Bases de datos II - OrientDB

v1, 2023-11-08

Secciones

1. Introducción . . . 1
2. Origen . . . 1
3. Casos de éxito . . . 1
4. Características principales . . . 1
5. Base de datos . . . 1
6. Tipos de datos . . . 2
7. Funcionamiento . . . 2
    7.1. Almacenamiento de datos . . . 2
    7.2. Relaciones . . . 4
8. Isolation levels . . . 5
9. Transacciones . . . 5
10. Replicación . . . 6
11. Sharding . . . 6
12. Diferencias con otras bases de datos . . . 6
13. Descarga e instalación . . . 7
14. Video de instalación y configuración . . . 10
15. Conexiones de bases de datos . . . 10
    15.1. Conexión mediante consola . . . 10
16. Operaciones básicas . . . 11
    16.1. Creación de clases . . . 11
    16.2. Creación de atributos . . . 11
    16.3. Creación de indices . . . 12
    16.4. Inserción de datos . . . 12
    16.5. Creación de aristas . . . 12
    16.6. Eliminación de datos . . . 14
    16.7. Actualización de datos . . . 15
17. Conclusiones . . . 16

1. Introducción

OrientDB es una bases de datos NoSQL de código abierto orientado a documentos y grafos desarrollada en Java.

2. Origen

Fue desarrollado por OrientDB Ltd. y está diseñado para ser una base de datos NoSQL de alto rendimiento y escalable.

3. Casos de éxito

Algunos de los clientes de OrientDB son Accenture, Comcast, Ericsson, las Naciones Unidas, Sky, entre otros.

4. Características principales

- Taxonomía: OrientDB es multi-modelo, lo que significa que admite múltiples modelos de datos, incluyendo documentos, grafos, clave-valor y objetos.
- Escalabilidad: OrientDB es escalable y puede funcionar en un clúster distribuido, lo que permite la distribución de datos en varios nodos para manejar grandes volúmenes de información.
- Transacciones ACID: OrientDB garantiza transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), lo que lo hace adecuado para aplicaciones que requieren alta integridad de datos.
- Compatibilidad: OrientDB es compatible con múltiples lenguajes de programación, incluyendo Java, Python y .NET; y el estándar SQL.

5. Base de datos

Se pueden crear múltiples bases de datos, de distintos tipos, pero no se puede trabajar con mas de una al mismo tiempo. Esto es porque el manejo de la estructura de directorios internamente permite que haya dos bases de datos con el mismo nombre en distintos directorios.

La URL de la base de datos se compone del motor de la base de datos y el nombre, de la siguiente forma: `<engine>:<db-name>`. El motor puede ser `plocal`, que utiliza al file system para almacenar los datos y contiene un LOG; `memory`, que almacena la base de

datos completamente en memoria; y `remote`, que permite una conexión remota para que sea posible acceder a ella a través de varios clientes. La sintaxis de esta última es `remote:<server>:[<port>]/db-name`, siendo el puerto por default el 2424.

6. Tipos de datos

OrientDB admite cuatro tipos de datos: *documento*, *blob*, *vertices* y *aristas*. Un registro puede ser cualquiera de estos elementos.

Los documentos son de tipado suave, y se pueden definir mediante schemas y añadir restricciones (como una base de datos relacional), pero también pueden ser utilizados sin un schema definido. Para las relaciones entre documentos, utilizamos `Map<String, Object>`.

Se denomina BLOB a los datos en formato binario. Los vértices almacenan la información, y las aristas conectan dos vértices de forma bidireccional. Ambos son documentos, excepto la versión lightweight de las aristas.

La información dentro de un registro puede ser de tipo: `Boolean`, `Integer`, `Short`, `Long`, `Float`, `Double`, `Date-time`, `String`, `Binary`, `Byte`, `Transient`, `Custom`, `Decimal`, `Any`, `Embedded`, `Link`, y distintos tipos de colecciones de estos últimos.

