Contenido Teorico.

RESUMEN CAPÍTULO IV

Libro: Fundamentos de base de datos 5ta edición.

Autores : ABRAHAM SILBERSCHATZ - HENRY F. KORTH - SUDARSHAN

Condiciones de la Integridad

Las condiciones que garantizan la integridad de los datos pueden ser de dos tipos:

1. Las restricciones de integridad de usuario: son condiciones específicas de una base de datos concreta; son las que se deben cumplir en una base de datos articular con unos usuarios concretos, pero que no son necesariamente relevantes en otra Base de Datos.
2. Las reglas de integridad de modelo: son condiciones propias de un modelo de datos, y se deben cumplir en toda base de datos que siga dicho modelo.

Los SGBD deben proporcionar la forma de definir las restricciones de integridad de usuario de una base de datos y una vez definida, debe velar por su cumplimiento. Las reglas de integridad del modelo, en cambio, no se deben definir para cada base de datos concreta, porque se consideran preestablecidas para todas las base de datos de un modelo. Un SGBD de un modelo determinado debe velar por el cumplimiento de las reglas de integridad preestablecidas por su modelo.

Reglas de Integridad

Regla de integridad de unicidad de la clave primaria

La regla de integridad de unicidad está relacionada con la definición de clave primaria que establece que toda clave primaria que se elija para una relación no debe tener valores repetidos por lo que el conjunto de atributos CP es la clave primaria de una relación R, entonces la extensión de R no puede tener en ningún momento dos tuplas con la misma combinación de valores para los atributos de CP.

Regla de integridad de entidad de la clave primaria

La regla de integridad de entidad de la clave primaria dispone que los atributos de la clave primaria de una relación no pueden tener valores nulos. Esta regla es necesaria para que los valores de las claves primarias puedan identificar las tuplas individuales de las relaciones. Si las claves primarias tuviesen

valores nulos, es posible que algunas tuplas no se pudieran distinguir. Un SGBD relacional tendrá que garantizar el cumplimiento de esta regla de integridad en todas las inserciones y en todas las modificaciones que afecten a atributos que pertenecen a la clave primaria de la relación.

Regla de integridad referencial

La regla de integridad referencial está relacionada con el concepto de clave foránea, lo que determina que todos los valores que toma una clave foránea deben ser valores nulos o valores que existen en la clave primaria que referencia. La necesidad de esta regla es debido a que las claves foráneas tienen por objetivo establecer una conexión con la clave primaria que referencian. Si un valor de una clave foránea no estuviese presente.

Restricción

La restricción en caso de borrado, consiste en no permitir borrar una tupla si tiene una clave primaria referenciada por alguna clave foránea y la restricción en caso de modificación consiste en no permitir modificar ningún atributo de la clave primaria de una tupla si tiene una clave primaria referenciada por alguna clave foránea.

Actualización en cascada

La actualización en cascada consiste en permitir la operación de actualización de la tupla, y en efectuar operaciones compensatorias que propaguen en cascada la actualización a las tuplas que la referenciaban; se actúa de este modo para mantener la integridad referencial. La actualización en cascada en caso de borrado consiste en permitir el borrado de una tupla t que tiene una clave primaria referenciada, y borrar también todas las tuplas que referencian t y la actualización en cascada en caso de modificación consiste en permitir la modificación de atributos de la clave primaria de una tupla t que tiene una clave primaria referenciada, y modificar del mismo modo todas las tuplas que referencian t.

Anulación

La anulación consiste en permitir la operación de actualización de la tupla y en efectuar operaciones compensatorias que pongan valores nulos a los atributos de la clave foránea de las tuplas que la referencian; esta acción se lleva a cabo para mantener la integridad referencial. Los SGBD relacionales permiten establecer que un determinado atributo de una relación no admite valores nulos, sólo se puede aplicar la política de anulación si los atributos de la clave foránea sí los admiten. Más concretamente, la anulación en caso de borrado consiste en permitir el borrado de una tupla t que tiene una clave referenciada y, además, modificar todas las tuplas que referencian t, de modo que los atributos de la clave foránea correspondiente tomen valores nulos y la anulación en caso de modificación consiste en permitir la modificación de atributos de la clave primaria de una tupla t que tiene una clave referenciada y, además, modificar todas las tuplas que referencian t, de modo que los atributos de la clave foránea correspondiente tomen valores nulos.

Regla de integridad de dominio

La regla de integridad de dominio está relacionada con la noción de dominio. Esta regla establece dos condiciones.

