CloudApps

MASTER

2.3 – Persistencia y Análisis de Datos

Tema 2 – BBDD multi-modelo y evolución de BBDD

Universidad Rey Juan Carlos

José Manuel Colmenar
josemanuel.colmenar@urjc.es
@jmcolmenarv

code URJC

Índice

- Introducción
- Bases de datos multi-modelo
- Soporte de JSON en BBDD relacionales: MySQL
- Evolución de esquemas
- Actualizaciones sin pérdida de servicio

Bibliografía

- Alex Petrov (2019). Database Internals. O'Reilly Media, Inc.
- Joel Ruisi (2018). Gaining Data Agility with Multi-Model Databases. O'Reilly Media, Inc.
- Piethein Strengholt (2020). Data Management at Scale. O'Reilly Media, Inc.
- Pethuru Raj; Ganesh Chandra Deka (2018). A Deep Dive into NoSQL Databases: The Use Cases and Applications. Academic Press.
- Peter Bakkum Kyle Banker Shaun Verch, Douglas Garrett, and Tim Hawkins (2016). MongoDB in Action, Second Edition: Covers MongoDB version 3.0. Manning Publications.
- Pramod J. Sadalage; Scott W. Ambler (2006). Refactoring Databases: Evolutionary Database Design. Published by Addison-Wesley Professional.
- Raj Malhotra (2019). Rapid Java Persistence and Microservices: Persistence Made Easy Using Java EE8, JPA and Spring. Published by Apress.

Introducción

- En desarrollo de aplicaciones, la **persistencia** se refiere a la capacidad de almacenar datos para su uso en el largo plazo.

- Existen diferentes paradigmas en el panorama de las bases de datos.

Introducción

- **Persistencia relacional**
  - Sigue un modelo estructurado basado en tablas.
  - Requiere un proceso previo de modelado para evitar redundancias.
  - Puede generar modelos complejos difíciles de evolucionar.
  - Muy madura: ORM, frameworks, …
- **Persistencia no relacional**
  - Basada en estructuras diferentes a tablas.
  - Esquemas flexibles.
  - No SQL en el sentido de “no solo SQL”.

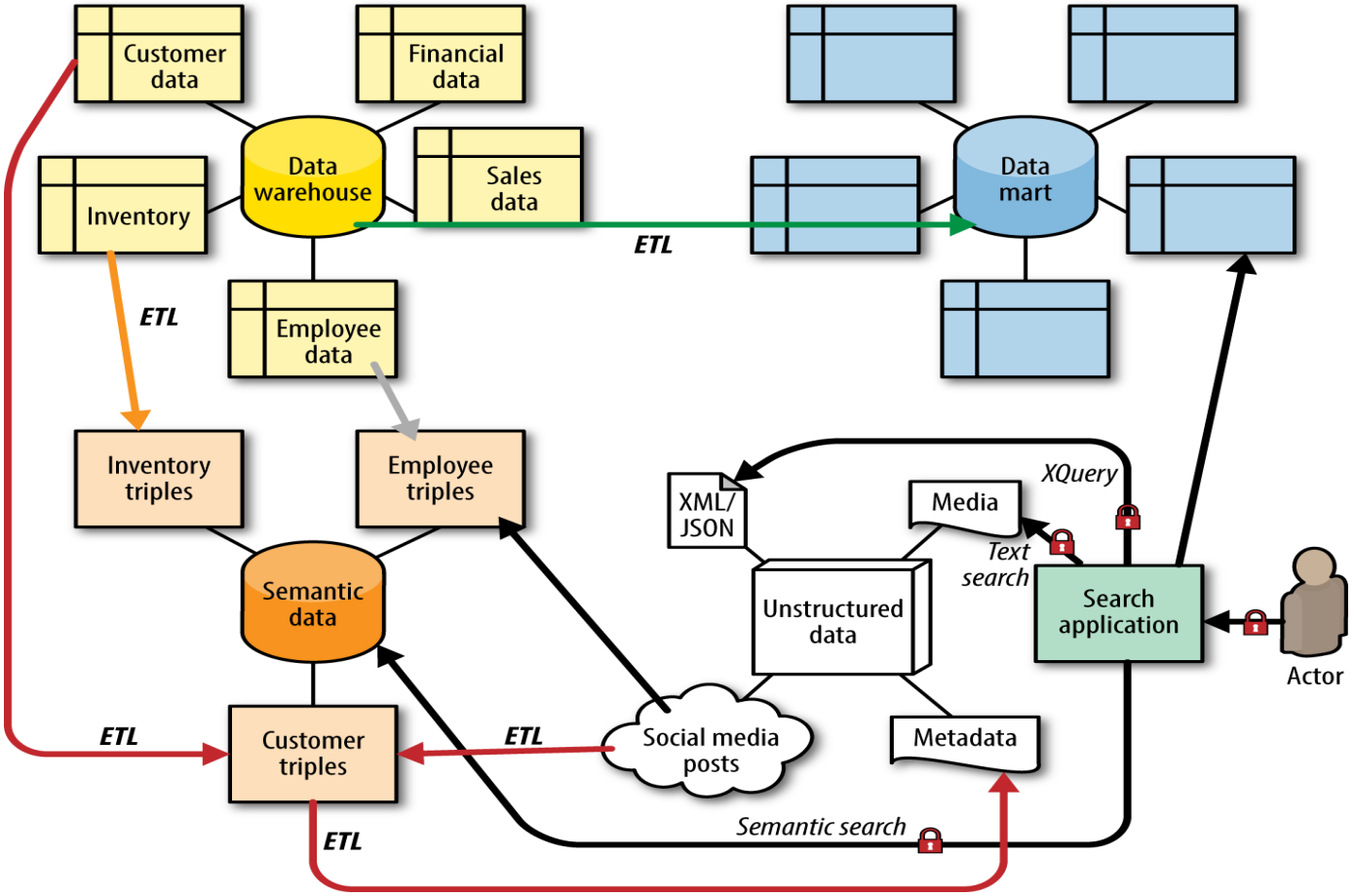
Introducción

- ¿Y si se necesita “un poco de todo”?
- ¿Existen soluciones adecuadas a los usos más comunes?

Bases de datos multi-modelo

- **Persistencia políglota**
  - Se utiliza la mejor implementación (modelo) para cada tipo de problema.
  - Se pueden mezclar diferentes fuentes de datos, pero su complejidad puede implicar un elevado coste de diseño y mantenimiento:
    - Mantenimiento de consistencia de datos.
    - Integridad.
    - Sincronización.
    - Integración de accesos de unas fuentes de datos a otras.
    - …

Bases de datos multi-modelo



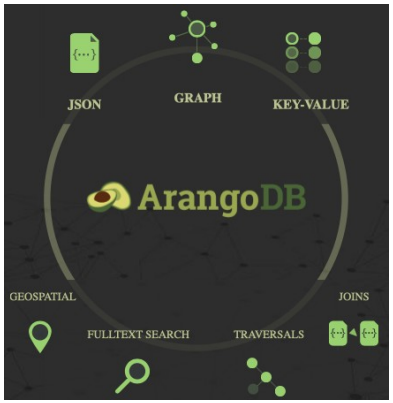
Complejidad a evitar. Fuente: Gaining Data Agility with Multi-Model Databases.