7. Funcionamiento

Cuando se genera un registro, OrientDB le asigna automáticamente un ID. Este ID está compuesto por dos elementos: el identificador del cluster y la posición del registro en el mismo. El formato es `#<cluster>:<position>`. Si el ID del cluster es positivo, se garantiza la persistencia del registro. En caso contrario, los registros son temporales, como los que aparecen en conjuntos de resultados de queries que utilizan proyecciones.

No es necesario generar un campo para estos ID, ya que la base de datos lo gestiona independientemente.

Si el registro es eliminado, su ID también se elimina, y no se vuelve a utilizar.

Cada registro tiene un número de versión que se incrementa cuando se actualiza, para evitar conflictos al momento del commit en una transacción.

7.1. Almacenamiento de datos

- Clases: Las clases se podrían pensar como una tabla del modelo relacional, o una

colección en una base de datos no relacional con taxonomía documental. Definen registros, de forma parecida a como una clase define a un objeto en POO. Las clases pueden ser schema-less, schema-full o un mix de ambas, y pueden heredar de otras clases. La herencia es por atributos, es decir, la clase hija tendrá los mismos atributos que la clase padre.

También existe el concepto de **clase abstracta**, que son utilizadas para definir otras clases, pero no se pueden instanciar.

Cada clase tiene sus clusters, que se definen automáticamente al momento de la creación de la clase. El cluster es un espacio físico de almacenamiento para la información. Toda clase no abstracta debe tener al menos un cluster definido, funcionando como default. Como una clase puede tener más de un cluster, una búsqueda sobre los elementos de esa clase se propaga por todos los elementos del cluster. Al insertar información, OrientDB selecciona el cluster donde se almacenará utilizando una estrategia predeterminada.

- Cluster: como se mencionó, los clusters son espacios de almacenamiento para los registros. OrientDB genera un cluster por core de la CPU para cada clase, para mejorar el paralelismo. El nombre del cluster corresponde al nombre de la clase en minúsculas, un guión bajo y un entero. Por ejemplo, para la clase `Persona`, OrientDB generará los clusters `persona`, `persona_1`, `persona_2`, etc.

Supongamos que tenemos la clase `Estudiante`, que puede heredar de la clase `Persona`. Dos clusters para esta clase pueden ser `estudiante100` y `estudiante101`. Como ambos clusters pertenecen a la misma clase, podemos obtener los datos de ambos (y de todos los demás clusters pertenecientes a esta clase) de la siguiente forma:

```
orientdb> SELECT FROM Estudiante
```

Incluso se pueden filtrar los resultados, de la misma forma que se haría en SQL

```
orientdb> SELECT FROM Estudiante WHERE nombre = 'Juan'
```

Si se quisiera obtener elementos de un cluster específico, se puede realizar de la siguiente manera:

```
orientdb> SELECT FROM CLUSTER:estudiante101
```

En este caso, la query es significativamente más rápida, ya que OrientDB restringe la búsqueda al cluster requerido.

7.2. Relaciones

OrientDB admite dos tipos de relaciones, **referenciadas** (como en el modelo relacional) y **embebidas** (como puede ser en el modelo documental). Estas relaciones también pueden ser schema-full o schema-less.

7.2.1. Relaciones referenciadas

Como suele suceder en bases de datos noSQL, no se admiten `joins`. En reemplazo a esto, OrientDB almacena referencias directas a los objetos de la relación, generando un grafo. Por ejemplo

```
                carrera
  Record A     ------------>    Record B
CLASS=Estudiante              CLASS=Carrera
  RID=5:23                      RID=10:2
```

En este caso, el registro A contiene la referencia al registro B. Esta referencia se denominó "carrera". Como las relaciones son bidireccionales, con la API `Graph` se puede representar esta bidireccionalidad, almacenando una arista por vértice.

