Sistemas de almacenamiento y procesado distribuido

Tema 2. Almacenamiento de datos

© 2023 Javier Esparza Peidro - jesparza@dsic.upv.es

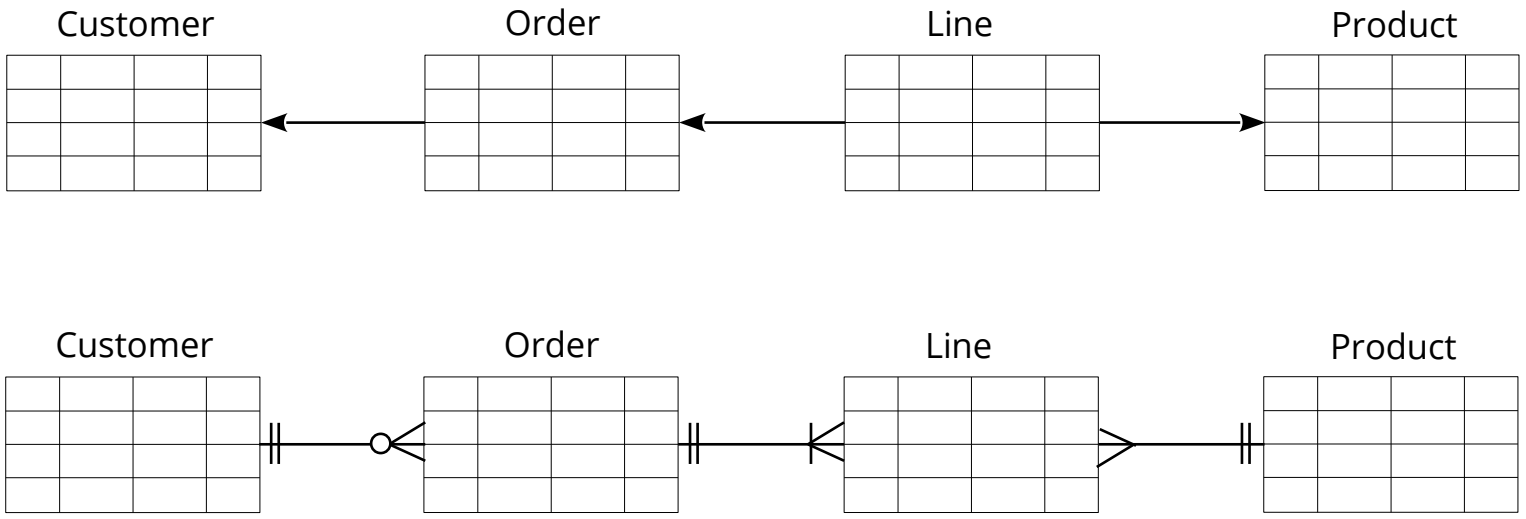
Contenido

- Introducción
- Agregados
  - Almacenes clave-valor
  - Almacenes de documentos
  - Almacenes de familias de columnas
- Indexación de datos
- Distribución de datos
- Persistencia políglota

Introducción

Modelo relacional

- Las bases de datos relacionales han dominado (y siguen) el mercado durante décadas
- Tablas (relaciones) con filas (tuplas), restricciones de integridad (dominio, clave, entidad, referencial)



Introducción

Modelo relacional

- Presentan muchas ventajas
  - Organizan la información de manera eficiente
  - Gestionan adecuadamente la concurrencia
  - Consistencia fuerte
  - Utilizan un modelo de datos estándar
  - Sirven para integrar múltiples aplicaciones

Introducción

Modelo relacional > MySQL

- https://www.mysql.com/
- Motor de bases de datos relacional clásico
- El servidor se ejecuta en un proceso independiente y sirve las consultas recibidas a través de la red
- Instalaremos la versión MySQL Community Server
- `mysqld` (servidor), `mysql` (CLI-*Command Line Interface*)

```
> mysqld
> mysql -u root -p
mysql> help
mysql> show databases;
mysql> use test;
```

```
mysql> create table xxx(...);
mysql> insert into xxx
       values(...);
mysql> select * from xxx;
mysql> quit
```

Introducción

Modelo relacional > MySQL > Python

- Se utiliza el driver MySQL Connector/Python
- `> pip install mysql-connector-python`
- Seguimos siempre los mismos pasos:
  1. Importar módulo `mysql.connector`
  2. Crear conexión contra la base de datos
  3. Ejecutar la consulta/sentencia y procesar resultados
  4. Confirmar los cambios, si es una actualización
  5. Cerrar conexión

Introducción

Modelo relacional > MySQL > Python

1. Importar módulo `mysql.connector`

```
import mysql.connector
```

2. Crear conexión contra la base de datos

```
con = mysql.connector.connect(
  host="localhost",
  user="root",
  password="root",
  database="test",
  port= 3306
)
```

Introducción

Modelo relacional > MySQL > Python

3. Ejecutar la consulta/sentencia y procesar resultados

```
cur = con.cursor()
cur.execute("create table users(id INT, name
VARCHAR(32))")
cur.execute("insert into users values (1, 'pepe')")
```

4. Confirmar cambios

```
con.commit()
```

5. Cerrar conexión

```
con.close();
```

Introducción

Modelo relacional > MySQL > Python

```
import mysql.connector
con = mysql.connector.connect(
  host="localhost",
  user="root",
  password="root",
  database="test",
  port= 3306
)
cur = con.cursor()
cur.execute("select * from users")
res = cur.fetchall()
for x in res: print(x)
con.close();
```

Introducción

Modelo relacional > SQLite

- https://www.sqlite.org
- Pequeño motor SQL compacto y versátil en C
- Incrustado en navegadores, móviles, etc.
- Serverless: no requiere proceso servidor
- Todos los datos se guardan en un fichero
- `sqlite3` (CLI-Command Line Interface)

Introducción

Modelo relacional > SQLite > Python

- Se utiliza el driver sqlite3
- Seguimos siempre los mismos pasos:
  1. Importar módulo `sqlite3`
  2. Crear conexión contra la base de datos
  3. Ejecutar la consulta/sentencia y procesar resultados
  4. Confirmar los cambios, si es una actualización
  5. Cerrar conexión

Introducción

Modelo relacional > SQLite > Python

1. Importar módulo `sqlite3`

```
import sqlite3
```

2. Crear conexión contra la base de datos

```
con = sqlite3.connect("test.db")
```

Introducción

Modelo relacional > SQLite > Python

3. Ejecutar la consulta/sentencia y procesar resultados

```
cur = con.cursor()
cur.execute("create table users(id INT, name
VARCHAR(32))")
cur.execute("insert into users values (1, 'pepe')")
```

4. Confirmar cambios

```
con.commit()
```

5. Cerrar conexión

```
con.close();
```

Introducción

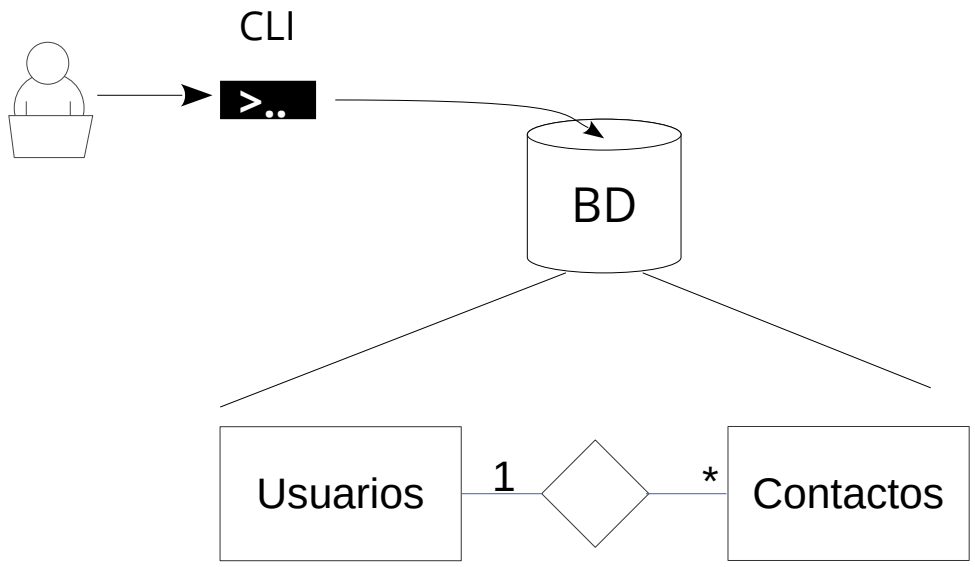
Modelo relacional > SQLite > Python

```
import sqlite3
con = sqlite3.connect("test.db")
cur = con.cursor()
cur.execute("select * from users")
res = cur.fetchall()
for x in res: print(x)
con.close();
```

Introducción

Ejercicio 1 > Bases de datos relacionales

- Diseñar una base de datos relacional (MySQL/SQLite) para una aplicación de contactos (usuarios, contactos)
- Implementar operaciones CRUD en herramienta CLI
  - Listar
  - Añadir
  - Actualizar
  - Eliminar



Introducción

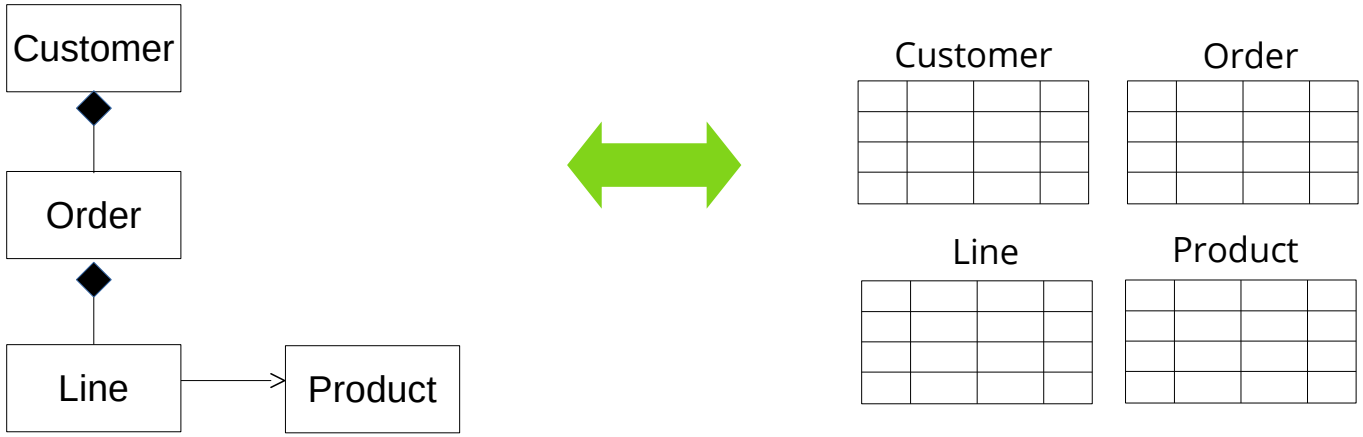
Modelo relacional > Problemas

- Presentan limitaciones importantes
  - Impedance mismatch
  - Escalabilidad
  - Anti-patrón para la integración de aplicaciones

Introducción

Modelo relacional > Problemas

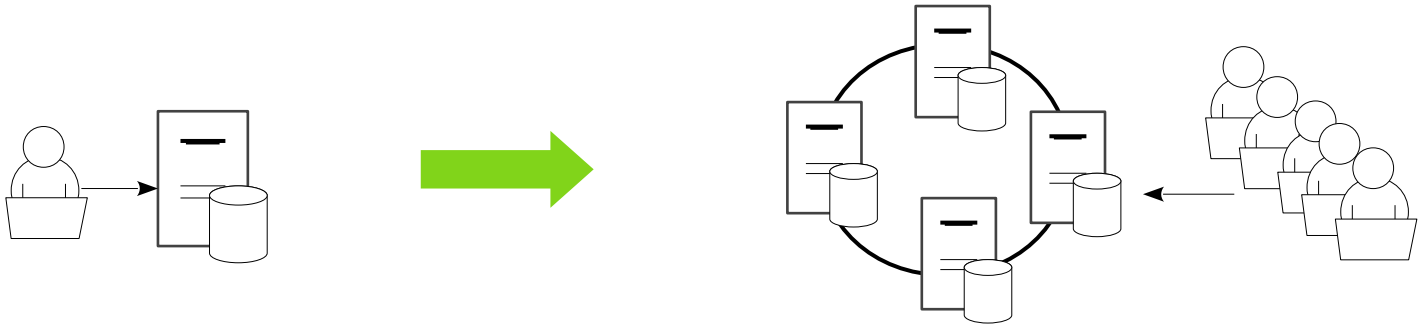
- Presentan limitaciones importantes
  - Impedance mismatch
    - El modelo de datos que se utiliza en la aplicación es diferente al modelo de datos relacional
    - Una tabla sólo soporta valores simples, sin estructura
    - Es necesario efectuar traducciones



Introducción

Modelo relacional > Problemas

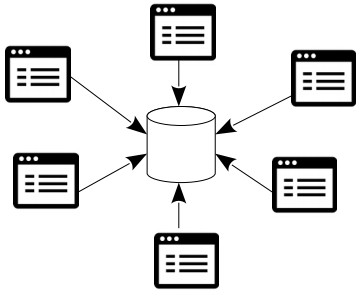
- Presentan limitaciones importantes
  - Escalabilidad
    - En cuanto al volumen de la información (Big Data)
    - En cuanto a la carga
    - Es necesario escalar: vertical vs horizontal
    - Las bases de datos relacionales son muy difíciles de escalar horizontalmente, hacen demasiado trabajo (transacciones, consistencia, integridad, etc.)



Introducción

Modelo relacional > Problemas

- Presentan limitaciones importantes
  - Anti-patrón para la integración de aplicaciones
    - La complejidad de la estructura de la base de datos es directamente proporcional al número de aplicaciones cliente
    - Modificar la estructura requiere mucho esfuerzo; en ocasiones no es posible
    - No todos los datos demandan las mismas garantías



Introducción

Buscando alternativas ...

- Se necesitan nuevas soluciones
  - Utilizar agregados
  - Distribuir los datos
  - Arquitecturas orientadas a servicios

Introducción

Buscando alternativas ...

- Se necesitan nuevas soluciones
  - Utilizar agregados
    - Intenta resolver el problema de impedance mismatching
    - Incrementar la productividad del desarrollador
    - No existe proceso de traducción
    - Distintos tipos: clave-valor, documentos, columna, etc.

Introducción

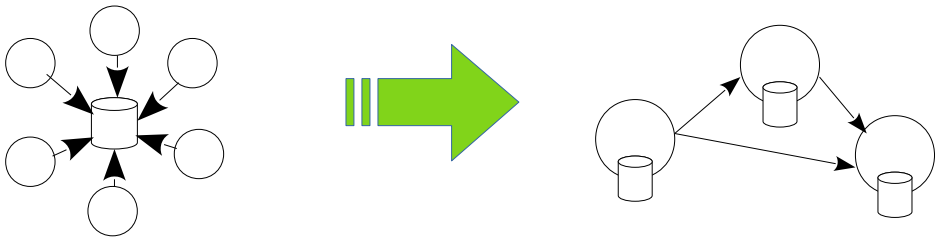
Buscando alternativas ...

- Se necesitan nuevas soluciones
  - Utilizar agregados
  - Distribuir los datos
    - Intenta resolver el problema de escalabilidad
    - Posibilita el tratamiento de datos a gran escala
    - Problemas de consistencia (Teorema CAP)
    - Relajar las garantías de los almacenes relacionales

Introducción

Buscando alternativas ...

- Se necesitan nuevas soluciones
  - Utilizar agregados
  - Distribuir los datos
  - Arquitecturas orientadas a servicios
    - Intenta resolver el problema de integración de aplicaciones
    - El sistema se rompe en servicios, cada servicio publica una interfaz y posee su base de datos privada
    - Utilizar distintos almacenes de datos para distintas circunstancias: persistencia políglota



Introducción

Buscando alternativas ...

- Se necesitan nuevas soluciones
  - Utilizar agregados
  - Distribuir los datos
  - Arquitecturas orientadas a servicios

Almacenes NoSQL

Introducción

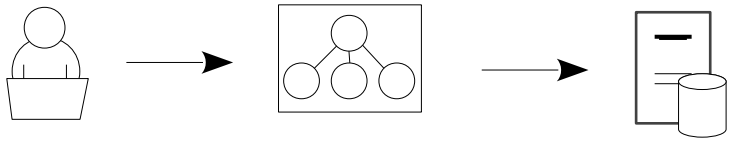
Almacenes NoSQL

- No existe definición formal: “Not Only SQL”
- Es un movimiento que abarca infinidad de iniciativas
- Propiedades comunes
  - No SQL como lenguaje de consulta/manipulación
  - Se suele trabajar con agregados
  - Los datos se organizan en esquemas flexibles
  - Preparados para trabajar en clúster
  - Relajan propiedades ACID en favor de rendimiento, escalabilidad y disponibilidad, propiedades BASE (Basically Available, soft state, eventually consistent)

Introducción

Almacenes NoSQL

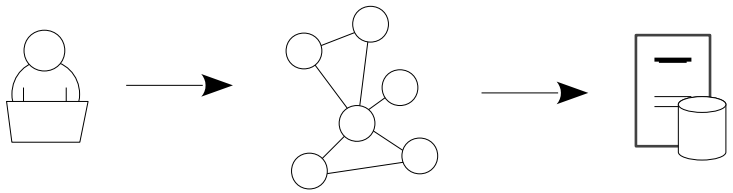
- Orientado a agregados vs orientado a grafos
  - La información relacionada se “empaqueta” en una unidad, el agregado
  - Se recuperan/manipulan agregados
  - El agregado es fácil de replicar-distribuir



Introducción

Almacenes NoSQL

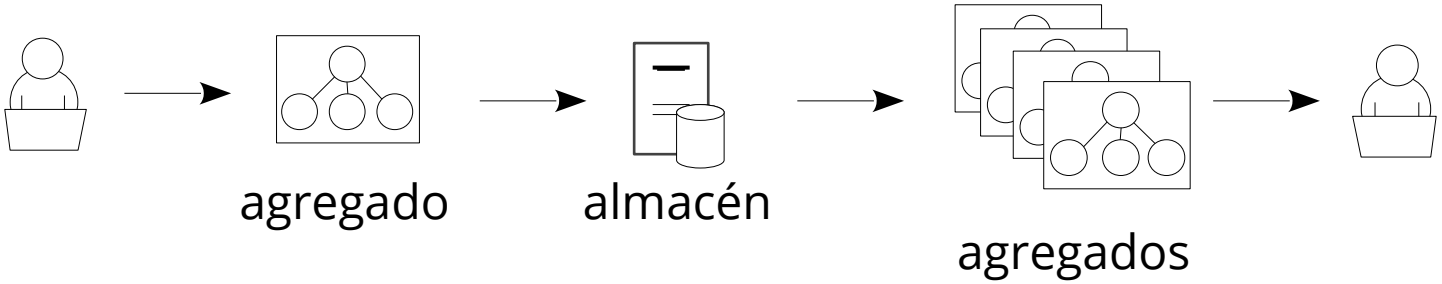
- Orientado a agregados vs orientado a grafos
  - Grafo: nodos + aristas
  - El énfasis está en las relaciones
  - Recuperación de relaciones complejas de manera eficiente



Agregados

¿Qué es?

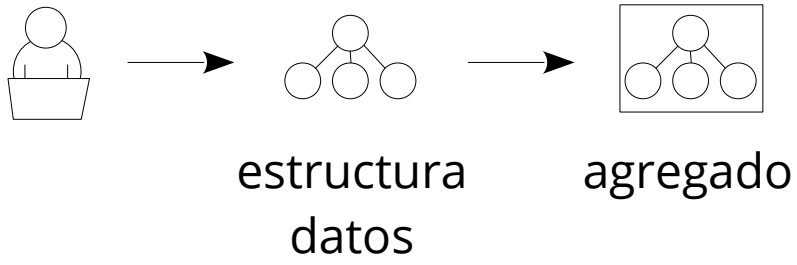
- Colección de objetos relacionados que tratamos como una unidad
  - Para su manipulación (CRUD)
  - Para su consistencia (operaciones atómicas)



Agregados

¿Qué es?

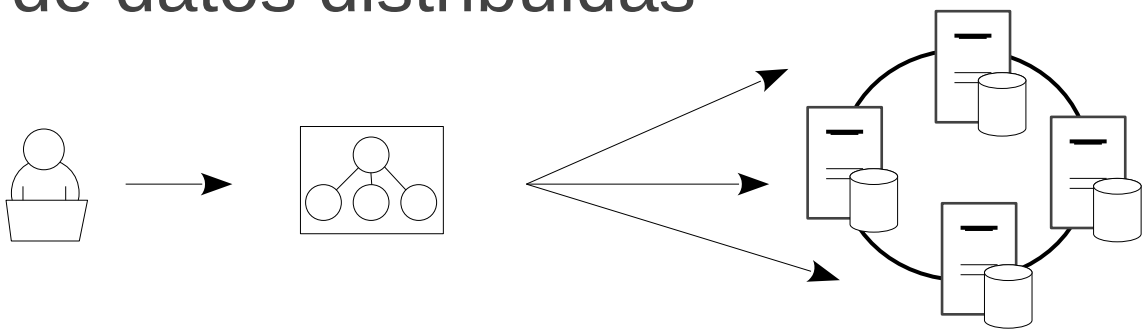
- Colección de objetos relacionados que tratamos como una unidad
  - Para su manipulación (CRUD)
  - Para su consistencia (operaciones atómicas)
- Más simple para el programador, su estructura coincide con las estructuras de datos en memoria



Agregados

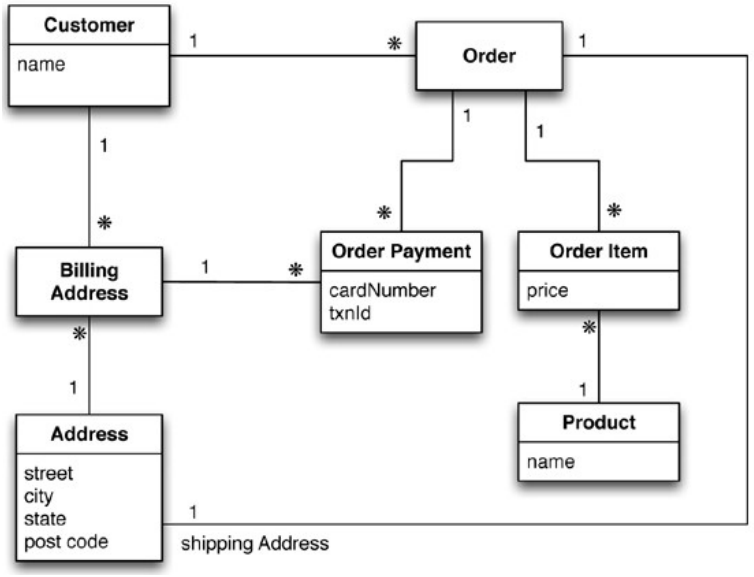
¿Qué es?