- La primera condición consiste en que un valor no nulo de un atributo Ai debe pertenecer al dominio del atributo Ai; es decir, debe pertenecer a dominio(Ai). Esta condición implica que todos los valores no nulos que contiene la base de datos para un determinado atributo deben ser del dominio declarado para dicho atributo.
- La segunda condición sirve para establecer que los operadores que pueden aplicarse sobre los valores dependen de los dominios de estos valores; es decir, un operador determinado sólo se puede aplicar sobre valores que tengan dominios que le sean adecuados.

Tipos de datos

Los campos de las tablas MySQL nos dan la posibilidad de elegir entre tres grandes tipos de contenidos:

- Datos numéricos,
- Datos para guardar cadenas de caracteres (alfanuméricos) y
- Datos para almacenar fechas y horas.

NUMÉRICOS ENTEROS

Comencemos por conocer las opciones que tenemos para almacenar datos que sean numéricos enteros (edades, cantidades, magnitudes sin decimales); poseemos una variedad de opciones:

Tipos de datos	Bytes	Valor mínimo	Valor máximo
TINYINT	1	-128	127
SMALLINT	2	-32768	32767
MEDIUMINT	3	-8388608	8388607
INT o INTEGER	4	-2147483648	2147483647
BIGINT	8	-9223372036854775808	9223372036854775807

NÚMEROS CON DECIMALES

Dejemos los enteros y pasemos ahora a analizar los valores numéricos con decimales.

Estos tipos de datos son necesarios para almacenar precios, salarios, importes de cuentas bancarias, etc. que no son enteros.

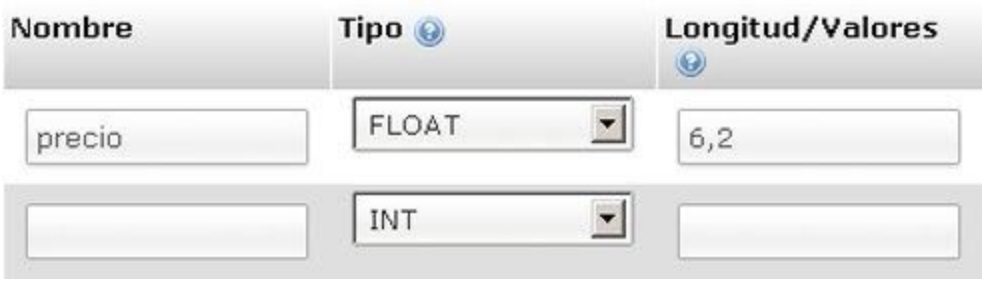
Tenemos que tener en cuenta que si bien estos tipos de datos se llaman "de coma flotante", por ser la coma el separador entre la parte entera y la parte decimal, en realidad MySQL los almacena usando un punto como separador.

En esta categoría, disponemos de tres tipos de datos: FLOAT, DOUBLE y DECIMAL.

La estructura con la que podemos declarar un campo FLOAT implica definir dos valores: la longitud total (incluyendo los decimales y la coma), y cuántos de estos dígitos son la parte decimal. Por ejemplo:

FLOAT (6.2)

Esta definición permitirá almacenar como minimo el valor -999.99 y como máximo 999.99 (el signo menos no cuenta, pero el punto decimal sí, por eso son seis digitos en total, y de ellos dos son los decimales).



La cantidad de decimales (el segundo número entre los paréntesis) debe estar entre 0 y 24, ya que ése es el rango de precisión simple.
En cambio, en el tipo de dato DOUBLE, al ser de doble precisión, sólo permite que la cantidad de decimales se defina entre 25 y 53.

Datos alfanuméricos

- CHAR
- VARCHAR
- BINARY
- VARBINARY
- TINYBLOB
- TINYTEXT
- BLOB
- TEXT
- MEDIUMBLOB
- MEDIUMTEXT
- LONGBLOB
- LONGTEX
- ENUM
- SET

Datos de fecha y hora

DATETIME
Un campo definido como DATETIME nos permitirá almacenar información acerca de un instante de tiempo, pero no sólo la fecha sino también su horario, en el formato:

AAAA-MM-DD HH:MM:SS
Siendo la parte de la fecha de un rango similar al del tipo DATE (desde el 1000-01-01 00:00:00 al 9999-12-31 23:59:59), y la parte del horario, de 00:00:00a 23:59:59.

TIME
Este tipo de cambio permite almacenar horas, minutos y segundos, en el formato HH:MM:SS, y su rango permitido va desde -839:59:59 hasta 839:59:59(unos 35 días hacia atrás y hacia adelante de la fecha actual). Esto lo hace ideal para calcular tiempos transcurridos entre dos momentos cercanos.

