CONCEPTOS BÁSICOS DE LAS BASES DE DATOS RELACIONALES

1. Base de Datos Relacional (RDBMS): Una base de datos relacional es un sistema de gestión de bases de datos que organiza los datos en tablas relacionadas entre sí. Utiliza un enfoque tabular para almacenar datos y garantiza la integridad y consistencia de los datos a través de relaciones.

2. Tabla: En una base de datos relacional, los datos se almacenan en tablas. Cada tabla representa una entidad o concepto, como usuarios, productos o pedidos. Cada fila de la tabla se llama registro y cada columna se llama campo o atributo.

3. Clave Primaria: La clave primaria es un campo o conjunto de campos en una tabla que identifica de manera única cada registro. Garantiza que no haya duplicados y es fundamental para establecer relaciones entre tablas.

4. Clave Foránea: Una clave foránea es un campo en una tabla que se relaciona con la clave primaria de otra tabla. Establece relaciones entre tablas y permite la consulta de datos relacionados.

5. Relación: Una relación es la asociación lógica entre dos tablas a través de las claves primarias y foráneas. Puede ser uno a uno, uno a muchos o muchos a muchos.

6. SQL (Structured Query Language): SQL es un lenguaje de programación utilizado para gestionar y consultar bases de datos relacionales. Permite realizar operaciones como SELECT (consulta), INSERT (inserción), UPDATE (actualización) y DELETE (eliminación) de datos.

7. Consulta: Una consulta es una instrucción SQL que se utiliza para recuperar datos de una o varias tablas. Las consultas SELECT son comunes para obtener información de una base de datos.

8. Normalización: La normalización es un proceso de diseño de bases de datos que reduce la redundancia de datos al dividir las tablas en estructuras más pequeñas y relacionadas. Esto mejora la eficiencia y la integridad de la base de datos.

9. Transacción: Una transacción es una secuencia de operaciones de base de datos que se ejecutan como una unidad. Deben ser atómicas (todas se realizan o ninguna), consistentes (mantienen la integridad de la base de datos), aisladas (no se ven afectadas por otras transacciones) y duraderas (los cambios persisten).

10.Índice: Un índice es una estructura de datos que mejora la velocidad de búsqueda de registros en una tabla. Acelera la recuperación de datos, pero también puede aumentar el tamaño de la base de datos.

11.Vista: Una vista es una representación virtual de una o varias tablas. Permite a los usuarios consultar datos de manera conveniente sin acceder directamente a las tablas subyacentes.

12.Integridad de Datos: La integridad de datos se refiere a la precisión y coherencia de los datos almacenados en la base de datos. Se puede lograr a través de restricciones, como claves primarias y foráneas, y mediante la aplicación de reglas de negocio.13.Respaldos y Restauración: Los respaldos son copias de seguridad de la base de datos que

permiten recuperar datos en caso de pérdida o daño. La restauración implica volver a cargar una copia de seguridad en la base de datos.

LENGUAJE SQL CONSULTAS A LA BASE DE DATOS CON PHP:

1. **SELECT:** Se utiliza para recuperar datos de una o varias tablas.
   - Sintaxis: **SELECT columna1, columna2 FROM tabla WHERE condición;**
   - **Ejemplo: Para seleccionar todos los registros de una tabla llamada "clientes": SELECT * FROM clientes;**
2. **INSERT:** Permite agregar nuevos registros a una tabla.
   - Sintaxis: **INSERT INTO tabla (columna1, columna2) VALUES (valor1, valor2);**
   - **Ejemplo: Para agregar un nuevo cliente a la tabla "clientes":**

     INSERT INTO clientes (nombre, email) VALUES ('Juan Pérez', 'juan@example.com');
3. **UPDATE:** Se utiliza para modificar registros existentes en una tabla.
   - Sintaxis: **UPDATE tabla SET columna1 = nuevo_valor1, columna2 = nuevo_valor2 WHERE condición;**
   - **Ejemplo: Incrementar el precio de todos los productos en un 10%.**

     UPDATE Productos

     SET Precio = Precio * 1.10;
4. **DELETE:** Elimina registros de una tabla.
   - Sintaxis: **DELETE FROM tabla WHERE condición;**
   - **Ejemplo: DELETE FROM clientes WHERE id = 2;**
5. **JOIN:** Combina filas de dos o más tablas relacionadas.
   - Sintaxis básica: **SELECT columna1, columna2 FROM tabla1 INNER JOIN tabla2 ON tabla1.columna_comun = tabla2.columna_comun;**

     Ejemplo: Para obtener información de pedidos junto con los nombres de los clientes que los realizaron:

     SELECT pedidos.id, clientes.nombre FROM pedidos

     JOIN clientes ON pedidos.cliente_id = clientes.id;
6. **GROUP BY:** Agrupa filas que tienen valores idénticos en una o más columnas.
   - Sintaxis: **SELECT columna, COUNT(*) FROM tabla GROUP BY columna;**
   - **Ejemplo:** Para contar la cantidad de clientes por país:

     SELECT pais, COUNT(*) as total_clientes FROM clientes GROUP BY pais;
7. **ORDER BY:** Ordena el resultado de la consulta en función de una o más columnas.
   - Sintaxis: **SELECT columna1, columna2 FROM tabla ORDER BY columna ASC/DESC;**
   - **Ejemplo:** Para ordenar clientes por nombre de forma ascendente:

     SELECT * FROM clientes ORDER BY nombre ASC;

8. **WHERE:** Se utiliza para filtrar filas en una consulta según una condición.
    - Sintaxis: **`SELECT columna1, columna2 FROM tabla WHERE condición;`**
    - **`Ejemplo:`** `Para seleccionar clientes con un nombre específico:`

      `SELECT * FROM clientes WHERE nombre = 'María';`
9. **LIMIT:** Limita el número de filas devueltas por la consulta.
    - Sintaxis: **`SELECT columna1, columna2 FROM tabla LIMIT cantidad;`**
10. **OFFSET:** Se combina con LIMIT para paginar los resultados.
    - Sintaxis: **`SELECT columna1, columna2 FROM tabla LIMIT cantidad OFFSET inicio;`**
11. **Copia de Datos entre Tablas:**

    • Se utiliza para transferir datos de una tabla a otra.

    • Ejemplo: Copiar datos de una tabla temporal a una tabla permanente.

    INSERT INTO TablaPermanente

    SELECT * FROM TablaTemporal;

Estos son algunos de los elementos SQL básicos que se utilizan comúnmente en el diseño web con PHP para interactuar con bases de datos y mostrar datos dinámicos en páginas web. Cada uno de estos elementos se utiliza en consultas SQL para realizar operaciones específicas en la base de datos.

