all properties for a simple update, the REST API lets you make changes to specific properties – leaving the rest unchanged. Be sure to replace the IP address, database name, user name, and password (when requested) with values provided by your instructor.

First, create a new record:

HTTP method: `POST`

Mime type: `application/json`

Request URL:
`http://<server_ip_address>:2480/document/<database_name>`

Message body: `{"@class": "Player", "Name": "Jim", "Age": "23"}`

You should receive a response similar to this:

```
{"@type":"d","@rid":"#305:0","@version":1,"@class":"Player"
,"Name":"Jim","Age":"23"}
```

Next, update the record:

HTTP method: `PATCH`. Use the record id that was just generated.

Mime type: `application/json`

Request URL:
`http://<server_ip_address>:2480/document/<database_name>/<r
ecord id>`

Message body: `{"@class": "Player", "Name": "Jim", "Age": "23"}`

You should receive a response similar to this:
`{"@type":"d","@rid":"#305:0","@version":2,"@class":"Player"
,"Name":"Jim","Age":"35"}`