Contenido

1. Conceptos básicos de bases de datos relacionales

Una **base de datos relacional** es un sistema organizado de almacenamiento de datos que utiliza tablas para representar la información y las relaciones entre diferentes conjuntos de datos. Se basa en el **modelo relacional**, propuesto por Edgar F. Codd en 1970.

Tablas y registros

La información en una base de datos relacional se organiza en **tablas** (también llamadas relaciones). Cada tabla está compuesta por **filas** y **columnas**.

- Una **fila** representa un **registro** o **tupla**, que es una instancia concreta de la entidad representada en la tabla.
- Una **columna** representa un **atributo** o **campo**, que es una propiedad específica de la entidad.

Ejemplo: una tabla llamada **Clientes** podría tener las siguientes columnas: ID_Cliente, Nombre, Apellido, Email.

Claves primarias y claves foráneas

Una **clave primaria** es un campo (o combinación de campos) que identifica de manera única cada fila dentro de una tabla. No puede contener valores nulos ni repetidos. Una **clave foránea** es un campo en una tabla que hace referencia a la clave primaria de otra tabla. Se utiliza para establecer relaciones entre tablas.

Ejemplo: en una base de datos de pedidos, la tabla **Pedidos** podría tener una clave foránea llamada ID_Cliente que hace referencia a la clave primaria de la tabla **Clientes**.

Relaciones entre tablas

Las bases de datos relacionales permiten establecer conexiones entre diferentes tablas. Existen tres tipos principales de relaciones:

- **Uno a uno (1:1)**: Un registro de la tabla A se asocia con un solo registro de la tabla B.
- **Uno a muchos (1:M)**: Un registro de la tabla A se asocia con varios registros de la tabla B, pero cada registro de B solo está vinculado a un registro de A.
- **Muchos a muchos (M:N)**: Varios registros de la tabla A pueden estar relacionados con varios registros de la tabla B. Para gestionar esta relación, se usa una **tabla intermedia**.

Ejemplo: Un cliente puede hacer muchos pedidos, pero cada pedido pertenece a un único cliente (relación 1:M).

Normalización

Es el proceso de estructurar una base de datos para minimizar la redundancia y mejorar la integridad de los datos. Se logra a través de las **formas normales**, que son reglas que guían el diseño de las tablas.

Las tres primeras formas normales más comunes son:

1. **Primera Forma Normal (1NF)**: Se eliminan los grupos repetitivos, asegurando que cada campo contenga solo un valor.
2. **Segunda Forma Normal (2NF)**: Se eliminan dependencias parciales, asegurando que cada campo dependa completamente de la clave primaria.
3. **Tercera Forma Normal (3NF)**: Se eliminan dependencias transitivas, asegurando que los campos no dependan de otros campos que no sean clave primaria.

Consultas SQL

Para interactuar con una base de datos relacional, se utiliza **SQL (Structured Query Language)**. Algunas operaciones básicas incluyen:

- **SELECT**: Recupera datos de una tabla.
  SELECT Nombre, Apellido FROM Clientes WHERE ID_Cliente = 1;
- **INSERT**: Agrega un nuevo registro.
  INSERT INTO Clientes (ID_Cliente, Nombre, Apellido, Email) VALUES (1, 'Ana', 'García', 'ana@email.com');
- **UPDATE**: Modifica registros existentes.
  UPDATE Clientes SET Email = 'nuevo@email.com' WHERE ID_Cliente = 1;
- **DELETE**: Elimina registros.
  DELETE FROM Clientes WHERE ID_Cliente = 1;

Integridad referencial y restricciones

Las bases de datos relacionales aseguran la integridad de los datos mediante restricciones como:

- **NOT NULL**: Evita que un campo tenga valores nulos.
- **UNIQUE**: Garantiza que no haya valores duplicados en una columna.
- **CHECK**: Establece condiciones que deben cumplir los valores de un campo.
- **FOREIGN KEY**: Define claves foráneas para asegurar la relación correcta entre tablas.