EJERCICIO PRÁCTICO de CONSULTAS

código sql de la base de datos:

```
-- Tabla "Departamento" para almacenar información sobre los departamentos
CREATE TABLE Departamento (
    ID_Departamento INT PRIMARY KEY,
    Nombre VARCHAR(255)
);
-- Tabla "Profesor" para almacenar información sobre los profesores
CREATE TABLE Profesor (
    ID_Profesor INT PRIMARY KEY,
    Nombre VARCHAR(255),
    Apellido VARCHAR(255),
    ID_Departamento INT,
    FOREIGN KEY (ID_Departamento) REFERENCES Departamento(ID_Departamento)
);
-- Tabla "Asignatura" para almacenar información sobre las asignaturas
CREATE TABLE Asignatura (
    ID_Asignatura INT PRIMARY KEY,
    Nombre VARCHAR(255),
    ID_Departamento INT,
    FOREIGN KEY (ID_Departamento) REFERENCES Departamento(ID_Departamento)
);
-- Tabla "Alumno" para almacenar información sobre los alumnos
CREATE TABLE Alumno (
    ID_Alumno INT PRIMARY KEY,
    Nombre VARCHAR(255),
    Apellido VARCHAR(255)
);
-- Tabla "Curso" para almacenar información sobre los cursos
CREATE TABLE Curso (
    ID_Curso INT PRIMARY KEY,
    Nombre VARCHAR(255)
);
-- Tabla intermedia "Matricula" para representar la relación entre Alumno y Curso
CREATE TABLE Matricula (
    ID_Matricula INT PRIMARY KEY,
    ID_Alumno INT,
    ID_Curso INT,
    FOREIGN KEY (ID_Alumno) REFERENCES Alumno(ID_Alumno),
    FOREIGN KEY (ID_Curso) REFERENCES Curso(ID_Curso)
);

-- Tabla intermedia "Ensenyada" para representar la relación entre Profesor y Asignatura
CREATE TABLE Ensenyada (
    ID_Ensenyada INT PRIMARY KEY,
    ID_Profesor INT,
    ID_Asignatura INT,
    FOREIGN KEY (ID_Profesor) REFERENCES Profesor(ID_Profesor),
    FOREIGN KEY (ID_Asignatura) REFERENCES Asignatura(ID_Asignatura)
);
```

Insertar algunos datos iniciales en las tablas de tu base de datos. Asumiré que tienes las siguientes tablas: "Alumno," "Profesor," "Curso," "Asignatura," "Matricula," y "Ensenyada." A continuación, te mostraré cómo insertar algunos registros de ejemplo en estas tablas:

```
-- Insertar datos en la tabla Curso
INSERT INTO Curso (Nombre) VALUES
    ('1º ESO'),
    ('2º ESO'),
    ('3º ESO');

-- Insertar datos en la tabla Asignatura
INSERT INTO Asignatura (Nombre, Departamento) VALUES
    ('Matemáticas', 'Ciencias'),
    ('Física', 'Ciencias'),
    ('Química', 'Ciencias'),
    ('Historia', 'Letras'),
    ('Geografía', 'Letras');

-- Insertar datos en la tabla Profesor
INSERT INTO Profesor (Nombre, Apellido, Departamento) VALUES
    ('Juan', 'García', 'Ciencias'),
    ('María', 'López', 'Ciencias'),
    ('Pedro', 'Martínez', 'Ciencias'),
    ('Ana', 'Rodríguez', 'Letras'),
    ('Luis', 'Pérez', 'Letras'),
    ('Laura', 'Fernández', 'Letras');

-- Insertar datos en la tabla Alumno
INSERT INTO Alumno (Nombre, Apellido) VALUES
    ('Carlos', 'Gómez'),
    ('Sofía', 'Hernández'),
    ('Miguel', 'Díaz'),
    ('Isabel', 'Ruiz'),
    ('Alejandro', 'Torres'),
    ('Lucía', 'Sánchez'),
    ('Daniel', 'Vargas'),
    ('Valentina', 'Luna'),
    ('Javier', 'Gutiérrez'),
    ('Paula', 'Mendoza');

-- Insertar datos en la tabla Matricula
INSERT INTO Matricula (ID_Alumno, ID_Curso) VALUES
    (1, 1),
    (2, 1),
    (3, 1),
    (4, 2),
    (5, 2),
    (6, 2),
    (7, 3),
    (8, 3),
    (9, 3),
    (10, 3);

-- Insertar datos en la tabla Ensenyada (Relación entre profesores y asignaturas)
-- Supongamos que los profesores Juan, María y Pedro enseñan Matemáticas, Física y Química,
respectivamente.
-- Y que Ana y Luis enseñan Historia y Geografía, respectivamente.

INSERT INTO Ensenyada (ID_Profesor, ID_Asignatura) VALUES
    (1, 1), -- Juan enseña Matemáticas
    (2, 2), -- María enseña Física
    (3, 3), -- Pedro enseña Química
    (4, 4), -- Ana enseña Historia
    (5, 5); -- Luis enseña Geografía
```

INNER JOIN (JOIN por defecto):

La cláusula INNER JOIN devuelve solo los registros que tienen valores coincidentes en ambas tablas.

```
SELECT columna1, columna2, ...
FROM tabla1
INNER JOIN tabla2
ON tabla1.columna_comun = tabla2.columna_comun;
```

Ejemplo:

Listar alumnos y sus cursos (suponiendo que hay una relación entre la tabla "Alumno" y la tabla "Curso" a través de la columna "ID_Curso"):

```
SELECT Alumno.Nombre, Alumno.Apellido, Curso.Nombre AS Curso
FROM Alumno
INNER JOIN Curso ON Alumno.ID_Curso = Curso.ID_Curso;
```

LEFT JOIN (o LEFT OUTER JOIN):

La cláusula LEFT JOIN devuelve todos los registros de la tabla izquierda (primera tabla) y los registros coincidentes de la tabla derecha (segunda tabla). Si no hay coincidencias en la tabla derecha, se rellenará con valores nulos.

```
SELECT columna1, columna2, ...
FROM tabla1
LEFT JOIN tabla2
ON tabla1.columna_comun = tabla2.columna_comun;
```

Ejemplo:

Listar todos los profesores y sus alumnos (incluso si algunos profesores no tienen alumnos):

```
SELECT Profesor.Nombre, Alumno.Nombre AS NombreAlumno
FROM Profesor
LEFT JOIN Matricula ON Profesor.ID_Profesor = Matricula.ID_Profesor
LEFT JOIN Alumno ON Matricula.ID_Alumno = Alumno.ID_Alumno;
```

RIGHT JOIN (o RIGHT OUTER JOIN):

La cláusula RIGHT JOIN funciona de manera similar a LEFT JOIN, pero devuelve todos los registros de la tabla derecha (segunda tabla) y los registros coincidentes de la tabla izquierda (primera tabla).

```
SELECT columna1, columna2, ...
FROM tabla1
RIGHT JOIN tabla2
ON tabla1.columna_comun = tabla2.columna_comun;
```

Es importante destacar que no todas las bases de datos admiten RIGHT JOIN de manera nativa, por lo que en algunos casos puedes lograr el mismo resultado intercambiando el orden de las tablas en una LEFT JOIN.