Las relaciones 1 a 1 y 1 a n (sin la API `Graph`) se definen utilizando referencias a través del tipo `LINK`. Para las relaciones 1 a M y M a M, OrientDB utiliza conjuntos de `LINK`, de tipo:

- `LINKLIST`: Una lista ordenada de links
- `LINKSET`: Un conjunto desordenado de links, sin repetidos
- `LINKMAP`: Un map ordenado de links, siendo la clave un `String` y el valor un `LINK`

La API `Graph` define que una arista sólo conecta a dos vértices, por lo que una relación 1 a n no se podría modelar de forma trivial. Para modelar este tipo de relaciones, se deben utilizar múltiples aristas.

7.2.2. Relaciones embebidas

Como cualquier otra base de datos de taxonomía documental, OrientDB puede almacenar relaciones embebidas. El DBMS almacena la relación dentro del registro que la contiene. Estas relaciones se pueden pensar como composición, de diagramas UML.

```
                carrera
```

```
  Record A     <>---------->   Record B
CLASS=Estudiante             CLASS=Carrera
  RID=5:23                     NO RID!
```

En este caso, el registro A contiene el registro B en la propiedad `carrera`. Solo se puede llegar a B a través de A. Por ejemplo:

```
orientdb> SELECT FROM Estudiante WHERE carrera.nombre = 'Sistemas'
```

Las relaciones 1 a 1 y n a 1 se pueden almacenar utilizando el tipo `EMBEDDED`. Por otra parte, las relaciones 1 a n y n a n se pueden almacenar utilizando una colección de registros, como

- `EMBEDDEDLIST`: Una lista ordenada de registros
- `EMBEDDEDSET`: Un conjunto desordenado de registros, sin repetidos
- `EMBEDDEDMAP`: Un map ordenado de registros, siendo la clave un `String` y el valor un `EMBEDDED`

8. Isolation levels

OrientDB ofrece dos tipos de isolation levels: Read Committed y Repeatable Read, siendo Read Committed el default, y el único disponible si se utiliza el protocolo `remote`. Esto quiere decir que no se tendrán `dirty reads`, pero puede cambiar un valor leido más de una vez

9. Transacciones

Existen dos tipos de transacciones:

- No transaction: cada operación se ejecuta instantáneamente
- Optimistic transaction: se utiliza MVCC. Se permiten muchas lecturas y escrituras, y la integridad de los datos se verifica en el commit. Si el registro se actualizó en medio de una transacción, se lanza una excepción a la aplicación, para que defina si desea reintentarla o abortarla.

  En una arquitectura distribuida, se utiliza un protocolo similar a `2 phase lock`. Si durante el commit se modifican los registros desde otra parte, la transacción falla y el usuario debe decidir cómo continuar. En cambio, si los registros se bloquean por otra transacción, la transacción falla pero se puede configurar para que se

reintente automáticamente.

Cuando se tienen diferentes nodos y se realiza el commit de una transacción en uno de ellos, todos los registros actualizados se envían al resto de los nodos del servidor, por lo que estos nodosos son responsables de realizar el commit de la transacción. En caso de fallo, se verifica el quorum. Si fue respetado, los nodos que fallaron son alineados a los que no. En caso contrario, todos los nodos realizan el rollback.

10. Replicación

OrientDB soporta la replicación Multi Master, en la que cada nodo es un maestro y puede leer y escribir en la base de datos. La única restricción es que cada replicación de la base de datos debe tener el mismo nombre en la base de datos distribuida. Cuando el servidor se inicia, envía una lista de sus bases de datos a todos los nodos del cluster. Si los nodos tienen bases de datos con los mismos nombres, se establece la replicación entre ellas.