- Colección de objetos relacionados que tratamos como una unidad
  - Para su manipulación (CRUD)
  - Para su consistencia (operaciones atómicas)
- Más simple para el programador, su estructura coincide con las estructuras de datos en memoria
- Más fácil de gestionar (replicar, consistencia) en bases de datos distribuidas



Agregados

Ejemplo > sin agregados



Customer

Id	Name
1	Martin

Order

Id	CustomerId	ShippingAddressId
99	1	77

Product

Id	Name
27	NoSQL Distilled

BillingAddress

Id	CustomerId	AddressId
55	1	77

OrderItem

Id	OrderId	ProductId	Price
100	99	27	32.45

Address

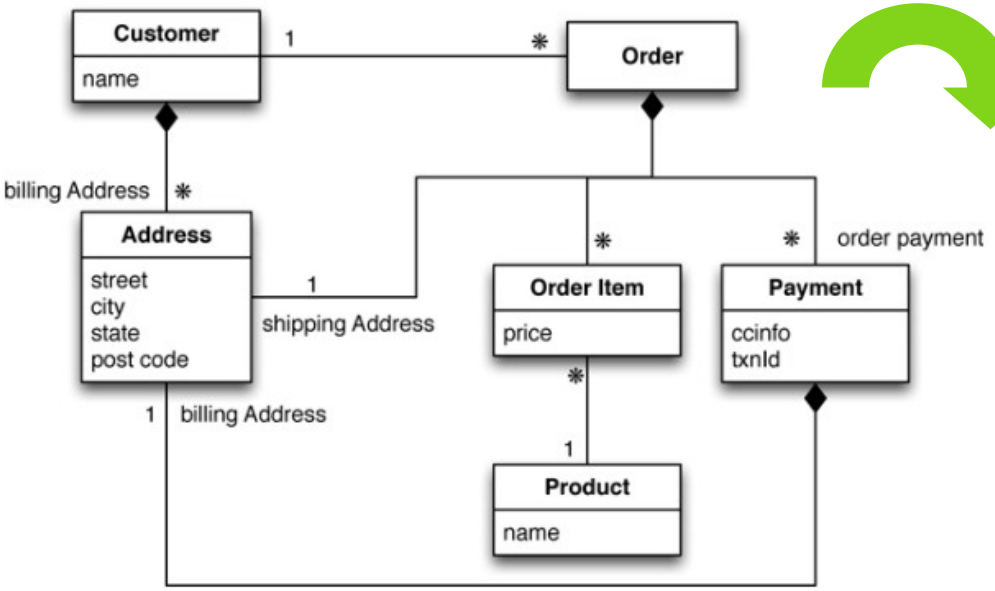
Id	City
77	Chicago

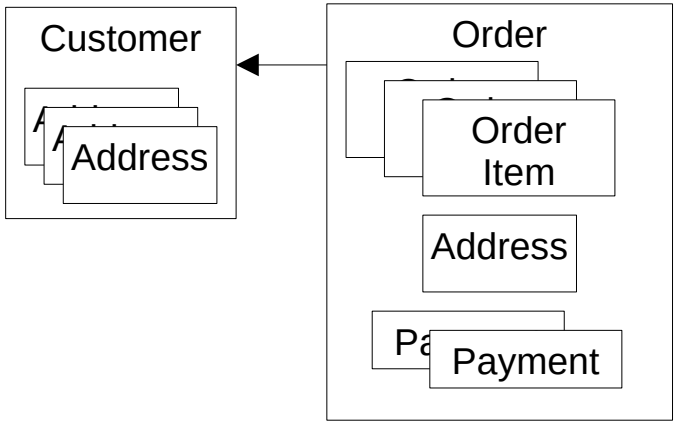
OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft

Agregados

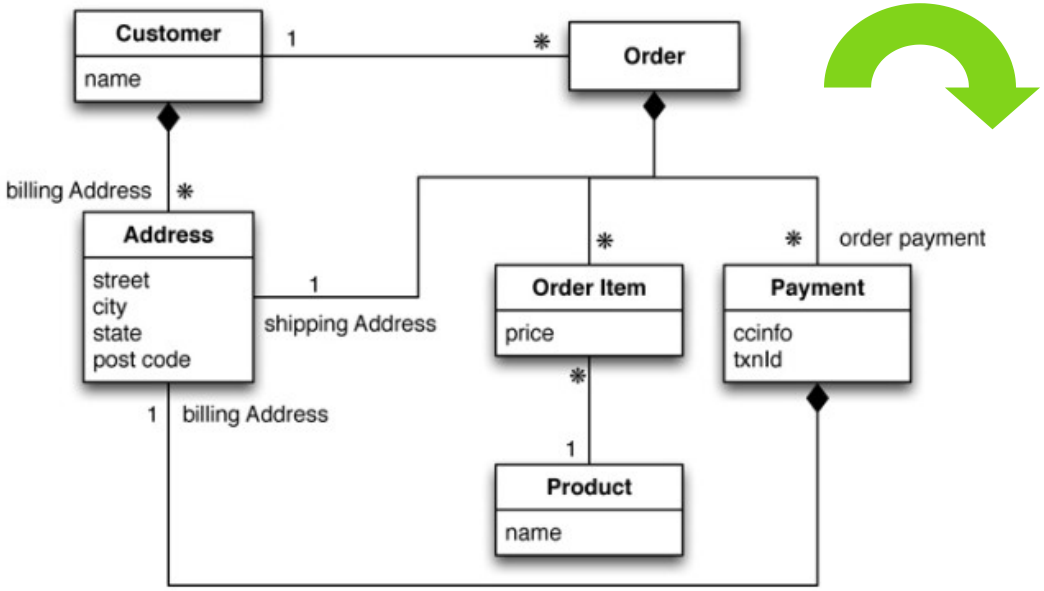
Ejemplo > con agregados (I)





Agregados

Ejemplo > con agregados (I)

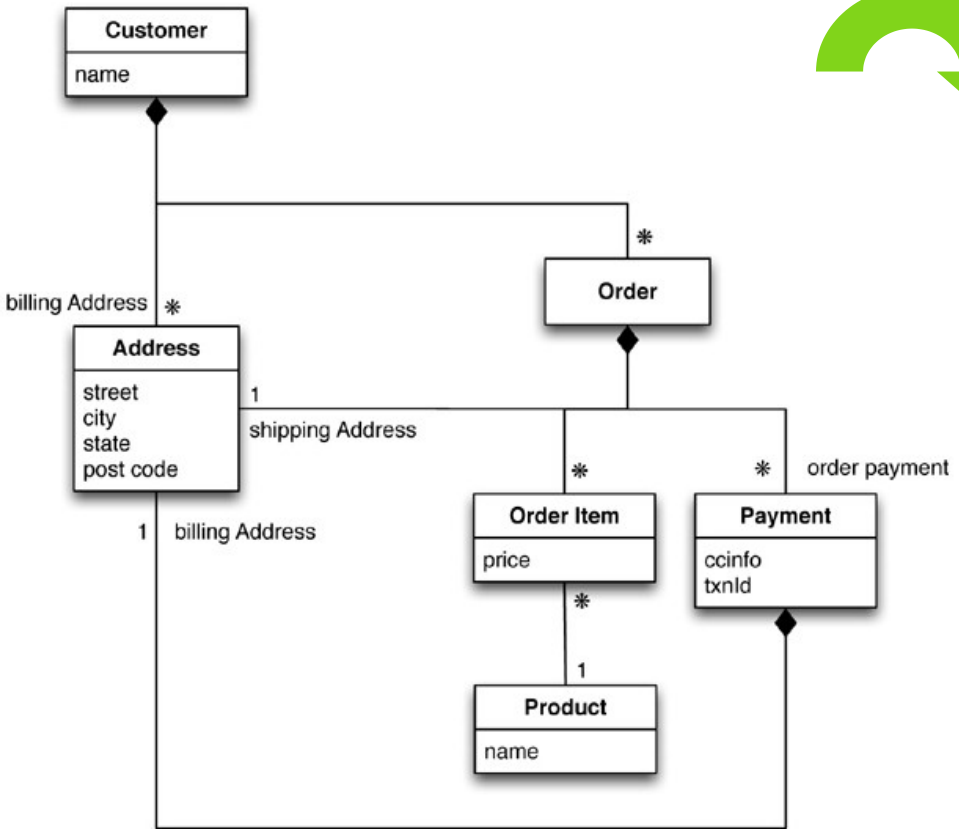


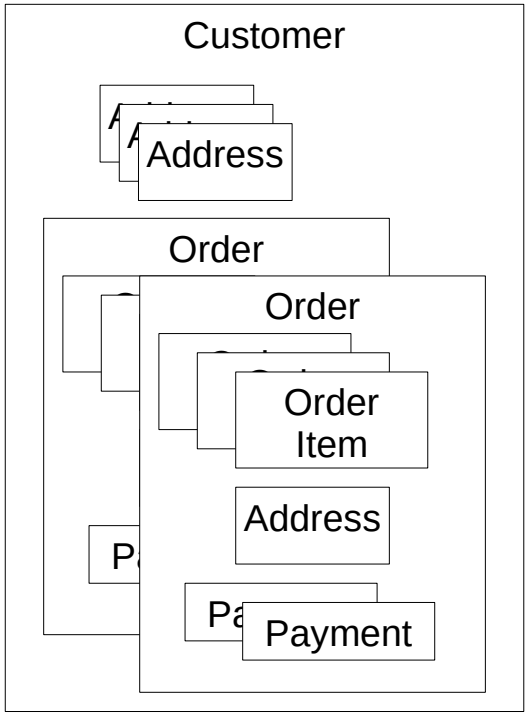

```
// in customers
{
"id":1,
"name":"Martin",
"billingAddress":[{"city":"Chicago"}]
}

// in orders
{
"id":99,
"customerId":1,
"orderItems":[
  {
  "productId":27,
  "price": 32.45,
  "productName": "NoSQL Distilled"
    }
  ],
"shippingAddress":[{"city":"Chicago"}]
"orderPayment":[
  {
    "ccinfo":"1000-1000-1000-1000",
    "txnId":"abelif879rft",
    "billingAddress": {"city": "Chicago"}
  }
  ],
}
```

Agregados

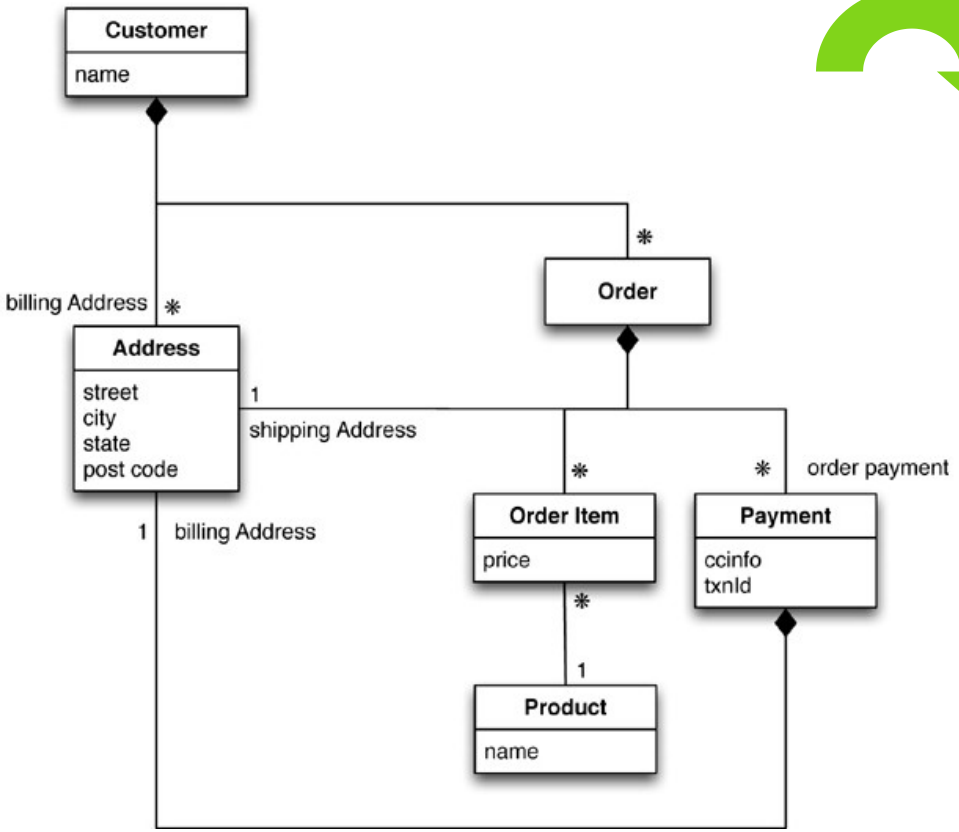
Ejemplo > con agregados (II)





Agregados

Ejemplo > con agregados (II)




```
// in customers
{
"id": 1,
"name": "Martin",
"billingAddress": [{"city": "Chicago"}],
"orders": [
  {
    "id":99,
    "orderItems":[
      {
       "productId":27,
       "price": 32.45,
       "productName": "NoSQL Distilled"
      }],
    "shippingAddress":[{"city":"Chicago"}]
    "orderPayment":[
      {
        "ccinfo":"1000-1000-1000-1000",
        "txnId":"abelif879rft",
        "billingAddress": {"city": "Chicago"}
      }],
  }]
}
```

Agregados

Consecuencias directas del agregado ...

- Guía el almacenamiento y la distribución de datos: determina qué datos se manipulan/almacenan juntos
- Ayuda en ciertas interacciones (e.g. recuperar pedido)
- Puede dificultar otras operaciones (e.g. analizar ventas de un producto en los últimos meses)
- Si el agregado no está claro, es mejor evitarlo (más versátil)
- Es difícil implementar garantías ACID sobre múltiples agregados, pero sí sobre uno solo

Agregados

Tipos de agregados

- Clave-valor
- Documentos
- Familias de columnas

Agregados

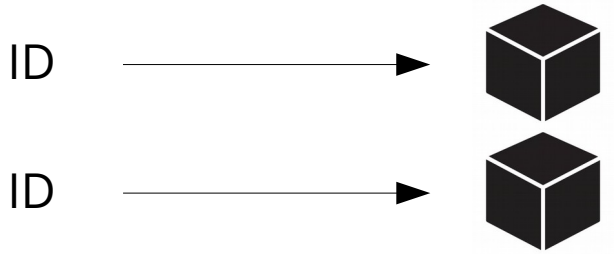
Tipos de agregados

- > Clave-valor
- Documentos
- Familias de columnas

Agregados

Tipos de agregados

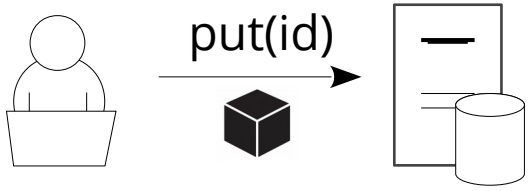
- Clave-valor
  - Cada agregado posee un ID único
  - El agregado es opaco (blob)

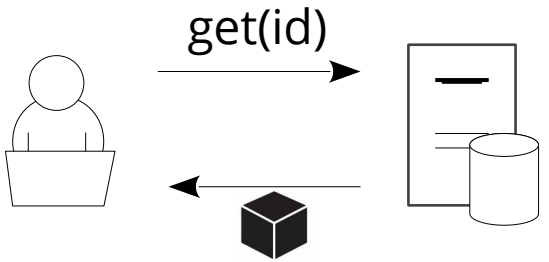


Agregados

Tipos de agregados

- Clave-valor
  - Cada agregado posee un ID único
  - El agregado es opaco (blob)
  - Recuperación/actualización por ID: get(), put()





Agregados

Almacenes clave-valor

- El almacén NoSQL más simple según su API
- Diccionario: put(key,value), get(key),delete(key)

- La clave suele ser un string, el valor es un blob
- Operaciones de actualización put(key,value) atómicas
- Algunos almacenes permiten recuperar (iterar) las claves
- Algunos almacenes permiten buscar en los valores (se parecen a los almacenes de documentos)

Agregados

Almacenes clave-valor

- Rendimiento muy alto y fácil de escalar
- Recomendados para almacenar info de sesiones, preferencias de usuario, carrito de la compra, etc.
- No recomendados para almacenar relaciones, transacciones con múltiples operaciones, consultas por valor, operaciones sobre conjuntos
- Ejemplos: Redis, DynamoDB, Memcached, etcd, Riak , ...

Agregados

Almacenes clave-valor > LevelDB

- https://github.com/google/leveldb
- Es un almacén clave-valor serverless muy rápido
- Inspirado por Bigtable (Google): chrome, autoCAD, ...
- Claves y valores de tipo string
- Las claves se almacenan en orden
- Soporta consultas de tipo rango
- Snapshots

Agregados

LevelDB > Python

- Instalar driver Plyvel

```
> pip install plyvel
```

- Ejemplo

```
import plyvel
db = plyvel.DB('./db', create_if_missing=True)
db.put(b'test', b'Hello wolrd!')
db.get(b'test')
db.delete(b'test')
db.close()

for key,value in db.iterator(start=b'start',
stop=b'end'):
  print(key)
```

Agregados

Almacenes clave-valor > Redis

- https://redis.io/
- Almacena toda la información en memoria
- Sirve la información de manera muy eficiente
- Cada cierto tiempo se escriben los cambios en disco (recuperación en caso de fallo)
- Soporta tipos de datos: string, tablas hash, listas, conjuntos, etc.
- Se utiliza como base de datos, cache, cola mensajes

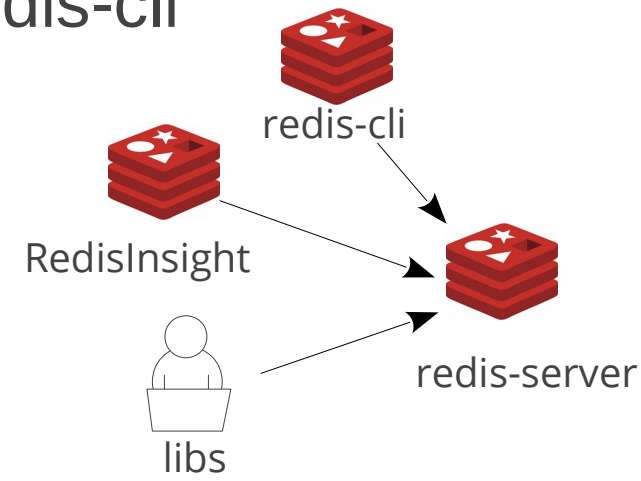
Agregados

Redis > Puesta en marcha

- (redis-server) > docker run -d --name redis-stack -p 6379:6379 -p 8001:8001 redis/redis-stack:latest
- (redis-cli) > docker exec -it redis-stack redis-cli

```
redis> set a a
redis> get a
redis> keys *
```

- (RedisInsight) http://localhost:8001
- (libs) > pip install redis

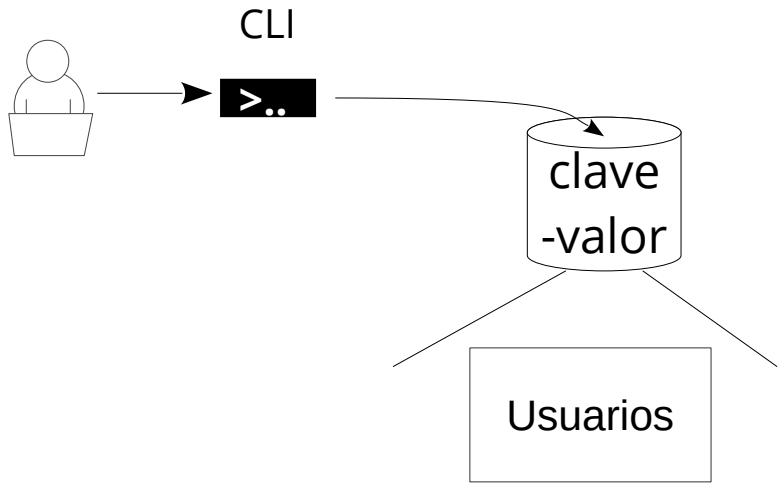



```
import redis
r = redis.Redis(host='localhost', port=6379, db=0)
r.set('test', 'Hello world!')
r.get('test')
keys = r.keys('*')
r.delete('a')
```

Agregados

Ejercicio 2 > Almacenes clave-valor

- Diseñar una aplicación CLI que permita gestionar un conjunto de usuarios en un almacén clave-valor (LevelDB/Redis)
  - Listar
  - Añadir
  - Actualizar
  - Eliminar



Agregados

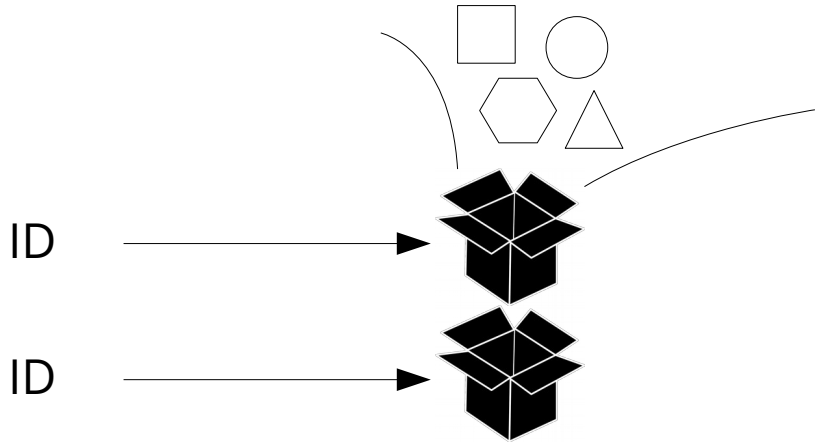
Tipos de agregados

- Clave-valor
- > Documentos
- Familias de columnas

Agregados

Tipos de agregados

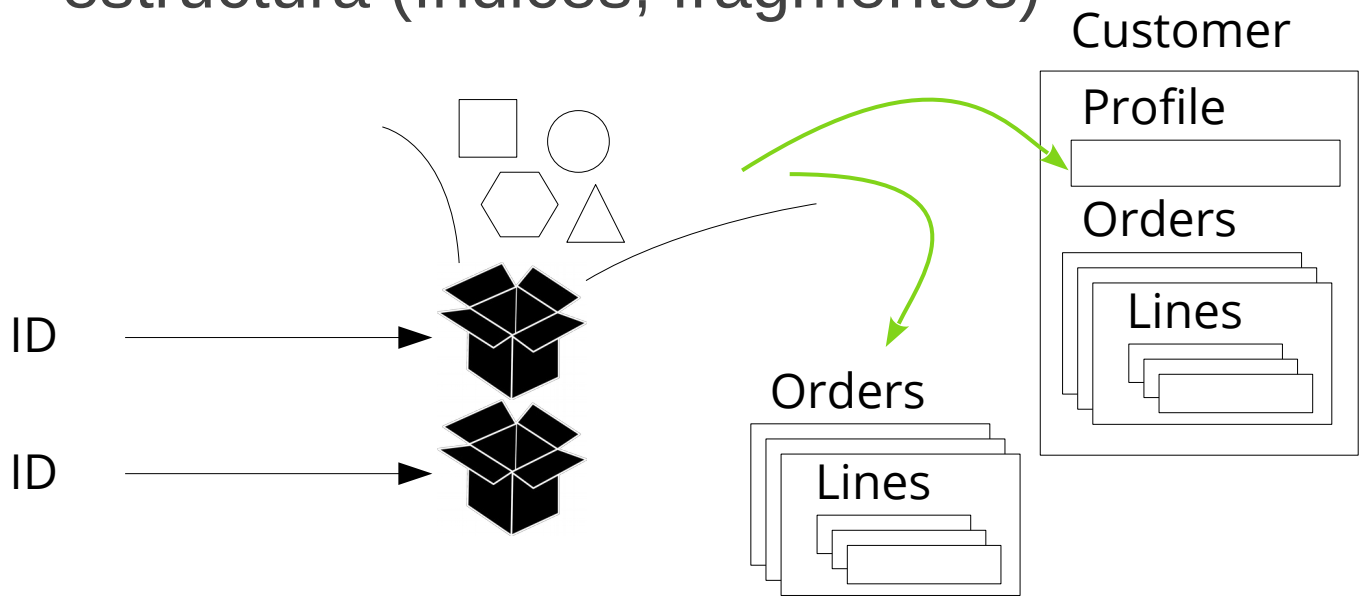
- Documentos
  - Cada agregado posee un ID único
  - El agregado tiene estructura interna



Agregados

Tipos de agregados

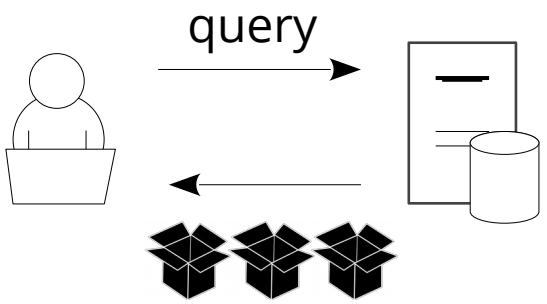
- Documentos
  - Cada agregado posee un ID único
  - El agregado tiene estructura interna
  - Recuperación/actualización por ID o en base a su estructura (índices, fragmentos)



Agregados

Tipos de agregados

- Documentos
  - Cada agregado posee un ID único
  - El agregado tiene estructura interna
  - Recuperación/actualización por ID o en base a su estructura (índices, fragmentos)
  - Suele proporcionarse un lenguaje de consulta sofisticado

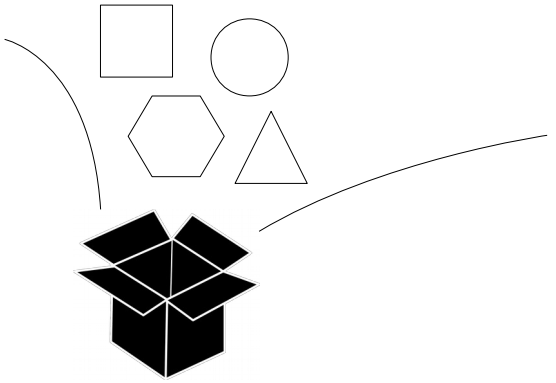


Agregados

Tipos de agregados

- Documentos vs clave-valor
  - La frontera es difusa
  - Depende del nivel de visibilidad de la estructura interna del agregado
  - Algunos almacenes clave-valor extienden con metainfo (Riak) para indexar, o permiten cierta estructura (listas, conjuntos) en el contenido (Redis)

VS



Agregados

Almacenes de documentos

- Almacenan documentos (objetos, XML, JSON, …)
- Los documentos no precisan la misma estructura
- Consultas basadas en la estructura de los documentos: lenguaje de consulta
- Recuperaciones parciales: proyecciones
- Operaciones sobre un documento atómicas: transacciones atómicas; no hay soporte para transacciones que operen sobre múltiples documentos
- Ejemplos: MongoDB, CouchDB, DynamoDB, etc.