PRÁCTICAS: Consultas a la base de datos con PHP

Código para conectar con la base de datos: **conectar.php**

```
// Establecer la conexión a la base de datos (reemplazar con tus datos)
$servername = "nombre_servidor";
$username = "nombre_usuario";
$password = "contraseña";
$dbname = "nombre_base_datos";
$conn = new mysqli($servername, $username, $password, $dbname);
// Verificar la conexión
if ($conn->connect_error) {
die("Conexión fallida: " . $conn->connect_error);
}
```

Listar Alumnos por Profesor: listar_alumnos_por_curso.php

```
<?php
// Incluye la conexión a la base de datos
include 'conexion.php';

// Realiza la consulta SQL para obtener la lista de cursos
$sqlCursos = "SELECT * FROM Curso";
$resultadoCursos = $conn->query($sqlCursos);
?>

<!DOCTYPE html>
<html>
<head>
    <title>Listar Alumnos por Curso</title>
</head>
<body>
    <h1>Listar Alumnos por Curso</h1>
    <form method="get" action="">
        <label for="curso">Seleccione un curso:</label>
        <select name="curso" id="curso">
            <?php while ($rowCurso = $resultadoCursos->fetch_assoc()): ?>
                <option value="<?php echo $rowCurso['ID_Curso']; ?>"><?php echo
$rowCurso['Nombre']; ?></option>
            <?php endwhile; ?>
        </select>
        <input type="submit" value="Mostrar Alumnos">
    </form>

    <?php
    // Verifica si se ha enviado el formulario para mostrar alumnos por curso
    if (isset($_GET['curso'])) {
        $idCurso = $_GET['curso'];

        // Realiza la consulta SQL para obtener la lista de alumnos por curso
        $sqlAlumnosPorCurso = "SELECT Alumno.Nombre AS NombreAlumno,
Alumno.Apellido AS ApellidoAlumno FROM Alumno INNER JOIN Matricula ON
Alumno.ID_Alumno = Matricula.ID_Alumno  WHERE Matricula.ID_Curso = $idCurso";
```

```
        $resultadoAlumnosPorCurso = $conn->query($sqlAlumnosPorCurso);

        // Muestra la lista de alumnos por curso en una tabla HTML
        if ($resultadoAlumnosPorCurso->num_rows > 0) {
            echo "<h2>Alumnos en el curso:</h2>";
            echo "<ul>";
            while ($row = $resultadoAlumnosPorCurso->fetch_assoc()) {
                echo "<li>" . $row["NombreAlumno"] . " " . $row["ApellidoAlumno"] .
"</li>";
            }
            echo "</ul>";
        } else {
            echo "No se encontraron alumnos en este curso.";
        }
    }
    ?>
</body>
</html>
```

Listar Alumnos por Profesor: **listar_alumnos_por_profesor.php**

```
<?php
// Incluye la conexión a la base de datos
include 'conexion.php';

// Realiza la consulta SQL para obtener la lista de profesores
$sqlProfesores = "SELECT * FROM Profesor";
$resultadoProfesores = $conn->query($sqlProfesores);
?>

<!DOCTYPE html>
<html>
<head>
    <title>Listar Alumnos por Profesor</title>
</head>
<body>
    <h1>Listar Alumnos por Profesor</h1>
    <form method="get" action="">
        <label for="profesor">Seleccione un profesor:</label>
        <select name="profesor" id="profesor">
            <?php while ($rowProfesor = $resultadoProfesores->fetch_assoc()): ?>
                <option value="<?php echo $rowProfesor['ID_Profesor']; ?>"><?php
echo $rowProfesor['Nombre'] . ' ' . $rowProfesor['Apellido']; ?></option>
            <?php endwhile; ?>
        </select>
        <input type="submit" value="Mostrar Alumnos">
    </form>

    <?php
    // Verifica si se ha enviado el formulario para mostrar alumnos por profesor
    if (isset($_GET['profesor'])) {
        $idProfesor = $_GET['profesor'];

        // Realiza la consulta SQL para obtener la lista de alumnos por profesor
        $sqlAlumnosPorProfesor = "SELECT Alumno.Nombre AS NombreAlumno,
Alumno.Apellido AS ApellidoAlumno FROM Alumno INNER JOIN Matricula ON
Alumno.ID_Alumno = Matricula.ID_Alumno INNER JOIN Ensenyada ON Matricula.ID_Profesor
= Ensenyada.ID_Profesor WHERE Ensenyada.ID_Profesor = $idProfesor";
```

```
        $resultadoAlumnosPorProfesor = $conn->query($sqlAlumnosPorProfesor);

        // Muestra la lista de alumnos por profesor en una tabla HTML
        if ($resultadoAlumnosPorProfesor->num_rows > 0) {
            echo "<h2>Alumnos del profesor:</h2>";
            echo "<ul>";
            while ($row = $resultadoAlumnosPorProfesor->fetch_assoc()) {
                echo "<li>" . $row["NombreAlumno"] . " " . $row["ApellidoAlumno"] .
"</li>";
            }
            echo "</ul>";
        } else {
            echo "No se encontraron alumnos para este profesor.";
        }
    }
    ?>
</body>
                                        </html>
```