Índices y optimización

Un **índice** mejora la velocidad de búsqueda en una base de datos, permitiendo acceso más rápido a los registros. Se pueden crear índices sobre columnas frecuentemente consultadas.
Ejemplo:

```
CREATE INDEX idx_nombre ON Clientes (Nombre);
```

El diseño de una base de datos relacional eficiente implica conocer estos conceptos y aplicarlos adecuadamente según las necesidades del sistema.

SQL

En bases de datos relacionales, SQL se divide en varios subconjuntos según el tipo de operación que realizan sobre los datos. Los tres más importantes son **DML (Data Manipulation Language)**, **DDL (Data Definition Language)** y **DCL (Data Control Language)**.

DML (Data Manipulation Language)

Incluye comandos que permiten manipular los datos almacenados en la base de datos. Son operaciones que afectan a los registros de las tablas sin modificar su estructura.

Comandos DML en Oracle XE

1. **INSERT**: Inserta nuevos registros en una tabla.

```
INSERT INTO Clientes (ID_Cliente, Nombre, Apellido, Email)
VALUES (1, 'Ana', 'García', 'ana@email.com');
```

2. **UPDATE**: Modifica registros existentes en una tabla.

```
UPDATE Clientes
SET Email = 'nuevo@email.com'
WHERE ID_Cliente = 1;
```

3. **DELETE**: Elimina registros de una tabla.

```
DELETE FROM Clientes WHERE ID_Cliente = 1;
```

4. **SELECT**: Consulta datos almacenados en la base de datos.

```
SELECT * FROM Clientes;
```

DDL (Data Definition Language)

Incluye comandos que definen la estructura de la base de datos, como la creación y modificación de tablas, índices y restricciones.

Comandos DDL en Oracle XE

1. **CREATE**: Crea objetos como bases de datos, tablas e índices.

```
CREATE TABLE Clientes (
    ID_Cliente NUMBER PRIMARY KEY,
    Nombre VARCHAR2(50),
    Apellido VARCHAR2(50),
    Email VARCHAR2(100) UNIQUE
);
```

2. **ALTER**: Modifica la estructura de una tabla (agregar, modificar o eliminar columnas).

```
ALTER TABLE Clientes ADD Telefono VARCHAR2(20);
```

3. **DROP**: Elimina completamente una tabla o cualquier otro objeto de la base de datos.

```
DROP TABLE Clientes;
```

4. **TRUNCATE**: Elimina todos los registros de una tabla sin afectar su estructura y sin posibilidad de reversión.

```
TRUNCATE TABLE Clientes;
```

DCL (Data Control Language)

Incluye comandos que gestionan los permisos y el control de acceso a los datos.

Comandos DCL en Oracle XE

1. **GRANT**: Otorga permisos a un usuario para realizar acciones específicas.

GRANT SELECT, INSERT ON Clientes TO usuario1;

2. **REVOKE**: Revoca los permisos previamente concedidos a un usuario.

REVOKE INSERT ON Clientes FROM usuario1;

Cada uno de estos conjuntos de comandos cumple una función específica en la gestión de una base de datos Oracle XE. Mientras que **DML** se usa para manipular los datos, **DDL** define la estructura de la base de datos y **DCL** gestiona los permisos y la seguridad.

2. TIPOS DE DATOS

Oracle XE soporta una amplia variedad de tipos de datos para manejar diferentes tipos de información. Se pueden clasificar en las siguientes categorías principales:

Tipos de Datos Numéricos

Se utilizan para almacenar valores numéricos, tanto enteros como decimales.