11. Sharding

12. Diferencias con otras bases de datos

con relacional: puede ser schemaless, no tenes que generar la PK o generar un campo para la misma, esta tiene clases y la otra no, esta se divide por clusters, ambas utilizan SQL pero esta lo utiliza modificado levemente

https://es.slideshare.net/WilmerGonzalez7/bases-de-datos-nosql-multimodelos-caso-de-estudio-orientdb

FEATURES	ORIENTDB	MONGODB	NEO4J	MYSQL (RDBMS)
Operational Database	X	X		X
Graph Database	X		X	
Document Database	X	X		
Object-Oriented Concepts	X			
Schema-full, Schema-less, Schema mix	X			
User and Role & Record Level Security	X			
Record Level Locking	X		X	X
SQL	X			X
ACID Transaction	X		X	X
Relationships (Linked Documents)	X		X	X
Custom Data Types	X	X		X
Embedded Documents	X	X		
Multi-Master Zero Configuration Replication	X			
Sharding	X	X		
Server Side Functions	X	X		X
Native HTTP Rest/ JSON	X	X		
Embeddable with No Restrictions	X			

13. Descarga e instalación

Descargue OrientDB desde la siguiente URL:

https://orientdb.org/download

Se descargará el archivo de la versión community. Descomprímalo en su sistema de archivos y abra un shell en el directorio.

Ahora ingrese a la carpeta que se descomprimió y a la subcarpeta "bin":

```
cd orientdb-community-3.2.24/bin
```

(cambia el nombre de la carpeta con la versión exacta que descargaste)

y luego, si estás en Linux/OSX, puede iniciar el servidor con `./server.sh`.

Si estás en Windows, inicia el servidor con `server.bat`.

Verás que el servidor inicia:

```
          .
          .`        `
         .`
         ,      `:.
        `,`    ,:`
        .,.   :,,
        .,,  ,,,
```

```
    .    .,.::::: ````                                 :::::::::
:::::::::
    ,`   .::,,,,::.,,,,,,,`;;                    .:    ::::::::::    :::
:::
    `,.  ::,,,,,,,:.,,.`  `                      .:    :::      :::  :::
:::
     ,,:,:,,,,,,,,::.   `       `        ``     .:    :::      :::  :::
:::
      ,,:.,,,,,,,,,: `::, ,,   ::,::`   : :,::` ::::   :::      :::  :::
:::
       ,:,,,,,,,,,,::,:   ,,  :.    :   ::    :   .:    :::      :::
:::::::
        :,,,,,,,,,,:,::   ,,  :     :  :     :   .:    :::      :::
:::::::::
  `      :,,,,,,,,,,:,::,  ,, .::::::::  :     :   .:    :::      :::  :::
:::
  `,...,,:,,,,,,,,,: .:,. ,, ,,        :     :   .:    :::      :::  :::
:::
    .,,,,::,,,,,,,:  `: , ,,  :     `   :     :   .:    :::      :::  :::
:::
      ...,::,,,,::.. `:  .,,  :,    :   :     :   .:    ::::::::::    :::
:::
           ,::::,,,. `:   ,,   :::::    :     :   .:    :::::::::
::::::::::
           ,,:` `,,.
          ,,,    .,`
        ,,,.     `,
       ``        `.
                 ``
www.orientdb.com
                 `


2023-11-04 16:47:51:200 INFO  Windows OS is detected, 262144 limit of
open files will be set for the disk cache. [ONative]
2023-11-04 16:47:51:232 INFO  Loading configuration from:
C:/Users/Gonza/Desktop/orientdb-community-3.2.24/config/orientdb-server-
config.xml... [OServerConfigurationLoaderXml]
2023-11-04 16:47:51:419 INFO  OrientDB Server v3.2.24 (build
${buildNumber}, branch UNKNOWN) is starting up... [OServer]
2023-11-04 16:47:51:732 INFO  14963257344 B/14270 MB/13 GB of physical
memory were detected on machine [ONative]
2023-11-04 16:47:51:732 INFO  Detected memory limit for current process
is 14963257344 B/14270 MB/13 GB [ONative]
2023-11-04 16:47:51:732 INFO  JVM can use maximum 2048MB of heap memory
[OMemoryAndLocalPaginatedEnginesInitializer]
```

```
2023-11-04 16:47:51:732 INFO  Because OrientDB is running outside a
container 2g of memory will be left unallocated according to the setting
'memory.leftToOS' not taking into account heap memory
[OMemoryAndLocalPaginatedEnginesInitializer]
2023-11-04 16:47:51:732 INFO  OrientDB auto-config DISKCACHE=10,174MB
(heap=2,048MB os=14,270MB) [orientechnologies]
2023-11-04 16:47:51:732 INFO  System is started under an effective user :
`Gonza` [OEngineLocalPaginated]
2023-11-04 16:47:51:903 INFO  WAL maximum segment size is set to 6,144 MB
[OrientDBDistributed]
2023-11-04 16:47:51:997 INFO  Databases directory:
C:\Users\Gonza\Desktop\orientdb-community-3.2.24\databases [OServer]
2023-11-04 16:47:52:013 INFO  Creating the system database 'OSystem' for
current server [OSystemDatabase]
2023-11-04 16:47:52:060 INFO  Page size for WAL located in
C:\Users\Gonza\Desktop\orientdb-community-3.2.24\databases\OSystem is set
to 4096 bytes. [CASDiskWriteAheadLog]
2023-11-04 16:47:52:107 INFO  DWL:OSystem: block size = 4096 bytes,
maximum segment size = 2161 MB [DoubleWriteLogGL]
2023-11-04 16:47:52:310 INFO  Storage
'plocal:C:\Users\Gonza\Desktop\orientdb-community-
3.2.24\databases/OSystem' is created under OrientDB distribution : 3.2.24
(build ${buildNumber}, branch UNKNOWN) [OLocalPaginatedStorage]
2023-11-04 16:47:53:512 INFO  Listening binary connections on
0.0.0.0:2424 (protocol v.38, socket=default) [OServerNetworkListener]
2023-11-04 16:47:53:512 INFO  Listening http connections on 0.0.0.0:2480
(protocol v.10, socket=default) [OServerNetworkListener]

+----------------------------------------------------------------+
|                WARNING: FIRST RUN CONFIGURATION                |
+----------------------------------------------------------------+
| This is the first time the server is running. Please type a    |
| password of your choice for the 'root' user or leave it blank |
| to auto-generate it.                                           |
|                                                                |
| To avoid this message set the environment variable or JVM      |
| setting ORIENTDB_ROOT_PASSWORD to the root password to use.    |
+----------------------------------------------------------------+

Root password [BLANK=auto generate it]: *
```

