5. Claves. Tipos de claves.

Cuando se habla de claves en una base de datos relacional, básicamente nos referimos a un valor o conjunto de valores que permiten identificar de manera única un registro dentro de una tabla. Esto es fundamental para garantizar que los datos puedan ser localizados y relacionados de manera correcta. Vamos a ver los diferentes tipos de claves que se usan en las bases de datos.

Clave primaria (Primary Key)

→ La clave primaria es la clave más importante de una tabla, ya que sirve para identificar de manera única cada fila o registro. No puede haber dos filas con el mismo valor en la clave primaria, y tampoco puede ser nula (esto quiere decir que todos los registros tienen que tener un valor en ese campo).

→ Por ejemplo, si tienes una tabla de usuarios, podrías tener un campo id_usuario que sería la clave primaria, y cada usuario tendría un valor único en ese campo: el usuario 1, el usuario 2, el usuario 3, etc.

```sql
-- SQL

CREATE TABLE usuarios (
    id_usuario INT PRIMARY KEY,
    nombre VARCHAR(100),
    email VARCHAR(100)
);
```

→ Aquí, id_usuario es la clave primaria. Esta clave es la que permite que, aunque haya dos usuarios con el mismo nombre, se puedan diferenciar gracias a su id_usuario.

Clave foránea (Foreign Key)

→ La clave foránea es la que se usa para crear relaciones entre dos tablas. Básicamente, es un campo en una tabla que hace referencia a la clave primaria de otra tabla. De esta manera, se puede conectar la información de dos tablas diferentes.

→ Imaginemos que tenemos dos tablas: una de usuarios y otra de préstamos de libros. La tabla de préstamos tendría un campo id_usuario, que sería la clave foránea y que se conectaría con el campo id_usuario de la tabla de usuarios.

```sql
-- SQL

CREATE TABLE prestamos (
    id_prestamo INT PRIMARY KEY,
    id_usuario INT,
    id_libro INT,
    fecha_prestamo DATE,
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario)
);
```

→ Aquí, id_usuario es una clave foránea que apunta a la tabla de usuarios. Esto permite saber qué usuario ha tomado qué préstamo, y esa relación se mantiene a través de la clave foránea.

Clave candidata (Candidate Key)

→ Una clave candidata es cualquier conjunto de campos que podría ser una clave primaria, pero no ha sido elegida como tal. Es como decir: "Este conjunto de columnas podría ser único, pero solo uno de ellos se elige como clave primaria".

→ Por ejemplo, si en una tabla de usuarios tuviéramos tanto el id_usuario como el email del usuario, podríamos decir que el email también es único (asumiendo que no permitimos que dos usuarios tengan el mismo email). En ese caso, tanto el id_usuario como el email serían claves candidatas, pero normalmente solo se elige una (generalmente el id_usuario).

Clave compuesta (Composite Key)

→ A veces, un solo campo no es suficiente para identificar de manera única un registro, así que se usa una clave compuesta. Esto significa que la clave primaria está formada por dos o más campos.

→ Un ejemplo sencillo es si tenemos una tabla de "detalles de pedidos" donde necesitamos una combinación de id_pedido y id_producto para identificar cada fila de manera única. Ni el id_pedido ni el id_producto por sí solos podrían ser una clave primaria, pero juntos sí.

```sql
-- SQL

CREATE TABLE detalles_pedido (
    id_pedido INT,
    id_producto INT,
    cantidad INT,
    PRIMARY KEY (id_pedido, id_producto)
);
```

→ Aquí, la clave compuesta está formada por id_pedido y id_producto.

Actividad 4

Imagina que estás diseñando una base de datos para gestionar el alquiler de vehículos en una empresa de transporte. La empresa necesita asegurarse de que todos los datos están organizados correctamente y que las relaciones entre las tablas son claras y seguras. A continuación, debes crear las tablas necesarias y definir los distintos tipos de claves (clave primaria, clave foránea, clave candidata, clave compuesta) que garantizarán la integridad de la base de datos.

Tabla Vehículos:

Esta tabla debe contener información sobre los vehículos disponibles para alquilar.

Define una clave primaria para identificar de manera única cada vehículo.

Propón también una clave candidata que podría ser única, pero que no se utiliza como clave primaria.

Tabla Clientes:

Esta tabla incluye los datos de los clientes que alquilan los vehículos.

Define una clave primaria para identificar de manera única a cada cliente.

Proporciona también un campo que pueda ser utilizado como clave candidata.

Tabla Alquileres:

Esta tabla gestiona la información sobre cada alquiler realizado.

Define una clave compuesta que permita identificar de manera única cada registro.

Incluye una clave foránea que establezca una relación entre esta tabla y la tabla de vehículos.