Listar Alumnos por Asignatura: **listar_alumnos_por_asignatura.php**

```
<?php
// Incluye la conexión a la base de datos
include 'conexion.php';

// Realiza la consulta SQL para obtener la lista de asignaturas
$sqlAsignaturas = "SELECT * FROM Asignatura";
$resultadoAsignaturas = $conn->query($sqlAsignaturas);
?>

<!DOCTYPE html>
<html>
<head>
    <title>Listar Alumnos por Asignatura</title>
</head>
<body>
    <h1>Listar Alumnos por Asignatura</h1>
    <form method="get" action="">
        <label for="asignatura">Seleccione una asignatura:</label>
        <select name="asignatura" id="asignatura">
            <?php while ($rowAsignatura = $resultadoAsignaturas->fetch_assoc()): ?>
                <option value="<?php echo $rowAsignatura['ID_Asignatura']; ?>"><?php
echo $rowAsignatura['Nombre']; ?></option>
            <?php endwhile; ?>
        </select>
        <input type="submit" value="Mostrar Alumnos">
    </form>

    <?php
    // Verifica si se ha enviado el formulario para mostrar alumnos por asignatura
```

```
    if (isset($_GET['asignatura'])) {
        $idAsignatura = $_GET['asignatura'];

        // Realiza la consulta SQL para obtener la lista de alumnos por asignatura utilizando JOIN
        $sqlAlumnosPorAsignatura = "SELECT Alumno.Nombre AS NombreAlumno, Alumno.Apellido AS ApellidoAlumno FROM Alumno INNER JOIN Matricula ON Alumno.ID_Alumno = Matricula.ID_Alumno INNER JOIN Ensenyada ON Matricula.ID_Profesor = Ensenyada.ID_Profesor WHERE Ensenyada.ID_Asignatura = $idAsignatura";

        $resultadoAlumnosPorAsignatura = $conn->query($sqlAlumnosPorAsignatura);

        // Muestra la lista de alumnos por asignatura en una tabla HTML
        if ($resultadoAlumnosPorAsignatura->num_rows > 0) {
            echo "<h2>Alumnos en la asignatura:</h2>";
            echo "<table border='1'>";
            echo "<tr><th>Nombre</th><th>Apellido</th></tr>";
            while ($row = $resultadoAlumnosPorAsignatura->fetch_assoc()) {
                echo "<tr><td>" . $row["NombreAlumno"] . "</td><td>" . $row["ApellidoAlumno"] . "</td></tr>";
            }
            echo "</table>";
        } else {
            echo "No se encontraron alumnos para esta asignatura.";
        }
    }
    ?>
</body>
</html>
```

Ejemplo:

Supongamos que tenemos dos tablas, "Clientes" y "Pedidos". La tabla "Clientes" contiene información sobre los clientes y la tabla "Pedidos" contiene información sobre los pedidos realizados por esos clientes. Ambas tablas están relacionadas por la columna "ID_Cliente".

1. **Clientes:**
   - ID_Cliente (Clave primaria)
   - Nombre
   - Email
2. **Pedidos:**
   - ID_Pedido (Clave primaria)
   - ID_Cliente (Clave foránea)
   - Fecha
   - Total

Para obtener una lista de pedidos con los nombres de los clientes que los realizaron pedidos, puedes usar la cláusula JOIN de la siguiente manera:

```
SELECT Pedidos.ID_Pedido, Clientes.Nombre, Pedidos.Fecha, Pedidos.Total
FROM Pedidos
JOIN Clientes ON Pedidos.ID_Cliente = Clientes.ID_Cliente;
```

Explicación:

- **SELECT Pedidos.ID_Pedido, Clientes.Nombre, Pedidos.Fecha, Pedidos.Total**: Esto selecciona las columnas que queremos recuperar en el resultado de la consulta.
- **FROM Pedidos**: Indica que estamos seleccionando datos de la tabla "Pedidos".
- **JOIN Clientes**: Especifica que queremos combinar datos de la tabla "Pedidos" con la tabla "Clientes".
- **ON Pedidos.ID_Cliente = Clientes.ID_Cliente**: Esta parte establece la relación entre las dos tablas utilizando la columna "ID_Cliente". La cláusula ON indica que estamos relacionando las filas donde el valor de "ID_Cliente" en la tabla "Pedidos" sea igual al valor de "ID_Cliente" en la tabla "Clientes".

El resultado de esta consulta mostrará una lista de pedidos con los nombres de los clientes que los realizaron, lo que facilita la comprensión de quién hizo cada pedido.

La cláusula JOIN es una herramienta poderosa para trabajar con bases de datos relacionales y permite realizar consultas complejas que involucran múltiples tablas de manera eficiente.

NOTAS SOBRE LOS CÓDIGOS Y SINTAXIS USADOS

Conexión a la base de datos

sintaxis: `$resultadoAlumnosPorCurso = $conn->query($sqlAlumnosPorCurso);`

representa una operación común en PHP que se utiliza para **ejecutar una consulta SQL** en una base de datos utilizando una conexión activa:

- **$resultadoAlumnosPorCurso**: Esta es una variable en la que se almacenará el resultado de la consulta. Después de ejecutar la consulta SQL, esta variable contendrá los datos recuperados de la base de datos.
- **$conn->query($sqlAlumnosPorCurso)**: Esto se desglosa de la siguiente manera:
  - **$conn** es una variable que generalmente representa una conexión activa a una base de datos. Esta variable debería haber sido previamente configurada para conectarse a la base de datos utilizando una extensión como MySQLi o PDO.
  - **->** es el operador de acceso a miembros en PHP. En este contexto, se utiliza para acceder a un método llamado query() que pertenece al objeto representado por la variable $conn. Este método se utiliza para ejecutar consultas SQL en la base de datos.
  - **$sqlAlumnosPorCurso** es una variable que contiene la consulta SQL que deseas ejecutar.

Entonces, en resumen, la línea de código está ejecutando la consulta SQL almacenada en **$sqlAlumnosPorCurso** en la base de datos representada por la **conexión $conn**, y el resultado de esa consulta se almacena en la variable

$resultadoAlumnosPorCurso. Luego, puedes utilizar esta variable para acceder a los datos recuperados de la base de datos y realizar acciones adicionales, como mostrarlos en una página web o procesarlos de alguna otra manera.

El método query()

El método query() es una función utilizada en PHP para ejecutar consultas SQL en una base de datos. Aquí tienes un resumen de este método:

- **Nombre del Método:** query()
- **Uso Principal:** Se utiliza para enviar una consulta SQL a una base de datos y ejecutarla.
- **Sintaxis Básica:** **$resultado = $conexion->query($consulta);**
- **Parámetros:**
  - **$conexion**: Representa una conexión activa a la base de datos previamente configurada utilizando extensiones como MySQLi o PDO.
  - **$consulta**: Contiene la consulta SQL que se desea ejecutar.
- **Resultado:** El método devuelve un objeto que representa el resultado de la consulta. Este objeto puede contener los datos recuperados de la base de datos y proporciona métodos para acceder a esos datos.
- **Uso Típico:** Después de ejecutar una consulta utilizando query(), se suele utilizar métodos adicionales del objeto de resultado para recuperar y manipular los datos, como fetch() o fetch_assoc().
- **Nota:** Es importante manejar los errores que puedan ocurrir durante la ejecución de la consulta utilizando manejo de excepciones o comprobación de errores para asegurarse de que la consulta se realizó correctamente.

En resumen, query() es una función fundamental para interactuar con bases de datos en PHP, ya que **permite ejecutar consultas SQL y recuperar datos de la base de datos para su posterior procesamiento**.