TIMESTAMP
Un campo que tenga definido el tipo de dato TIMESTAMP sirve para almacenar una fecha y un horario, de manera similar a DATETIME, pero su formato y rango de valores serán diferentes.
El formato de un campo TIMESTAMP puede variar entre tres opciones:

- AAAA-MM-DD HH:MM:SS
- AAAA-MM-DD
- AA-MM-DD

Es decir, la longitud posible puede ser de 14, 8 o 6 dígitos, según qué información proporcionemos.

Atributos de los campos

- NULL O NOT NULL
- VALOR PREDETERMINADO (DEFAULT)

Prepared Statements

Se pueden crear instrucciones preparadas en las que algunos valores estén sustituidos por “?”, lo que indica que los valores reales se proporcionarán más tarde. Algunos sistemas de bases de datos compilan las consultas cuando se preparan; cada vez que se ejecuta la consulta (con valores nuevos), la base de datos puede volver a emplear la forma previamente compilada de la consulta. El fragmento de código de la Figura 4.6 muestra la manera de utilizar las instrucciones preparadas.

```
PreparedStatement pStmt = con.prepareStatement(
                    "insert into cuenta values(?,?,?)");
pStmt.setString(1, "C-9732");
pStmt.setString(2, "Navacerrada");
pStmt.setInt(3, 1200);
pStmt.executeUpdate();
pStmt.setString(1, "C-9733");
pStmt.executeUpdate();
```

Figura 4.6 Instrucciones preparadas en código JDBC.

La función setString (y otras funciones parecidas para otros tipos de datos básicos de SQL) permite especificar el valor de los parámetros. Las instrucciones preparadas son el método preferido de ejecucion de las consultas de SQL, cuando utilizan valores que introducen los usuarios. Supóngase que el valor de las variables número_cuenta, nombre_sucursal, and saldo ha sido introducido por un usuario y hay que insertar la fila correspondiente en la relación cuenta. Supóngase que, en lugar de utilizar una instrucción preparada, se crea una consulta mediante la concatenación de las cadenas de caracteres de la manera siguiente:

```
"insert into cuenta values(' "+ número_cuenta + -', ' "+ nombre_sucursal + -', "
                    + saldo + ")"
```

y la consulta se ejecuta directamente. Ahora, si el usuario escribiera una sola comilla en el campo del número de cuenta o en el del nombre de la sucursal, la cadena de caracteres de la consulta tendría un error.

Instrucciones preparadas en código JDBC. de sintaxis. Es bastante probable que alguna sucursal tenga un apóstrofo en su nombre (especialmente si es un nombre irlandés como O’Donnell). Peor todavía, intrusos informáticos maliciosos pueden “inyectar” consultas de SQL propias escribiendo los caracteres adecuados en la cadena de caracteres. Una inyección de SQL de ese tipo puede dar lugar a graves fallos de seguridad.

La adición de caracteres de escape para tratar con los apóstrofos de las cadenas de caracteres es una manera de resolver este problema. El uso de las instrucciones preparadas es una manera más sencilla de hacerlo, ya que el método setString añade caracteres de escape de manera implícita. Además,

cuando hay que ejecutar varias veces la misma instrucción con valores diferentes, las instrucciones preparadas suelen ejecutarse mucho más rápido que varias instrucciones de SQL por separado. JDBC también proporciona una interfaz CallableStatement que permite la llamada a los procedimientos almacenados y a las funciones de SQL (que se describen más adelante, en el Apartado 4.6). Esta interfaz desempeña el mismo rol para las funciones y los procedimientos que prepareStatement para las consultas.

```
CallableStatement cStmt1 = con.prepareCall("{? = call alguna_función(?)}");
CallableStatement cStmt2 = con.prepareCall("{call algún_procedimiento(?,?)}");
```

Los tipos de datos de los valores devueltos por las funciones y de los parámetros externos de los procedimientos deben registrarse empleando el método registerOutParameter(), y pueden recuperarse utilizando métodos get parecidos a los de los conjuntos de resultados.