6. Normalización. Formas normales.

La normalización es el proceso de organizar los datos en una base de datos para reducir la redundancia y evitar problemas como las anomalías de actualización (cuando se cambian datos en un lugar, pero no en otro) o las anomalías de eliminación (cuando se borra información que no debería haberse perdido). Se hace mediante la aplicación de un conjunto de reglas, que son conocidas como formas normales.

Existen varias formas normales (normal forms), pero las más importantes son las primeras tres.

Primera forma normal (1FN)

- → Una tabla está en primera forma normal cuando cada columna contiene solo valores atómicos, es decir, que no hay valores repetidos dentro de una misma celda. En resumen, no debería haber listas o conjuntos de valores dentro de una celda.
- → Por ejemplo, si en una tabla de pedidos guardamos varios productos en la misma celda, estaríamos rompiendo la primera forma normal:

Incorrecto:

En este ejemplo, los productos están agrupados en una sola celda, lo que rompe la primera forma normal. Los valores deben ser atómicos (no listas o conjuntos de valores).

ID_PEDIDO	PRODUCTOS	TOTAL
1	Producto1, Producto2, Prod3	200

Correcto:

Para cumplir con la primera forma normal, cada producto debe estar en su propia fila, eliminando los valores no atómicos.

ID_PEDIDO	PRODUCTO	TOTAL
1	Producto1	200
1	Producto2	200
1	Producto3	200

Segunda forma normal (2FN)

- → Una tabla está en segunda forma normal cuando cumple con la primera forma normal y, además, todos los atributos no clave dependen totalmente de la clave primaria. Esto quiere decir que no debe haber atributos que dependan solo de una parte de una clave compuesta (si la hay).
- → Por ejemplo, si tenemos una clave compuesta en una tabla de detalles de pedidos con id_pedido e id_producto, y tenemos también un campo como nombre_producto, eso estaría mal, porque el nombre_producto depende solo de id_producto y no del conjunto completo de la clave primaria (que es id_pedido e id_producto juntos).
- → Para solucionar esto, se tendría que separar el nombre_producto en una tabla aparte.

Tercera forma normal (3FN)

- → Una tabla está en tercera forma normal cuando cumple con la segunda forma normal y, además, no existen dependencias transitivas. Esto significa que los atributos no clave solo deben depender de la clave primaria, no de otros atributos no clave.
- → Por ejemplo, si en una tabla de usuarios tenemos los campos id_usuario, ciudad, y código_postal, pero código_postal depende de ciudad (y no directamente de id_usuario), entonces hay una dependencia transitiva. Lo correcto sería dividir esa tabla en dos: una para usuarios y otra para ciudades y códigos postales.

7. Construcción del modelo lógico de datos.

Cuando hablamos de modelo lógico de datos en el contexto de una aplicación web, nos referimos a la representación estructurada de cómo se van a organizar los datos en el servidor. En otras palabras, es como un mapa que te dice cómo serán las tablas, qué datos (columnas) contendrán, y cómo estarán relacionadas entre sí. El modelo lógico es lo que permite que la base de datos funcione de manera eficiente y que los datos se puedan acceder y modificar fácilmente.

Este modelo lógico se construye tomando el modelo conceptual (que es más abstracto y se enfoca en el diseño global de las entidades y relaciones) y transformándolo en algo más técnico y concreto. Vamos a verlo en pasos, empezando por la creación de tablas, la definición de columnas y las claves.

7.1. Especificación de tablas.

El primer paso para construir el modelo lógico es especificar las tablas. Cada entidad importante del sistema se convierte en una tabla. Si estamos hablando de una aplicación de gestión de una biblioteca, por ejemplo, las entidades principales serían cosas como usuarios, libros y préstamos. Estas entidades se transforman en tablas dentro de la base de datos.

Por ejemplo:

- → usuarios: sería una tabla que contiene los datos de las personas que usan la biblioteca.
- → libros: sería una tabla que almacena los detalles de cada libro disponible.
- → préstamos: sería una tabla que registra los libros que los usuarios han tomado prestados.

Cada tabla se relaciona con una entidad del mundo real y contiene la información más importante sobre ella. Aquí lo importante es que las tablas se construyen de manera que cada una se encargue de manejar una cosa específica, evitando redundancias y manteniendo la integridad de los datos.

7.2. Definición de columnas.

Una vez que se han definido las tablas, el siguiente paso es definir las columnas. Las columnas son los atributos o propiedades que describen cada entidad. Para que quede claro, cada columna en una tabla representa un tipo de dato que va a ser almacenado.

Por ejemplo, en la tabla de usuarios, las columnas podrían ser:

- → id_usuario (un identificador único para cada usuario)
- → nombre (el nombre del usuario)
- → email (el correo electrónico del usuario)
- → tipo_usuario (puede ser "estudiante", "profesor", etc.)