Agregados

Almacenes de documentos > MongoDB

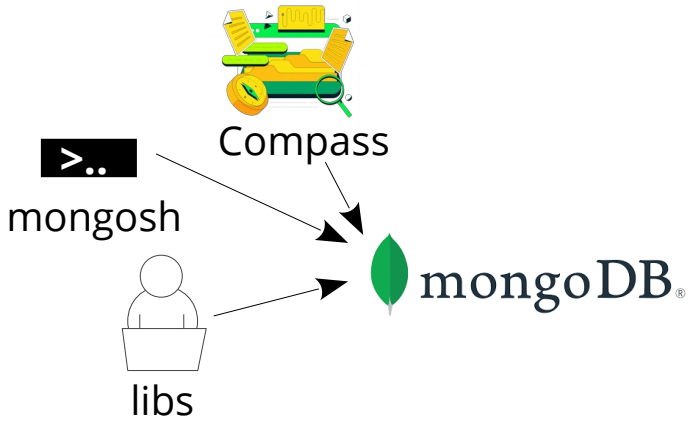
- https://www.mongodb.com/
- Repositorio de documentos NoSQL open source
- Características clave:
  - Alto rendimiento: índices muy rápidos
  - Alta disponibilidad: con replicación, recuperación automática
  - Escalado automático: utilizando el concepto de sharding (distribución horizontal de datos)

Agregados

MongoDB > Puesta en marcha

- (MongoDB) > docker run --name mongodb -d -p 27017:27017 mongodb/mongodb-community-server
- (mongosh) > docker exec -it mongodb mongosh
- (libs) > pip install pymongo

```
import pymongo
client = pymongo.MongoClient('localhost', 27017)
users = client.mydb.users.find()
for usr un users: print(usr.email)
client.close()
```



Agregados

MongoDB > MongoDB shell

- Intérprete interactivo de JavaScript que nos permite interactuar con el servidor

```
> help
```

Lista comandos disponibles

- mongo vs mongosh
- Permite acceder a MongoDB utilizando una API JS
- No es la misma API que el driver de Python (parecido)
- Muy útil para explorar y administrar el servidor
- Lo usaremos en los próximos ejemplos

Agregados

MongoDB > Base de datos

- Listar todas las bases de datos

```
> show dbs
```

- Seleccionar una base de datos para trabajar con ella

```
> use mibd
```

- La base de datos actual siempre está en la variable db
- Una base de datos se crea la primera vez que se inserta en ella

```
> use nuevabd    > db.micoleccion.insert({ ... })
```

Agregados

MongoDB > Documentos

- Un registro en MongoDB es un documento
- Un documento es una estructura de datos compuesta por pares campo-valor
- Muy parecido a un objeto JSON

```
{
  nombre: "Pepe",
  edad: 30,
  estado: "casado",
  grupos: ["deportes", "noticias"]
}
```

campo: valor

Agregados

MongoDB > Documentos

- Los documentos son almacenados en formato BSON (representación binaria de JSON)
- El valor de un campo puede ser cualquier tipo de datos soportado por BSON, otro documento, un array o un array de documentos

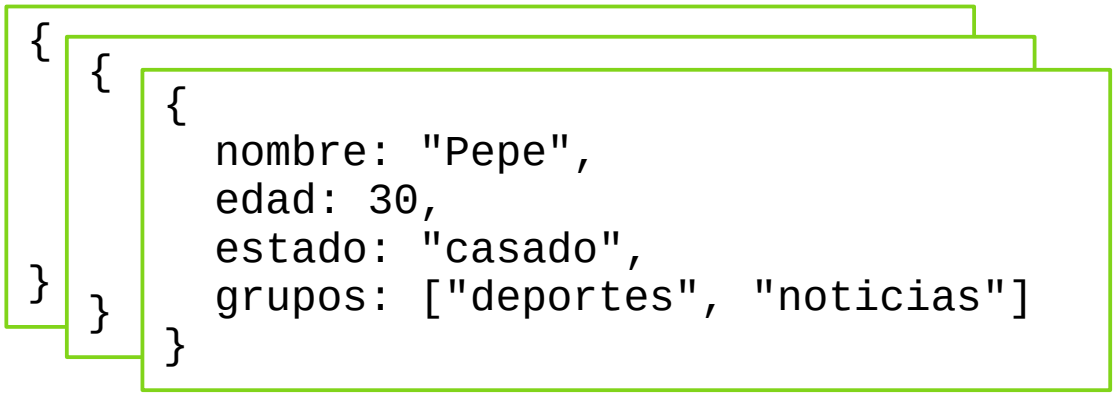
```
{
  nombre: "Pepe",
  direccion : {
    calle : "El Cid",
    cp : "03800",
    coord : [ -73.9557413, 40.7720266 ],
   }
}
```

Documento embebido (subdocumento)

Agregados

MongoDB > Colecciones

- Los documentos se guardan en colecciones
- Una colección es el análogo a una tabla en bbdd relacionales
- Una colección no fuerza ninguna estructura (esquema) en sus documentos (pueden ser diferentes)
- Generalmente todos los documentos en una colección poseen una estructura similar, y comparten índices



colección

Agregados

MongoDB > Colecciones

- Mostrar las colecciones de la base de datos actual

```
> show collections
```

- Acceder a una colección en la base de datos actual

```
> db.usuarios
```

Agregados

MongoDB > Colecciones > Insertar

- Usamos db.collection.insertOne(doc, [opts])
- doc: documento a insertar
- opts: opcional. Opciones adicionales

```
> db.usuarios.insertOne({ "nombre": "Pepe",
    "edad": "30", "estado": "casado" });
```

- Si la colección no existe se creará automáticamente
- Devuelve info sobre el documento insertado (insertedId)
- db.collection.insertMany()

Agregados

MongoDB > Colecciones > Insertar 

- Los documentos almacenados en una colección deben tener un identificador único _id (ObjectId) que es la clave primaria en dicha colección
- Si no los genera el cliente, lo generará automáticamente el servidor

```
> db.usuarios.insertOne({ _id: ObjectId(),
   "nombre": "Pepe", "edad": "30",
   "estado": "casado" });
```

Agregados

MongoDB > Colecciones > Buscar

- Usamos db.collection.find([query], [projection])
  - query: opcional. Criterios de búsqueda
  - projection: opcional. Limita los campos a recuperar
- Obtenemos un cursor con los resultados

```
> var cursor = db.usuarios.find();
```

- En MongoDB shell se iteran y muestran automáticamente los primeros 20 documentos. Para mostrar más usar it

```
> db.usuarios.find();
> it
```

Agregados

MongoDB > Colecciones > Buscar 

- Cursor
- Apunta a los resultados de una query
- cursor.next() cursor.hasNext()
- Otros métodos

```
> var cursor = db.usuarios.find();
> while (cursor.hasNext()) {
    print(cursor.next());
}
```

Agregados

MongoDB > Colecciones > Buscar

- Query
- Especificar condiciones de igualdad

```
{ <campo1>: <valor1>, <campo2>: <valor2>, ...}
```

```
> db.usuarios.find({_id: 1});
> db.usuarios.find({nombre: 'XXX', apellidos:
'YYY'});
```

- Si <campo> está dentro de un documento embebido entonces se especifica con ., y con comillas

```
> db.usuarios.find({nombre: 'XXX',
'direccion.ciudad': 'Alcoy'});
```

Agregados

MongoDB > Colecciones > Buscar

- Query
- Especificar condiciones con operadores de consulta

```
> db.usuarios.find({edad: {$gt: 50} });
> db.productos.find({
    color: {$in: ['rojo','verde']}
  });
```

- $eq, $gt, $gte, $lt, $lte, $ne, $in, $nin
- $or, $and, $not, $nor
- $exists, $type, $mod, $regexp, $text, $where, $all, $elementMatch, $size, $slice, ...

Agregados

MongoDB > Colecciones > Modificar

- Usamos db.collection.updateOne(query, update, [opts])
  - query: determina los documentos a modificar; usamos las mismas condiciones que con las búsquedas
  - update: especifica las modificaciones a aplicar
  - opts: opcional. Opciones adicionales

```
> db.usuarios.updateOne({ _id: "0001"},
  { $set: { "name", "Juan" } }
);
```

query

modificaciones

- Devuelve(.matchedCount,.modifiedCount,.upsertedId,..)
- db.collection.updateMany()

Agregados

MongoDB > Colecciones > Modificar

- Modificaciones
- Se especifican con operadores de modificación : $inc, $mul, $rename, $setOrInsert, $set, $unset, $min, $max, $currentDate, $addToSet, $pop, $pull, $each, $slice, $sort, …

```
> db.usuarios.update( { name: "Pepe"},
  { $set: { "name", "Juan" }, $inc: { "edad": 1 }}
);
```

- Algunos operadores crean el campo si no existe
- Las modificaciones son atómicas en un documento

Agregados

MongoDB > Colecciones > Eliminar

- Usamos db.collection.deleteOne(query,opts)
  - query: determina los documentos a eliminar; usamos las mismas condiciones que con las búsquedas
  - opts: opcional. Opciones adicionales

```
> db.usuarios.deleteOne( { name: "Pepe"} );
```

- Devuelve info (.deletedCount,…)
- db.collection.deleteMany()

Agregados

MongoDB > Python

- Hasta ahora hemos accedido a MongoDB usando el MongoDB shell
- Para acceder a MongoDB desde Python es necesario utilizar el driver adecuado PyMongo
- Similar a MongoDB shell (JS) pero distinta API
- Instalación

```
> pip install pymongo
```

Agregados

MongoDB > Python

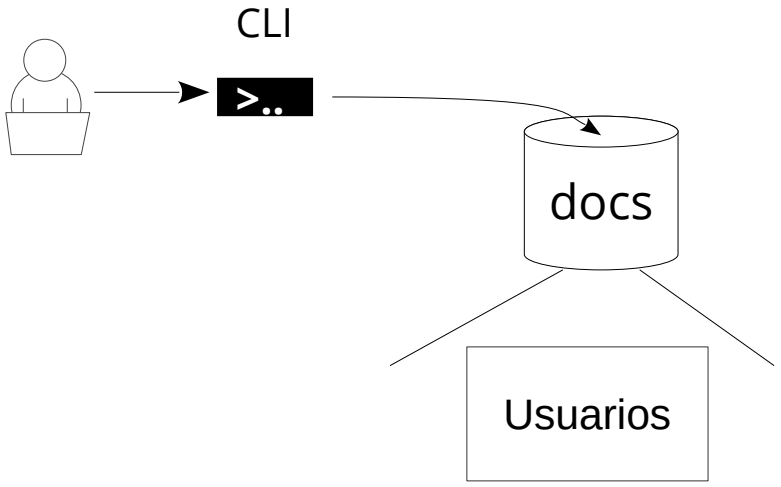
- Ejemplo

```
import pymongo
client = pymongo.MongoClient('localhost', 27017)
db = client.mydb  # db = client['mydb']
col = db.mycollection  # col = db['mycollection']
db.usuarios.insert_one({'nombre': 'Pepe', 'edad': 50});
cur = db.usuarios.find(filter={'edad': {'$gt': 50}},
      projection={'nombre':1}, 'skip'=1, 'limit'=1)
for doc in cur: print(doc.nombre)
db.usuarios.update_many(
  {'nombre': 'Pepe'},{'$set':{'edad':50})
db.usuarios.delete_many({'nombre': 'Pepe'})
client.close()
```

Agregados

Ejercicio 3 > Almacenes de documentos

- Diseñar una aplicación CLI que permita gestionar un conjunto de usuarios en un almacén de documentos (MongoDB)
  - Listar
  - Añadir
  - Actualizar
  - Eliminar



Agregados

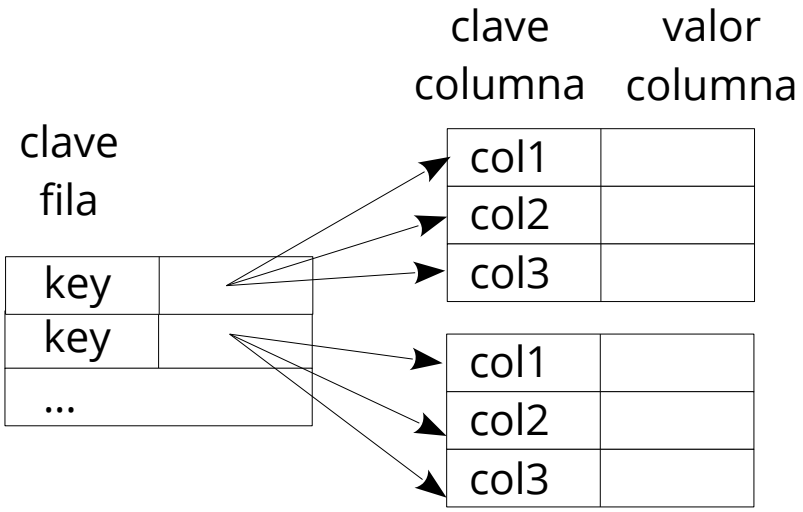
Tipos de agregados

- Clave-valor
- Documentos
- > Familias de columnas

Agregados

Tipos de agregados

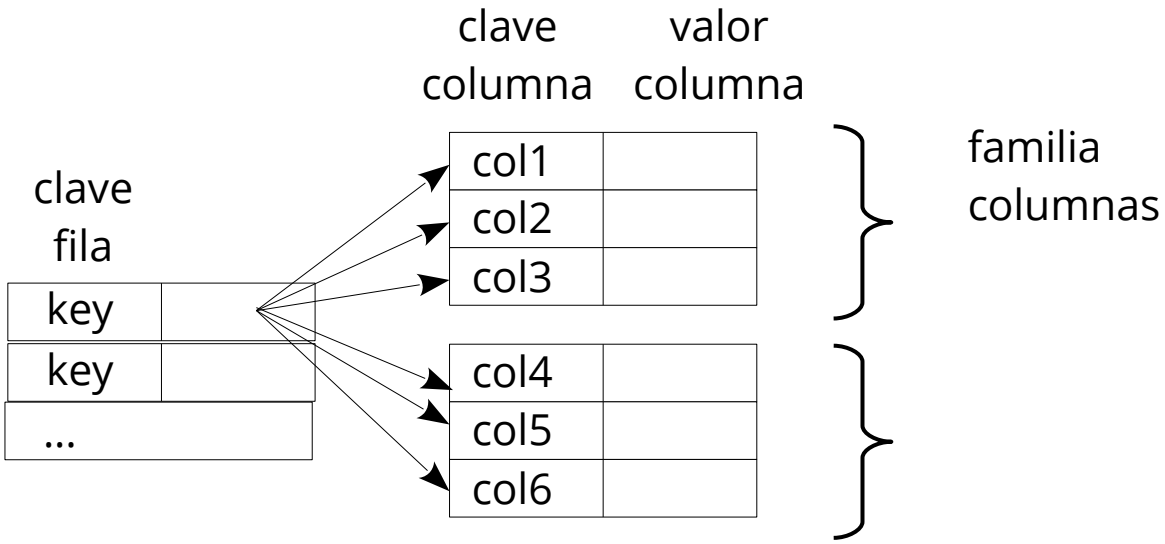
- Familias de columnas
  - El agregado es accedido a través de una clave única
  - El agregado contiene una colección de columnas
  - Es un mapa de dos niveles: primer nivel por clave, devuelve un mapa de columnas



Agregados

Tipos de agregados

- Familias de columnas
  - Las columnas se agrupan en famillias
  - Datos en una familia se acceden conjuntamente



Agregados

Almacenes de familias de columnas

- Utilizan tablas, filas y columnas
- En una tabla, cada fila posee una clave única
- En una tabla, cada fila puede contener un número variable de columnas (billones de columnas dinámicas)
- Operaciones de join no están disponibles
- Ejemplos: Bigtable, Cassandra, HBase, etc.

Agregados

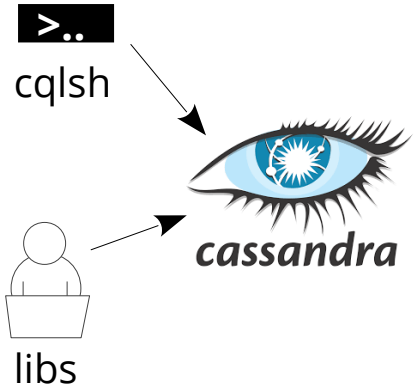
Almacenes de familias de columnas > Cassandra

- https://cassandra.apache.org/
- Almacén de familias de columnas open source
- Facebook: combinación de Dynamo y Bigtable
- Características clave:
  - Distribuido y descentralizado: clúster de nodos con el mismo rol, se pueden añadir/eliminar de manera dinámica
  - Particionado: los datos se distribuyen a partir de su clave
  - Tolerante a fallos: los datos se replican automáticamente en distintos nodos

Agregados

Cassandra > Puesta en marcha

- (cassandra-server) > docker run -d --name cassandra -p 9042:9042 cassandra
- (cqlsh) > docker exec -it cassandra cqlsh
- (libs) > pip install cassandra-driver




```
from cassandra.cluster import Cluster
cluster = Cluster(['192.168.1.1'])
session = cluster.connect('mykeyspace')
rows = session.execute("SELECT name,age,email FROM users")
for row in rows:
  print(row.name, row.age, row.email)
session.shutdown()
```

Agregados

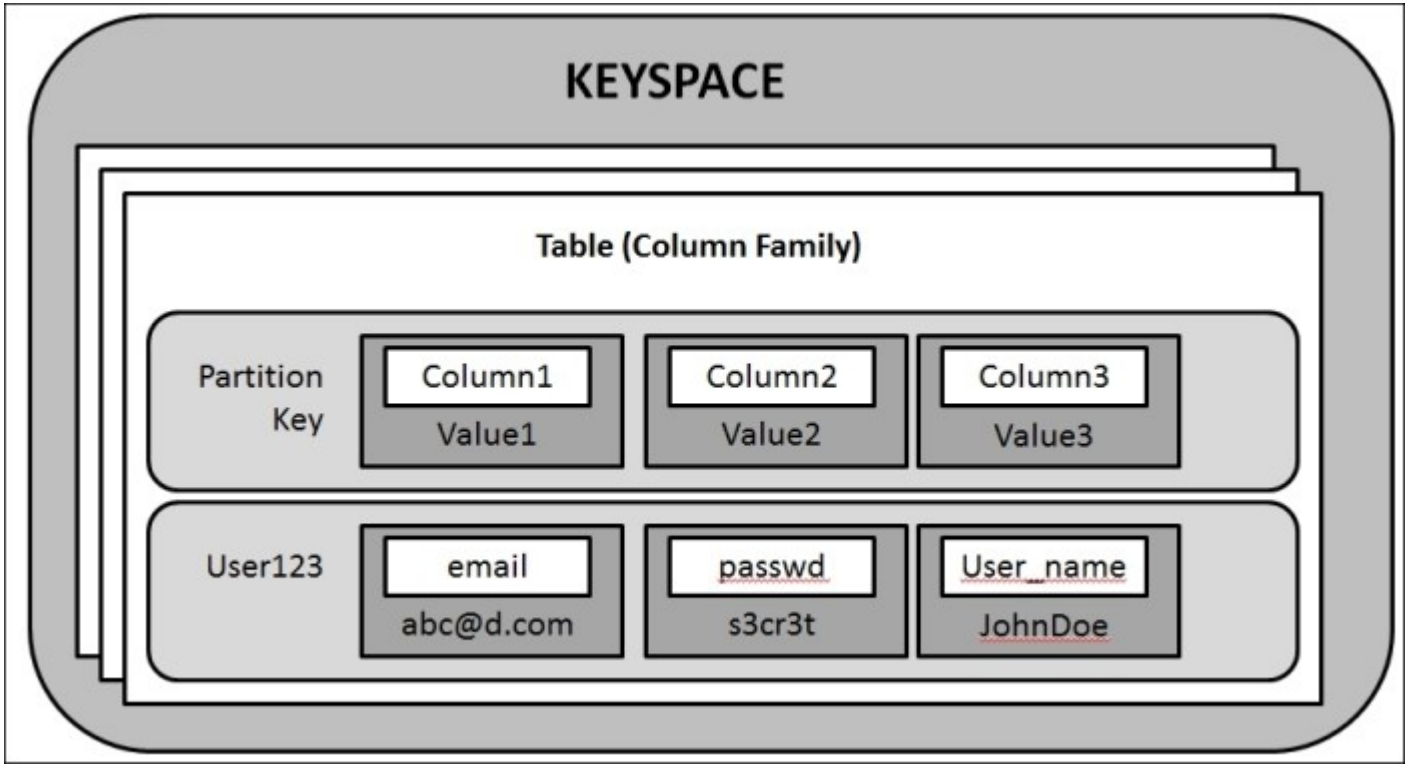
Cassandra > CQLSH

- Cliente que permite interactuar con Cassandra
- Utiliza CQL (Cassandra Query Language)
  - Muy parecido a SQL: tablas, filas y columnas
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)

Agregados

Cassandra > Modelo de datos

- Un keyspace contiene tablas, una tabla contiene particiones, una partición contiene filas, una fila contiene columnas



Agregados

Cassandra > Keyspace

- Contiene la configuración de un clúster

```
> DESCRIBE keyspaces;
> CREATE KEYSPACE IF NOT EXISTS mystore WITH REPLICATION =
{'class' : 'SimpleStrategy', 'replication_factor' : '1' };
> DESCRIBE keyspaces;
> USE mystore;
> DROP KEYSPACE mykeyspace;
```

Agregados

Cassandra > Tablas

- Contiene las filas

```
> DESCRIBE tables;
> CREATE TABLE IF NOT EXISTS users (id int PRIMARY KEY,
name text, email text, age int);
> DESCRIBE tables;
> DESCRIBE table users;
> DROP TABLE users;
```

Agregados

Cassandra > Filas y columnas

- Contienen los datos

```
> INSERT INTO users(id,name,email,age) VALUES
(1,'pepe','pepe',40);
> UPDATE users SET name='juan' WHERE id=1;
> DELETE users WHERE id=1;
> SELECT * FROM users;
```

Agregados

Cassandra > Python

- Se configura la conexión con el clúster, se abre una sesión sobre un keyspace, se utiliza CQL

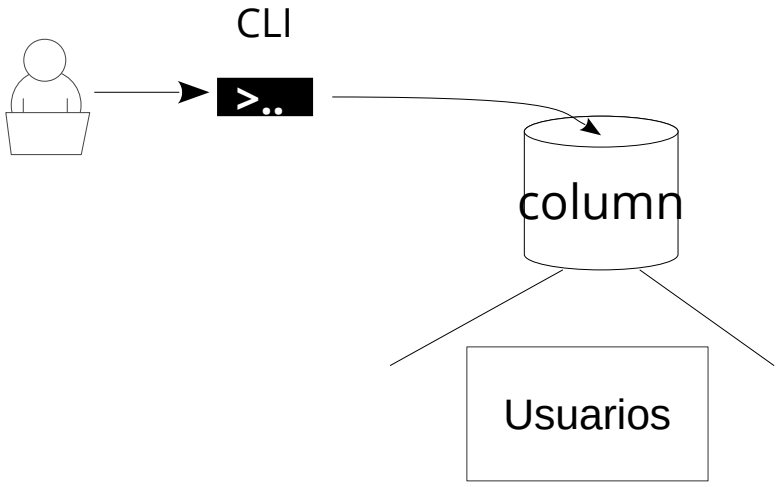
```
> pip install cassandra-driver

from cassandra.cluster import Cluster
cluster = Cluster(['192.168.1.1'])
session = cluster.connect('mykeyspace')
rows = session.execute("SELECT name,age,email FROM users")
for row in rows:
  print(row.name, row.age, row.email)
session.shutdown()
```