Los métodos fetch() y fetch_assoc()

Los métodos fetch() y fetch_assoc() son utilizados en PHP para **recuperar filas de resultados de una consulta SQL** ejecutada en una base de datos. Ambos métodos se aplican a un objeto que representa el resultado de una consulta, y cada uno tiene sus propias características. Aquí te explico cada uno de ellos:

1. fetch():

- **Uso Principal:** fetch() se utiliza para recuperar una fila de resultados de la consulta. Esta fila se devuelve como un array indexado, lo que significa que los valores se acceden por posición numérica en lugar de nombres de columnas.
- **Sintaxis Básica:** $fila = $resultado->fetch();
- **Resultado:** $fila contendrá la siguiente fila de resultados como un array indexado, o false si no hay más filas que recuperar.

- **Ejemplo:**

```
$fila = $resultado->fetch();
echo $fila[0]; // Accede al primer valor de la fila
echo $fila[1]; // Accede al segundo valor de la fila, y así sucesivamente
```

2. `fetch_assoc()`:

- **Uso Principal:** fetch_assoc() se utiliza para **recuperar una fila de resultados de la consulta**. A diferencia de fetch(), esta fila se devuelve como un array asociativo, lo que significa que los valores se acceden por nombres de columnas.
- **Sintaxis Básica:** `$filaAsociativa = $resultado->fetch_assoc();`
- **Resultado:** $filaAsociativa contendrá la siguiente fila de resultados como un array asociativo, donde las claves son los nombres de las columnas y los valores son los valores de esas columnas. Si no hay más filas, se devuelve null.
- **Ejemplo:**

```
$filaAsociativa = $resultado->fetch_assoc();
echo $filaAsociativa["columna1"]; // Accede al valor de la columna "columna1"
echo $filaAsociativa["columna2"]; // Accede al valor de la columna "columna2", y así sucesivamente
```

En resumen, **fetch()** se utiliza para recuperar filas de resultados como arrays indexados, mientras que **fetch_assoc()** se utiliza para obtener filas como arrays asociativos, lo que hace que sea más conveniente acceder a los valores de las columnas utilizando los nombres de las columnas en lugar de índices numéricos.

La elección entre estos métodos depende de cómo desees manipular los datos recuperados de la base de datos.

bloque de código:

```
if (isset($_GET['curso'])) { $idCurso = $_GET['curso'];
```

Este bloque de código PHP está diseñado para realizar una serie de acciones basadas en una solicitud GET enviada a través de la URL. Aquí está una explicación paso a paso de lo que hace este código:

1. **if (isset($_GET['curso']))** {: Esta línea verifica si el parámetro "curso" se ha enviado a través de la URL utilizando la superglobal $_GET. Si se ha enviado, significa que el usuario ha seleccionado un curso y desea ver la lista de alumnos en ese curso.
2. **$idCurso = $_GET['curso'];**: Si se ha enviado el parámetro "curso", esta línea recoge el valor de "curso" desde la URL y lo almacena en la variable $idCurso para su posterior uso en la consulta SQL.

El bloque de codigo:

```
if ($resultadoAlumnosPorAsignatura->num_rows > 0) {
     echo "<h2>Alumnos en la asignatura:</h2>";
     echo "<table border='1'>";
     echo "<tr><th>Nombre</th><th>Apellido</th></tr>";
     while ($row = $resultadoAlumnosPorAsignatura->fetch_assoc()) {
     echo "<tr><td>" . $row["NombreAlumno"] . "</td><td>" .
     $row["ApellidoAlumno"] . "</td></tr>";
}
echo "</table>";
} else {
echo "No se encontraron alumnos para esta asignatura.";
}
}
```

Este bloque de código PHP se encarga de mostrar los resultados de una consulta SQL que busca y lista los alumnos que están inscritos en una asignatura específica. Aquí está una explicación paso a paso de lo que hace este código:

1. **`if ($resultadoAlumnosPorAsignatura->num_rows > 0) {`**: Esta línea verifica si el objeto $resultadoAlumnosPorAsignatura contiene al menos una fila de resultados. En otras palabras, comprueba si hay alumnos inscritos en la asignatura seleccionada.

2. **`echo "<h2>Alumnos en la asignatura:</h2>";`**: Si se encontraron resultados (alumnos inscritos en la asignatura), se muestra un encabezado en HTML (<h2>) que indica que se mostrará una lista de alumnos en la asignatura.

3. **`echo "<table border='1'>";`**: Se inicia una tabla HTML (<table>) con un borde visible para organizar y mostrar los resultados en un formato tabular.

4. **`echo "<tr><th>Nombre</th><th>Apellido</th></tr>";`**: Se crea una fila de encabezado de tabla (<tr>) con dos columnas de encabezado (<th>) que indican las columnas que se mostrarán en la tabla: "Nombre" y "Apellido".

5. **`while ($row = $resultadoAlumnosPorAsignatura->fetch_assoc()) {`**: Se inicia un bucle while que recorre cada fila de resultados devueltos por la consulta. El método fetch_assoc() se utiliza para obtener cada fila como un array asociativo, donde las claves son los nombres de las columnas.

6. **`echo "<tr><td>" . $row["NombreAlumno"] . "</td><td>" $row["ApellidoAlumno"] . "</td></tr>";`**: Dentro del bucle, se crea una nueva fila de tabla (<tr>) para cada alumno y se muestran sus datos (nombre y apellido) en celdas de tabla (<td>).

7. **`echo "</table>";`**: Se cierra la tabla HTML después de mostrar todos los alumnos.

8. **`} else {`**: Si la consulta no devuelve ningún resultado (es decir, no hay alumnos inscritos en la asignatura), se ejecuta este bloque de código.
9. **`echo "No se encontraron alumnos para esta asignatura.";`**: Se muestra un mensaje en HTML que indica que no se encontraron alumnos inscritos en la asignatura seleccionada.

En resumen, **este bloque de código verifica si la consulta SQL devolvió resultados** y, en caso afirmativo, muestra una tabla HTML con la lista de alumnos en la asignatura, incluyendo sus nombres y apellidos. Si no se encuentran alumnos en la asignatura, se muestra un mensaje de "No se encontraron alumnos".