Luego, deberás ingresar una nueva contraseña para el usuario `root`.

14. Video de instalación y configuración

Instalación y configuración

15. Conexiones de bases de datos

Existen dos métodos para conectarse a un servidor y comenzar a trabajar con las bases de datos en OrientDB:

1. Acceso a través del navegador en el puerto 2480 (http://localhost:2480/): Esta opción no solo te permite crear y administrar bases de datos, sino también editar y visualizar gráficos directamente desde la página web.
2. Utilización de la consola de OrientDB.

15.1. Conexión mediante consola

Si estás utilizando Linux/OSX, puedes iniciar la consola ejecutando `./console.sh` desde la carpeta "bin".

En el caso de Windows, inicia la consola ejecutando `console.bat`.

Luego, para conectarte al servidor, puedes usar el siguiente comando (asegúrate de reemplazar `servidor`, `usuario` y `contraseña`):

```
connect remote:servidor usuario contraseña
```

Para crear una base de datos, utiliza el comando `create database`. Asegúrate de especificar una URL para la base de datos y un nombre de usuario.

```
create database plocal:/ruta/a/la/base-de-datos usuario
```

Para listar las bases de datos existentes, ejecuta el siguiente comando:

```
list databases
```

Para conectarte a una base de datos, puedes utilizar el siguiente comando:

```
connect remote:servidor/base_de_datos usuario
```

Asegúrate de reemplazar `servidor`, `base_de_datos` y `usuario` con los valores correspondientes.

Para ver las clases existentes, utiliza el siguiente comando:

```
classes
```

16. Operaciones básicas

OrientDB es compatible con el conocido lenguaje de consultas SQL y soporta consultas en lenguaje Gremlin para trabajar con datos de grafo.