Bases de datos multi-modelo

- Una alternativa son las **bases de datos multi-modelo**.
  - Soportan múltiples modelos de datos en un solo *backend* integrado.
  - Utilizan estándares de datos y consultas apropiados a cada modelo.
  - Proporcionan consultas fluidas (*seamless querying*) entre los modelos soportados.
  - Soporte de modelo relacional, grafos, documentos, clave-valor, ...
- Las bases de datos multi-modelo proporcionan diferentes modelos de datos a diferentes necesidades sin la complejidad añadida de la persistencia políglota.
- Se pueden ir modelando datos según se necesitan en el proceso de desarrollo.

Bases de datos multi-modelo

- **ArangoDB**: https://www.arangodb.com
  - JSON, grafos, clave-valor.
  - Open source.
- **Couchbase**: https://www.couchbase.com
  - JSON, clave-valor.
  - Open source.
- **DataStax**: https://www.datastax.com
  - Basada en Cassandra.
  - Clave-valor, tablas, grafos.
- **MarkLogic**: https://www.marklogic.com
  - XML, JSON, grafos, texto, geoespacial, binario, SQL.
- **Oracle**: https://www.oracle.com/a/tech/docs/multimodel19c-wp.pdf
  - XML, JSON, grafos, texto, geoespacial, binario, SQL.



Alternativa: JSON en modelo relacional

- En ocasiones existe la necesidad de unir un modelo relacional con información no estructurada.
- Una combinación interesante puede ser incorporar **campos JSON dentro de una tabla**.
- Existen diferentes SGBD que soportan campos JSON. Por ejemplo:
  - MySQL:
    - https://dev.mysql.com/doc/refman/8.0/en/json.html
  - PostgreSQL:
    - https://www.postgresql.org/docs/12/datatype-json.html
  - Oracle:
    - https://docs.oracle.com/en/database/oracle/oracle-database/19/adjsn/index.html
- El soporte de JSON implica que **se validan los datos que se guardan**.

JSON en MySQL

- MySQL proporciona el **tipo de datos JSON**.
  - Valida automáticamente la corrección de los datos.
  - Optimiza el almacenamiento interno del campo, cuyo límite es similar a LONGBLOB o LONGTEXT.
- También proporciona **multitud de funciones** para realizar operaciones sobre los documentos almacenados.
  - https://dev.mysql.com/doc/refman/8.0/en/json-function-reference.html
- **No es posible crear índices** sobre campos JSON.
  - Se pueden crear campos generados automáticamente a partir de elementos del campo JSON.
  - Esos campos generados sí se pueden indexar.
  - https://dev.mysql.com/doc/refman/8.0/en/create-table-secondary-indexes.html

JSON en MySQL

- Ejemplo: tabla de productos con diferentes campos.

```
-- Creación de tabla:
CREATE TABLE producto(
   id BIGINT PRIMARY KEY AUTO_INCREMENT,
   datos JSON NOT NULL
)ENGINE=InnoDB;

-- Inserción de datos:
INSERT INTO producto(datos) values
('{"nombre": "Macbook", "marca": "Apple", "etiquetas":["Portátil","Mac"]}'),
('{"nombre": "P30 Lite 4", "marca": "Huawei", "etiquetas":["Móvil"]}'),
('{"nombre": "iPhone 11 Pro", "marca": "Apple", "etiquetas":["Móvil"]}');

-- Inserción que no funciona por no ser documento válido (falta } al final):
INSERT INTO producto(datos) values
('{"nombre": "EliteBook", "marca": "HP", "etiquetas":["Portátil"]');

-- Inserción correcta:
INSERT INTO producto(datos) values
('{"nombre": "EliteBook", "marca": "HP", "etiquetas":["Portátil"]}');
```

JSON en MySQL

Funciones para creación de JSON, con validación:

- JSON_ARRAY([val[, val] ...]) devuelve un array de valores.

```
SELECT JSON_ARRAY(25, "hola", NULL, FALSE, CURTIME());
```

- JSON_OBJECT([key, val[, key, val] ...]) devuelve un objeto JSON siempre que esté correctamente construido.

```
SELECT JSON_OBJECT('id', 44, 'nombre', 'Pantalla');
SELECT JSON_OBJECT('id', 44, 'marcadores', JSON_ARRAY('19-22','25-20','38-12'));
```

- JSON_QUOTE(string) añade comillas a cadenas de caracteres, “escapando” los símbolos especiales que contenga la cadena.

```
SELECT JSON_QUOTE('El invitado dijo "Hola"');
```

- JSON_UNQUOTE(json_val) devuelve la cadena de caracteres de un valor del documento, pero sin comillas.

```
SELECT JSON_UNQUOTE('"Quita comillas"');
```

JSON en MySQL

- El *path* es la ruta o camino hasta un valor de un documento JSON.
- Se utiliza el $ seguido de punto y el selector (etiqueta) requerido.
  - $.marca : indica el contenido de la etiqueta *marca*.
  - $[1] : indica la segunda posición del array.
- Ejemplo: [23, {"x": [1, 2, 3], "un número": 5}, [2, 10]]
  - $[0] devuelve 23.
  - $[1] devuelve {"x": [1, 2, 3], "un número": 5}.
  - $[3] devuelve NULL porque se refiere a un elemento que no existe.
  - $[1].x devuelve [1, 2, 3].
  - $[1].x[1] devuelve 2.
  - $[1].”un número” devuelve 5.

JSON en MySQL

Funciones de búsqueda y parseo en JSON:

- JSON_EXTRACT(json_doc, path[, path] ...) parsea el documento y devuelve un array con los valores que se indican en los paths. Si solamente hay un path, devuelve un valor sin array.
- column->path : es un alias de JSON_EXTRACT.
- Se pueden utilizar en cualquier sentencia SELECT, DELETE o UPDATE.

```
-- Productos marca Apple:
SELECT * FROM producto p
WHERE JSON_EXTRACT(p.datos,"$.marca") = 'Apple';

-- Misma consulta:
SELECT * FROM producto p
WHERE p.datos->"$.marca" = 'Apple';

-- Borra productos Huawei:
DELETE FROM producto WHERE datos->"$.marca" = "Huawei";
```

JSON en MySQL

Funciones de búsqueda y parseo en JSON:

- JSON_CONTAINS(target, candidate[, path]) devuelve 1 si el documento *candidate* se encuentra en target o, en caso de proporcionar path, si el candidato se encuentra en ese path. 0 en caso contrario.
- JSON_CONTAINS_PATH(json_doc, one_or_all, path[, path] ...)
  - Si se pasa 'one', devuelve 1 se existe al menos un path.
  - Si se pasa 'all', devuelve 1 se existen todos los paths.

```
-- Productos con etiqueta 'Móvil':
-- El literal requiere comillas dobles dentro de las simples.
SELECT * FROM producto p WHERE
JSON_CONTAINS(p.datos,'"Móvil"','$.etiquetas')=1;
```

JSON en MySQL

Funciones para modificación de JSON:

- JSON_SET(json_doc, path, valor[, path, valor] ...) reemplaza los valores existentes (UPDATE) y añade los valores que no existen (INSERT).
- JSON_INSERT(json_doc, path, valor[, path, valor] ...) inserta valores sin reemplazar los existentes.
- JSON_REPLACE(json_doc, path, valor[, path, valor] ...) reemplaza solamente valores existentes.
- JSON_ARRAY_APPEND(json_doc, path, val[, path, val] ...) añade valores al final de los arrays indicados en los *path*.

```
-- Actualización cambiando el valor de una etiqueta:
UPDATE producto SET datos = JSON_REPLACE(datos, '$.marca', 'Apple $$')
WHERE JSON_EXTRACT(datos,"$.marca") = 'Apple';

-- Actualización añadiendo elementos al array JSON:
UPDATE producto SET datos = JSON_ARRAY_APPEND(datos, '$.etiquetas', 'Material plástico')
WHERE JSON_EXTRACT(datos,"$.nombre") = 'EliteBook';
```

JSON en MySQL

Ejemplo: tabla de productos con diferentes campos **en Spring Data**.