Para la tabla de libros, las columnas podrían ser:

- → id_libro (un identificador único para cada libro)
- → título (el título del libro)
- → autor (el autor del libro)
- → año_publicación (el año en que se publicó el libro)

Cada columna debe definirse con un tipo de dato específico (por ejemplo, enteros para los identificadores, cadenas de texto para nombres, fechas para los campos de tiempo). Además, es importante definir si una columna puede estar vacía (nula) o si debe tener siempre un valor (no nula).

7.3. Especificación de claves.

Las claves aseguran que los datos sean únicos y permiten relacionar una tabla con otra. Existen varios tipos de claves que se deben tener en cuenta:

Clave primaria (Primary Key)

→ La clave primaria es una columna (o combinación de columnas) que identifica de manera única cada fila en una tabla. En la tabla de usuarios, por ejemplo, el id_usuario sería la clave primaria, porque es único para cada usuario.

```sql
-- SQL

CREATE TABLE usuarios (
    id_usuario INT PRIMARY KEY,
    nombre VARCHAR(100),
    email VARCHAR(100),
    tipo_usuario VARCHAR(50)
);
```

Clave foránea (Foreign Key)

→ La clave foránea es una columna en una tabla que sirve para hacer referencia a la clave primaria de otra tabla. Es lo que permite relacionar tablas entre sí. Por ejemplo, en la tabla de préstamos, habría una clave foránea que se conectaría con la tabla de usuarios, para saber qué usuario ha tomado un determinado libro.

```sql
-- SQL

CREATE TABLE prestamos (
    id_prestamo INT PRIMARY KEY,
    id_usuario INT,
    id_libro INT,
    fecha_prestamo DATE,
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario)
);
```

7.4. Conversión a formas normales. Dependencias.

La normalización es el proceso mediante el cual organizamos las columnas y las tablas para eliminar redundancias y dependencias innecesarias. La idea es que cada tabla almacene únicamente datos relacionados, y que no haya información duplicada. Para lograr esto, se aplican una serie de reglas conocidas como formas normales.

Primera forma normal (1FN)

➔ Para que una tabla esté en primera forma normal, cada columna debe contener valores atómicos, es decir, que no puede haber listas o conjuntos de valores en una sola celda. Por

ejemplo, no podrías tener una columna de "productos" con varios productos en una sola celda. En su lugar, cada producto tendría que estar en su propia fila.

Segunda forma normal (2FN)

- ➔ Una tabla está en segunda forma normal si cumple con la primera forma normal y además todos los atributos no clave dependen totalmente de la clave primaria. Esto significa que no puede haber una columna en la tabla que dependa solo de una parte de la clave primaria (si hay una clave compuesta). Por ejemplo, si en la tabla de detalles de pedido tenemos una clave compuesta de id_pedido y id_producto, todas las demás columnas (como "cantidad") deberían depender de esa clave compuesta, no solo de id_pedido o id_producto.

Tercera forma normal (3FN)

- ➔ Para que una tabla esté en tercera forma normal, debe cumplir con la segunda forma normal y no debe haber dependencias transitivas. Esto quiere decir que ningún atributo no clave puede depender de otro atributo no clave. Por ejemplo, si tienes una tabla de usuarios con las columnas id_usuario, ciudad, y código_postal, sería un error que el código postal dependa de la ciudad, ya que ambos dependen del usuario. En este caso, lo ideal sería separar esas dependencias en una tabla aparte.

Las dependencias en una base de datos hacen referencia a la forma en que las columnas dependen unas de otras. Las dependencias funcionales son las más comunes. Por ejemplo, en la tabla de usuarios, el nombre y email dependen del id_usuario, porque este último identifica de manera única a cada usuario.

Importante

- Es importante manejar bien las dependencias para evitar inconsistencias en los datos. Si un dato depende de otro (como el código postal de una ciudad), es mejor separarlos en tablas distintas para mantener la integridad de los datos.

Actividad 5

Relaciona cada concepto con la descripción o ejemplo que le corresponde:

1. Clave primaria

2. Clave foránea

3. Clave compuesta

4. Primera forma normal (1FN)

5. Segunda forma normal (2FN)

6. Tercera forma normal (3FN)

a. Combinación de dos o más columnas que juntas identifican de manera única un registro en la tabla.

b. Columna en una tabla que referencia a la clave primaria de otra tabla para crear una relación entre ambas.

c. Cada columna contiene valores atómicos, es decir, no puede haber listas o múltiples valores en una celda.

d. Identificador único para cada registro de una tabla.

e. Asegura que todas las columnas no clave dependan completamente de la clave primaria.

f. Asegura que no existan dependencias transitivas entre columnas no clave.

8. El modelo físico de datos. Ficheros de datos.

Cuando hablamos del modelo físico de datos, nos referimos a la forma en la que los datos se almacenan realmente en el servidor. Hasta ahora, el modelo lógico nos ha permitido organizar los datos en tablas, definir columnas y establecer relaciones entre esas tablas. Pero todo eso es algo abstracto, ya que las tablas no existen como tal en el disco duro de un servidor. En realidad, lo que hay son ficheros que contienen esos datos.