16.1. Creación de clases

A la hora de crear clases en OrientDB, puedes hacerlo de diversas formas:

Creación de una clase genérica

```
CREATE CLASS Estudiante
```

Creación de una clase que hereda de un vértice

```
CREATE CLASS Usuario EXTENDS V
```

Creación de una clase que hereda de una arista

```
CREATE CLASS Sigue EXTENDS E
```

16.2. Creación de atributos

Si deseas agregar atributos a una clase antes de ingresar datos, puedes hacerlo de la siguiente manera:

```
CREATE PROPERTY Estudiante.legajo STRING
CREATE PROPERTY Estudiante.nombre STRING
CREATE PROPERTY Estudiante.apellido STRING
```

```
CREATE PROPERTY Estudiante.nacimiento DATE
```

16.3. Creación de indices

Para crear indices puedes hacerlo de la siguiente forma:

```
CREATE INDEX Estudiante.legajo UNIQUE
```

16.4. Inserción de datos

La inserción de datos en OrientDB se puede realizar de diversas formas, similar a como se hace en una base de datos SQL:

Inserción utilizando la sintaxis de columnas y valores:

```
INSERT INTO Estudiante (legajo, nombre, apellido) VALUES (1, 'Juan',
'Perez')
```

Otra forma de inserción, utilizando la sintaxis de pares clave-valor:

```
INSERT INTO Estudiante SET legajo = 1, nombre = 'Juan', apellido =
'Perez'
```

También puedes utilizar la sintaxis de contenido JSON para la inserción de datos:

```
INSERT INTO Estudiante CONTENT {'legajo': 1, 'nombre': 'Juan',
'apellido': 'Perez'}
```

16.5. Creación de aristas

La creación de aristas en OrientDB te permite establecer relaciones entre diferentes vértices en tu base de datos.

A continuación, se presenta un ejemplo de cómo crear aristas.

Agreguemos usuarios a la clase `Usuario` de la siguiente manera:

```
INSERT INTO Usuario SET nombre_usuario = 'juan_perez55', correo =
'juanperez55@gmail.com';
INSERT INTO Usuario SET nombre_usuario = 'carlos_rodr32', correo =
'carlosrodriguez32@gmail.com';
```

Al observar los datos en la consola, veremos lo siguiente:

```
+----+-----+-------+--------------+--------------------------+
|#   |@RID |@CLASS |nombre_usuario|correo                    |
+----+-----+-------+--------------+--------------------------+
|0   |#22:0|Usuario|juan_perez55  |juanperez55@gmail.com     |
|1   |#23:0|Usuario|carlos_rodr32 |carlosrodriguez32@gmail.com|
+----+-----+-------+--------------+--------------------------+
```

Para crear una arista que conecte a estos dos usuarios, ejecutamos el siguiente comando:

```
CREATE EDGE Sigue FROM (SELECT FROM Usuario WHERE nombre_usuario =
'juan_perez55') TO (SELECT FROM Usuario WHERE nombre_usuario =
'carlos_rodr32');
```

Una vez realizado esto, podemos ver los datos de la clase `Sigue`:

```
+----+-----+------+-----+-----+
|#   |@RID |@CLASS|out  |in   |
+----+-----+------+-----+-----+
|0   |#26:0|Sigue |#22:0|#23:0|
+----+-----+------+-----+-----+
```

Como se puede observar, hemos establecido una relación de `Sigue` entre los usuarios `juan_perez55` y `carlos_rodr32`. En términos coloquiales, podríamos decir que `juan_perez55` sigue a `carlos_rodr32`. Es importante destacar que en este contexto, `Sigue` no representa una relación simétrica, lo que significa que `carlos_rodr32` no sigue automáticamente a `juan_perez55`.