EJEMPLO: Conectar y mostrar datos de dos tablas relacionadas: "usuarios" y "pedidos"

ejemplo-join.php

```
//  Ejemplo de Consulta SQL con JOIN
// Establecer la conexión a la base de datos (reemplazar con tus datos)
$servername = "nombre_servidor";
$username = "nombre_usuario";
$password = "contraseña";
$dbname = "nombre_base_datos";
$conn = new mysqli($servername, $username, $password, $dbname);
// Verificar la conexión
if ($conn->connect_error) {
die("Conexión fallida: " . $conn->connect_error);
}
// Consulta SQL para obtener información de usuarios y sus pedidos
$sql = "SELECT usuarios.nombre AS nombre_usuario, pedidos.producto
FROM usuarios
INNER JOIN pedidos ON usuarios.id = pedidos.usuario_id";
$result = $conn->query($sql);if ($result->num_rows > 0) {
// Mostrar los resultados en una tabla HTML
echo "<table>
<tr>
<th>Usuario</th>
<th>Producto</th>
</tr>";
```

```
while ($row = $result->fetch_assoc()) {
echo "<tr>
<td>" . $row["nombre_usuario"] . "</td>
<td>" . $row["producto"] . "</td>
</tr>";
}
echo "</table>";
} else {
echo "No se encontraron registros.";
}
// Cerrar la conexión a la base de datos
$conn→close();
```

Código sql para implementar las bases de datos del ejemplo

```
-- Crear la tabla "usuarios"
CREATE TABLE usuarios (
id INT AUTO_INCREMENT PRIMARY KEY,
nombre VARCHAR(255) NOT NULL,
correo VARCHAR(255) NOT NULL
);
-- Insertar datos en la tabla "usuarios"
INSERT INTO usuarios (nombre, correo) VALUES
('Usuario 1', 'usuario1@example.com'),
('Usuario 2', 'usuario2@example.com');
-- Crear la tabla "pedidos"
CREATE TABLE pedidos (
id INT AUTO_INCREMENT PRIMARY KEY,
usuario_id INT NOT NULL,
producto VARCHAR(255) NOT NULL,
cantidad INT NOT NULL
);
-- Insertar datos en la tabla "pedidos"
INSERT INTO pedidos (usuario_id, producto, cantidad) VALUES
(1, 'Producto A', 3),
(1, 'Producto B', 2),
(2, 'Producto C', 1),
(2, 'Producto A', 4);
```

Del ejemplo anterior, incluyendo dos registros por cada tabla ("usuarios" y "pedidos"). Asegúrate de reemplazar nombre_base_datos por el nombre de tu base de datos y configura los otros valores de conexión según tu entorno.

Este código SQL creará dos tablas, "usuarios" y "pedidos", e insertará dos registros en cada tabla para que se pueda comenzar a trabajar con ellas en tu aplicación web.

LAS CONSULTAS MÚLTIPLES Y LAS UNIONES (JOINS) EN SQL

Son técnicas que te permiten combinar datos de múltiples tablas para obtener resultados más completos y útiles.
Aquí tienes un resumen y varios ejemplos de consultas múltiples y uniones: Consultas Múltiples y Uniones:

• Las consultas múltiples y las uniones permiten combinar datos de dos o más tablas en una única consulta.
• Se utilizan para recuperar información relacionada de múltiples tablas, lo que es esencial en bases de datos relacionales.
• Las cláusulas comunes para combinar tablas son **INNER JOIN, LEFT JOIN, RIGHT JOIN y FULL JOIN.**
• Se especifican las relaciones entre las tablas utilizando columnas compartidas, generalmente claves primarias y externas.
• Las consultas múltiples y uniones ayudan a extraer datos relacionados de manera eficiente y generar informes complejos.

INNER JOIN:

• Combina registros que tienen valores coincidentes en ambas tablas.
• Ejemplo: Obtener una lista de pedidos con detalles de productos.

```
SELECT Pedidos.ID, Productos.Nombre
FROM Pedidos
INNER JOIN DetallesDePedido ON Pedidos.ID = DetallesDePedido.PedidoID;
```

LEFT JOIN:

• Devuelve todos los registros de la tabla izquierda y los registros coincidentes de la tabla derecha.
• Ejemplo: Obtener una lista de todos los empleados y sus departamentos, incluyendo aquellos sin asignar a un departamento.

```
SELECT Empleados.Nombre, Departamentos.Nombre AS Departamento
FROM Empleados
LEFT JOIN Departamentos ON Empleados.DepartamentoID = Departamentos.ID;
```

RIGHT JOIN:

• Devuelve todos los registros de la tabla derecha y los registros coincidentes de la tabla izquierda.
• Ejemplo: Obtener una lista de todos los departamentos y los empleados asignados a ellos, incluyendo aquellos sin empleados.

```
SELECT Departamentos.Nombre AS Departamento, Empleados.Nombre
FROM Departamentos
RIGHT JOIN Empleados ON Departamentos.ID = Empleados.DepartamentoID;
```

FULL JOIN:

• Devuelve todos los registros cuando hay una coincidencia en una de las tablas.
• Ejemplo: Obtener una lista de todos los clientes y sus pedidos, incluyendo clientes sin pedidos y pedidos sin cliente.

```
SELECT Clientes.Nombre, Pedidos.ID
FROM Clientes
FULL JOIN Pedidos ON Clientes.ID = Pedidos.ClienteID;
```

Auto-Uniones:

• También puedes unir una tabla consigo misma para relaciones en la misma tabla.

• Ejemplo: Encontrar empleados que tienen el mismo administrador.

```
SELECT e1.Nombre AS Empleado, e2.Nombre AS Supervisor
FROM Empleados e1
INNER JOIN Empleados e2 ON e1.SupervisorID = e2.ID;
```

Estos ejemplos ilustran cómo utilizar diferentes tipos de uniones para combinar datos de múltiples tablas en SQL y extraer información relevante de una base de datos relacional.

Ejemplo: consulta SQL

Supongamos que tenemos una base de datos de una tienda en línea con las siguientes tablas: "**Productos**" y "**Pedidos**". Queremos realizar una consulta para obtener información sobre los productos más populares que se han vendido en el último mes:

```
SELECT
    Productos.Nombre AS NombreDelProducto,
    COUNT(Pedidos.ID) AS CantidadVendida,
    SUM(Pedidos.Precio) AS IngresosTotales
FROM Productos
JOIN DetallesDePedido ON Productos.ID = DetallesDePedido.ProductoID
JOIN Pedidos ON DetallesDePedido.PedidoID = Pedidos.ID
WHERE Pedidos.Fecha >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
GROUP BY Productos.Nombre
HAVING CantidadVendida > 10
ORDER BY CantidadVendida DESC;
```

Explicación de la consulta:

• **Selecciona tres columnas** para mostrar en los resultados: el nombre del producto, la cantidad vendida y los ingresos totales.

• Utiliza la cláusula **FROM** para especificar las tablas involucradas en la consulta (Productos, DetallesDePedido y Pedidos) y establece las relaciones entre ellas mediante las cláusulas JOIN.

• Usa **WHERE para filtrar los registros** y seleccionar solo aquellos con pedidos realizados en el último mes.• Agrupa los resultados utilizando GROUP BY para obtener la cantidad vendida por producto.

• Aplica una condición de **filtrado adicional con HAVING** para mostrar solo los productos que se han vendido más de 10 veces en el último mes.