Agregados

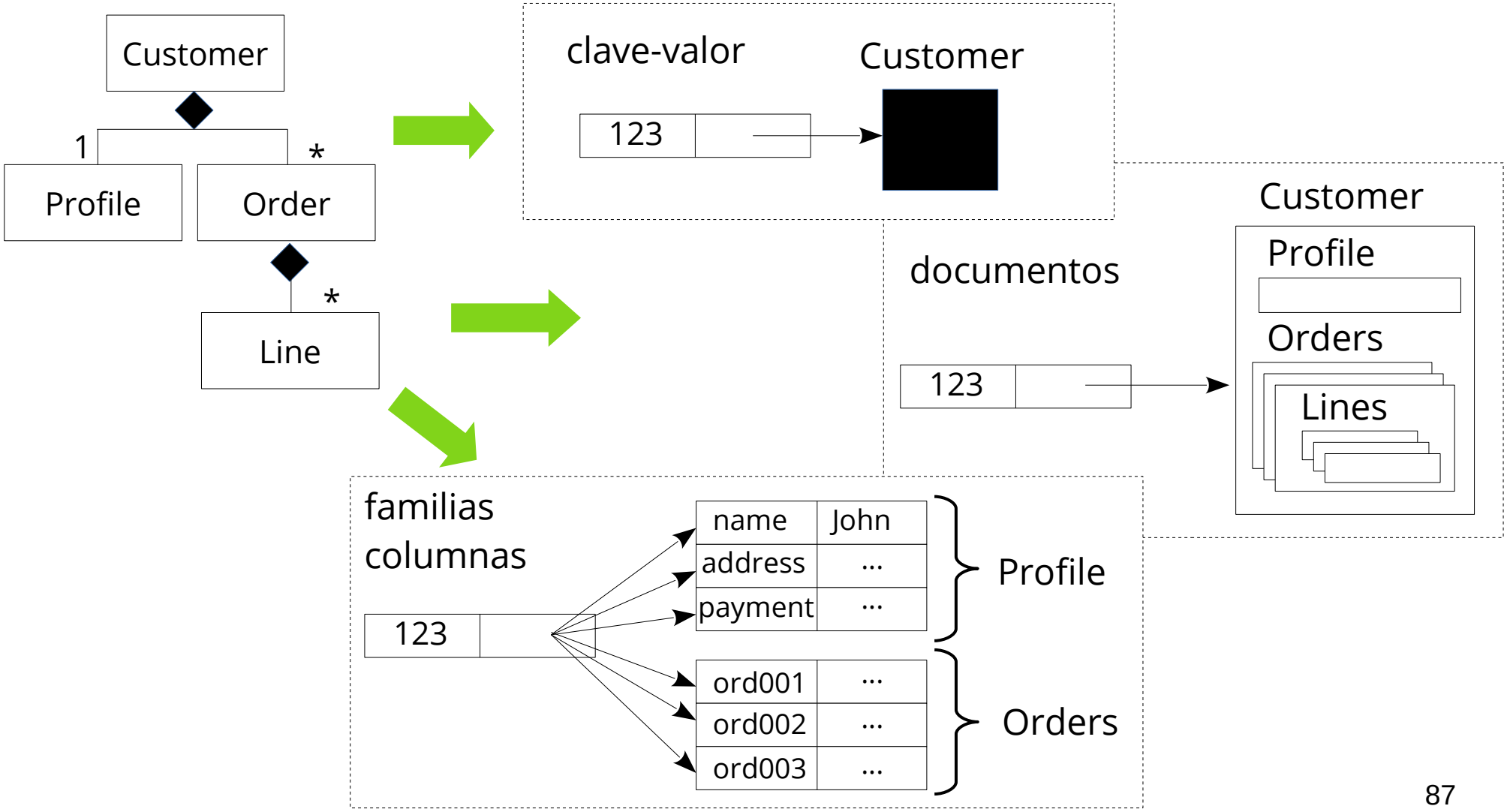
Ejercicio 4 > Almacenes de familias de columnas

- Diseñar una aplicación CLI que permita gestionar un conjunto de usuarios en un almacén de familias de columnas (Cassandra)
  - Listar
  - Añadir
  - Actualizar
  - Eliminar



Agregados

Tipos de agregados > Resumen



Agregados

Características

- Esquemas flexibles
- Relaciones
- Diseño de agregados
- Vistas materializadas

Agregados

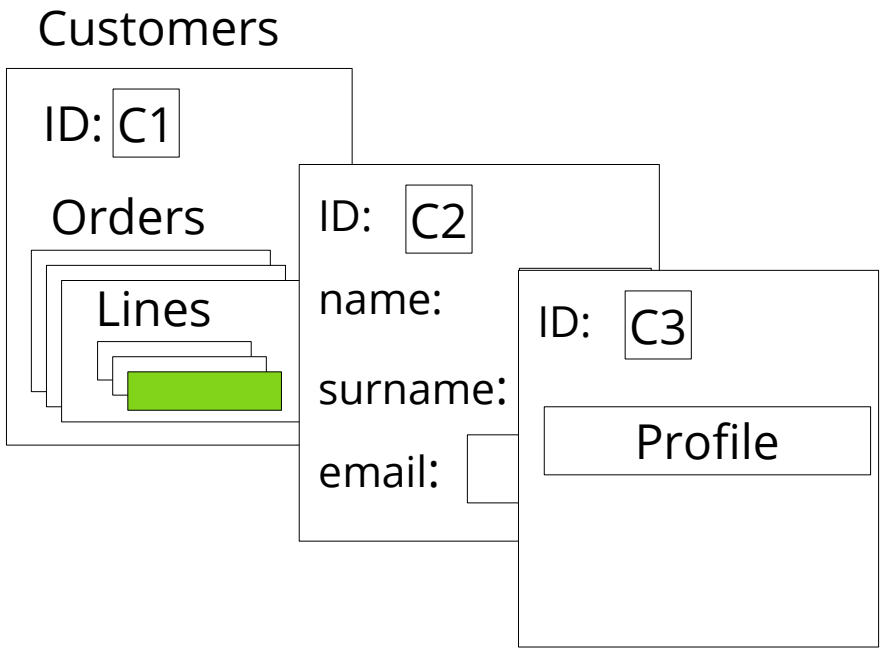
Características

- > Esquemas flexibles
- Relaciones
- Diseño de agregados
- Vistas materializadas

Agregados

Esquemas flexibles

- Los agregados no fuerzan una estructura interna fija



Agregados

Esquemas flexibles > Ventajas e inconvenientes

- ✔ Almacenamiento de datos no uniformes: libertad y flexibilidad
- ✔ La estructura de los datos puede evolucionar
- ✗ Suele existir siempre un esquema implícito, que está embebido en el código
- ✗ Si hay múltiples clientes, ese esquema implícito debe estar en todos ellos
- ✗ Si no hay esquema, no hay optimización, no hay validación

Agregados

Características

- Esquemas flexibles
- > Relaciones
- Diseño de agregados
- Vistas materializadas

Agregados

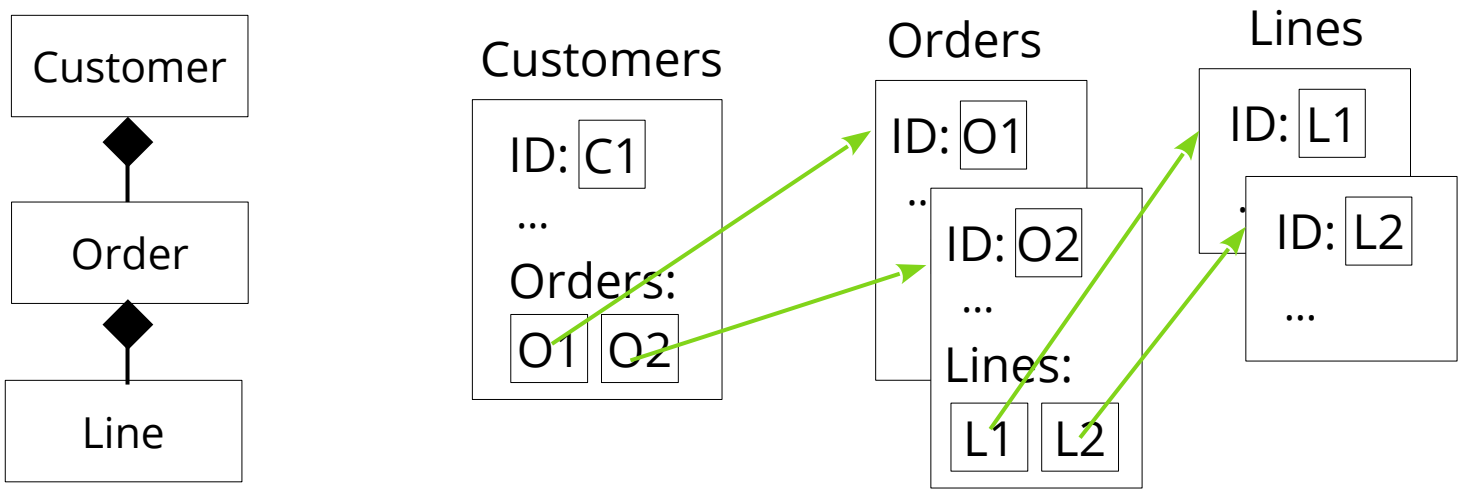
Relaciones

- No suele haber soporte nativo para modelar relaciones entre agregados
- El almacén no puede optimizar, validar relaciones
- Hay que implementarlo “manualmente”
- Dos aproximaciones: referencias vs anidamiento

Agregados

Relaciones > Referencias

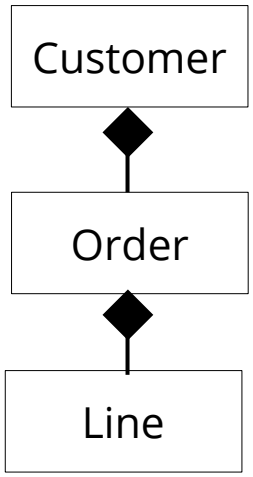
- Se almacena el ID del agregado referenciado

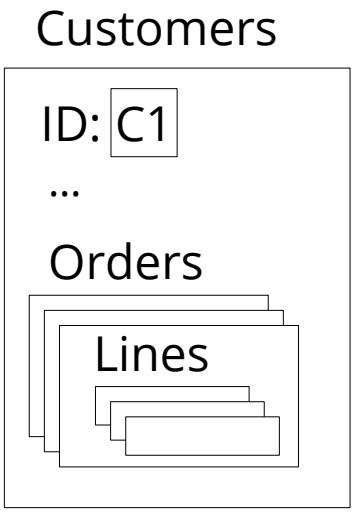


Agregados

Relaciones > Anidamiento

- Se anidan los agregados

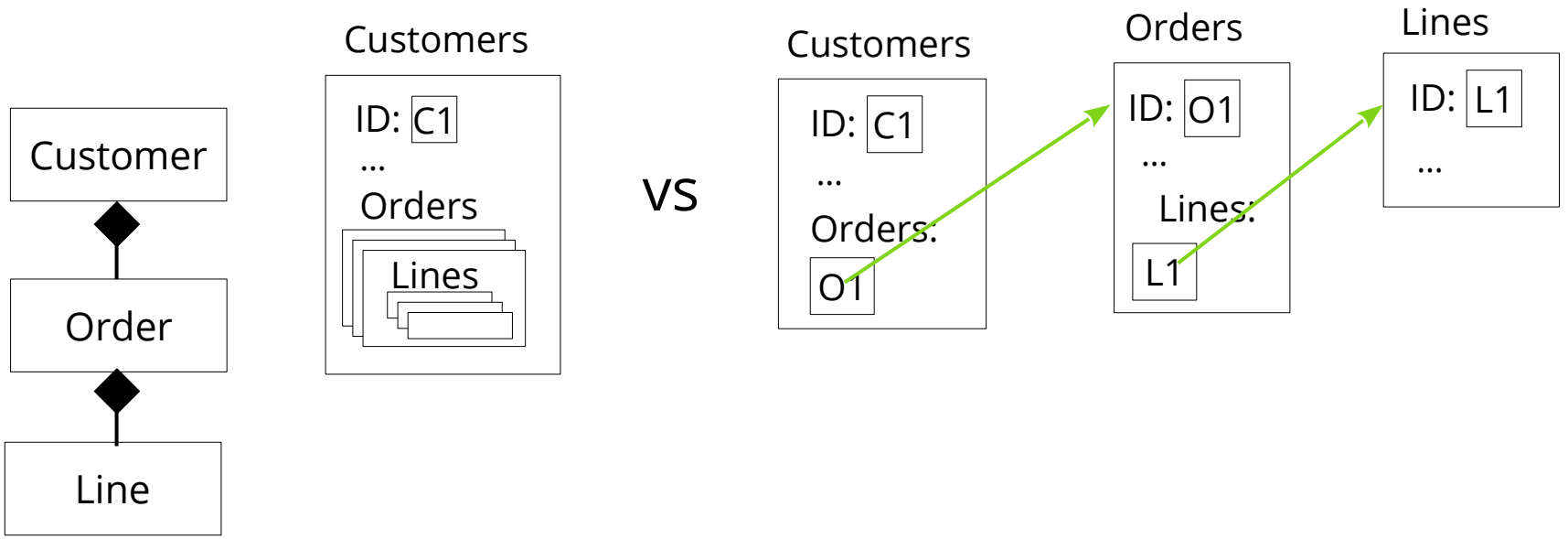




Agregados

Relaciones > Referencias vs anidamiento

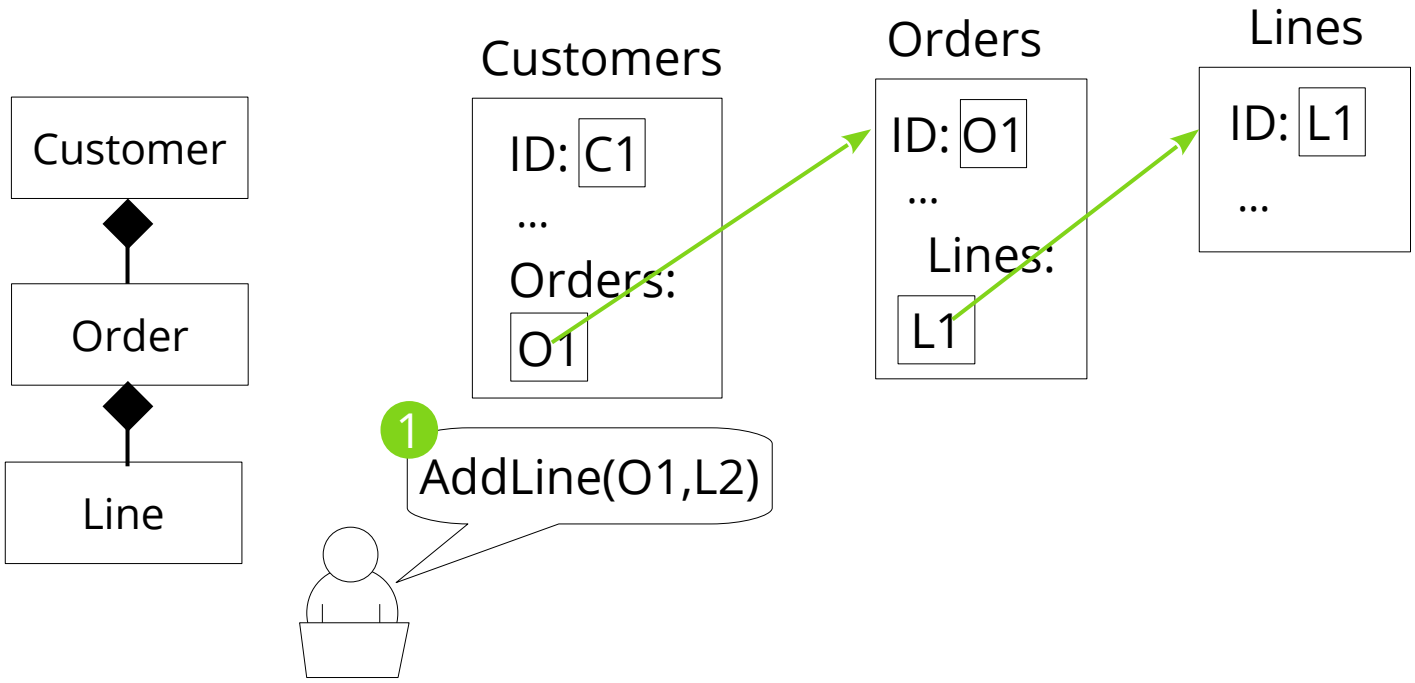
- Consecuencias en el rendimiento
  - Con anidamiento, toda la info en una consulta
  - Con referencias, son necesarias varias consultas



Agregados

Relaciones > Referencias vs anidamiento

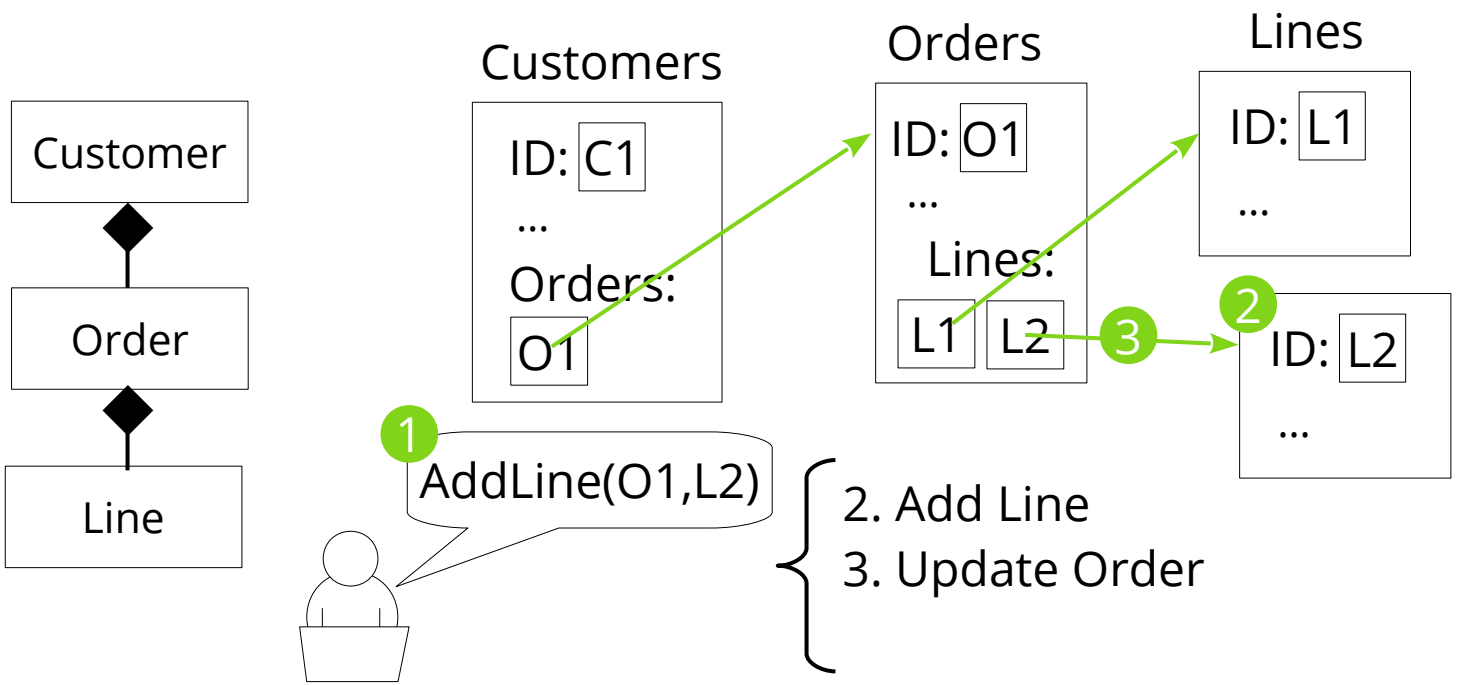
- Consecuencias en la consistencia
  - Con referencias y actualizaciones en varios agregados: alta probabilidad de inconsistencias



Agregados

Relaciones > Referencias vs anidamiento

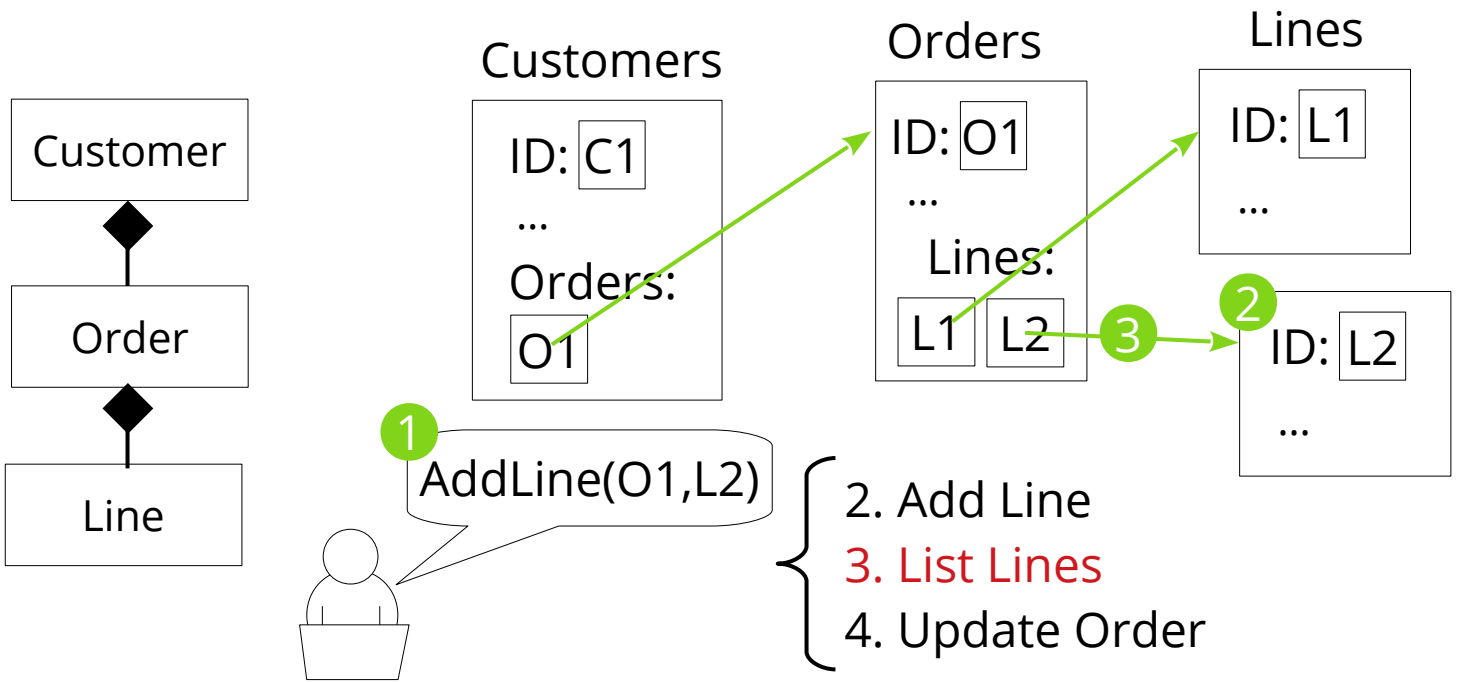
- Consecuencias en la consistencia
  - Con referencias y actualizaciones en varios agregados: alta probabilidad de inconsistencias



Agregados

Relaciones > Referencias vs anidamiento

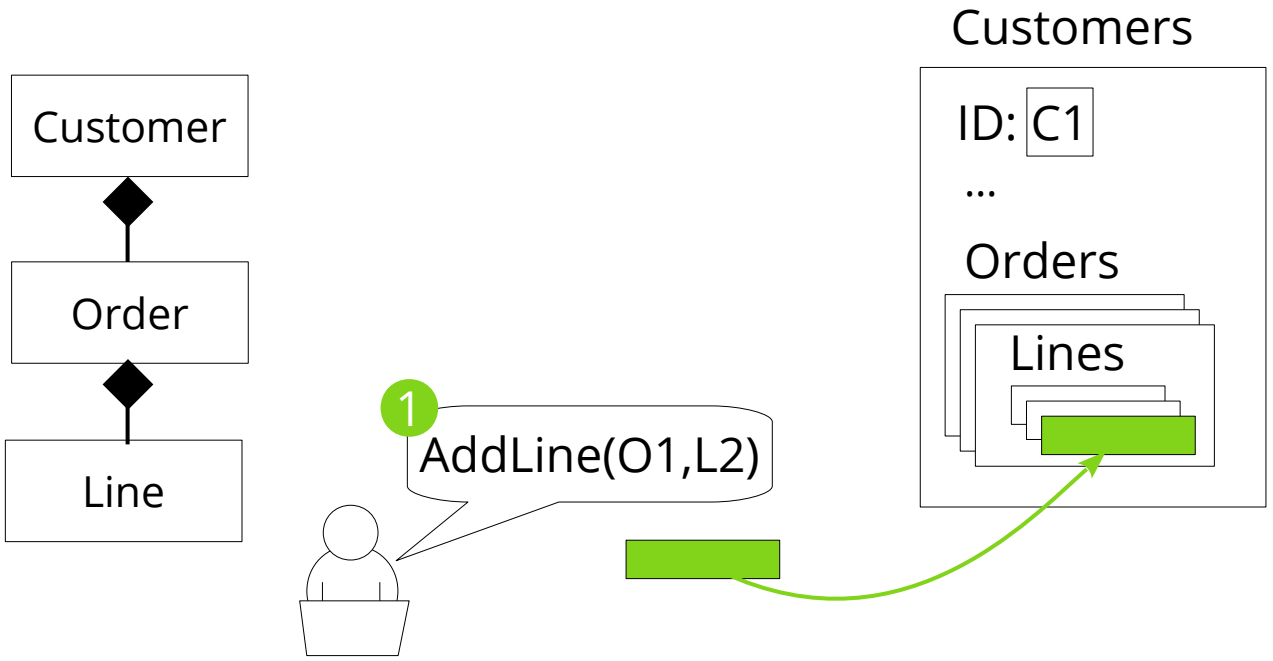
- Consecuencias en la consistencia
  - Con referencias y actualizaciones en varios agregados: alta probabilidad de inconsistencias



Agregados

Relaciones > Referencias vs anidamiento

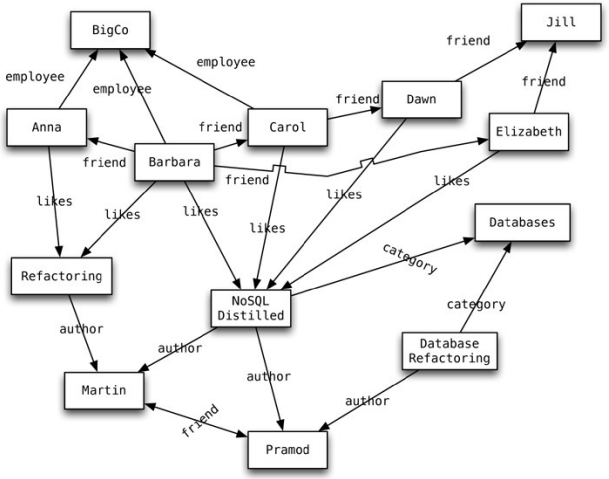
- Consecuencias en la consistencia
  - Con anidamiento la actualización es atómica en un agregado: minimiza la probabilidad de inconsistencias