- **NUMBER(p, s)**: Número de precisión variable.
  - p: Número total de dígitos (máximo 38).
  - s: Número de dígitos a la derecha del punto decimal.
  - Ejemplo: NUMBER(10,2) almacena hasta 10 dígitos, con 2 después del decimal.
- **INTEGER**: Equivalente a NUMBER(38,0). Almacena enteros sin decimales.
  - Ejemplo: INTEGER almacena valores como 100, -50, 2500.
- **FLOAT (p)**: Número de punto flotante.
  - p define la precisión en bits.
  - Ejemplo: FLOAT(10) almacena números con precisión de 10 dígitos.
- **BINARY_FLOAT** y **BINARY_DOUBLE**: Números de punto flotante en precisión simple y doble.
  - BINARY_FLOAT: Más rápido pero menos preciso.
  - BINARY_DOUBLE: Mayor precisión para cálculos científicos.

Tipos de Datos de Caracteres

Se utilizan para almacenar texto.

- **CHAR(n)**: Cadena de longitud fija de n caracteres (máximo 2000 bytes).
    - Ejemplo: CHAR(10) almacena exactamente 10 caracteres.
- **VARCHAR2(n)**: Cadena de longitud variable, con un máximo de n caracteres (hasta 4000 bytes).
    - Ejemplo: VARCHAR2(100) almacena hasta 100 caracteres sin desperdiciar espacio.
- **NCHAR(n)** y **NVARCHAR2(n)**: Versiones Unicode de CHAR y VARCHAR2 para almacenar caracteres multibyte.

Tipos de Datos de Fecha y Hora

Se utilizan para almacenar fechas y horas.

- **DATE**: Almacena fecha y hora con precisión de segundos.
    - Ejemplo: DATE almacena valores como 2025-02-17 14:30:00.
- **TIMESTAMP [(n)]**: Similar a DATE, pero permite mayor precisión en los segundos (n define la cantidad de dígitos en la fracción de segundo, hasta 9).
    - Ejemplo: TIMESTAMP(3) almacena 2025-02-17 14:30:15.123.
- **TIMESTAMP WITH TIME ZONE**: Almacena fecha, hora y zona horaria.
    - Ejemplo: TIMESTAMP '2025-02-17 14:30:00 -05:00'.
- **TIMESTAMP WITH LOCAL TIME ZONE**: Similar al anterior, pero ajusta automáticamente la zona horaria según el usuario.
- **INTERVAL YEAR TO MONTH**: Almacena diferencias de tiempo en años y meses.
    - Ejemplo: INTERVAL '3-6' YEAR TO MONTH (3 años y 6 meses).
- **INTERVAL DAY TO SECOND**: Almacena diferencias de tiempo en días, horas, minutos y segundos.
    - Ejemplo: INTERVAL '5 12:30:45' DAY TO SECOND (5 días, 12 horas, 30 minutos y 45 segundos).

Tipos de Datos LOB (Large Objects)

Se utilizan para almacenar grandes volúmenes de datos como documentos, imágenes o vídeos.

- **CLOB (Character Large Object)**: Almacena hasta 128 TB de texto.
    - Ejemplo: CLOB se usa para almacenar documentos XML o JSON largos.
- **BLOB (Binary Large Object)**: Almacena datos binarios (imágenes, vídeos, audio).
    - Ejemplo: BLOB se usa para guardar archivos multimedia.
- **NCLOB**: Similar a CLOB, pero para caracteres Unicode.
- **BFILE**: Referencia un archivo binario almacenado fuera de la base de datos.

Tipos de Datos Especiales

Se usan en aplicaciones específicas.

- **RAW(n)**: Almacena datos binarios de longitud fija (máximo 2000 bytes).

    - Ejemplo: RAW(16) se usa para almacenar valores hash o claves encriptadas.
- **ROWID**: Representa la ubicación física de una fila en la base de datos.
    - Ejemplo: ROWID se usa en consultas internas para identificar registros de manera única.
- **UROWID**: Similar a ROWID, pero para tablas organizadas con índices.