• Finalmente, ordena los resultados en orden descendente según la cantidad vendida utilizando **ORDER BY.**

Esta consulta muestra los productos más populares en términos de ventas y los ingresos generados en el último mes, lo que puede ser útil para tomar decisiones comerciales basadas en datos.

LA AGRUPACIÓN EN BASES DE DATOS SQL

Es una técnica que permite combinar registros en conjuntos más pequeños basados en un criterio específico y realizar cálculos o agregaciones en cada uno de estos conjuntos. Resumen del concepto de agrupación:

Concepto de Agrupación en Bases de Datos SQL:

• Agrupación (**GROUP BY**): La cláusula GROUP BY se utiliza para **dividir los registros en grupos** basados en los valores de una o más columnas.

• Agregaciones: Después de agrupar los registros, se pueden aplicar funciones de agregación, como SUM, COUNT, AVG, MAX o MIN, a las columnas de cada grupo. Estas funciones resumen los datos dentro de cada grupo.

• Resumen de Datos: La agrupación se utiliza para resumir grandes conjuntos de datos y obtener información relevante sobre esos datos. Es común en informes y análisis de datos.

Ejemplo 1 :

Supongamos que tenemos una tabla de "Pedidos" que almacena información sobre pedidos de productos. Queremos saber cuántos pedidos se realizaron por cada cliente:

```
SELECT ClienteID, COUNT(*) AS TotalDePedidos
FROM Pedidos
GROUP BY ClienteID;
```

En este ejemplo:

• Utilizamos GROUP BY ClienteID para agrupar los pedidos por el ID del cliente.
• Luego, utilizamos la función COUNT(*) para contar el número de pedidos en cada grupo de cliente.
• Obtenemos un resultado que muestra el ID del cliente y la cantidad total de pedidos que ese cliente ha realizado.

Este es solo un ejemplo básico de agrupación. También se pueden realizar otras operaciones de agregación, como SUM para calcular el total de ventas por cliente o AVG para obtener el promedio de valores en cada grupo. La agrupación es una herramienta poderosa en SQL para resumir y analizar datos en bases de datos grandes y complejas.

Ejemplo 2 de Agrupación:

Supongamos que tenemos una base de datos de una tienda en línea con dos tablas: "Pedidos" y "DetallesDePedido". Queremos obtener un informe de ventas mensuales que muestre la cantidad de productos vendidos y los ingresos por mes. Aquí está la consulta SQL:

```
SELECT
YEAR(Pedidos.Fecha) AS Año,
MONTH(Pedidos.Fecha) AS Mes,
COUNT(DetallesDePedido.ProductoID) AS CantidadVendida,
SUM(DetallesDePedido.Precio * DetallesDePedido.Cantidad) AS Ingresos
FROM Pedidos
INNER JOIN DetallesDePedido ON Pedidos.ID = DetallesDePedido.PedidoID
GROUP BY YEAR(Pedidos.Fecha), MONTH(Pedidos.Fecha)
ORDER BY Año, Mes;
```

Explicación de la Consulta:

• **YEAR(Pedidos.Fecha) AS Año, MONTH(Pedidos.Fecha) AS Mes:** Estas expresiones extraen el año y el mes de la fecha de cada pedido. Estos valores se renombran como "Año" y "Mes" para facilitar su referencia.
• **COUNT(DetallesDePedido.ProductoID) AS CantidadVendida:** Esta parte de la consulta cuenta la cantidad de registros en la tabla "DetallesDePedido" para cada grupo (mes y año), lo que representa la cantidad de productos vendidos en ese período.
• **SUM(DetallesDePedido.Precio * DetallesDePedido.Cantidad) AS Ingresos**: Esta parte calcula los ingresos totales para cada grupo sumando el producto del precio y la cantidad de productos vendidos en cada registro.
• **FROM Pedidos INNER JOIN DetallesDePedido ON Pedidos.ID = DetallesDePedido.PedidoID:** Establece la relación entre las tablas "Pedidos" y "DetallesDePedido" utilizando una unión (JOIN) basada en la columna "PedidoID".
• **GROUP BY YEAR(Pedidos.Fecha), MONTH(Pedidos.Fecha)**: Agrupa los resultados por año y mes, lo que significa que los cálculos de cantidad vendida e ingresos se realizan para cada combinación única de año y mes.
• **ORDER BY Año, Mes**: Ordena los resultados en orden ascendente por año y mes para obtener un informe ordenado cronológicamente.

Esta consulta proporcionará un informe de ventas mensuales que muestra la cantidad de productos vendidos y los ingresos totales por mes y año. Es un ejemplo más completo de cómo utilizar la agrupación en SQL para resumir y analizar datos en bases de datos relacionales.

LAS VISTAS EN BASES DE DATOS SQL

Son **objetos virtuales que representan conjuntos de datos seleccionados de una o más tablas.** Permiten **simplificar la complejidad de las consultas SQL** al proporcionar una vista lógica de los datos. Estas vistas son útiles porque permiten a los usuarios y aplicaciones acceder a un conjunto de datos específico o realizar operaciones complejas de consulta sin necesidad de acceder directamente a las tablas subyacentes.

Concepto de Vistas en Bases de Datos SQL:

• Una vista es una consulta SQL almacenada que se comporta como una tabla virtual.
• Proporciona una forma personalizada de ver datos almacenados en una o más tablas.
• Las vistas pueden ocultar detalles complejos y datos confidenciales.
• Se utilizan para simplificar consultas frecuentes o para restringir el acceso a ciertos datos.
• Las vistas no almacenan datos físicamente; simplemente almacenan la definición de la consulta.
• Permiten a los usuarios y aplicaciones acceder a datos de manera más conveniente y segura, ya que pueden ocultar detalles de la estructura de la base de datos.
• Se utilizan para simplificar consultas complejas, aplicar políticas de seguridad y proporcionar vistas personalizadas de los datos.

Ejemplo: Creación de una Vista Simple:

Supongamos que tenemos una tabla "Empleados" con información sobre los empleados de una empresa, y queremos crear una vista que muestre solo los nombres y salarios de los empleados. Aquí está cómo crear y utilizar esta vista:

```
-- Crear la vista
CREATE VIEW VistaEmpleados AS
SELECT Nombre, Salario
FROM Empleados;
-- Utilizar la vista
SELECT * FROM VistaEmpleados;
```

• En este ejemplo, creamos una vista llamada "VistaEmpleados" que selecciona los nombres y salarios de la tabla "Empleados".
• Luego, podemos utilizar la vista como si fuera una tabla real para recuperar los datos necesarios.