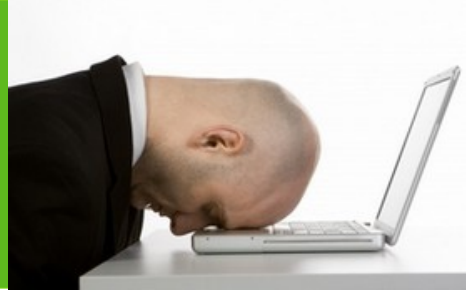
Agregados

Relaciones > Relaciones complejas

- Cuando las relaciones son complejas no se puede usar anidamiento: los agregados no ayudan
- Los almacenes relacionales tampoco son buena opción: muchos joins son costosos
- Los almacenes orientados a grafos optimizan la recuperación de información hiperrelacionada

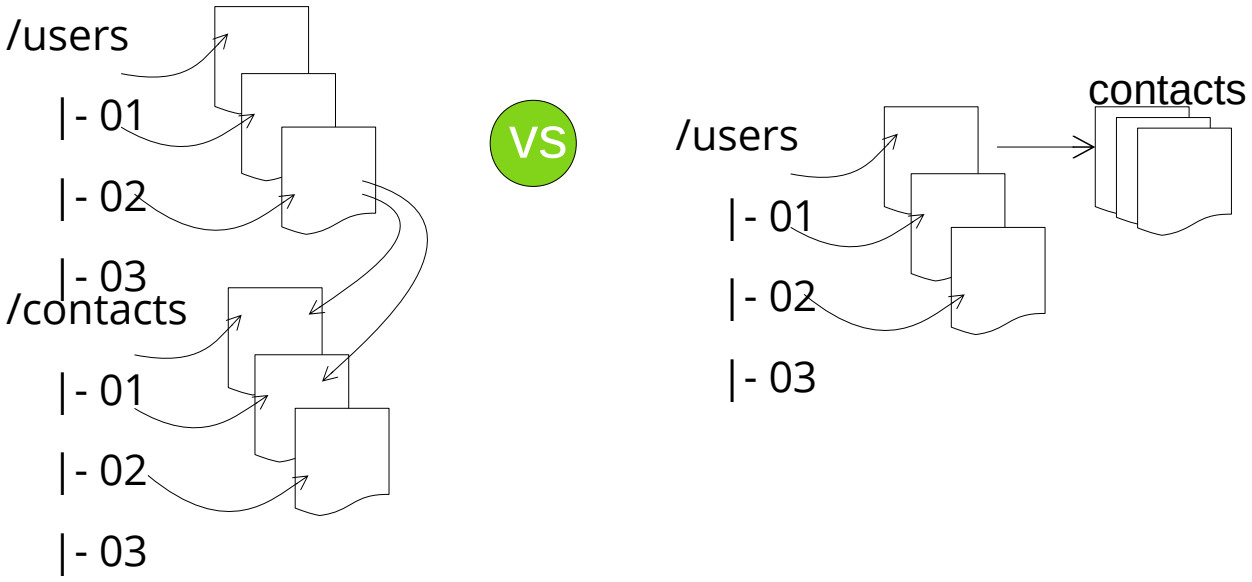


Agregados

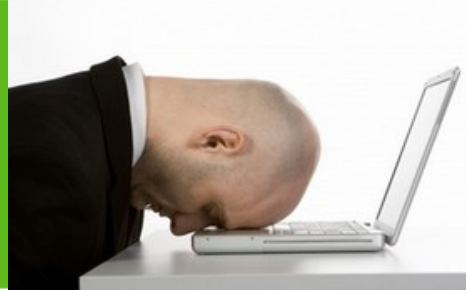


Ejercicio 5 > Relaciones en almacén clave-valor

- Ampliar el ejercicio 2; ahora cada usuario tiene sus propios contactos
- Operaciones CRUD sobre contactos
- Referencias vs anidamiento

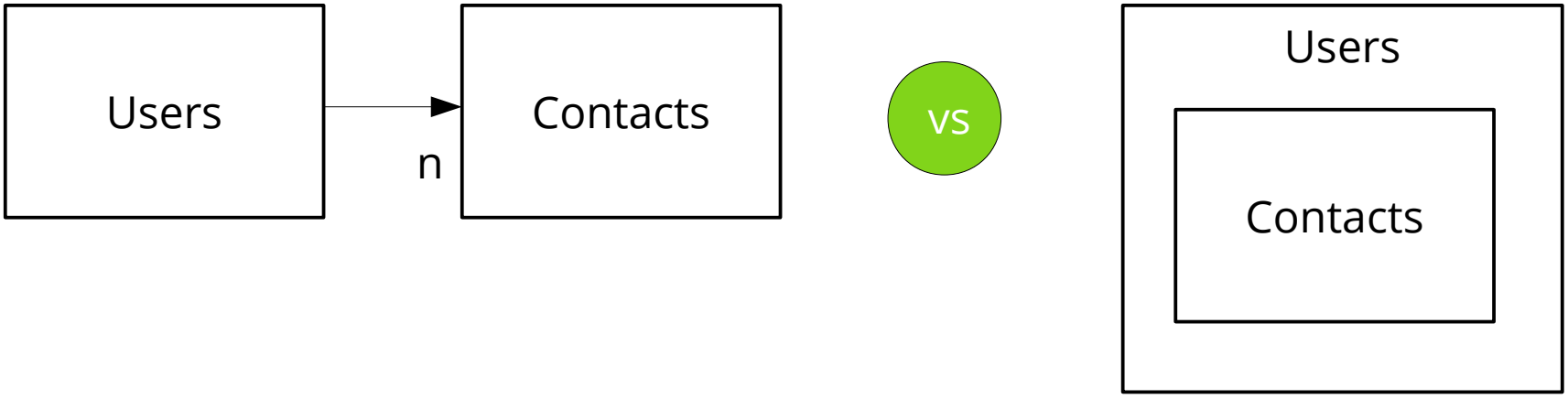


Agregados



Ejercicio 6 > Relaciones en almacén documentos

- Ampliar el ejercicio 3; ahora cada usuario tiene sus propios contactos
- Operaciones CRUD sobre contactos
- Referencias vs anidamiento



Agregados

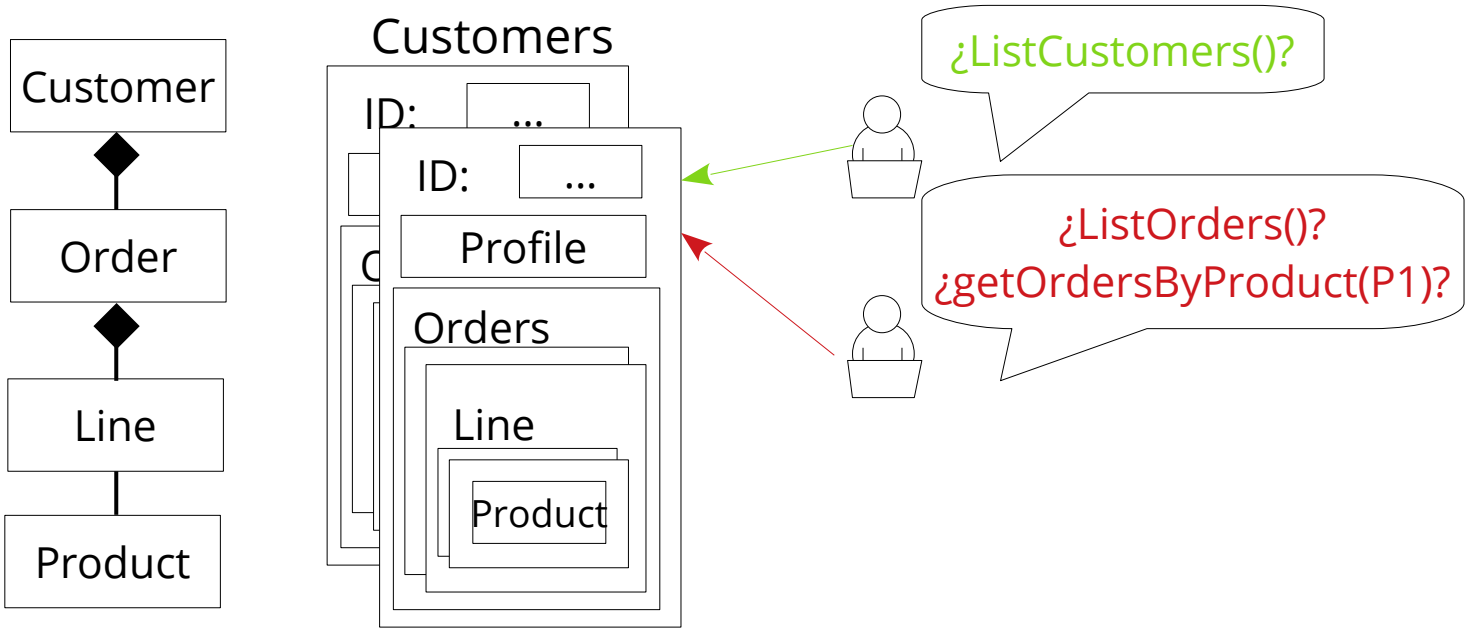
Características

- Esquemas flexibles
- Relaciones
- > Diseño de agregados
- Vistas materializadas

Agregados

Diseño de agregados

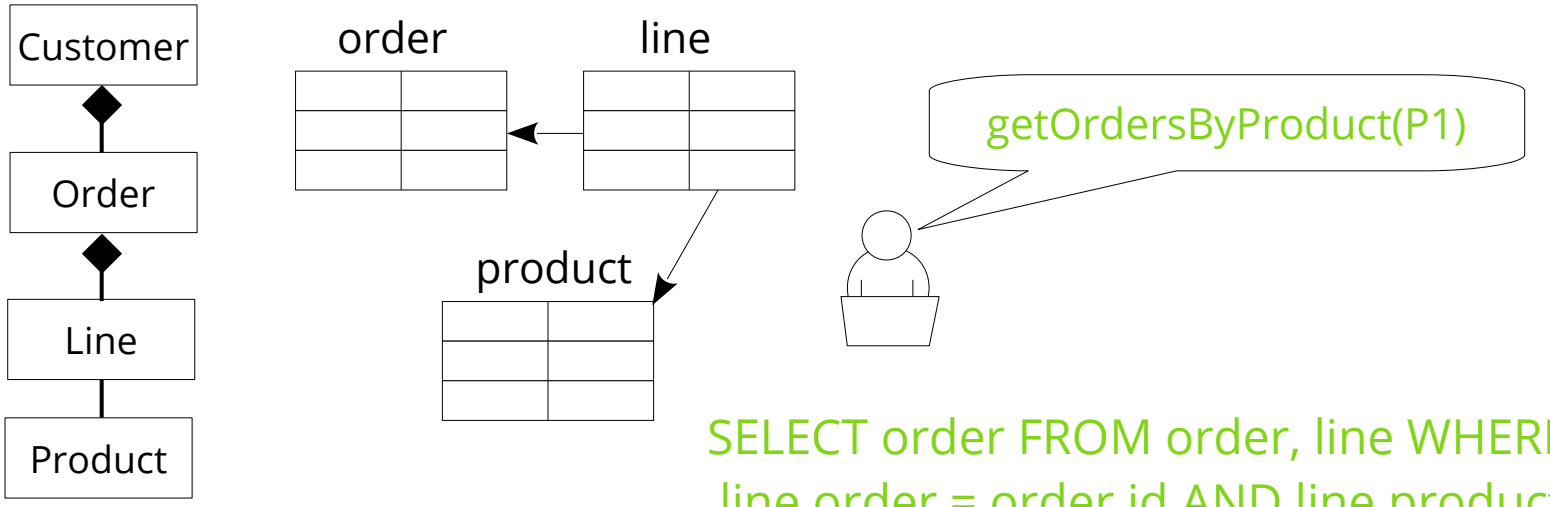
- El diseño del agregado condiciona el rendimiento en los accesos



Agregados

Diseño de agregados

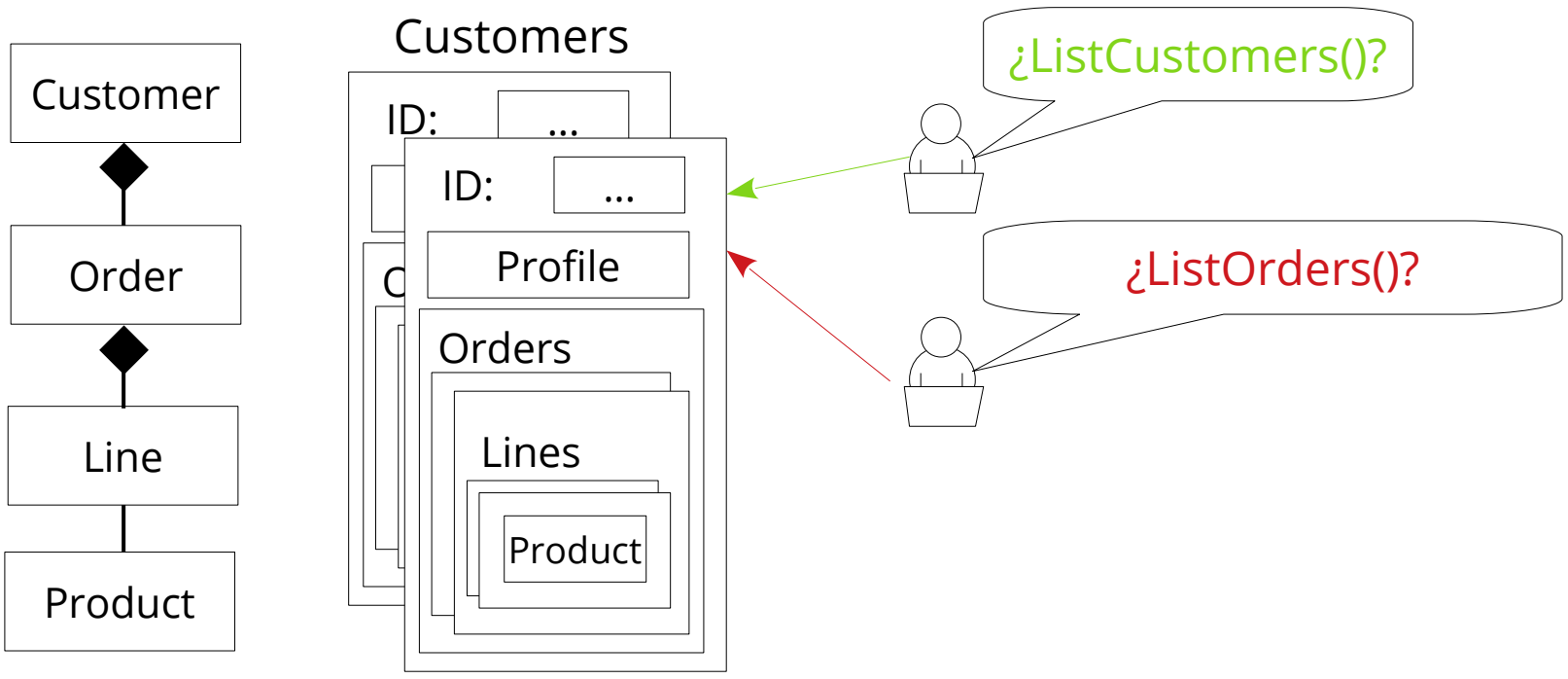
- El diseño del agregado condiciona el rendimiento en los accesos
- En un almacén relacional no tiene tanto impacto



Agregados

Diseño de agregados

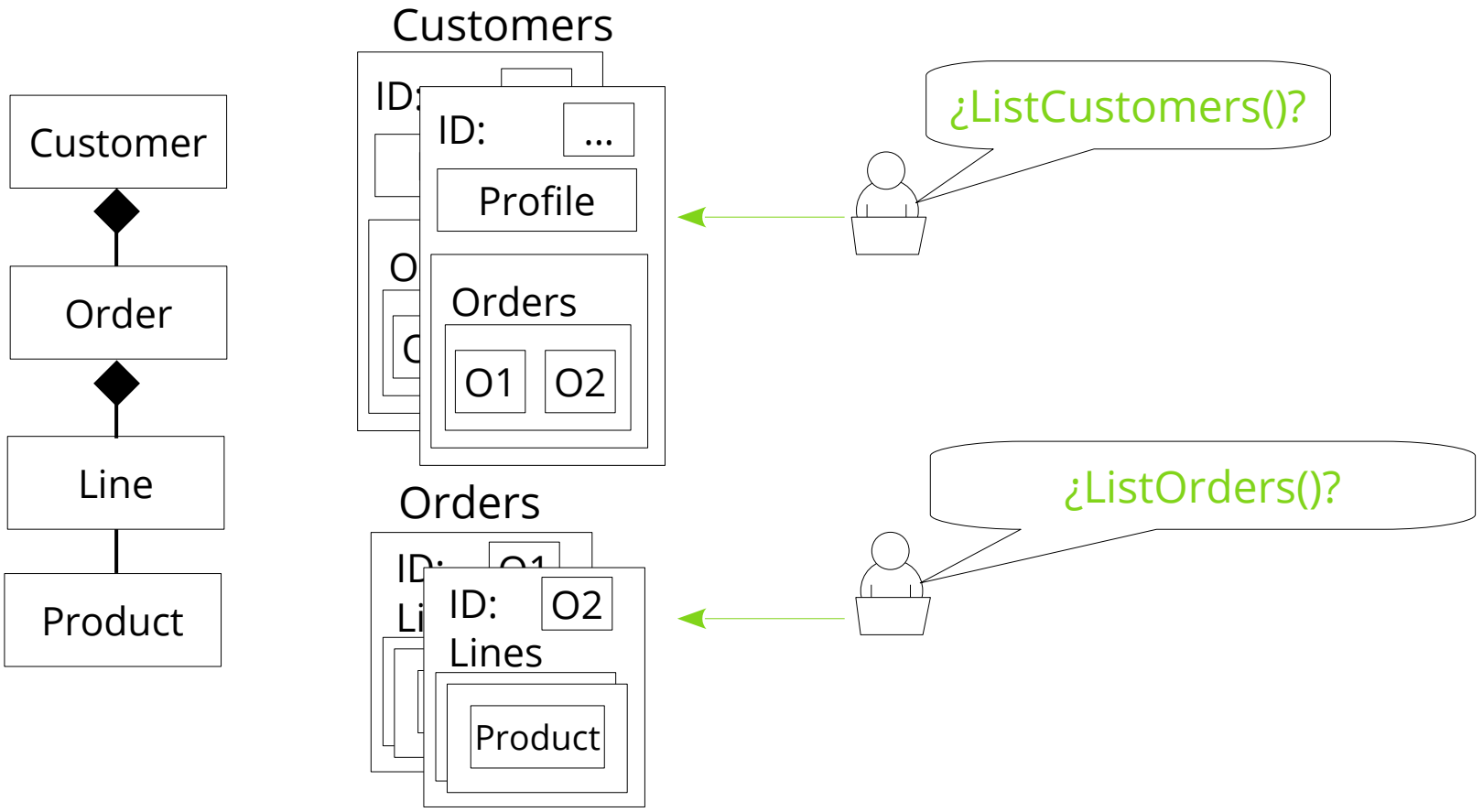
- Los agregados deben diseñarse pensando en el tipo de accesos que se efectuarán



Agregados

Diseño de agregados

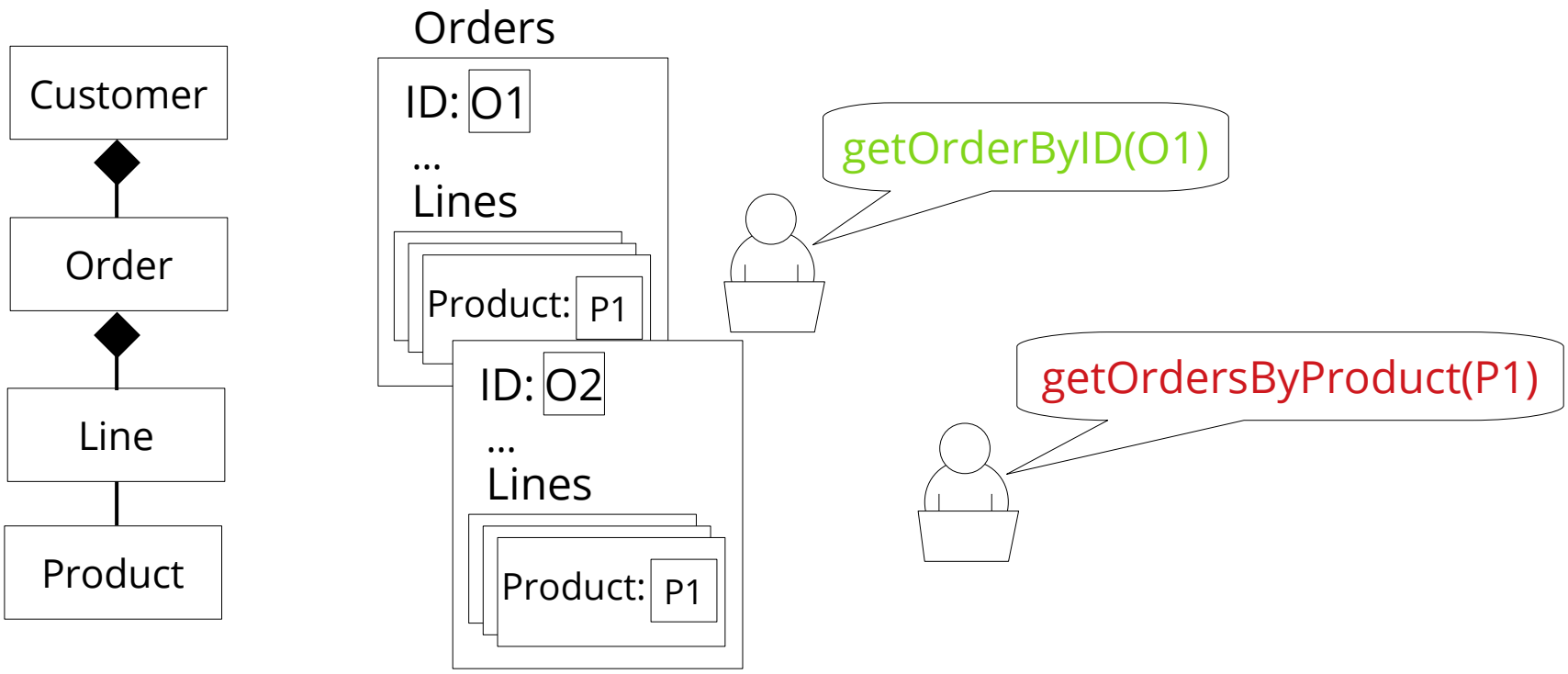
- Los agregados deben diseñarse pensando en el tipo de accesos que se efectuarán



Agregados

Diseño de agregados

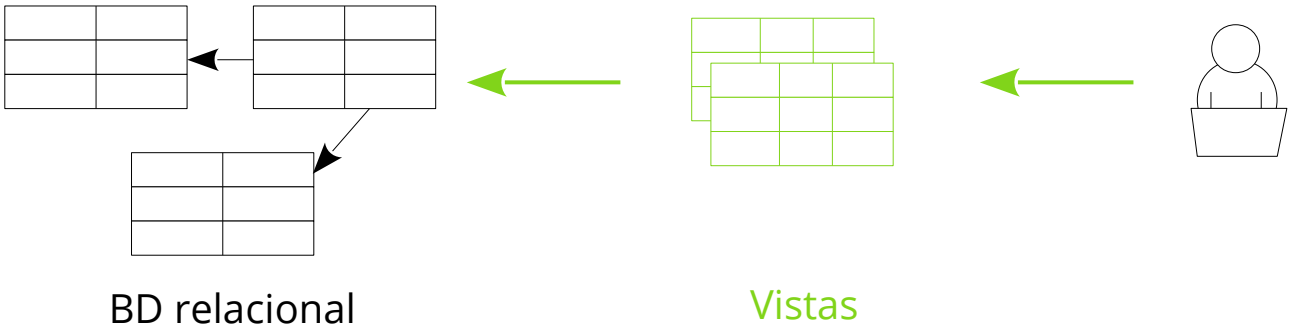
- Ciertos tipos de accesos seguirán siendo costosos y/o complejos



Agregados

Diseño de agregados

- Ciertos tipos de accesos seguirán siendo costosos y/o complejos
- Los almacenes relacionales disponen de vistas; los almacenes con agregados pueden usar vistas materializadas