El modelo físico, por tanto, se centra en cómo los datos se guardan en el sistema, cómo se accede a ellos, y cómo se organiza todo para que sea eficiente y rápido a la hora de hacer consultas o modificar los datos. Vamos a desglosar cada punto para entender bien cómo funciona.

8.1. Descripción de los ficheros de datos.

Los ficheros de datos son los archivos en los que realmente se guardan los datos de las tablas que hemos definido en el modelo lógico. En el contexto de una base de datos, estos ficheros almacenan información de manera estructurada para que se pueda recuperar y manipular de manera eficiente.

Por ejemplo, cuando creas una tabla en una base de datos como MySQL o PostgreSQL, lo que está sucediendo por debajo es que el sistema está creando un fichero (o varios) donde va a guardar cada registro de esa tabla. Estos ficheros están organizados de manera que se pueda acceder a ellos rápidamente, ya sea para leer, modificar o eliminar datos.

Los ficheros de datos pueden estar en diferentes formatos dependiendo del sistema de gestión de bases de datos (SGBD) que se esté utilizando, pero en todos los casos el objetivo es el mismo: almacenar y recuperar la información de forma eficiente.

8.2. Tipos de ficheros.

Hay diferentes tipos de ficheros de datos que se pueden usar para almacenar la información en una aplicación web. Los más comunes son los siguientes:

- ➔ Ficheros secuenciales: Este tipo de fichero almacena los datos de forma continua, uno tras otro, como si fueran líneas en un archivo de texto. Es el tipo más sencillo, pero también el más limitado, ya que, si necesitas buscar un dato en particular, tendrás que recorrer todo el fichero desde el principio hasta que lo encuentres. Por eso, los ficheros secuenciales suelen ser útiles solo cuando se necesita leer todos los datos de forma secuencial o cuando no es necesario hacer búsquedas rápidas.
- ➔ Ficheros indexados: En este tipo de fichero, se añade una estructura de índices que permite encontrar los datos de forma rápida. Es como si tuvieras un libro con un índice al principio que te dice en qué página está cada tema. De esta forma, cuando necesitas encontrar un dato, no tienes que recorrer todo el fichero, sino que usas el índice para ir directamente a la parte del fichero donde está el dato.
- ➔ Ficheros de acceso directo (aleatorio): Estos ficheros permiten acceder a cualquier registro directamente, sin necesidad de recorrer el fichero secuencialmente. Esto se logra dividiendo el fichero en bloques y usando un mecanismo para calcular dónde está cada registro. Los ficheros de acceso directo son muy eficientes cuando se necesitan hacer muchas consultas rápidas, pero pueden ser más complicados de gestionar que los secuenciales.

Pie de imagen: Esquema con ejemplos de los tipos de ficheros (Elaboración propia).

8.3. Modos de acceso.

Dependiendo del tipo de fichero que se esté utilizando, hay diferentes modos de acceso a los datos. Estos son algunos de los más comunes:

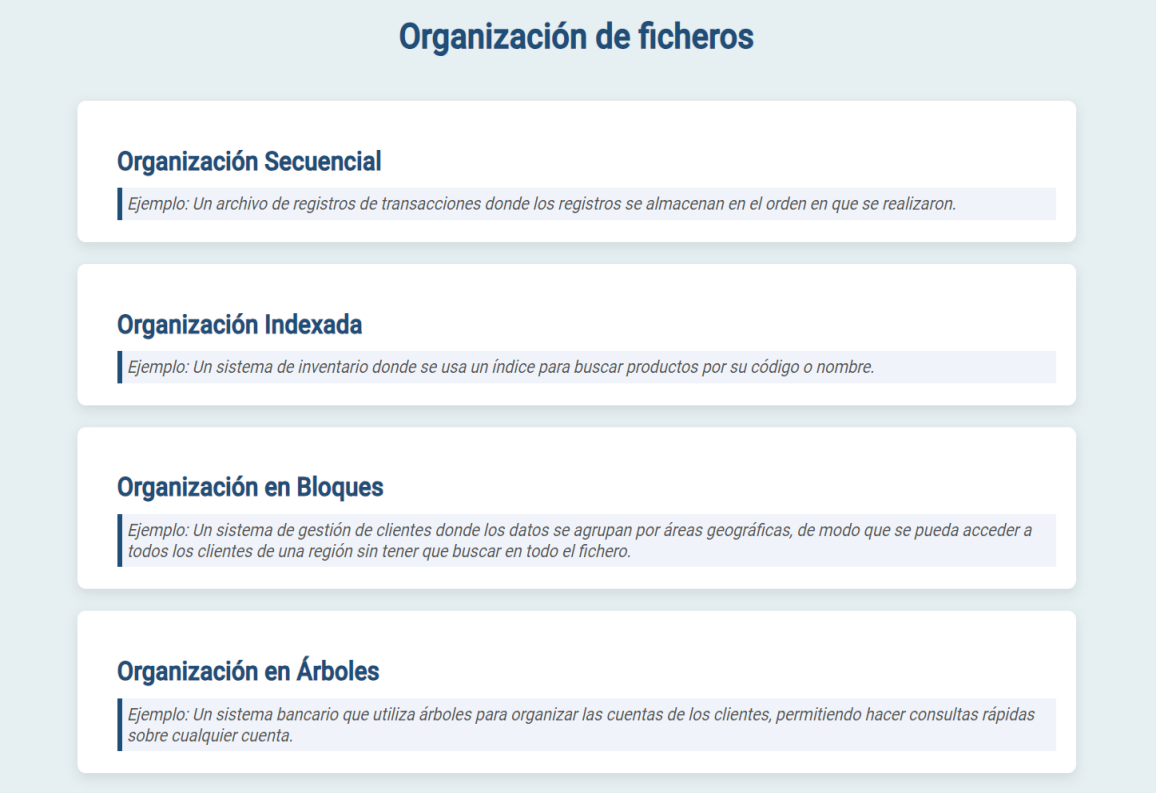
- ➔ Acceso secuencial: Este es el modo de acceso más simple. Los datos se leen en el orden en que están almacenados. Es muy eficiente si necesitas procesar todos los datos de principio a fin, pero si solo necesitas buscar un dato en particular, puede ser muy lento, ya que tienes que recorrer todos los registros uno por uno.
- ➔ Acceso directo (aleatorio): Este modo de acceso permite ir directamente al dato que necesitas sin tener que recorrer los registros anteriores. Se utiliza mucho en bases de datos que necesitan hacer consultas rápidas, ya que permite acceder a cualquier registro de forma eficiente.
- ➔ Acceso indexado: En este caso, se usa un índice que apunta a la ubicación de los datos en el fichero. Este índice permite que las búsquedas sean rápidas, ya que se puede ir directamente a la parte del fichero donde están los datos. Es como usar el índice de un libro para encontrar una sección específica sin tener que leer todo el libro.

Pie de imagen: Esquema con ejemplos de modos de acceso (Elaboración propia).

8.4. Organización de ficheros.

La organización de los ficheros es clave para asegurar que los datos se puedan almacenar y recuperar de forma eficiente. Hay diferentes formas de organizar los ficheros, y la elección de una u otra depende del tipo de datos que se estén manejando y del tipo de consultas que se van a hacer con mayor frecuencia.

- ➔ Organización secuencial: En este tipo de organización, los datos se almacenan uno tras otro, en el orden en que se insertan. Es útil si los datos se van a leer de manera secuencial (uno detrás del otro), pero si se necesita hacer muchas búsquedas específicas o modificaciones, no es muy eficiente.
- ➔ Organización indexada: Aquí, además de almacenar los datos, se crea un índice que permite acceder rápidamente a los registros. El índice actúa como una tabla de contenido, lo que hace que las búsquedas y las actualizaciones sean más rápidas. Este tipo de organización es muy útil cuando se necesita hacer muchas consultas basadas en claves o en campos específicos.
- ➔ Organización en bloques: En este tipo de organización, los datos se dividen en bloques, y cada bloque contiene varios registros. Los bloques se organizan de manera que se pueda acceder a ellos de forma directa o secuencial. Esto permite hacer búsquedas y modificaciones de manera eficiente, ya que no es necesario leer todo el fichero para encontrar un registro específico.
- ➔ Organización en árboles: Esta es una de las organizaciones más avanzadas. Se utiliza una estructura de árbol (normalmente un árbol B+ en bases de datos) que permite acceder a los datos de forma eficiente. Los árboles permiten hacer búsquedas, inserciones y eliminaciones de manera rápida, ya que la estructura está diseñada para minimizar el número de operaciones necesarias para encontrar un registro.



Pie de imagen: Esquema con ejemplos de tipos de organización (Elaboración propia).

Actividad 6

Imagina que eres responsable de diseñar el sistema de almacenamiento de una tienda en línea. ¿Por qué elegirías ficheros de acceso directo en lugar de ficheros secuenciales para gestionar los datos de los pedidos? Reflexiona sobre cómo la velocidad de respuesta afecta la experiencia del usuario en una tienda en línea. Busca información sobre cómo los grandes comercios electrónicos optimizan el acceso a los datos.

Piensa en un proyecto personal o académico donde tengas que almacenar grandes cantidades de información. ¿Qué tipo de organización de ficheros crees que sería más eficiente para tu caso y por qué? Investiga más sobre cómo distintos sistemas de bases de datos (MySQL, PostgreSQL, etc.) manejan la organización de ficheros y cómo eso podría aplicarse a tu proyecto.