- **pom.xml**

```
<groupId>es.urjccode</groupId>
  <artifactId>springdata_json_column</artifactId>
  <version>1.0-SNAPSHOT</version>

  <!-- Spring Boot necesario como proyecto padre -->
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId> <version>2.6.0</version>
  </parent>

  <dependencies>

    <!-- Spring data -->
    <dependency>  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-data-jpa</artifactId> </dependency>

    <!-- Driver MySQL -->
    <dependency>  <groupId>mysql</groupId>
                  <artifactId>mysql-connector-java</artifactId> </dependency>

    <!-- Objetos JSON -->
    <dependency> <groupId>org.json</groupId> <artifactId>json</artifactId>
                 <version>20211205</version> </dependency>
    ...
 </dependencies>
```

JSON en MySQL

Ejemplo: tabla de productos con diferentes campos **en Spring Data**.

- Entidad **Producto**: el campo “datos” es un String pero en la BD es JSON.
  - Anotación `@Column(columnDefinition="json")`

```
@Entity
public class Producto {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    // El objeto JSON se maneja como String en la entidad
    @Column(columnDefinition="json")
    private String datos;

    public Producto() {  }

    public Producto(String datos) { this.datos = datos; }

    public long getId() { return id; }

    public void setId(long id) { this.id = id; }

    public String getDatos() { return datos; }

    public void setDatos(String datos) { this.datos = datos; }
   ...
 }
```

JSON en MySQL

Ejemplo: tabla de productos con diferentes campos **en Spring Data**.

- **application.properties** con **ddl-auto=update**.

```
spring.datasource.url=jdbc:mysql://localhost/test
spring.datasource.username=root
spring.datasource.password=pass
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
```

- No es necesario crear la tabla, aunque su código sería el siguiente:

```
-- Tabla de productos:
CREATE TABLE producto(
   id BIGINT PRIMARY KEY NOT NULL,
   datos JSON NOT NULL
)ENGINE=InnoDB;
```

JSON en MySQL

Ejemplo: tabla de productos con diferentes campos **en Spring Data**.

- Cargador de datos con diferentes elementos: **inserción**

```
@Controller
public class DatabaseLoader implements CommandLineRunner {

    @Autowired
    private ProductoRepository repository;

    @Override
    public void run(String... args) {

        // Guardando algunos productos
        repository.save(new Producto("{\"nombre\": \"Macbook\", \"marca\": \"Apple\", " +
            "\"etiquetas\":[\"Portátil\",\"Mac\"]}"));
        repository.save(new Producto("{\"nombre\": \"P30 Lite 4\", \"marca\": \"Huawei\", " +
            "\"etiquetas\":[\"Móvil\"]}"));
        repository.save(new Producto("{\"nombre\": \"iPhone 11 Pro\", \"marca\": \"Apple\", " +
            "\"etiquetas\":[\"Móvil\"]}"));
        repository.save(new Producto("{\"nombre\": \"Teclado 105\", \"marca\": \"Logitech\"}"));

        // Producto a partir del toString de un objeto JSON
        JSONObject jo = new JSONObject();
        jo.put("nombre", "EliteBook");
        jo.put("marca", "HP");
        JSONArray jsonArray = new JSONArray();
        jsonArray.put("Portátil");
        jsonArray.put("Ligero");
        jo.put("etiquetas",jsonArray);
        repository.save(new Producto(jo.toString()));
   ...
```

JSON en MySQL

Ejemplo: tabla de productos con diferentes campos **en Spring Data**.

- Cargador de datos con diferentes elementos: **consultas**

```
...
        // Recupera productos
        List<Producto> productos = repository.findAll();
        System.out.println("Productos con findAll():");
        System.out.println("-------------------------------");
        productos.forEach(System.out.println);

        // Recupera productos por marca
        String marca = "Apple";
        productos = repository.findByMarca(marca);
        System.out.println("Productos "+marca+":");
        System.out.println("-------------------------------");
        productos.forEach(System.out.println);

        // Recupera productos por etiqueta
        String etiqueta = "Portátil";
        productos = repository.findByEtiqueta(etiqueta);
        System.out.println("Productos con etiqueta \""+etiqueta+"\":");
        System.out.println("-------------------------------");
        productos.forEach(System.out.println);
    }  // run

}
```

JSON en MySQL

Ejemplo: tabla de productos con diferentes campos **en Spring Data**.

- **Repositorio**. Se utilizan funciones JSON de MySQL a través de la palabra reservada **FUNCTION**.