Agregados

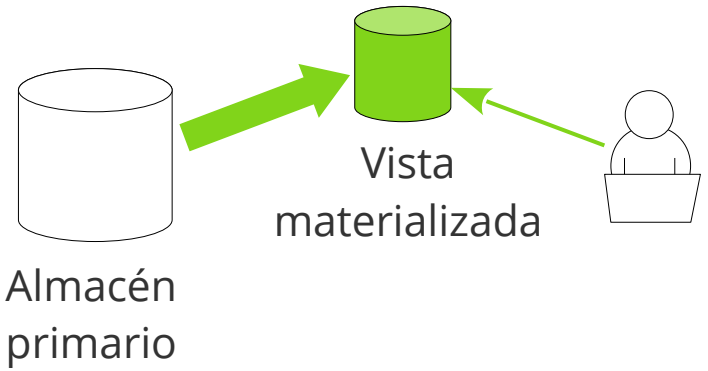
Características

- Esquemas flexibles
- Relaciones
- Diseño de agregados
- > Vistas materializadas

Agregados

Vistas materializadas

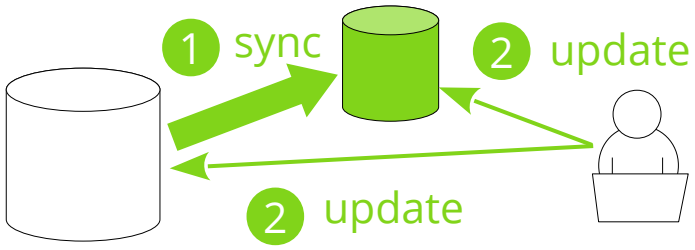
- Una vista materializada es una vista que se computa por adelantado, y se cachea
- Permite implementar consultas complejas sobre agregados de manera eficiente



Agregados

Vistas materializadas

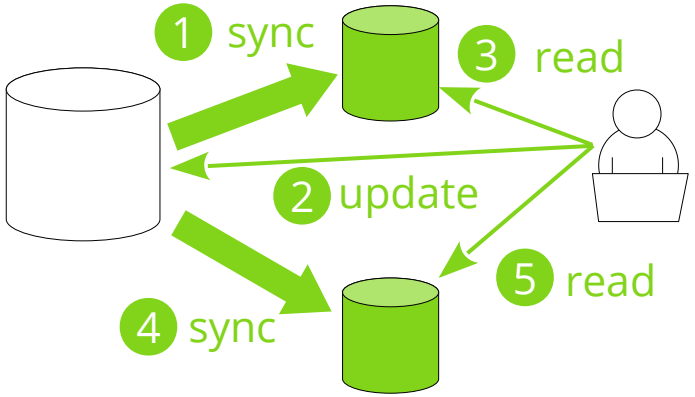
- Aproximación eager (voraz) vs batch
  - En las actualizaciones, se actualiza la vista y el almacén: penalización en rendimiento
  - Apropiado si se requieren datos actualizados, y si hay más lecturas que escrituras



Agregados

Vistas materializadas

- Aproximación eager (voraz) vs batch
  - Se construyen las vistas periódicamente: no hay penalización en las actualizaciones
  - Los datos estarán desactualizados hasta la próxima construcción de vista



Agregados

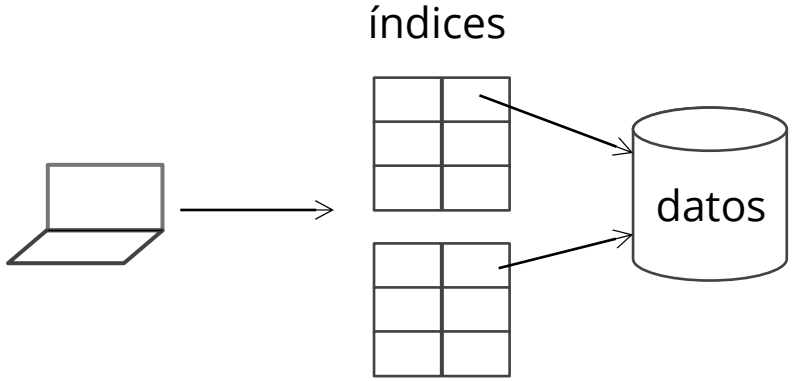
Vistas materializadas

- Pueden construirse externamente o internamente
- Es habitual incluir vistas en el mismo agregado

Indexación de datos

¿Cómo se organiza internamente la información?

- Escrituras y lecturas rápidas
- Para acelerar las lecturas se utilizan índices
- Son estructuras de datos adicionales, que mantienen referencias a los datos, acelerando los accesos
- Distintos tipos de índices con distintas propiedades
- Mantener un índice es costoso, ralentiza las escrituras



Indexación de datos

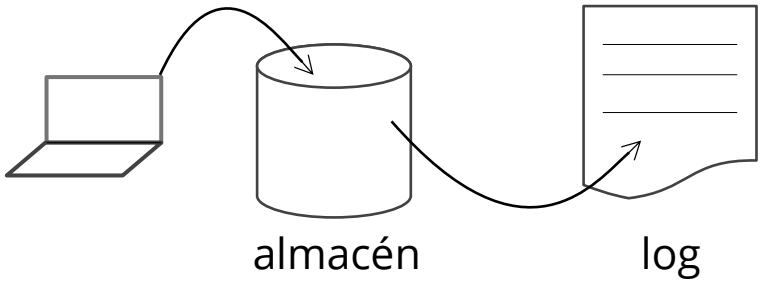
Tipos de almacenes según sus índices

- Almacenes basados en logs
  - Índices de tipo diccionario
- Almacenes basados en páginas
  - Índices de tipo árbol balanceado

Indexación de datos

Almacenes basados en logs

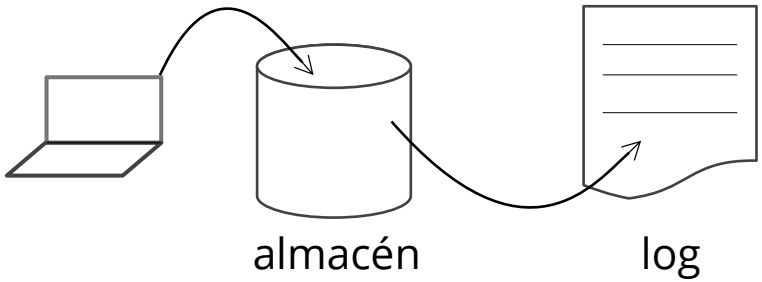
- Las operaciones de escritura se añaden al final de un log (append-only): sistemas de ficheros optimizados
- Tenemos toda la historia de operaciones: posibilidad de reconstruir el almacén en cierto instante de tiempo
- Las operaciones de escritura son muy eficientes
- Las operaciones de lectura son muy costosas: hay que recorrer todo el fichero



Indexación de datos

Ejercicio 7

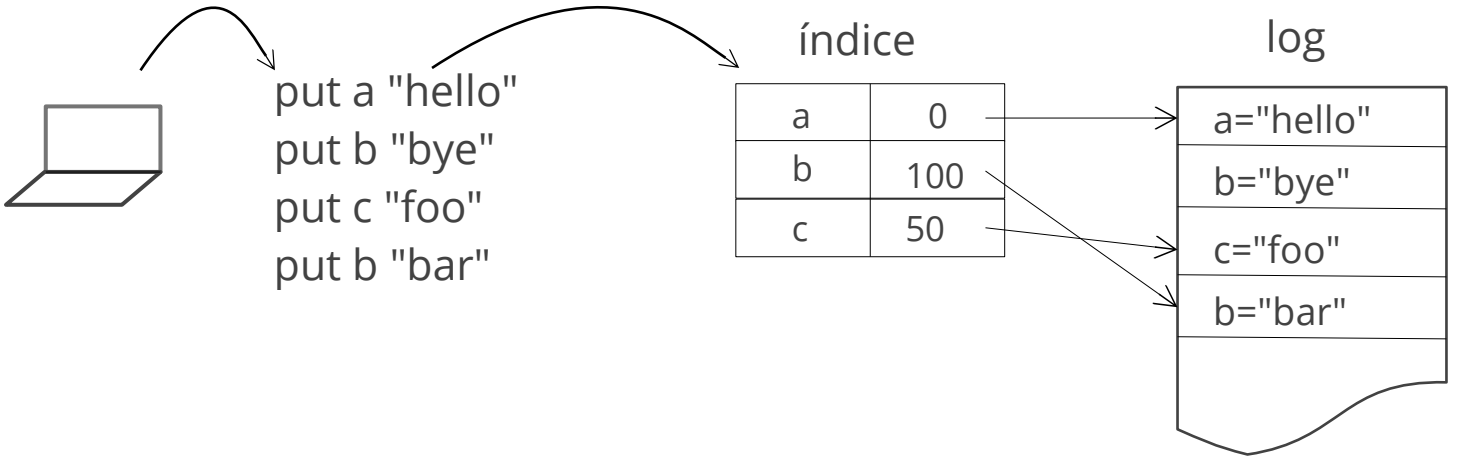
- Implementar un almacén clave-valor basado en log
- create, put, get, list



Indexación de datos

Almacenes basados en logs > Índices

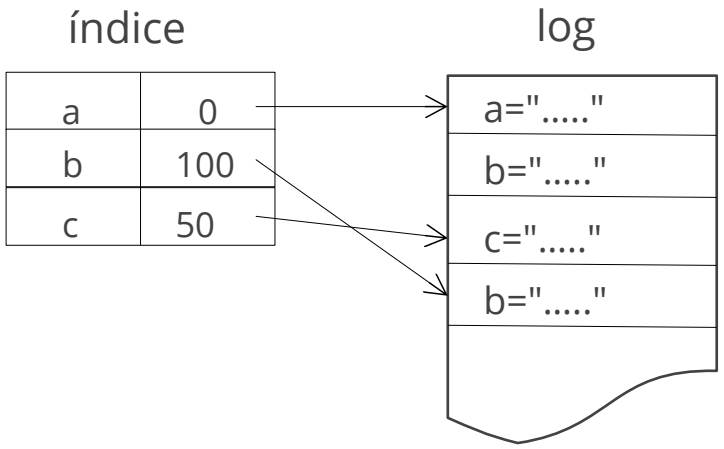
- Mejorar las lecturas: con un diccionario en memoria
- Una entrada por cada elemento insertado
- La entrada apunta a la posición de la última escritura de ese elemento



Indexación de datos

Ejercicio 8

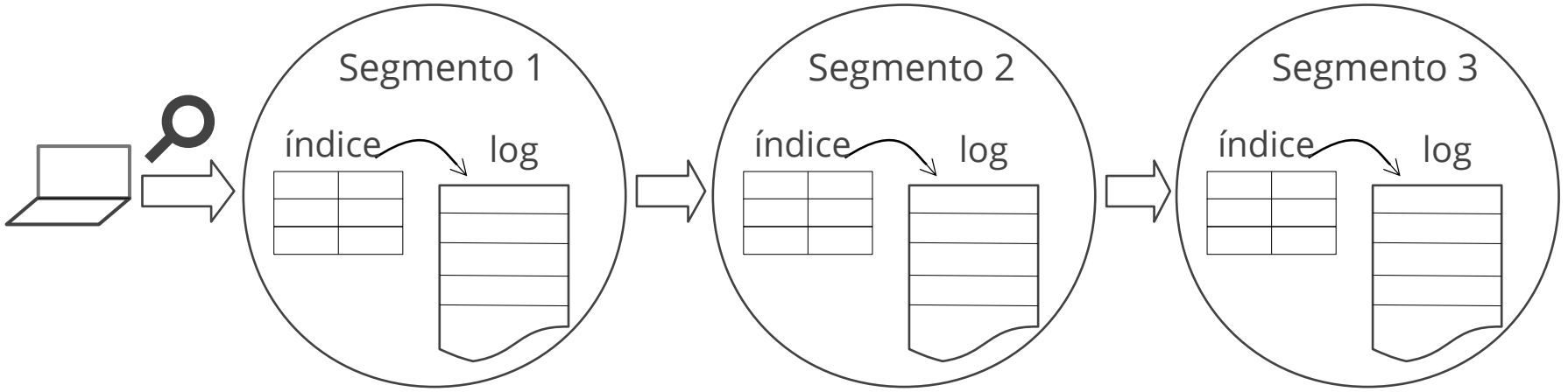
- Añadir al almacén un índice con un diccionario



Indexación de datos

Almacenes basados en logs > Segmentos

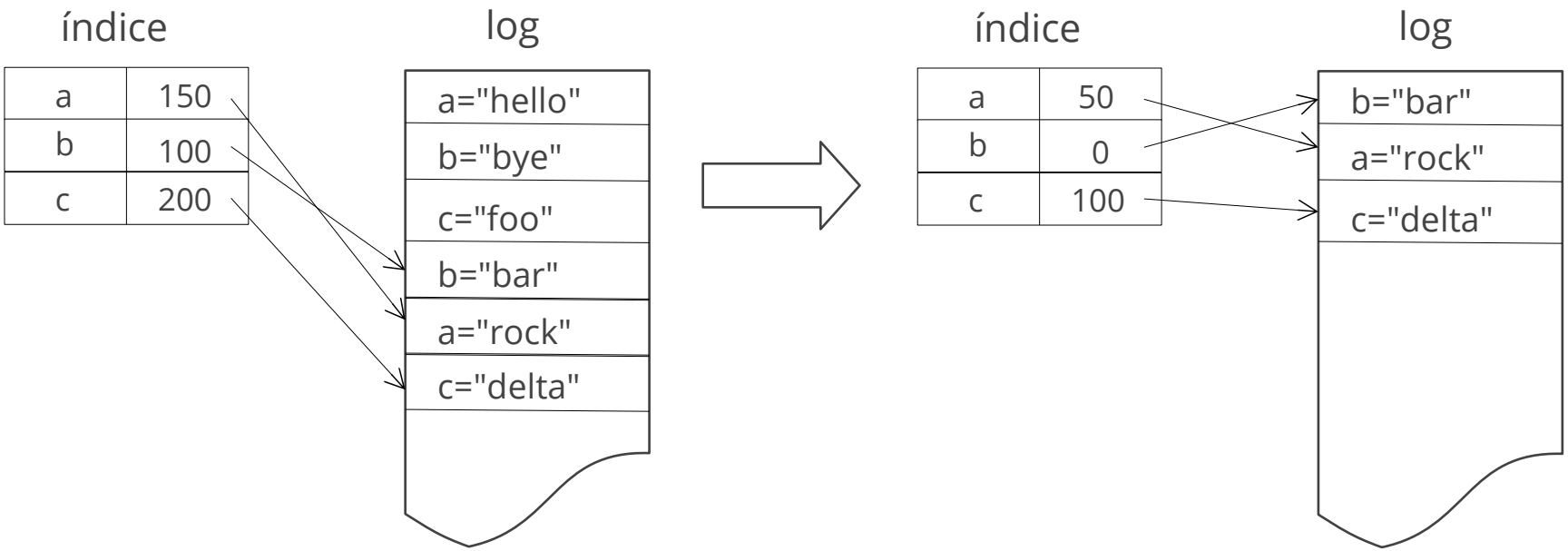
- El log puede crecer de manera desmesurada
- Se fragmenta en segmentos de similar longitud
- Cada segmento posee su log y su índice en memoria
- Cadena de segmentos
- Búsquedas: primero el más reciente



Indexación de datos

Almacenes basados en logs > Segmentos

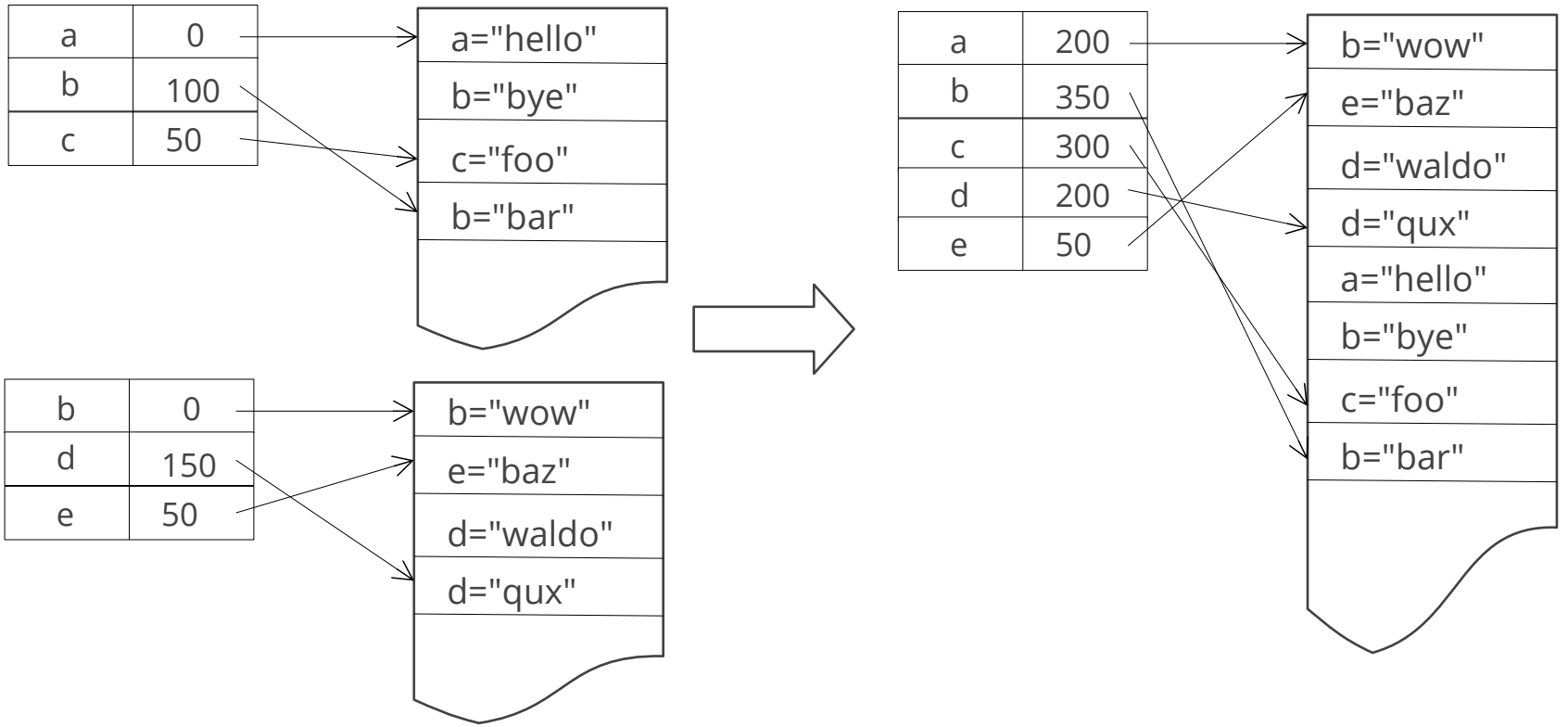
- Se pueden compactar



Indexación de datos

Almacenes basados en logs > Segmentos

- Se pueden fusionar



Indexación de datos

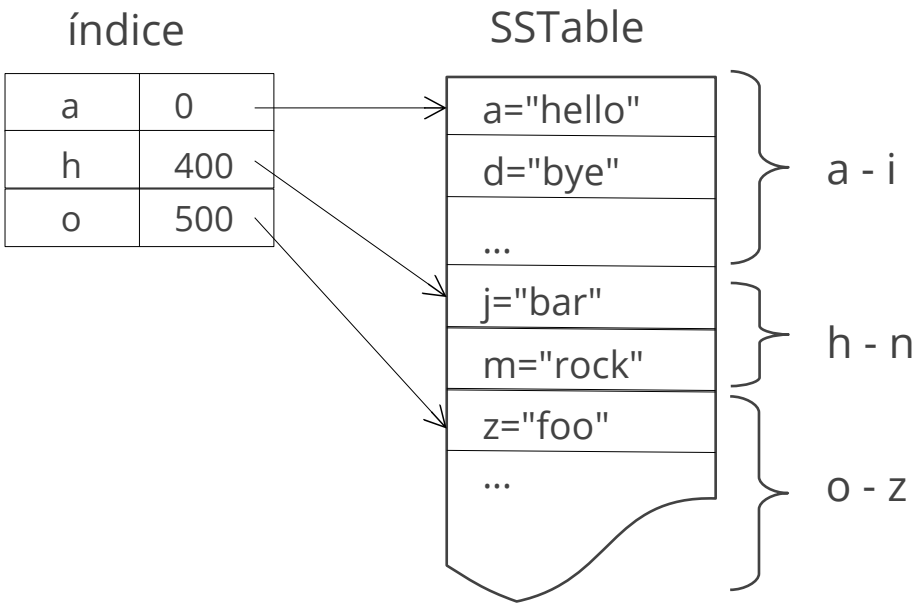
Almacenes basados en logs > Segmentos

- Problemas
  - Índices en memoria muy gandes
  - Consultas basadas en rangos muy difíciles
- La solución: SSTables (Sorted String Tables)

Indexación de datos

Almacenes basados en logs > SSTables

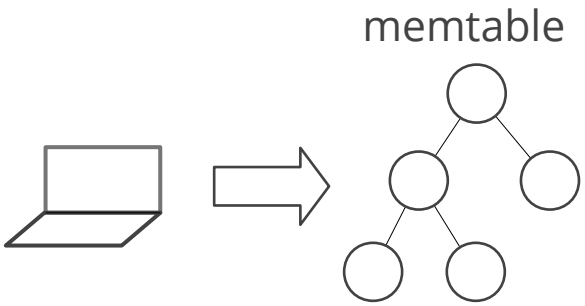
- Los datos en el log ordenados y no repetidos
- El índice mantiene una entrada por cada rango
- Búsquedas: binaria dicotómica (O(log n))



Indexación de datos

Almacenes basados en logs > memtable

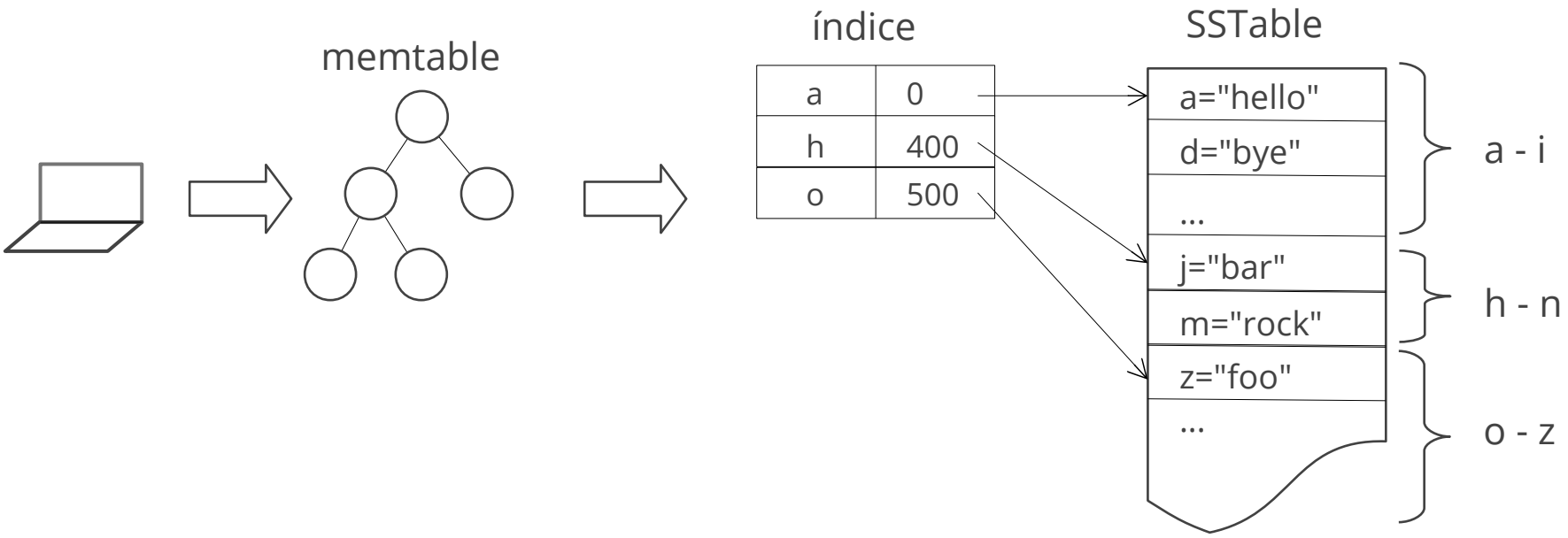
- Para mantener el orden, los datos se guardan provisionalmente en memoria, en árbol ordenado (rojo-negro, AVL, etc.): memtable