¿Cómo crees que el uso de índices en ficheros de bases de datos impacta el tiempo de carga y la capacidad de respuesta de una aplicación web que usas regularmente (como redes sociales o plataformas de streaming)? Piensa en tus propias interacciones con aplicaciones web y busca cómo esas plataformas gestionan grandes volúmenes de datos de usuarios y contenidos, como ocurre con Netflix o Facebook.

9. Transformación de un modelo lógico en un modelo físico de datos.

La transformación de un modelo lógico en un modelo físico de datos es el proceso en el que se pasa de la planificación (cómo queremos que los datos se organicen y se relacionen) a la implementación real (cómo se almacenarán esos datos en el servidor). En otras palabras, aquí es donde hacemos que las ideas abstractas del modelo lógico se conviertan en algo tangible dentro de una base de datos, como las tablas y los registros.

El modelo lógico es una representación de las entidades y cómo se relacionan entre sí, mientras que el modelo físico es cuando tomamos esas entidades y las implementamos como tablas reales en un sistema de bases de datos, como PostgreSQL, MySQL, o SQL Server.

¿Cómo se hace esta transformación?

Vamos a explicarlo paso por paso para que quede claro. Por ejemplo, en el contexto de una aplicación de gestión de una tienda de música online cuyas entidades serían “artistas”, “álbumes” y “canciones”:

1. Definir las tablas en el SGBD:

Lo primero es convertir las entidades del modelo lógico en tablas concretas dentro de la base de datos. Si tenemos una entidad artista en el modelo lógico, esto se traduce en una tabla física llamada artistas dentro del sistema de bases de datos.

→ Un ejemplo en MySQL sería crear una tabla para almacenar los datos de los artistas:

```sql
-- SQL

CREATE TABLE artistas (
    id_artista INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100),
    género VARCHAR(50),
    país VARCHAR(50)
);
```

→ Con esto, ya tenemos la entidad artista convertida en una tabla real que se puede usar para almacenar información en la base de datos.

2. Asignar tipos de datos específicos:

En el modelo físico, es necesario especificar los tipos de datos exactos que se van a usar para cada columna. En el modelo lógico, sabemos que el nombre del artista es un texto, pero en el modelo físico tenemos que definir si será un texto de longitud fija (como CHAR) o variable (como VARCHAR), y decidir el tamaño máximo.

→ Por ejemplo, en la tabla de álbumes, podríamos definir las columnas de la siguiente manera:

```sql
-- SQL

CREATE TABLE albumes (
    id_album INT AUTO_INCREMENT PRIMARY KEY,
    título VARCHAR(100),
    año_lanzamiento YEAR,
    id_artista INT,
    FOREIGN KEY (id_artista) REFERENCES artistas(id_artista)
);
```

→ Aquí decidimos que id_album será un número entero, y el título tendrá un máximo de 100 caracteres. Además, el año de lanzamiento será del tipo YEAR, que es específico para almacenar años.

3. Implementar restricciones:

Las restricciones aseguran que los datos se mantengan coherentes y se relacionen de manera correcta. Se definen claves primarias (PRIMARY KEY) para que no haya dos registros iguales, y claves foráneas (FOREIGN KEY) para mantener las relaciones entre tablas.

→ En la tabla de canciones, por ejemplo, podríamos agregar una clave foránea para relacionarla con el álbum al que pertenece:

```sql
-- SQL

CREATE TABLE canciones (
    id_cancion INT AUTO_INCREMENT PRIMARY KEY,
    título VARCHAR(100),
    duración TIME,
    id_album INT,
    FOREIGN KEY (id_album) REFERENCES albumes(id_album)
);
```

→ De esta manera, sabemos que cada canción está vinculada a un álbum gracias a la clave foránea id_album, que hace referencia a la tabla de albumes.

4. Índices y optimización:

En esta etapa del modelo físico, también se pueden definir índices para mejorar la eficiencia de las consultas. Si, por ejemplo, necesitamos buscar a los artistas por su país de origen con frecuencia, sería útil crear un índice en la columna país de la tabla de artistas.´

```sql
-- SQL

CREATE INDEX idx_pais ON artistas(pais);
```

Este índice hará que las búsquedas de artistas por país sean mucho más rápidas, aunque ocupa espacio adicional en disco.

5. Elegir el sistema de almacenamiento:

Dependiendo del SGBD que estemos usando, tendremos que elegir el motor de almacenamiento adecuado. En MySQL, por ejemplo, tenemos InnoDB o MyISAM como opciones. Si necesitamos integridad referencial (claves foráneas), lo mejor es usar InnoDB, ya que soporta todas las relaciones entre tablas.