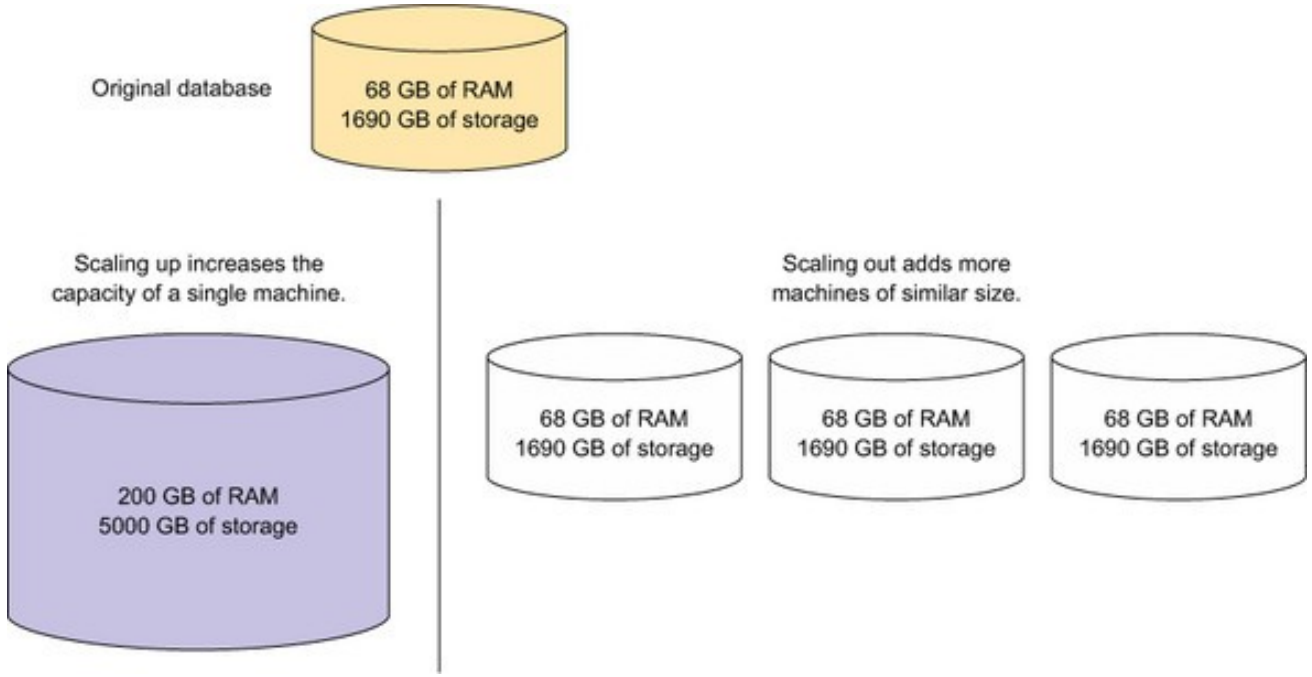
```
public interface ProductoRepository extends JpaRepository<Producto, Long> {

// Consulta por la marca del campo datos
@Query("select p from Producto p where FUNCTION('JSON_EXTRACT',p.datos,'$.marca') = ?1")
List<Producto> findByMarca(String marca);

// Consulta por la marca del campo datos consulta nativa
@Query(value = "select * from producto p where JSON_EXTRACT(p.datos,'$.marca') = ?1", nativeQuery = true)
List<Producto> findByMarcaNativa(String marca);

// Consulta por una etiqueta dada (hacen falta las comillas)
@Query("select p from Producto p where FUNCTION('JSON_CONTAINS',p.datos, " +
        "JSON_QUOTE(?1),'$.etiquetas') = 1")
List<Producto> findByEtiqueta(String etiqueta);

// Alternativa: usar el CONCAT
//    @Query("select p from Producto p where FUNCTION('JSON_CONTAINS',p.datos, " +
//           "CONCAT('\"',?1,'\"'),'$.etiquetas') = 1")

}
```

Código en GitHub:

https://github.com/MasterCloudApps/2.3.Persistencia-y-analisis-de-datos/tree/master/tema2/ejemplo1_springdata_json_column

Escalabilidad en BBDD

- Escalabilidad horizontal (*scale out*) vs vertical (*scale up*).



Fuente: MongoDB in Action, Second Edition: Covers MongoDB version 3.0 by Peter Bakkum Kyle Banker Shaun Verch, Douglas Garrett, and Tim Hawkins. Manning Publications, 2016

Escalabilidad horizontal

- La escalabilidad horizontal implica la inclusión de más máquinas similares.
- En términos de BBDD, se trata de añadir más nodos de similares características:
  - Mismo SGBD.
  - Mismos datos (replicados o troceados).
  - ¿Mismo modelo de datos?

Escalabilidad horizontal

Replicación de bases de datos:

- Varias bases de datos pueden trabajar juntas para:
  - Conseguir tolerancia a fallos.
    - Una BD secundaria toma el relevo cuando la primaria falla.
  - Balancear la carga.
    - Las consultas se distribuyen entre varias BBDD.

Escalabilidad horizontal

Tolerancia a fallos:

- Un servidor denominado **primario** o **maestro** atiende las peticiones de lectura y escritura.
- Un servidor denominado **secundario** o **esclavo** reproduce todas las operaciones de escritura del maestro.
- Cuando la conexión con el primario se pierde, el secundario toma el relevo garantizando la disponibilidad de la aplicación.
- Modelo que siguen los *replica sets* de MongoDB.

Escalabilidad horizontal

Balanceo de carga:

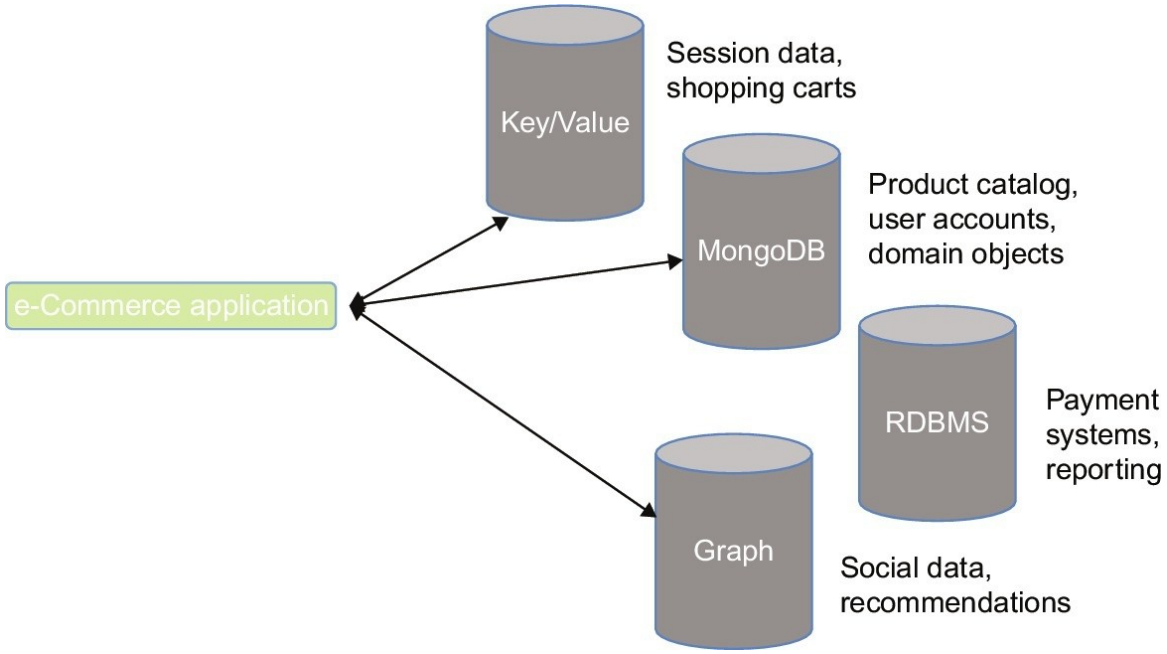
- Múltiples servidores se reparten las lecturas (balanceo).
  - Se puede conseguir con *haproxy*, por ejemplo.
    - http://www.haproxy.org , https://www.haproxy.com
- Para balancear las escrituras hay que recurrir al *sharding*.
  - Los datos se dividen en base a algún criterio y cada porción la gestiona un servidor diferente.

Escalabilidad vertical

- La escalabilidad vertical implica la mejora de recursos (CPU, memoria, disco) de la máquina actual.
- En términos de BBDD, se trata de soportar el crecimiento de una misma BD:
  - ¿Un solo SGBD?
  - ¿Mismo modelo de datos?

Escalabilidad vertical

- La persistencia políglota permite el escalado vertical de un sistema.
  - Hay una sola base de datos desde el punto de vista lógico.
  - Se necesita conocimiento sobre los diferentes modelos que se integren.