Indexación de datos

Almacenes basados en logs > memtable

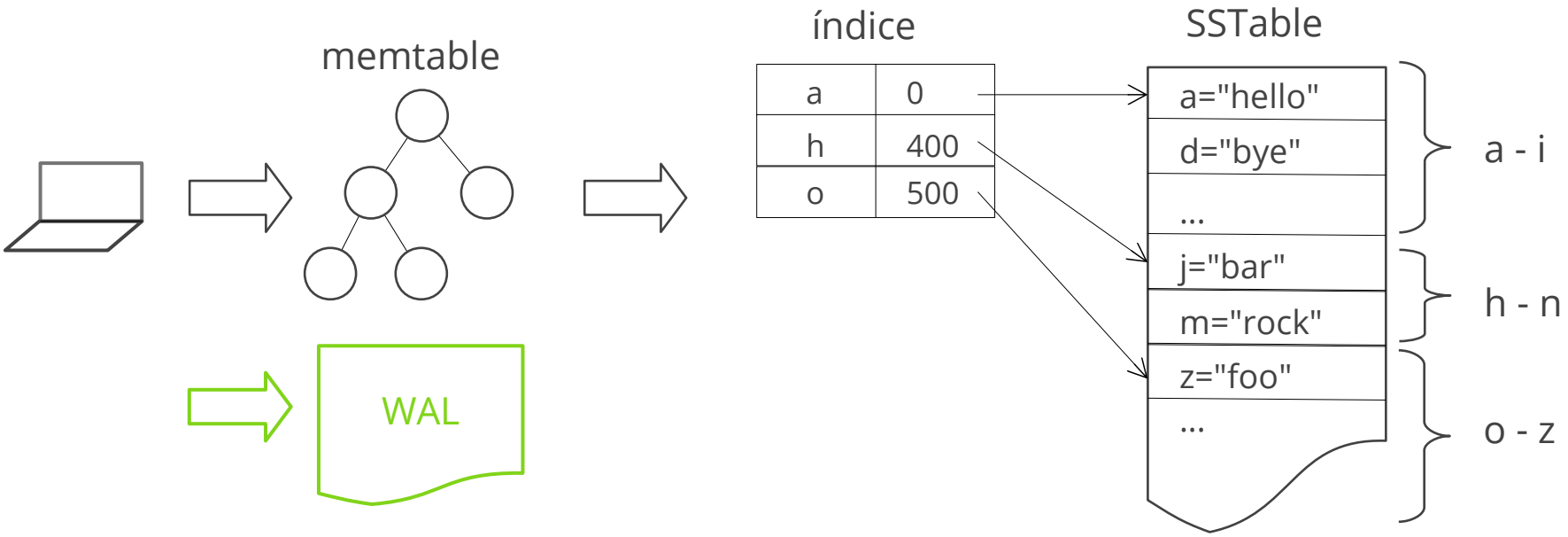
- Para mantener el orden, los datos se guardan provisionalmente en memoria, en árbol ordenado (rojo-negro, AVL, etc.): memtable
- Cuando alcanza cierto tamaño, se escribe en SSTable



Indexación de datos

Almacenes basados en logs > memtable

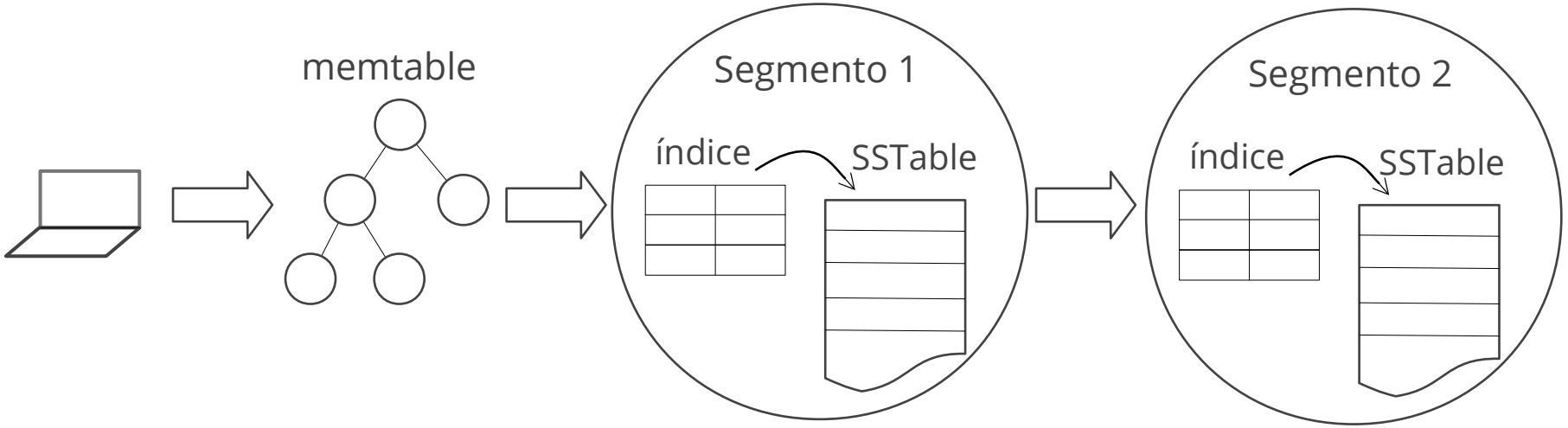
- Si el sistema falla, se pierde la memtable
- Para evitarlo se crea un log de escrituras (WAL)
- Si el sistema falla, se restaura la memtable



Indexación de datos

Almacenes basados en logs > LSM-Tree

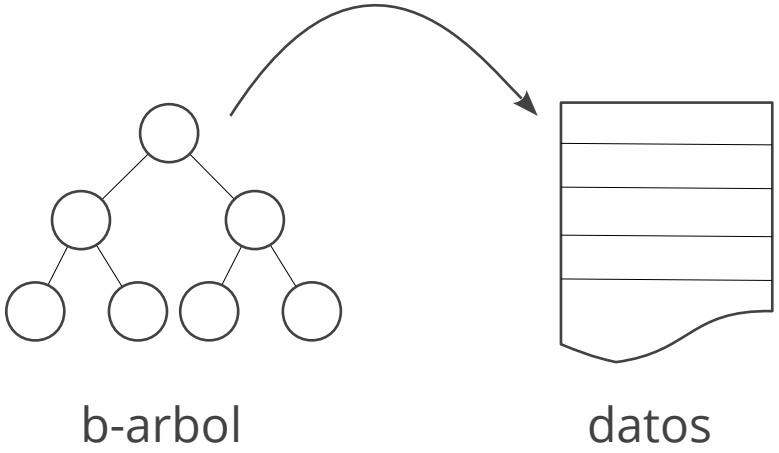
- Combinándolo todo: Log-Structured Merge-Tree
- Búsqueda en cadena de segmentos (SSTable + índice)
- LevelDB, Big-Table, Cassandra, HBase, SQLite, …



Indexación de datos

Almacenes basados en páginas

- Son los almacenes más comunes
- Construyen sus índices utilizando b-árboles
- B-árbol: árbol balanceado que mantiene la información ordenada y garantiza un número de accesos a disco mínimo



Indexación de datos

Almacenes basados en páginas > B-árbol

- B-árbol de grado n
- Estructura de los nodos internos

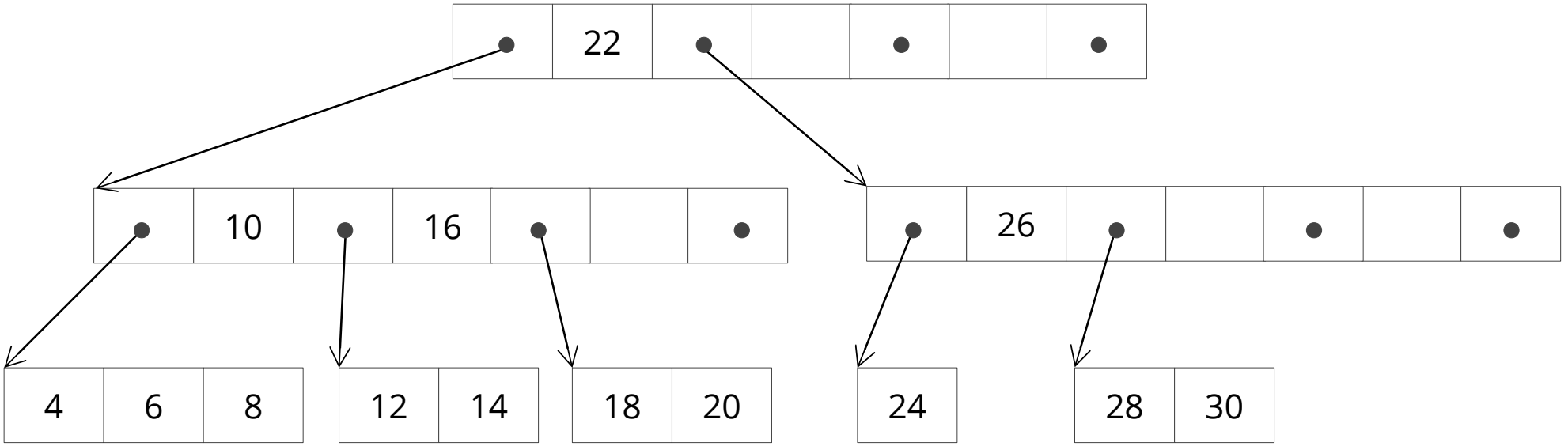
p_1	k_1	p_2	k_2	.....	k_{n-1}	p_n

  - pi : apunta al hijo i, $1 \leq i \leq n$
  - ki : valores de las claves (k1< k2< … < kn-1) que verifican:
    - ∀ clave c ∈ subárbol p1 : c < k1
    - ∀ clave c ∈ subárbol pi, $2 \leq i \leq n-1$: ki-1 ≤ c < ki
    - ∀ clave c ∈ subárbol pn: kn-1 ≤ c

Indexación de datos

Almacenes basados en páginas > B-árbol

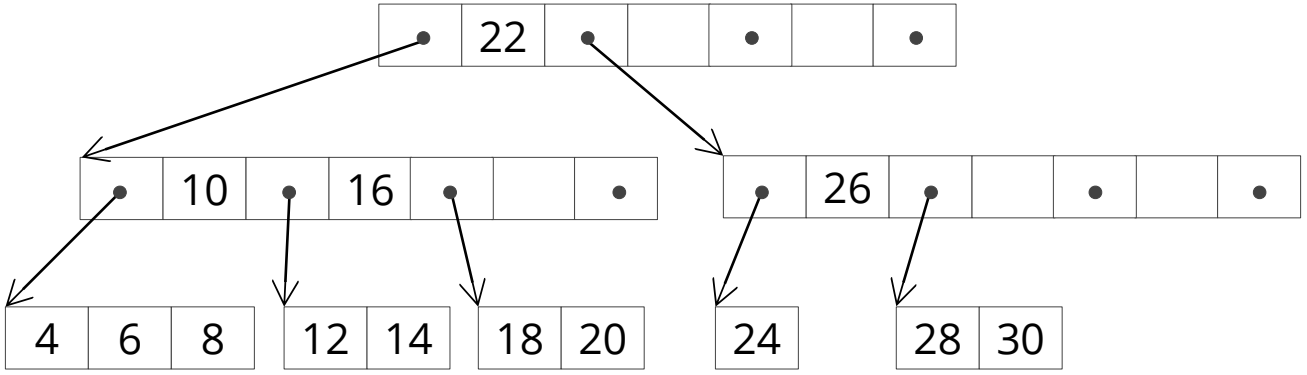
- B-árbol de grado n
- Ejemplo de B-árbol de grado 4



Indexación de datos

Almacenes basados en páginas > Funcionamiento

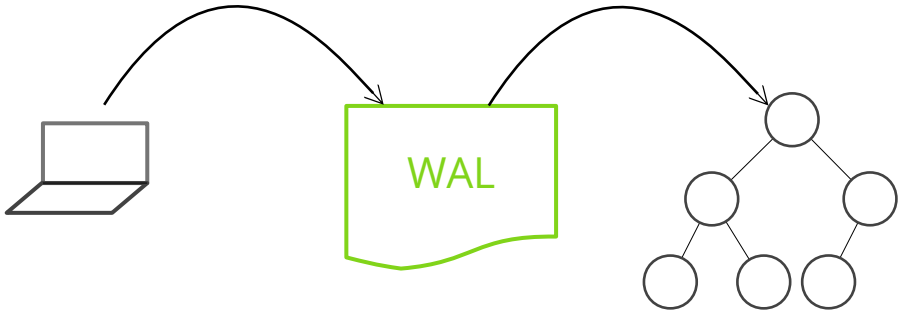
- La base de datos se divide en bloques (≈ 4KB)
- En cada bloque se almacena un nodo del b-árbol
- Los datos pueden residir en el bloque o fuera
- Lecturas/escrituras de bloques enteros O(log n)



Indexación de datos

Almacenes basados en páginas > WAL

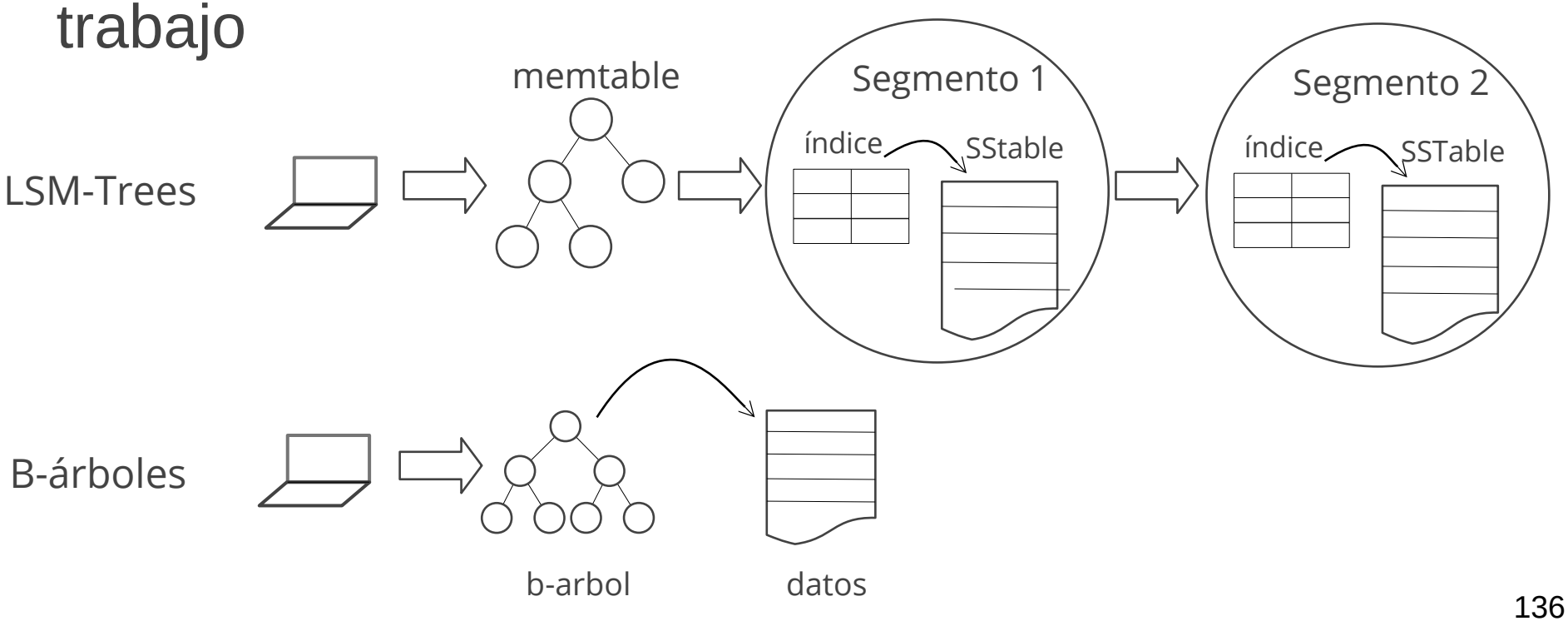
- La escritura de datos en disco es costosa, podría fallar
- Write-ahead log: se registran todas las escrituras antes de escribir en b-árbol
- Permite recuperar escrituras en caso de fallos



Indexación de datos

LSM-Tree vs B-árbol

- LSM-Trees más rápidos en escrituras
- B-árbol más rápido en lecturas (en el peor caso?)
- No hay datos concluyentes, depende de la carga de trabajo



Distribución de datos

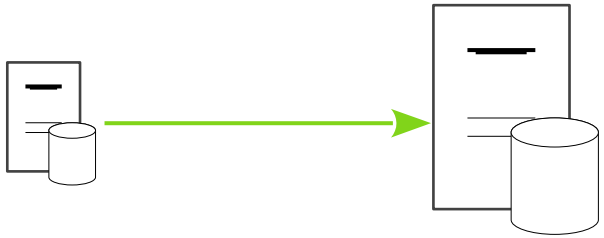
Cuando el volumen de datos crece: escalar

- Escalado vertical (scale up)
- Escalado horizontal (scale out)

Distribución de datos

Cuando el volumen de datos crece: escalar

- Escalado vertical (scale up)
  - Adquirir un servidor más potente
  - Costoso económicamente
  - Existen límites

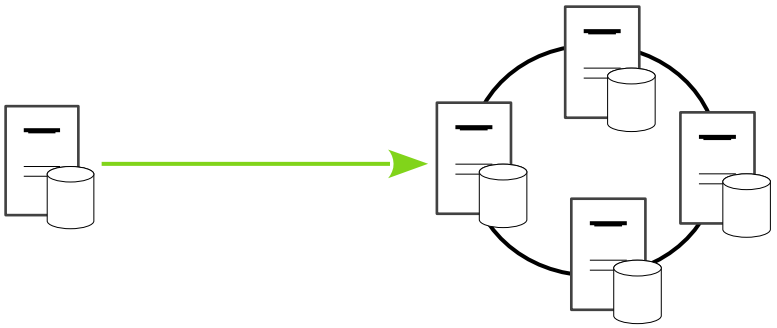


- Escalado horizontal (scale out)

Distribución de datos

Cuando el volumen de datos crece: escalar

- Escalado vertical (scale up)
- Escalado horizontal (scale out)
  - Clúster de servidores económicos
  - No hay límites
  - Se distribuyen los datos: distintos modelo de distribución de datos



Distribución de datos

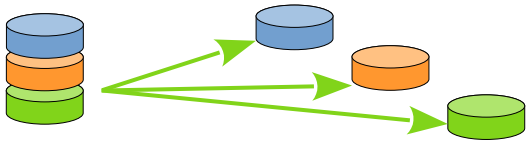
Modelos de distribución de datos

- Particionado
- Replicación
- Replicación + particionado

Distribución de datos

Modelos de distribución de datos

- Particionado
  - Se particionan los datos: particiones
  - Cada partición se almacena en un nodo diferente

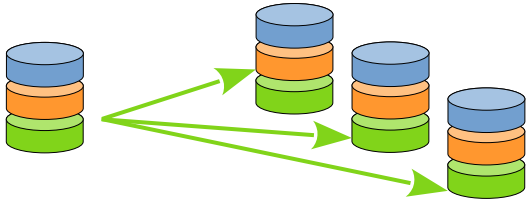


- Replicación
- Replicación + particionado

Distribución de datos

Modelos de distribución de datos

- Particionado
- Replicación
  - Los mismos datos se almacenan en distintos nodos

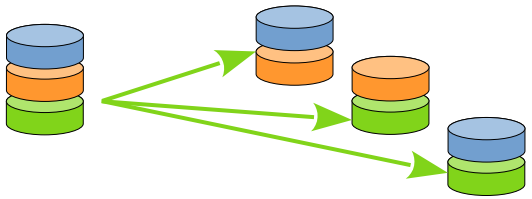


- Replicación + particionado

Distribución de datos

Modelos de distribución de datos

- Particionado
- Replicación
- Replicación + particionado
  - Se combinan ambas aproximaciones



Distribución de datos

Particionado

- Distribuir los datos del almacén entre distintos nodos
- La base de datos se divide en distintas particiones lógicas, que se gestionan de manera independiente
- Beneficios
  - Se incrementa la escalabilidad: repartir volumen de datos, paralelizar operaciones
  - Se mejora el rendimiento: tamaño menor, optimizar por tipo de acceso
  - Se mejora la seguridad: proteger particiones sensibles

Distribución de datos

Particionado > Tipos

- Particionado vertical
- Particionado horizontal (sharding)

Distribución de datos

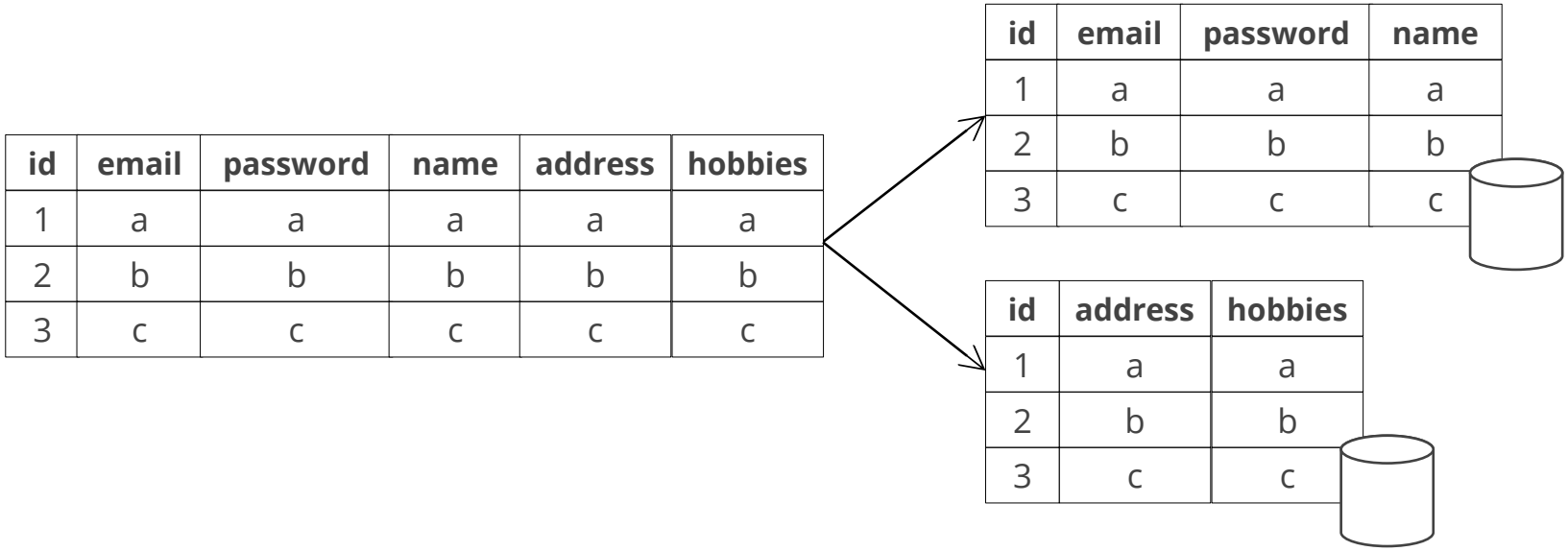
Particionado > Tipos

- Particionado vertical
  - Cada tabla/colección en una partición

Distribución de datos

Particionado > Tipos

- Particionado vertical
  - Cada tabla/colección en una partición
  - Es habitual romper una tabla/colección en trozos

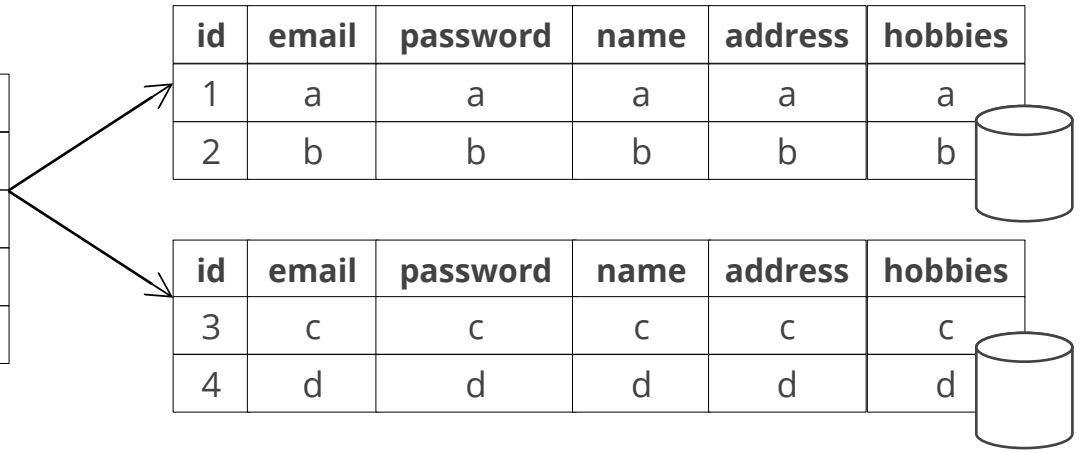


Distribución de datos

Particionado > Tipos

- Particionado horizontal (sharding)
  - Los elementos se distribuyen a partir de la clave de particionado

id	email	password	name	address	hobbies
1	a	a	a	a	a
2	b	b	b	b	b
3	c	c	c	c	c
4	d	d	d	d	d