6. Particionamiento (si es necesario):

Si estamos manejando grandes volúmenes de datos, como cientos de miles de canciones, puede ser útil particionar la tabla de canciones en diferentes partes físicas para mejorar el rendimiento. Esto no es algo que se defina en el modelo lógico, pero es importante en el modelo físico si estamos trabajando con grandes bases de datos.

Actividad 7

Imagina que estás desarrollando el sistema de gestión de una biblioteca digital. Tu tarea es transformar el modelo lógico de datos, que incluye las entidades "usuarios", "libros" y "préstamos", en un modelo físico dentro de un sistema de gestión de bases de datos (SGBD). Para ello, deberás crear las tablas correspondientes en SQL, asignar tipos de datos adecuados para cada columna, implementar restricciones con claves primarias y foráneas, y definir índices que optimicen las búsquedas más frecuentes. Redacta el código SQL necesario para cada paso y explica brevemente tus decisiones de diseño.

10. Herramientas para la realización de modelos de datos.

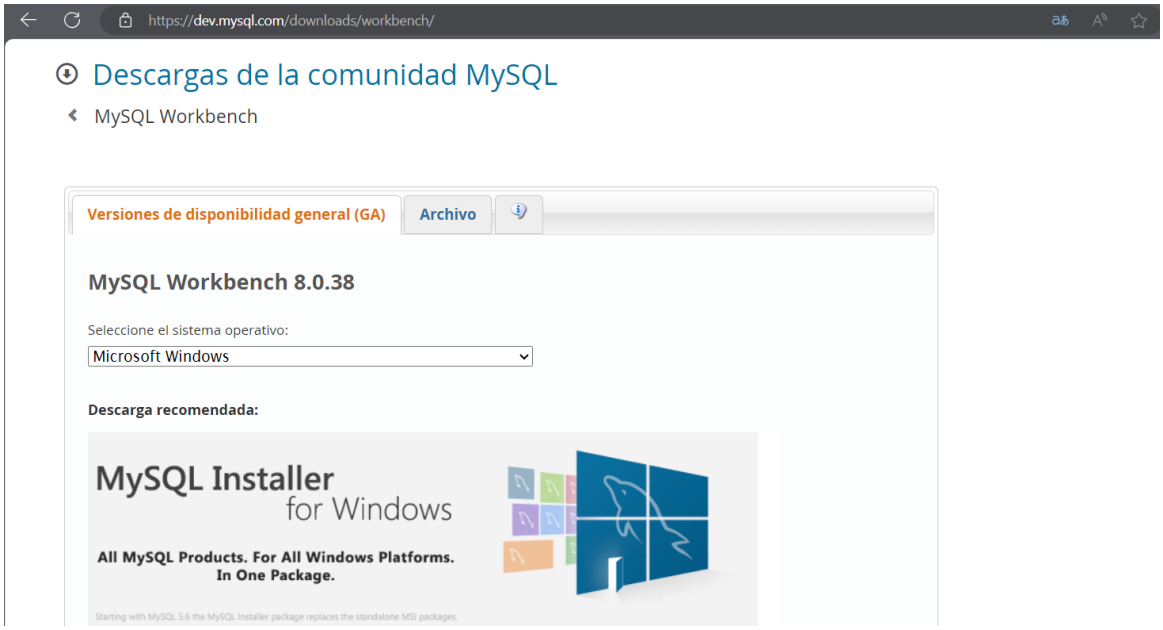
Hay varias herramientas que facilitan la creación y gestión de modelos de datos. Estas herramientas permiten diseñar las bases de datos de manera visual, generar el código SQL automáticamente y gestionar el rendimiento y el almacenamiento de los datos. Algunas de las más comunes son:

1. MySQL Workbench

Es una de las herramientas más populares y es gratuita, perfecta para diseñar bases de datos en MySQL. Algunas de sus funciones incluyen:

- Crear diagramas entidad-relación.
- Generar el código SQL automáticamente.
- Administrar la base de datos de manera visual.
- Ejecutar consultas SQL directamente.

Por ejemplo, puedes diseñar un diagrama con todas tus tablas de la tienda de música (artistas, álbumes, canciones) y luego exportarlo como un archivo SQL para implementarlo en el servidor.