Ejemplo 2: Vista con Uniones y Filtrado:

Supongamos que tenemos dos tablas, "Clientes" y "Pedidos", y queremos crear una vista que muestre la información del cliente y sus pedidos. Aquí está cómo hacerlo:

```
-- Crear la vista
CREATE VIEW VistaClientesPedidos AS
SELECT C.ClienteID, C.Nombre AS NombreCliente, P.PedidoID, P.FechaPedido
FROM Clientes C
JOIN Pedidos P ON C.ClienteID = P.ClienteID;
-- Utilizar la vista
SELECT * FROM VistaClientesPedidos;
```

En este ejemplo, creamos una vista llamada "**VistaClientesPedidos**" que **combina datos de las tablas "Clientes" y "Pedidos"** mediante una unión.

• La vista muestra información del cliente (ID y nombre) y detalles de los pedidos (ID y fecha).

• Podemos utilizar esta vista para consultar fácilmente la información del cliente y sus pedidos sin escribir consultas complicadas de unión.

Ejemplo 3: Vista con Agregación:

Supongamos que queremos crear una vista que muestre el número de productos en stock en una tienda en línea. Aquí está cómo hacerlo:

```
-- Crear la vista
CREATE VIEW VistaInventario AS
SELECT Categoria,
COUNT(*) AS CantidadEnStock
FROM Productos
WHERE EnStock = 1
GROUP BY Categoria;
-- Utilizar la vista
SELECT * FROM VistaInventario;
```

• En este ejemplo, creamos una vista llamada "**VistaInventario**" que muestra la cantidad de productos en stock por categoría.

• Utilizamos la función de agregación **COUNT** para contar los productos en stock en cada categoría.

• La vista filtra los productos que están en stock (EnStock = 1) y agrupa los resultados por categoría.

Estos ejemplos ilustran cómo crear y utilizar vistas en bases de datos SQL para simplificar consultas y proporcionar una vista personalizada de los datos. Las vistas son útiles para mejorar la modularidad y seguridad de las bases de datos, así como para facilitar el acceso a los datos requeridos.

Crear un Archivo PHP para Consultar y Mostrar la Vista

Ahora, crea un archivo PHP que consulte esta vista y muestre los resultados en una página web:

```
<?php
// Conexión a la base de datos
$servername = "tu_servidor";
$username = "tu_usuario";
$password = "tu_contraseña";
$dbname = "db_news";
$conn = new mysqli($servername, $username, $password, $dbname);
// Verificar la conexión
if ($conn->connect_error) {
die("Error de conexión: " . $conn->connect_error);
}
// Consulta SQL para seleccionar datos desde la vista
$sql = "SELECT * FROM VistaNoticiasCategorias";
$result = $conn->query($sql);
// Verificar si hay resultados
if ($result->num_rows > 0) {
// Mostrar los resultados en una página web
echo "<html><head><title>Noticias y Categorías</title></head><body>";
echo "<h1>Noticias y Categorías</h1>";
echo "<table><tr><th>Título</th><th>Fecha</th><th>Categoría</th></tr>";
while ($row = $result->fetch_assoc()) {
echo "<tr><td>" . $row["tituloNoticia"] . "</td><td>" . $row["fechaNoticia"] .
"</td><td>" . $row["nombreCategoria"] . "</td></tr>";
}
echo "</table>";
echo "</body></html>";
} else {
echo "No se encontraron resultados.";
}
// Cerrar la conexión
$conn->close();
?>
```

Este código PHP consulta la vista "VistaNoticiasCategorias" y muestra los resultados en una tabla en una página web.

LAS FUNCIONES AVANZADAS EN BASES DE DATOS SQL

Son operaciones que **permiten realizar cálculos complejos,** manipulación de cadenas de texto, manejo de fechas y otros procesamientos sofisticados en los datos almacenados en la base de datos.

Aquí tienes un resumen y varios ejemplos de funciones avanzadas en SQL. ejemplos ilustran cómo utilizar funciones avanzadas en SQL para realizar cálculos complejos y manipulaciones de datos en una base de datos.

Las funciones avanzadas son fundamentales para realizar análisis de datos y obtener información valiosa de los registros almacenados.

Funciones de Agregación:

• **SUM** (columna): Calcula la suma de los valores en una columna.
• **AVG** (columna): Calcula el promedio de los valores en una columna.
• **MAX** (columna): Devuelve el valor máximo en una columna.
• **MIN** (columna): Devuelve el valor mínimo en una columna.

Ejemplo de Función de Agregación (SUM):

```
SELECT Categoria, SUM(Precio) AS TotalVentas
FROM Productos
GROUP BY Categoria;
```

Esta consulta calcula el total de ventas por categoría de productos.

Funciones Matemáticas:

• **ABS**(valor): Devuelve el valor absoluto de un número.
• **ROUND**(valor, decimales): Redondea un número al número especificado de decimales.
• **SQRT**(valor): Calcula la raíz cuadrada de un número.

Funciones de Manipulación de Cadenas de Texto:

• **CONCAT**(cadena1, cadena2): Concatena dos cadenas de texto.
• **SUBSTRING**(cadena, inicio, longitud): Extrae una subcadena de una cadena.
• **UPPER**(cadena): Convierte una cadena a mayúsculas.
• **LOWER**(cadena): Convierte una cadena a minúsculas.

Ejemplo de Función de Manipulación de Cadenas (CONCAT):

```
SELECT CONCAT(FirstName, ' ', LastName) AS NombreCompleto
FROM Empleados;
```

Esta consulta concatena los nombres y apellidos de los empleados en una columna "NombreCompleto".

Funciones de Fecha y Hora:

• **NOW()**: Devuelve la fecha y hora actuales.
• **DATEADD(intervalo, cantidad, fecha):** Agrega una cantidad de intervalo (días, meses, años, etc.) a una fecha.
• **DATEDIFF**(intervalo, fecha1, fecha2): Calcula la diferencia entre dos fechas
en el intervalo especificado.

Ejemplo de Función de Fecha (NOW):

```
SELECT PedidoID, FechaEntrega, DATEDIFF(DAY, FechaPedido, FechaEntrega)
AS DiasParaEntrega
FROM Pedidos;
```

Esta consulta calcula la cantidad de días que tomó entregar cada pedido.

Funciones de Control de Flujo:

• CASE WHEN condición THEN resultado ELSE otroResultado END:
Realiza una evaluación condicional en una consulta SQL.
• COALESCE(valor1, valor2, ...): Devuelve el primer valor no nulo de una lista de valores.

Ejemplo de Función de Control de Flujo (CASE):

```
SELECT Nombre,
CASE WHEN Edad < 18 THEN 'Menor de Edad'
WHEN Edad >= 18 AND Edad < 65 THEN 'Adulto'
ELSE 'Mayor de Edad' END AS CategoriaEdad
FROM Personas;
```

Esta consulta clasifica a las personas en categorías de edad basadas en su edad.