Ejemplo de persistencia políglota. Fuente: A Deep Dive into NoSQL Databases: The Use Cases and Applications.

Evolución de bases de datos

- Desde el punto de vista del desarrollo, es interesante considerar la **evolución de la BD**.
- Ventajas de un enfoque evolutivo:
  - Se minimiza la "basura": si un requisito deja de hacer falta, todo su desarrollo queda obsoleto.
  - Se evitan las modificaciones sobre un mismo requisito (*rework*): si un requisito va a cambiar, se anticipan sus cambios a priori, así como toda su evolución.
  - Se reduce el riesgo: el sistema siempre funciona y siempre se está probando con una BD operativa.
  - La BD es siempre la mejor: se aplican refactorizaciones para mantener el mejor esquema.
  - Se reduce el esfuerzo total porque se destina a las tareas actuales y no a las futuras.

Evolución de bases de datos

- Desventajas de un enfoque evolutivo:
  - Menor adaptación a las técnicas ágiles en profesionales de BBDD.
  - Curva de aprendizaje mayor si además hay que cambiar de mentalidad.
  - Es necesario el apoyo de herramientas específicas de evolución de bases de datos.

Evolución de bases de datos

(Recordatorio) **Estrategia de desarrollo con BBDD en Java + Spring Data**:

- Fase 1: Desarrollo: Usar base de datos en memoria (H2).
- Fase 2: Desarrollo: Usar base de datos persistente (MySQL) con *create-drop*.
- Fase 3: Instalación: Usar base de datos persistente con *update*.
- Fase 4: **Actualización**: Usar herramientas específicas.

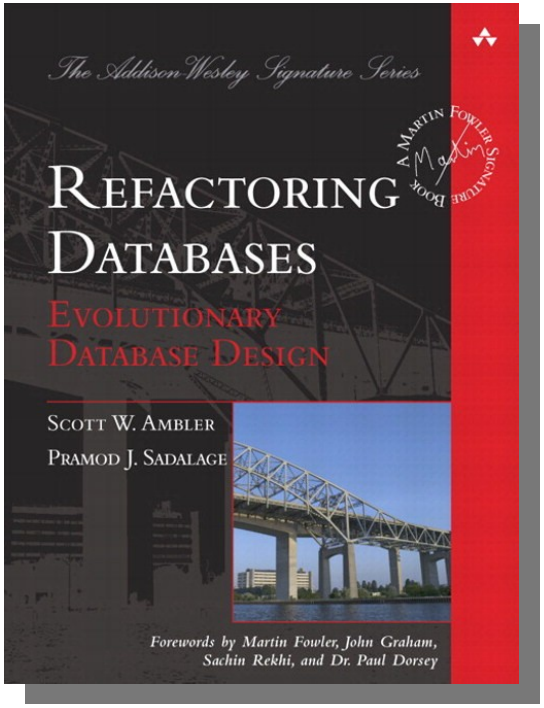
Evolución de bases de datos

- Fase 4: **Actualización**: Usar herramientas específicas.
  - Si se desarrolla una nueva versión de una aplicación y esa nueva versión tiene algún cambio en las entidades no se puede instalar sin más.
  - Es necesario hacer cambios en la BBDD para el nuevo esquema, pero la BBDD ya tiene datos.
  - Hay que usar herramientas específicas.

Evolución de bases de datos

- **Migración** del esquema de la BBDD:
  - Cuando una aplicación con BBDD se empieza a usar en producción, se empiezan a guardar datos en la misma.
  - Es posible que la aplicación evolucione:
    - Incluyendo nuevas entidades.
    - Modificando las entidades existentes.
  - Para actualizar la aplicación en producción, antes hay que migrar la BBDD al nuevo esquema.

Evolución de bases de datos



Refactoring Databases: Evolutionary Database Design

by Scott W. Ambler and Pramod J. Sadalage

Addison Wesley Professional
ISBN#: 0-321-29353-3

http://www.ambysoft.com/books/refactoringDatabases.html

Evolución de bases de datos

- Existen (al menos) dos herramientas externas que facilitan la evolución del esquema:

http://flywaydb.org/

Database Migrations Made Easy

Gestiona la migración con sentencias SQL específicas de la BBDD

http://www.liquibase.org/

Source Control for your Database

Gestiona la migración con XML genérico independiente de la de la BBDD

Evolución de bases de datos

- **Flyway y Liquibase** almacenan la migración/evolución de la base de datos en una tabla:
  - `flyway_schema_history` en Flyway.
  - `DATABASECHANGELOG` en Liquibase.
- Cada fila de la tabla almacena los datos de una versión: id, fecha de evolución, script utilizado, usuario que realizó la evolución, estado, etc.
- Estas herramientas admiten diferentes formatos para los scripts de evolución: SQL, XML, YAML, …
- Es posible utilizar ambas herramientas desde **línea de comandos**.

Evolución de bases de datos

- **Flyway y Liquibase** se pueden utilizar desde Java/Spring.
- Los scripts con los cambios se almacenan en el **directorio** `resources` siguiendo la sintaxis indicada por la herramienta.
- Cuando la aplicación Spring se inicia, se pueden ejecutar de **forma automática** estos cambios para convertir el esquema en la versión deseada.
  - Se compara el estado de la BD con la serie de cambios reflejados en el código.
- Ejemplos:
  - https://github.com/spring-projects/spring-boot/tree/1.3.x/spring-boot-samples/spring-boot-sample-flyway
  - https://github.com/spring-projects/spring-boot/tree/1.3.x/spring-boot-samples/spring-boot-sample-liquibase

Evolución de bases de datos

Ejemplo: Flyway en Java/Spring con MySQL.

- **pom.xml :**

```
...
  <!-- Spring Boot necesario como proyecto padre -->
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId> <version>2.6.0</version>
  </parent>

  <dependencies>

    <!-- Spring data -->
    <dependency>  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-data-jpa</artifactId> </dependency>

    <!-- Driver MySQL -->
    <dependency>  <groupId>mysql</groupId>
                  <artifactId>mysql-connector-java</artifactId> </dependency>

   <!-- Flyway -->
    <dependency> <groupId>org.flywaydb</groupId> <artifactId>flyway-core</artifactId>
    </dependency>
 ...
 </dependencies>
```

Evolución de bases de datos

Ejemplo: Flyway en Java/Spring con MySQL.