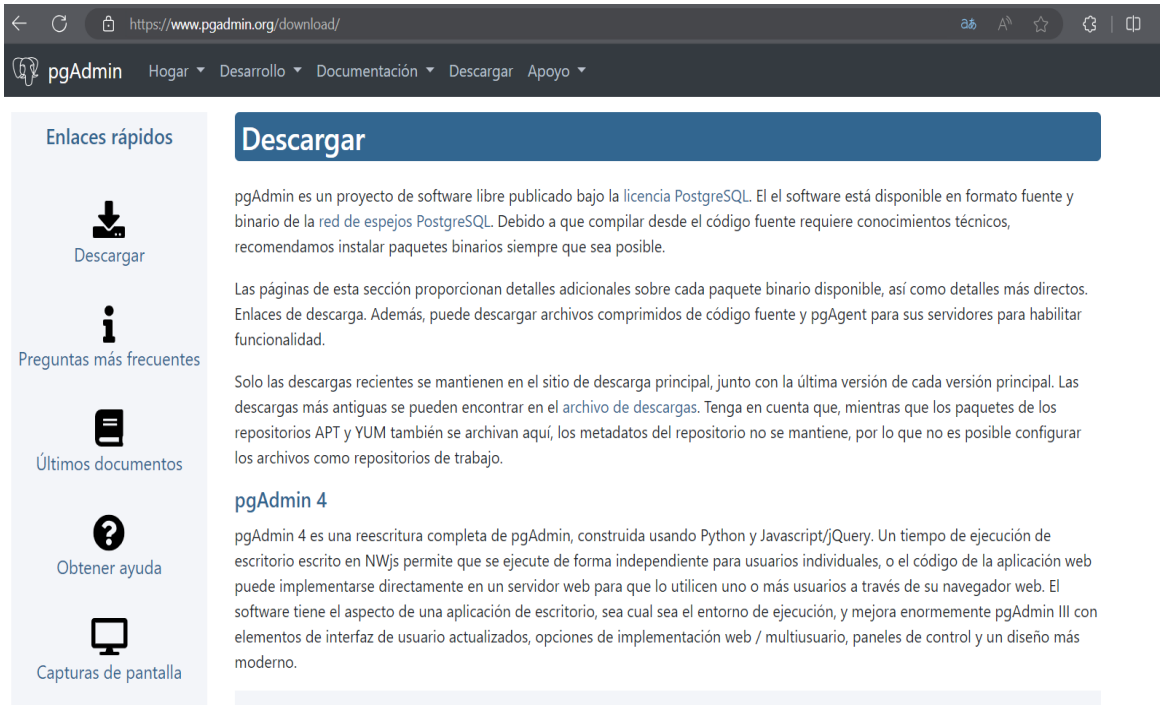
Pie de imagen: Captura de la página de descarga.

2. pgAdmin

Si trabajas con PostgreSQL, pgAdmin es una herramienta imprescindible. Tiene funcionalidades similares a MySQL Workbench, pero específicamente para PostgreSQL. Te permite:

- Crear y modificar esquemas de bases de datos.
- Ejecutar consultas SQL.
- Administrar servidores PostgreSQL.

Es una herramienta muy útil para proyectos que usan PostgreSQL como SGBD.



Pie de imagen: Captura de la página de descarga.

3. Microsoft SQL Server Management Studio (SSMS)

Si estás trabajando en un entorno empresarial y usas SQL Server, la herramienta oficial de Microsoft es SSMS. Es muy poderosa y permite:

- Diseñar bases de datos de forma visual.
- Optimizar el rendimiento de las consultas.
- Realizar copias de seguridad y restauraciones.

SSMS es ideal para proyectos grandes que requieren un sistema de bases de datos robusto y con muchas funcionalidades avanzadas.

Pie de imagen: Captura de la página de descarga.

4. Lucidchart

Lucidchart es una herramienta online que, aunque no está centrada solo en bases de datos, es muy útil para hacer diagramas de entidades-relación. Funciona muy bien para visualizar y colaborar en proyectos de bases de datos. Además, puedes integrarlo con varias bases de datos populares y tiene plantillas predefinidas para empezar rápidamente.

Pie de imagen: Captura de la página de descarga.

Actividad 8

Descarga alguno de los programas que se han mencionado e interactúa con sus herramientas.

11. Prueba de autoevaluación.

¿Qué es un modelo de datos en aplicaciones web?

a) Un algoritmo de seguridad

b) La estructura que organiza y manipula datos

c) Un diseño gráfico de la interfaz de usuario

¿Cuál es la primera fase del ciclo de vida de un dato?

a) Almacenamiento

b) Eliminación

c) Creación

¿Qué tipo de dato es 3.14?

a) Entero

b) Real

c) Cadena

¿Qué es un registro en el contexto de datos?

a) Una colección de datos relacionados

b) Un tipo de dato dinámico

c) Un único valor de cadena

¿Cuál es un ejemplo de dato dinámico?

a) El nombre de un usuario

b) El saldo de una cuenta bancaria

c) El ID de un producto

Los _____ son fundamentales para identificar de manera única un registro en una tabla.

El _____ es un patrón utilizado para estructurar los datos en base a entidades y relaciones.

La _____ permite actualizar datos en tiempo real sin recargar toda la página.

El modelo _____ organiza la información en tablas con filas y columnas.

El _____ de los datos comienza con su creación y finaliza con su eliminación.

Sistemas de gestión de bases de datos (SGBD)