Distribución de datos

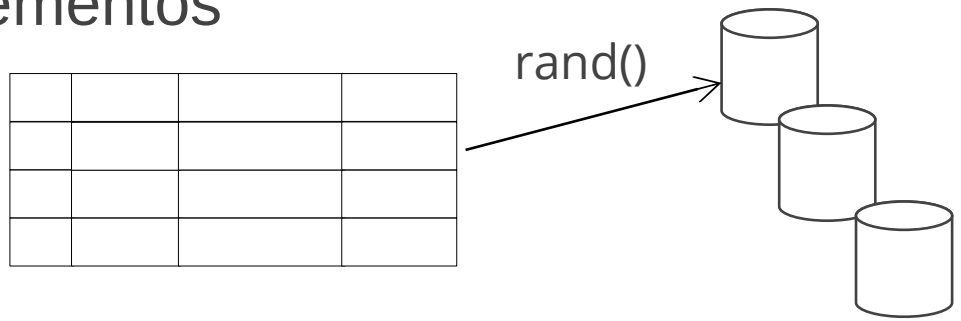
Particionado > Tipos

- Particionado horizontal (sharding)
  - Los elementos se distribuyen a partir de la clave de particionado
  - A partir de la clave de particionado, los elementos deben de distribuirse de manera equitativa
  - Distintos criterios
    - Aleatorio
    - Por valor
    - Por rango
    - Por hash

Distribución de datos

Particionado > Tipos

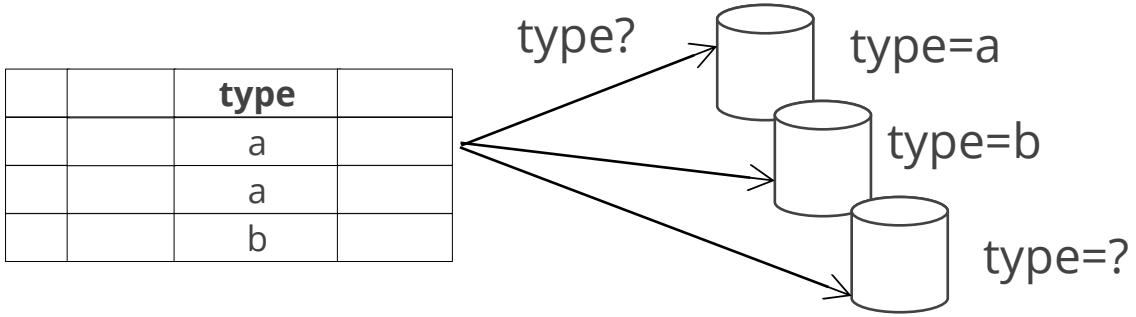
- Particionado horizontal (sharding)
  - Los elementos se distribuyen a partir de la clave de particionado
  - A partir de la clave de particionado, los elementos deben de distribuirse de manera equitativa
  - Distintos criterios
    - Aleatorio: problemas recordando dónde están los elementos



Distribución de datos

Particionado > Tipos

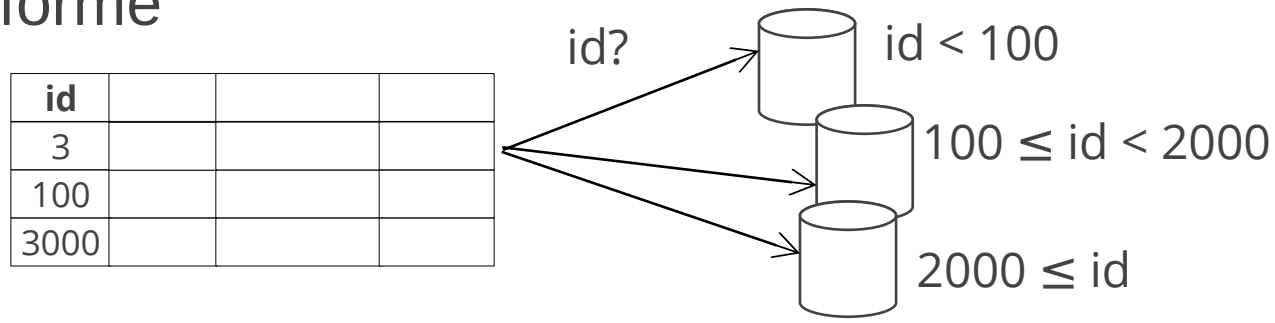
- Particionado horizontal (sharding)
  - Los elementos se distribuyen a partir de la clave de particionado
  - A partir de la clave de particionado, los elementos deben de distribuirse de manera equitativa
  - Distintos criterios
    - Por valor: no garantiza una distribución uniforme



Distribución de datos

Particionado > Tipos

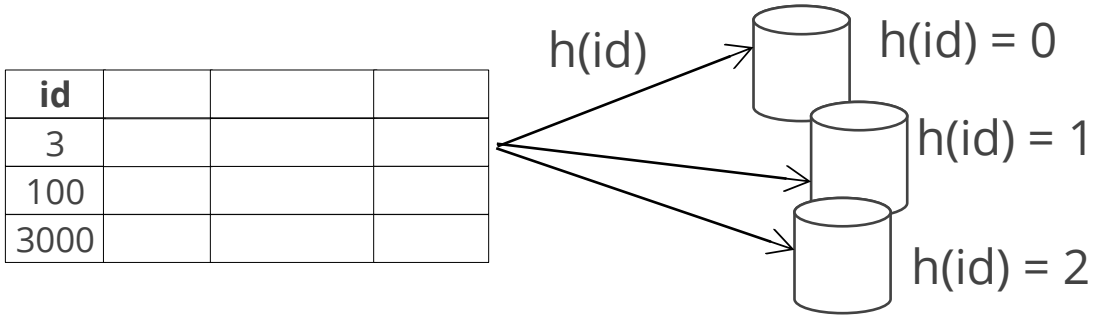
- Particionado horizontal (sharding)
  - Los elementos se distribuyen a partir de la clave de particionado
  - A partir de la clave de particionado, los elementos deben de distribuirse de manera equitativa
  - Distintos criterios
    - Por rango: consultas por rangos. No garantiza distribución uniforme



Distribución de datos

Particionado > Tipos

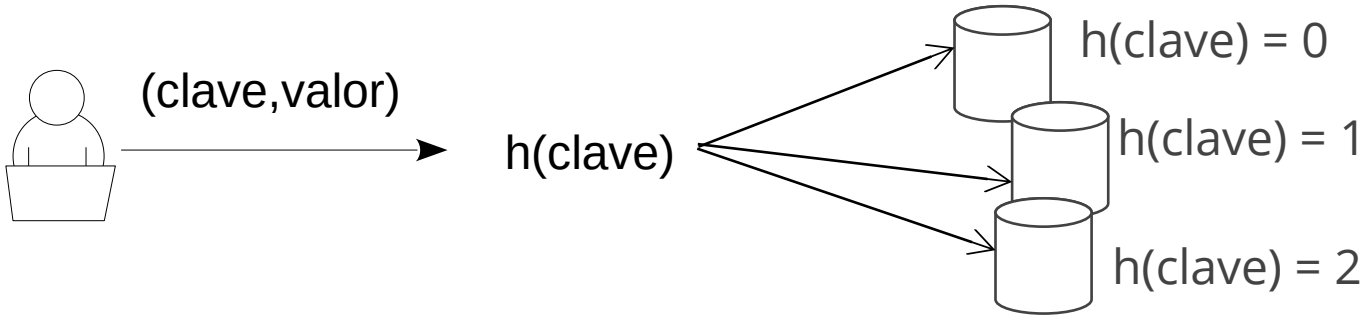
- Particionado horizontal (sharding)
  - Los elementos se distribuyen a partir de la clave de particionado
  - A partir de la clave de particionado, los elementos deben de distribuirse de manera equitativa
  - Distintos criterios
    - Por hash: consultas por rango costosas



Distribución de datos

Ejercicio 9 > Particionado

- A partir del diccionario implementado en el último ejercicio, crear una versión que utilice particionado horizontal utilizando la hash de las claves de los elementos



Distribución de datos

Replicación

- Técnica que mantiene copia de los mismos datos en distintos nodos interconectados
- Beneficios
  - Escalabilidad: repartir carga de trabajo entre nodos
  - Fiabilidad: reconfiguración de nodos automática
  - Latencia: servir desde nodos más próximos
- El problema es la consistencia de los datos: todas las réplicas tienen los mismos datos

Distribución de datos

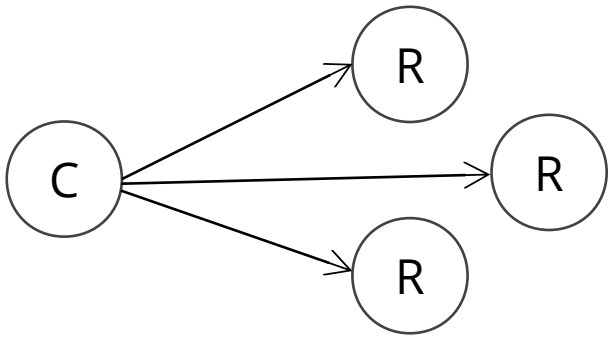
Replicación > Tipos

- Activa
- Pasiva
- Multi-líder

Distribución de datos

Replicación > Tipos

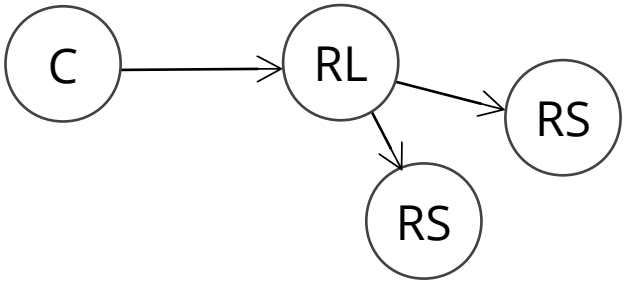
- Activa
  - Todas las peticiones difundidas en el mismo orden a todas las réplicas
  - Es necesario contar con un protocolo difusión atómica ordenada: muy costoso

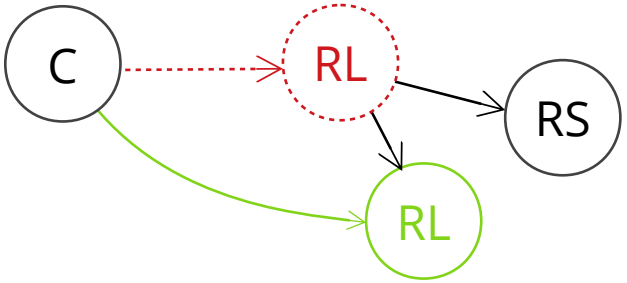


Distribución de datos

Replicación > Tipos

- Pasiva
  - Maestro-esclavo, líder-seguidor
  - Petición al líder; después difunde a seguidores
  - Si el líder falla, es reemplazado por un seguidor

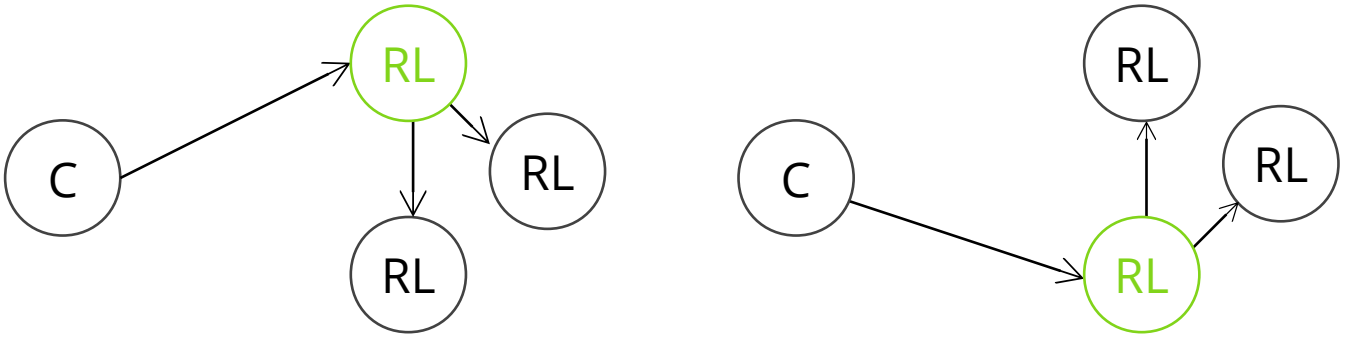




Distribución de datos

Replicación > Tipos

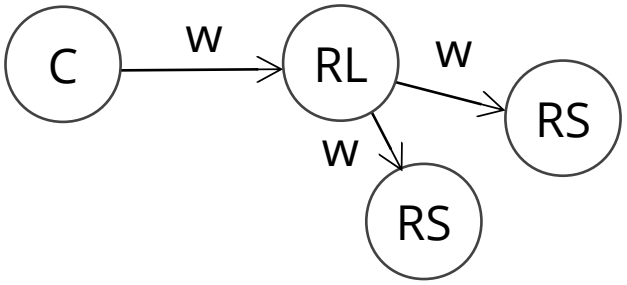
- Multi-líder
  - Coordinador-cohorte, peer-to-peer
  - Petición a un líder; se coordina con el resto
  - Es necesario un protocolo de consenso: muy costoso



Distribución de datos

Replicación pasiva

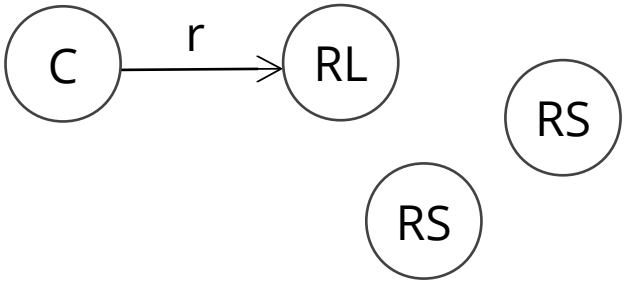
- Utilizada por muchos almacenes (MySQL, MongoDB,..)
- Operaciones escritura al líder. Líder propaga

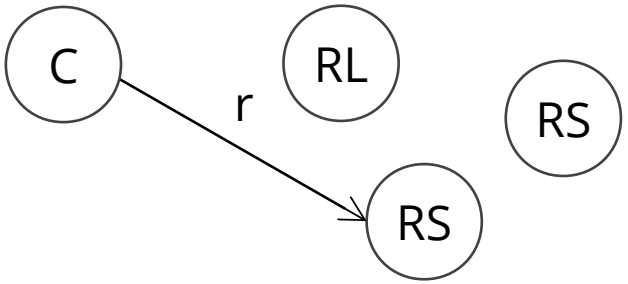


Distribución de datos

Replicación pasiva

- Utilizada por muchos almacenes (MySQL, MongoDB,..)
- Operaciones escritura al líder. Líder propaga
- Operaciones lectura a cualquier réplica.





Distribución de datos

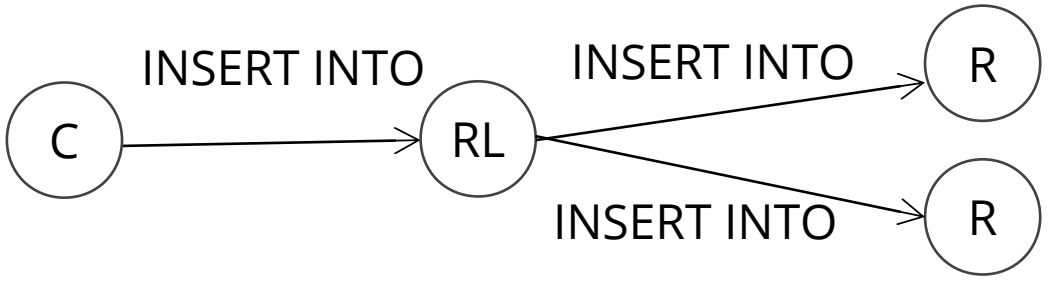
Replicación pasiva > Tipos de operaciones

- Sentencias
- Físicas (WAL)
- Lógicas

Distribución de datos

Replicación pasiva > Tipos de operaciones

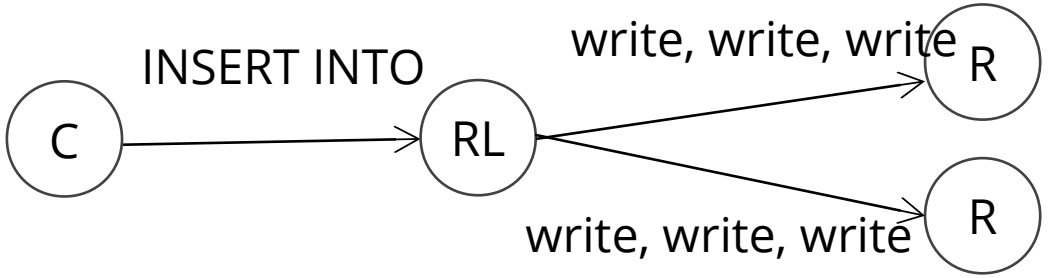
- Sentencias: alto nivel. Mismas operaciones que el cliente. Indeterminismo en now(), time(), rand()



Distribución de datos

Replicación pasiva > Tipos de operaciones

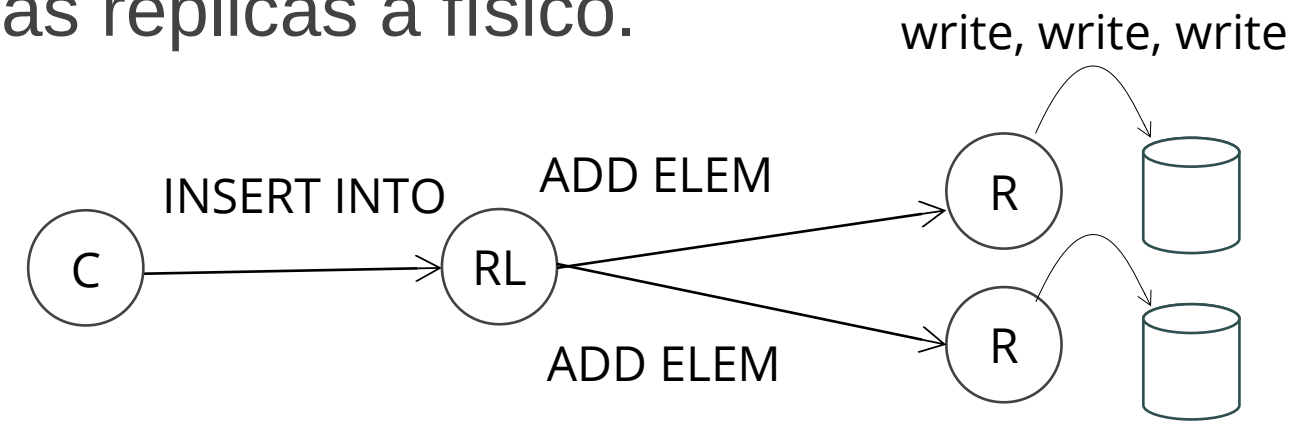
- Sentencias: alto nivel. Mismas operaciones que el cliente. Indeterminismo en now(), time(), rand()
- Físicas (WAL): bajo nivel (log). Más operaciones y fuerte acoplamiento con formato físico.



Distribución de datos

Replicación pasiva > Tipos de operaciones

- Sentencias: alto nivel. Mismas operaciones que el cliente. Indeterminismo en now(), time(), rand()
- Físicas (WAL): bajo nivel (log). Más operaciones y fuerte acoplamiento con formato físico.
- Lógicas: alto nivel. Lenguaje abstracto. Traducción en las réplicas a físico.



Distribución de datos

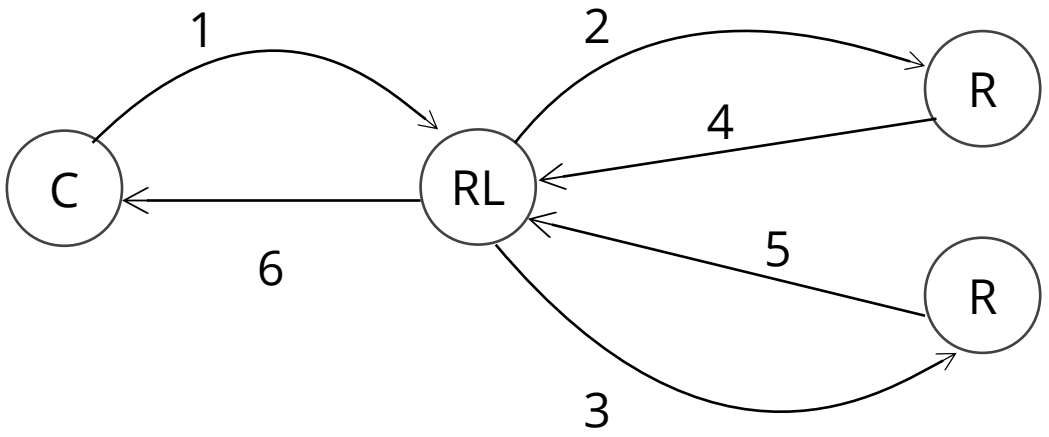
Replicación pasiva > Tipos de propagación

- Síncrona
- Asíncrona

Distribución de datos

Replicación pasiva > Tipos de propagación

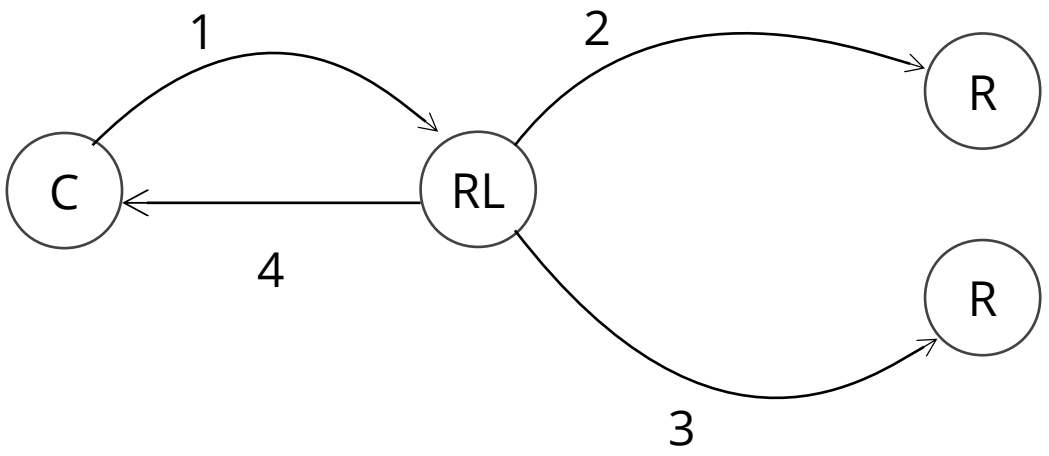
- Síncrona: se espera confirmación. Garantía de aplicación. Tiempo respuesta alto.



Distribución de datos

Replicación pasiva > Tipos de propagación

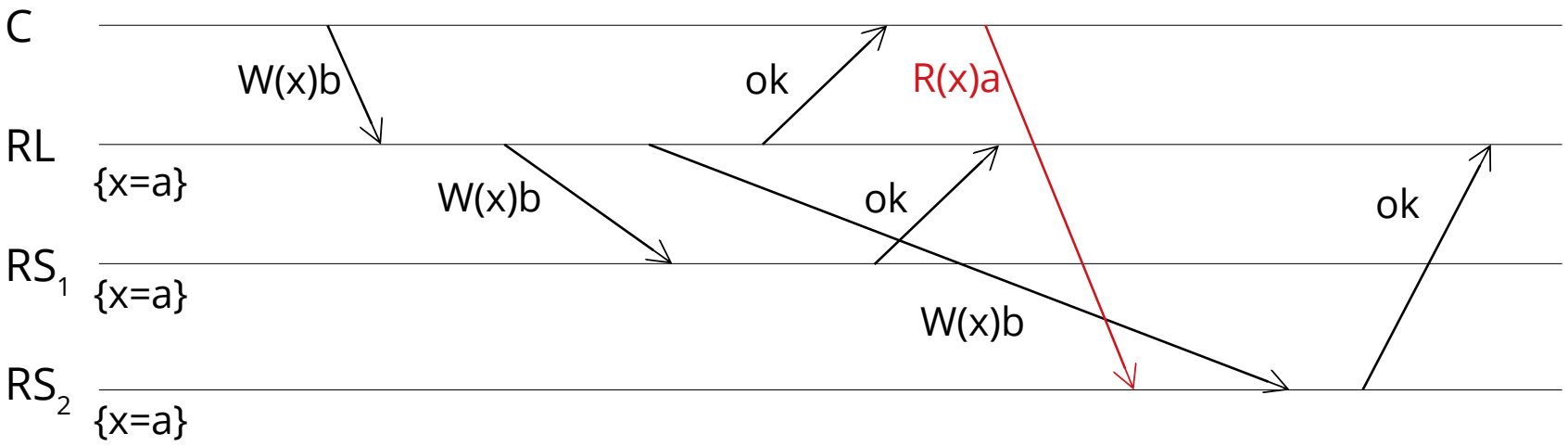
- Síncrona: se espera confirmación. Garantía de aplicación. Tiempo respuesta alto.
- Asíncrona: no se espera confirmación. Escrituras rápidas. Pueden encontrarse inconsistencias.



Distribución de datos

Replicación pasiva > Tipos de propagación

- Síncrona: se espera confirmación. Garantía de aplicación. Tiempo respuesta alto.
- Asíncrona: no se espera confirmación. Escrituras rápidas. Pueden encontrarse inconsistencias.



Distribución de datos

Ejercicio 10 > Replicación

- A partir del diccionario implementado en el último ejercicio, crear una versión que utilice replicación pasiva síncrona por medio de sentencias