Además, los datos en la clase `Usuario` también se actualizan para reflejar la relación:

```
+----+-----+-------+--------------+--------------------------+---------
+--------+
```

```
|#   |@RID |@CLASS |nombre_usuario|correo
|out_Sigue|in_Sigue|
+----+-----+-------+--------------+-------------------------+---------
+--------+
|0   |#22:0|Usuario|juan_perez55  |juanperez55@gmail.com      |[#26:0]  |
|
|1   |#23:0|Usuario|carlos_rodr32 |carlosrodriguez32@gmail.com|
|[#26:0] |
+----+-----+-------+--------------+-------------------------+---------
+--------+
```

Como se puede ver, la relación de `Sigue` se refleja en las propiedades de la clase `Usuario`. Esta es la forma en que OrientDB gestiona y representa las relaciones en su base de datos.

16.6. Eliminación de datos

La eliminación de datos en OrientDB se asemeja a la que se realiza en SQL.

A continuación, se presenta un ejemplo que ilustra cómo eliminar datos.

Agreguemos dos estudiantes más a la clase `Estudiante`:

```
INSERT INTO Estudiante CONTENT {'legajo': 2, 'nombre': 'Maria',
'apellido': 'Gonzalez'}
INSERT INTO Estudiante CONTENT {'legajo': 3, 'nombre': 'Carlos',
'apellido': 'Ramirez'}
```

Al ejecutar una consulta `SELECT * FROM Estudiante`, obtenemos los siguientes resultados en la consola:

```
+----+-----+----------+------+------+--------+
|#   |@RID |@CLASS    |legajo|nombre|apellido|
+----+-----+----------+------+------+--------+
|0   |#22:2|Estudiante|2     |Maria |Gonzalez|
|1   |#23:1|Estudiante|3     |Carlos|Ramirez |
|2   |#25:1|Estudiante|1     |Juan  |Perez   |
+----+-----+----------+------+------+--------+
```

Para eliminar un estudiante con un legajo específico, como el estudiante con legajo igual a 2 (Carlos Ramirez), utilizamos el siguiente comando:

```
DELETE FROM Estudiante WHERE legajo = 2
```

Tras la eliminación, al ejecutar nuevamente la consulta `SELECT * FROM Estudiante`, observamos que el estudiante Carlos Ramirez ha sido eliminado:

```
+----+-----+----------+------+------+--------+
|#   |@RID |@CLASS    |legajo|nombre|apellido|
+----+-----+----------+------+------+--------+
|0   |#23:1|Estudiante|3     |Carlos|Ramirez |
|1   |#25:1|Estudiante|1     |Juan  |Perez   |
+----+-----+----------+------+------+--------+
```

16.7. Actualización de datos

La actualización de datos en OrientDB se logra utilizando el comando `UPDATE`, que te permite modificar registros existentes.

A continuación, se presenta un ejemplo de cómo actualizar datos.

Si ejecutamos la consulta `SELECT * FROM Estudiante` obtenemos los siguientes resultados en la consola:

```
+----+-----+----------+------+------+--------+
|#   |@RID |@CLASS    |legajo|nombre|apellido|
+----+-----+----------+------+------+--------+
|0   |#23:1|Estudiante|3     |Carlos|Ramirez |
|1   |#25:1|Estudiante|1     |Juan  |Perez   |
+----+-----+----------+------+------+--------+
```

Para actualizar un registro, utilizamos el comando `UPDATE`. En este ejemplo, cambiamos el apellido de Carlos Ramirez a 'Rodriguez' con la siguiente consulta:

```
UPDATE Estudiante SET apellido = 'Rodriguez' WHERE legajo = 3
```

Al ejecutar nuevamente la consulta `SELECT * FROM Estudiante`, observamos los siguientes resultados en la consola:

```
+----+-----+----------+------+------+---------+
|#   |@RID |@CLASS    |legajo|nombre|apellido |
```

```
+----+-----+----------+------+------+---------+
|0   |#23:1|Estudiante|3     |Carlos|Rodriguez|
|1   |#25:1|Estudiante|1     |Juan  |Perez    |
+----+-----+----------+------+------+---------+
```

De esta manera, hemos actualizado el apellido de Carlos Ramirez a 'Rodriguez' en la base de datos.

17. Conclusiones