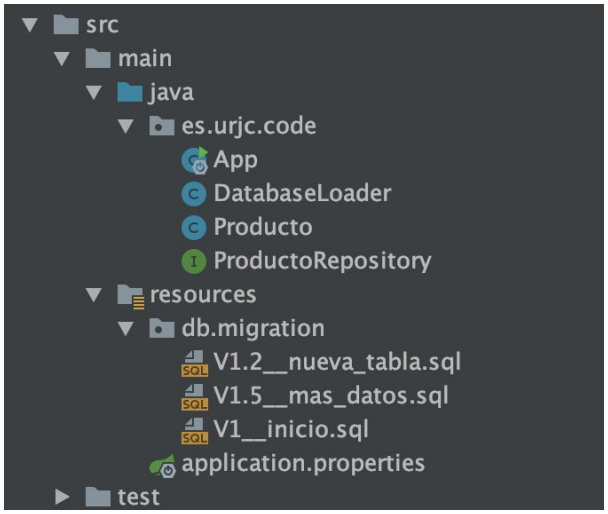
- Los scripts de migración se guardan en `resources/db/migration`.
- El nombre de los scripts de migración debe empezar por V seguido de la versión y, tras dos guiones bajos "__", la descripción del script.
  - Ejemplo: `V1__inicio.sql`
- Si se intenta incluir a posteriori una migración entre versiones, se genera un error:

```
Caused by: org.flywaydb.core.api.FlywayException: Validate failed:

Detected resolved migration not applied to database: 1.2
```

- La tabla de migraciones no se ve afectada por la opción `spring.jpa.hibernate.ddl-auto=create-drop`

Evolución de bases de datos

Ejemplo: Flyway en Java/Spring con MySQL.



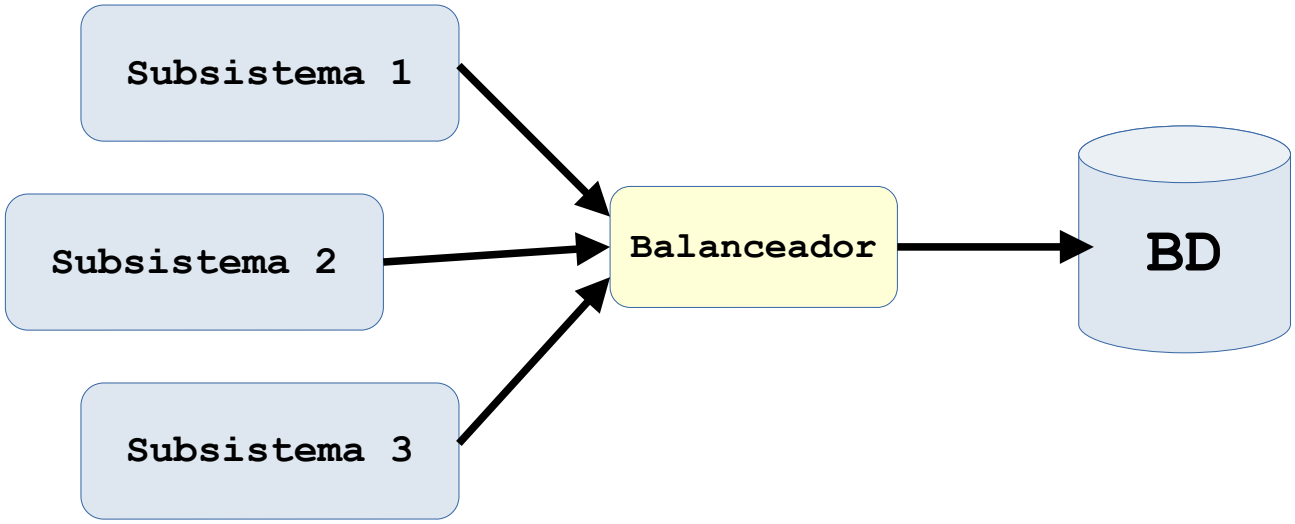

```
main] es.urjc.code.App                          : Starting App on mp-dep2-d116-01.escet.urjc.es with PID 32287 (/Use
main] es.urjc.code.App                          : No active profile set, falling back to default profiles: default
main] .s.d.r.c.RepositoryConfigurationDelegate  : Bootstrapping Spring Data repositories in DEFAULT mode.
main] .s.d.r.c.RepositoryConfigurationDelegate  : Finished Spring Data repository scanning in 33ms. Found 1 repositor
main] o.f.c.internal.license.VersionPrinter     : Flyway Community Edition 6.0.6 by Redgate
main] com.zaxxer.hikari.HikariDataSource        : HikariPool-1 - Starting...
main] com.zaxxer.hikari.HikariDataSource        : HikariPool-1 - Start completed.
main] o.f.c.internal.database.DatabaseFactory   : Database: jdbc:mysql://localhost/test (MySQL 8.0)
main] o.f.core.internal.command.DbValidate      : Successfully validated 3 migrations (execution time 00:00.008s)
main] o.f.c.i.s.JdbcTableSchemaHistory          : Creating Schema History table `test`.`flyway_schema_history` ...
main] o.f.core.internal.command.DbMigrate       : Current version of schema `test`: << Empty Schema >>
main] o.f.core.internal.command.DbMigrate       : Migrating schema `test` to version 1 - inicio
main] o.f.core.internal.command.DbMigrate       : Migrating schema `test` to version 1.2 - nueva tabla
main] o.f.core.internal.command.DbMigrate       : Migrating schema `test` to version 1.5 - mas datos
main] o.f.core.internal.command.DbMigrate       : Successfully applied 3 migrations to schema `test` (execution time
main] o.hibernate.jpa.internal.util.LogHelper   : HHH000204: Processing PersistenceUnitInfo [name: default]
main] org.hibernate.Version                     : HHH000412: Hibernate Core {5.4.6.Final}
```

Código en GitHub:

https://github.com/MasterCloudApps/2.3.Persistencia-y-analisis-de-datos/tree/master/tema2/ejemplo2_springdata_flyway_mysql

Actualizaciones sin parada

- Los sistemas con alta demanda usualmente ejecutan varias instancias de un subsistema que acceden a la BD a través de un balanceador de carga.



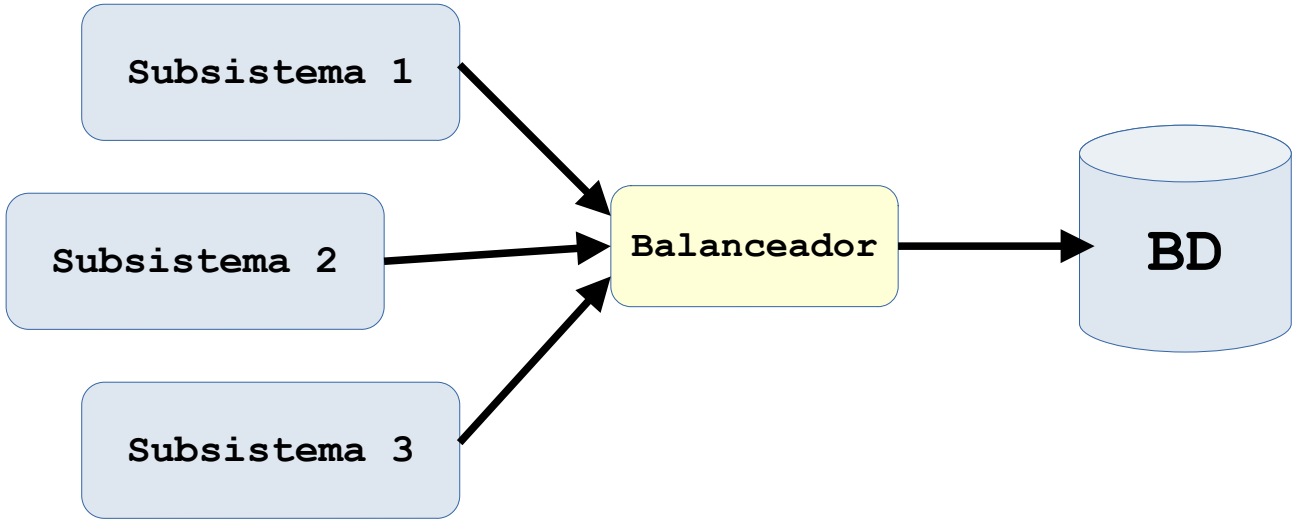
Actualizaciones sin parada

- La actualización de alguna funcionalidad requiere actualizar todas las instancias del subsistema.