Ejemplo de codigo empleando PreparedStatement

```
import com.mysql.jdbc.jdbc2.optional.MysqlDataSource;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class Main {

  public static void main(String[] args) throws SQLException {
    MysqlDataSource ds = new MysqlDataSource();
    ds.setDatabaseName("mysql");
    ds.setUser("root");

    try (Connection conn = ds.getConnection()) {
      try (Statement stmt = conn.createStatement()) {
        stmt.executeUpdate("CREATE TABLE IF NOT EXISTS products (name VARCHAR(40),
price INT)");
      }

      try (PreparedStatement stmt = conn.prepareStatement("INSERT INTO products VALUES
(?, ?)")) {
        stmt.setString(1, "bike");
        stmt.setInt(2, 10900);
        stmt.executeUpdate();
        }

      try (PreparedStatement stmt = conn.prepareStatement("SELECT * FROM products
WHERE name = ?")) {
        stmt.setString(1, "shoes");
```

```
        ResultSet rs = stmt.executeQuery();
        rs.next();
        System.out.println(rs.getInt(2));
      }
    }
  }
}
```

ResultSet

Un ResultSet, o conjunto de resultados, contiene los resultados de una consulta SQL, por lo que es un conjunto de filas obtenidas desde una base de datos, así como Metadatos sobre la consulta y los tamaños de cada columna.

```
import java.sql.*;

/**
ResultSetExample.java
**/
PreparedStatement stmt = null;
ResultSet rs = null;

stmt = con.prepareStatement("SELECT * FROM personas");
rs = stmt.executeQuery();

while(rs.next()) {
   for (int x=1;x<=rs.getMetaData().getColumnCount();x++)
     System.out.print(rs.getString(x)+ "\t");

   System.out.println("");
}
```

ORM (Object Relational Mapping)

Documento interesante de ORMs:
http://revistatelematica.cujae.edu.cu/index.php/tele/article/view/23/21

Introducción:
Hoy hablamos del Mapeo Objeto-Relacional como se conocen comúnmente, ORM (del inglés Object Relational Mapping). Permite convertir los datos de tus objectos en un formato correcto para poder guardar la información en una base de datos (mapeo) creándose una base de datos virtual donde los datos que se encuentran en nuestra aplicación, quedan vinculados a la base de datos (persistencia).

Supongamos que tenemos una tabla de clientes. En nuestra aplicación queremos hacer las funciones básicas sobre base de datos CRUD (del inglés Create, Read, Update and Delete) Crear, Obtener, Actualizar y Borrar. Cada operación corresponde con una sentencia SQL.

- Creear: INSERT
- Obtener: SELECT
- Actualizar: UPDATE
- Borrar: DELETE

Si queremos insertar un cliente nuevo y no utilizamos un ORM, el código quedaría de la siguiente manera si utilizamos C#:

```
String query = "INSERT INTO clientes (id,nombre,email,pais) VALUES (@id, @nombre, @email, @pais)";

command.Parameters.AddWithValue("@id","1")
command.Parameters.AddWithValue("@nombre","nombre")
command.Parameters.AddWithValue("@email","email")
command.Parameters.AddWithValue("@pais","pais")
command.ExecuteNonQuery();
```

En cambio si utilizamos ORM seria de la siguiente forma:

```
var cliente = new Cliente();
cliente.Id = "1";
cliente.Nombre = "nombre";
cliente.Email = "email";
cliente.Pais = "pais";
session.Save(customer);amos un ORM, el código se puede reducir de la siguiente manera:
```

Como podéis comprobar se ha reducido considerablemente el código.
Pero lo más importante de todo, es que ahora modificamos la tabla de nuestra base de datos y añadimos un campo más como por ejemplo el apellido de nuestro cliente. En los dos casos, tendríamos que añadir

a nuestra clase Cliente la propiedad correspondiente al apellido pero, si no utilizas un ORM te tocará revisar todas las sentencias INSERT, SELECT y UPDATE para introducir dicho campo en cada una de ellas.
En cambio si utilizas un ORM , lo único que tendrás que hacer será añadir la propiedad a la clase correspondiente. La sentencia save seguirá siendo la misma, el ORM se encargará de modificar los INSERT, SELECT y UPDATE por vos.
Además de lo que hemos visto hasta el momento, gracias a los ORM nos abstraemos del motor de base de datos que estamos utilizando es decir, si en algún momento queremos cambiar de base de datos, nos resultará sumamente sencillo. En algunos casos con solo cambiar un par de líneas de código, estaremos cambiando de motor de base de datos.

Existen varios ORM en el mercado. Todo dependerá del lenguaje de programación que estemos utilizando y de nuestras necesidades. A continuación podemos ver los ORM para algunos lenguajes de programación:

- Hibernate (Java)
- MyBatis (Java)
- Ebean (Java)
- Entity Famework (.NET)
- NHibernate (.NET)
- MyBatis.NET (.NET)
- Doctrine (PHP)
- Propel (PHP)
- Rocks (PHP)
- Torpor (PHP)
- Sequelize (javascript)