3. COMANDOS DDL

Creación de una tabla

```
CREATE TABLE empleados (
    id NUMBER(5) PRIMARY KEY,  -- Clave primaria, número de hasta 5 dígitos
    nombre VARCHAR2(50) NOT NULL,  -- Campo obligatorio
    fecha_nacimiento DATE,  -- Tipo fecha
    salario NUMBER(10,2) CHECK (salario > 0), -- Número con dos decimales y restricción
    departamento_id NUMBER(3) REFERENCES departamentos(id) -- Clave foránea
);
```

Explicación:

- id es la clave primaria.
- nombre es un **campo obligatorio** (NOT NULL).
- fecha_nacimiento es de tipo **DATE**.
- salario tiene una **restricción** (CHECK) que impide valores negativos.
- departamento_id es una **clave foránea** (REFERENCES), que enlaza con la tabla departamentos.

Modificación de la tabla

Añadir una nueva columna

```
ALTER TABLE empleados ADD email VARCHAR2(100);
```

Esto agrega un nuevo campo email a la tabla empleados.

Modificar el tipo de un campo

```
ALTER TABLE empleados MODIFY salario NUMBER(12,2);
```

Aumenta el tamaño del campo salario de NUMBER(10,2) a NUMBER(12,2).

Eliminar una columna

```
ALTER TABLE empleados DROP COLUMN email;
```

Elimina la columna email.

Truncar la tabla

TRUNCATE TABLE empleados;

Explicación:

- Elimina todos los registros de la tabla, pero mantiene la estructura.
- No se puede revertir (no genera registros en el log de transacciones).

Borrar la tabla

DROP TABLE empleados;

Explicación:

- Elimina completamente la tabla y su estructura.
- Se pierden los datos y la tabla deja de existir en la base de datos.

4. Constraints

En **Oracle XE**, las limitaciones (**constraints**) permiten definir reglas sobre los datos almacenados en las tablas para garantizar su integridad. A continuación, te detallo los principales tipos de restricciones que puedes aplicar a los campos de una tabla, con ejemplos explicados.

1. PRIMARY KEY (Clave primaria)

Garantiza que cada fila en la tabla tiene un valor único y no nulo en la columna o columnas definidas.

Ejemplo:

```
CREATE TABLE empleados (
    id NUMBER(5) PRIMARY KEY,  -- No permite valores duplicados ni nulos
    nombre VARCHAR2(50) NOT NULL
);
```

- La columna id no puede contener valores repetidos ni NULL.
- Se puede definir sobre una única columna o múltiples columnas (clave primaria compuesta).

Clave primaria compuesta:

```
CREATE TABLE asistencia (
    empleado_id NUMBER(5),
    fecha DATE,
    PRIMARY KEY (empleado_id, fecha)  -- Ambos campos juntos deben ser únicos
);
```

FOREIGN KEY (Clave foránea)

Asegura la integridad referencial entre dos tablas, impidiendo que se inserten valores en una tabla si no existen en la tabla referenciada.

Ejemplo:

```
CREATE TABLE departamentos (
    id NUMBER(3) PRIMARY KEY,
    nombre VARCHAR2(50) NOT NULL
);
```

```
CREATE TABLE empleados (
    id NUMBER(5) PRIMARY KEY,
    nombre VARCHAR2(50) NOT NULL,
    departamento_id NUMBER(3),
    CONSTRAINT fk_departamento FOREIGN KEY (departamento_id)
        REFERENCES departamentos(id) ON DELETE CASCADE
);
```

- departamento_id en empleados debe existir en departamentos.
- **ON DELETE CASCADE**: Si un departamento es eliminado, sus empleados también se eliminan.

NOT NULL

Impide que una columna almacene valores NULL.

Ejemplo:

```
CREATE TABLE clientes (
    id NUMBER(5) PRIMARY KEY,
    nombre VARCHAR2(100) NOT NULL  -- Obligatorio, no puede ser NULL
);
```

- La columna nombre **siempre debe tener un valor** al insertar un nuevo registro.