- ***Rolling update***: consiste en apagar, actualizar y reiniciar las instancias del subsistema sin detener el servicio:
  - Conviven, al menos, dos versiones del sistema a la vez.
  - La BD debe soportar accesos concurrentes de todas las versiones simultáneas de la aplicación.

Actualizaciones sin parada

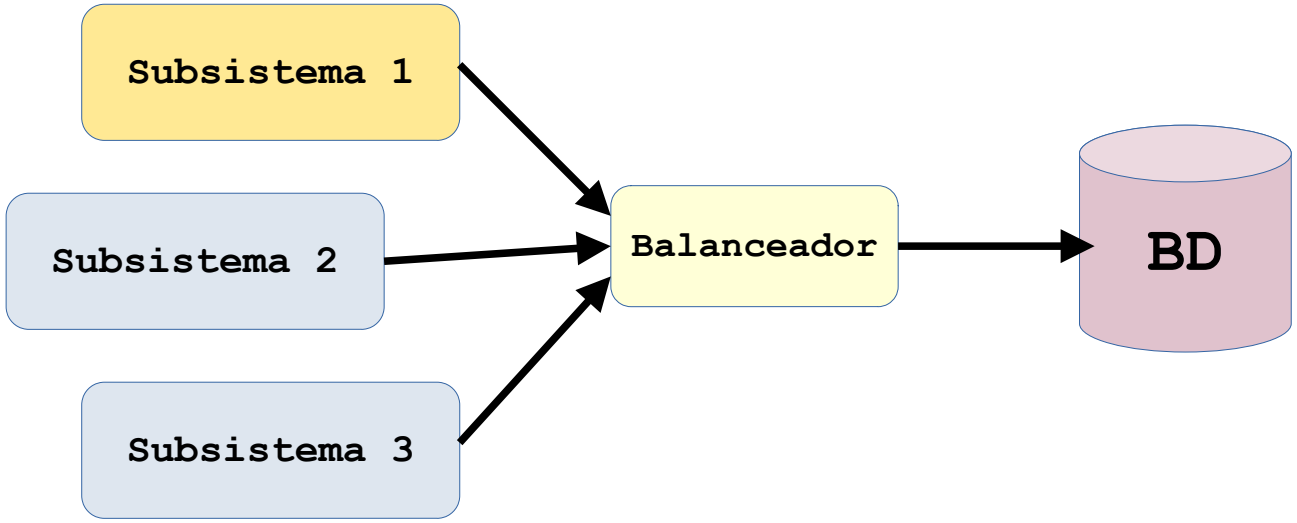
Rolling update



Inicio: instancias y BD previas a la actualización

Actualizaciones sin parada

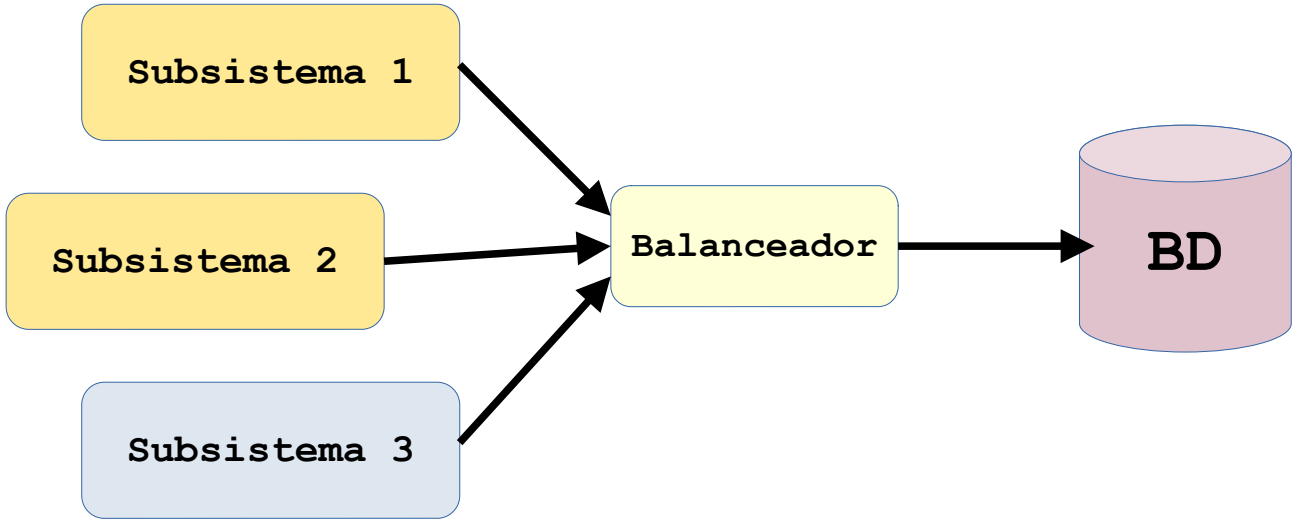
Rolling update



En proceso: instancias con diferentes versiones y BD soportando ambas instancias.

Actualizaciones sin parada

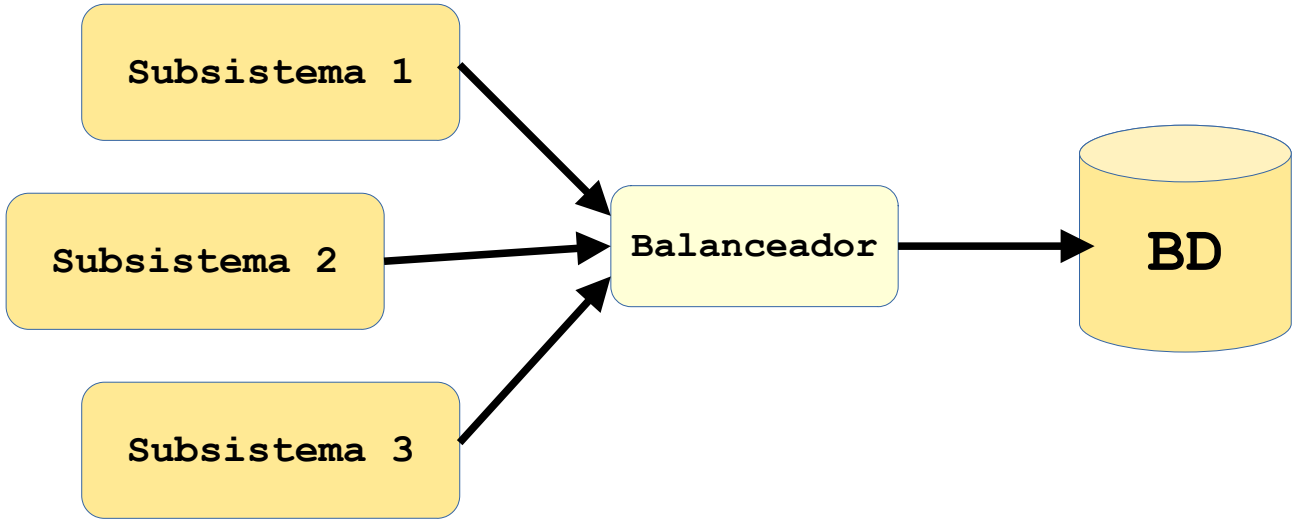
Rolling update



En proceso: instancias con diferentes versiones y BD soportando ambas instancias.