UNIQUE

Asegura que todos los valores en una columna sean únicos, pero permite valores NULL.

Ejemplo:

```
CREATE TABLE productos (
    id NUMBER(5) PRIMARY KEY,
    codigo VARCHAR2(20) UNIQUE,  -- No se pueden repetir valores
    nombre VARCHAR2(50) NOT NULL
);
```

- codigo **no puede repetirse**, pero puede tener NULL.

Diferencia entre PRIMARY KEY y UNIQUE:

- Una tabla solo puede tener una **PRIMARY KEY**.
- Puede tener múltiples **UNIQUE** en diferentes columnas.

CHECK

Permite definir una condición lógica que los valores de una columna deben cumplir.

Ejemplo:

```
CREATE TABLE empleados (
    id NUMBER(5) PRIMARY KEY,
    salario NUMBER(10,2) CHECK (salario > 0),  -- Solo valores positivos
    edad NUMBER(2) CHECK (edad BETWEEN 18 AND 65) -- Solo edades válidas
);
```

- **salario** debe ser mayor que 0.
- **edad** debe estar entre 18 y 65 años.

DEFAULT

Define un valor por defecto si no se especifica otro al insertar datos.

Ejemplo:

```
CREATE TABLE pedidos (
    id NUMBER(5) PRIMARY KEY,
    estado VARCHAR2(20) DEFAULT 'Pendiente',   -- Si no se inserta, usa 'Pendiente'
    fecha_pedido DATE DEFAULT SYSDATE  -- Toma la fecha actual
);
```

- Si no se proporciona estado, se asigna 'Pendiente'.
- fecha_pedido usa la fecha actual del sistema.

IDENTITY (Autoincremento)

Genera valores secuenciales automáticamente (en Oracle 12c+).

Ejemplo:

```
CREATE TABLE clientes (
    id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  -- Autoincrementado
    nombre VARCHAR2(100) NOT NULL
```

```
);
```

- id se genera automáticamente sin necesidad de usar SEQUENCES.

Alternativa en versiones antiguas de Oracle (como XE 11g):

```
CREATE SEQUENCE clientes_seq START WITH 1 INCREMENT BY 1;

CREATE TABLE clientes (
    id NUMBER(5) PRIMARY KEY,
    nombre VARCHAR2(100) NOT NULL
);
```

```
INSERT INTO clientes (id, nombre) VALUES (clientes_seq.NEXTVAL, 'Juan Pérez');
```

Ejemplo Completo con todas las restricciones

```
CREATE TABLE empleados (
    id NUMBER(5) GENERATED ALWAYS AS IDENTITY PRIMARY KEY, -- Autoincremental y clave primaria
    nombre VARCHAR2(50) NOT NULL,  -- No puede ser NULL
    email VARCHAR2(100) UNIQUE,  -- No se puede repetir
    salario NUMBER(10,2) CHECK (salario > 0),  -- No permite valores negativos
    fecha_contratacion DATE DEFAULT SYSDATE,  -- Por defecto, la fecha actual
    departamento_id NUMBER(3),
    CONSTRAINT fk_departamento FOREIGN KEY (departamento_id) REFERENCES departamentos(id) ON DELETE CASCADE
);
```

Resumen de restricciones

Restricción	Función
PRIMARY KEY	Garantiza valores únicos y no nulos en una columna o conjunto de columnas.
FOREIGN KEY	Vincula una columna con otra tabla, asegurando integridad referencial.
NOT NULL	Impide que una columna tenga valores nulos.
UNIQUE	Garantiza que los valores de la columna sean únicos (permite NULL).

Restricción	Función
CHECK	Define condiciones lógicas para los valores de una columna.
DEFAULT	Especifica un valor predeterminado si no se proporciona otro.
IDENTITY	Genera valores secuenciales automáticamente (versión 12c+)