Actualizaciones sin parada

Rolling update



Completado: instancias actualizadas y BD soportando nueva versión.

Actualizaciones sin parada

- El mecanismo de actualización depende de las operaciones a realizar:
  - *Backward-compatible*
  - *Backward-incompatible*

- Software de evolución de BBDD como *Flyway* puede ayudar.

Actualizaciones sin parada

Backward-compatible

- Operaciones compatibles con ambas versiones de la BD.
- No es necesario dividir la actualización de la BD en múltiples pasos.
- Ejemplos:
  - Añadir tabla, vista, columna.
  - Eliminar columna no usada.
  - Eliminar restricciones.

Actualizaciones sin parada

Añadir tabla o vista:

- Es *backward-compatible*
- Instancias de versión anterior no usan la nueva.
- Las instancias de la versión anterior ejecutan a la vez que las de la nueva versión:
  - Sus escrituras puede que se tengan que "limpiar" para que se tengan en cuenta en la nueva versión.

Actualizaciones sin parada

Añadir columna:

- Si permite nulos es *backward-compatible*.
- Si no permite nulos se puede definir valor por defecto para ser *backward-compatible*.
- Si no permite nulos ni valor por defecto, tres pasos:
  1. Añadir columna sin valor por defecto y actualizar todas las instancias de la aplicación.
  2. Ejecutar un script para rellenar todos los datos.
  3. Añadir la restricción NOT NULL.
- Los tres pasos anteriores se pueden hacer a la vez con Flyway.

Actualizaciones sin parada

Eliminar columna no usada ni en la nueva ni en la anterior versión:

- Es *backward-compatible*.

Eliminar restricción:

- Es *backward-compatible* porque las versiones anteriores pueden seguir escribiendo como lo hacían.
- Durante *rolling update* la nueva versión puede escribir datos que incumplan la restricción eliminada.
  - Puede provocar fallos en las instancias previas.
  - No hay una solución buena salvo ¡actualizar rápido!

Actualizaciones sin parada

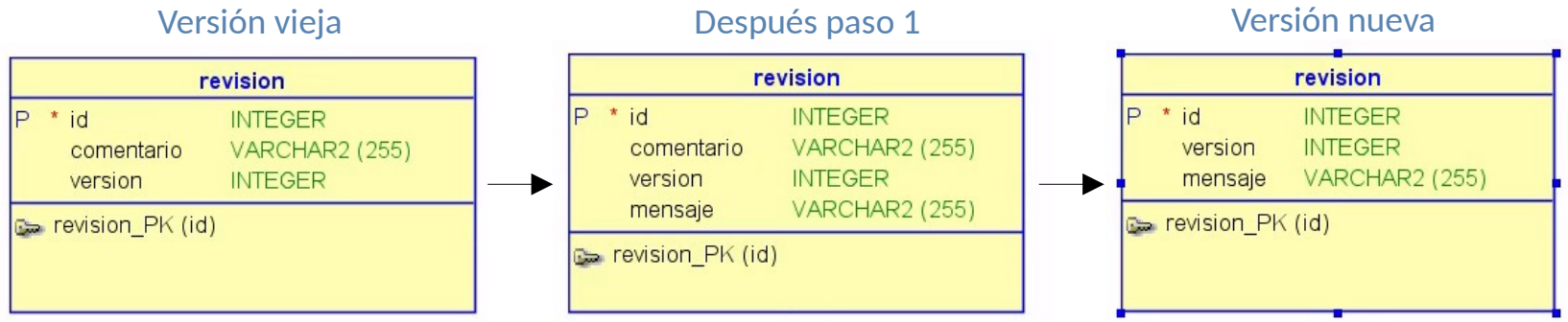
Backward-incompatible:

- Cambios en el esquema que hacen que las versiones anteriores de la aplicación no puedan usar la BD.
- Normalmente se ejecutan dividiendo los cambios entre *backward-compatible* y *backward-incompatible*:
  - Ejecución de la parte *backward-compatible*.
    - Suele implicar añadir tablas/columnas.
  - Se actualizan todas las instancias.
- Ejecución de la parte *backward-incompatible*.
  - Las instancias están actualizadas, por lo que no hay problema.
  - Puede requerir “limpiar” los elementos de transición.

Actualizaciones sin parada

Renombrar columna:

- Proceso en tres pasos:
  1. Añadir columna con nuevo nombre copiando datos de la vieja columna.
  2. Actualizar todas las instancias de la aplicación (*rolling update*).
  3. Eliminar columna vieja.



Actualizaciones sin parada

Renombrar columna:

- Problema: en *rolling update* conviven versiones viejas y nuevas de la aplicación.
  - La versión vieja usa la columna vieja.
  - La versión nueva usa la columna nueva.
  - Asegurar que ambas usan los mismos datos y ninguna operación de escritura se pierde.
- Soluciones:
  - Sincronización con disparadores de BD.
  - Sincronización a través de código.

Actualizaciones sin parada

Renombrar columna con disparadores:

- Proceso en tres pasos:
  1. Añadir columna con nuevo nombre copiando datos de la vieja columna. Añadir disparadores para sincronizar ambas columnas.
  2. Actualizar todas las instancias de la aplicación (*rolling update*).
  3. Eliminar columna vieja y los disparadores.

- Si se utiliza Flyway, los dos primeros pasos se pueden hacer a la vez.

Actualizaciones sin parada

Renombrar columna sincronizando a través código:

- Proceso en cuatro pasos:
  1. Añadir columna con nuevo nombre copiando datos de la vieja columna.
  2. Crear una versión de la aplicación (v1) que lee y escribe usando ambas columnas (vieja y nueva). Además debe sincronizar los cambios hechos por instancias de la versión vieja.
  3. Actualizar todas las instancias de la aplicación (*rolling update*) a v1.
  4. Las instancias v1 usan ya la columna nueva, así que se actualizan todas las instancias (*rolling update*) a v2, versión que solamente usa la nueva columna.
  5. Eliminar columna vieja.

- Si se utiliza Flyway, los dos primeros pasos se pueden hacer a la vez.

Actualizaciones sin parada

Renombrar tabla o vista:

- Proceso similar a renombrar columna.

Cambio de tipo de datos de columna:

- Similar a renombrar columna añadiendo la conversión de datos al nuevo tipo.

Eliminación de columna, tabla o vista usada por la versión vieja:

- No se puede con *rolling update*. Requiere actualización de todas las instancias primero.