Fundamento de Base de Datos

TESH
HUIXQUILUCAN

4.- Normalización de base de datos

La normalización en las bases de datos es un proceso de diseño y organización de las tablas y sus relaciones para reducir la redundancia y dependencia de los datos. El objetivo principal es asegurar que la base de datos esté estructurada de manera eficiente, minimizando la duplicación de información y manteniendo la integridad de los datos. Este proceso implica aplicar una serie de reglas conocidas como formas normales.

Objetivos de la Normalización

I. **Eliminar la Redundancia de Datos**: Reducir la duplicación innecesaria de datos en múltiples tablas.
II. **Asegurar la Integridad de los Datos**: Mantener la consistencia y precisión de los datos.
III. **Optimizar las Consultas y el Rendimiento**: Facilitar la ejecución de consultas y mejorar la eficiencia del sistema.
IV. **Simplificar el Mantenimiento y la Modificación**: Hacer más fáciles y seguras las actualizaciones y modificaciones de los datos.

4.1 Conceptos básicos

Relación: En el contexto de bases de datos, una relación es una tabla que contiene datos. Cada relación tiene un nombre único y está compuesta por columnas (atributos) y filas (tuplas).

I. **Atributo**: Es una columna en una tabla. Representa una propiedad o característica de la entidad que se está modelando.

II. **Tupla**: Es una fila en una tabla. Representa un registro único de datos.

III. **Clave Primaria**: Un atributo o conjunto de atributos que identifica de manera única a cada tupla en una tabla. No puede contener valores nulos.

IV. **Clave Foránea**: Un atributo o conjunto de atributos en una tabla que se refiere a la clave primaria de otra tabla. Se utiliza para establecer y reforzar la relación entre las tablas.

Formas Normales

Las formas normales son reglas que una base de datos debe seguir para estar correctamente normalizada. Las principales formas normales son:

Primera Forma Normal (1NF)

- **Definición**: Una tabla está en 1NF si todos los atributos contienen valores atómicos (indivisibles) y cada valor de atributo es único en cada tupla.
- **Requisitos**:
  - Cada columna debe contener solo un valor.
  - Todos los valores de una columna deben ser del mismo tipo de datos.
  - Cada columna debe tener un nombre único.

Segunda Forma Normal (2NF)

- **Definición**: Una tabla está en 2NF si está en 1NF y todos los atributos no clave dependen completamente de la clave primaria.
- **Requisitos**:
  - Eliminar las dependencias parciales. Esto significa que no debe haber atributos que dependan solo de una parte de una clave primaria compuesta.

Tercera Forma Normal (3NF)

- **Definición**: Una tabla está en 3NF si está en 2NF y no tiene dependencias transitivas.
- **Requisitos**:
  - Eliminar las dependencias transitivas. Esto significa que un atributo no clave no debe depender de otro atributo no clave.

Normalización Avanzada

Para bases de datos más complejas, se pueden aplicar formas normales adicionales:

Forma Normal de Boyce-Codd (BCNF)

- **Definición**: Una tabla está en BCNF si está en 3NF y para cada dependencia funcional $X \rightarrow Y$, X es una superclave.
- **Requisitos**:
  - Toda dependencia funcional no trivial debe tener una superclave como determinante.

Cuarta Forma Normal (4NF)

- **Definición**: Una tabla está en 4NF si está en BCNF y no tiene dependencias multivaluadas no triviales.
- **Requisitos**:
    - Eliminar las dependencias multivaluadas. Esto significa que una tabla no debe tener conjuntos de atributos independientes que dependan de la clave primaria.

Quinta Forma Normal (5NF)

- **Definición**: Una tabla está en 5NF si está en 4NF y no tiene dependencias de unión.
- **Requisitos**:
    - Eliminar las dependencias de unión. Esto significa que la tabla no debe ser descomponible en dos o más tablas que luego se pueden unir para reconstruir la tabla original sin pérdida de información.

Beneficios de la Normalización

- **Reducción de Redundancia**: Minimiza la duplicación de datos y reduce el tamaño de la base de datos.
- **Mejora de la Integridad de los Datos**: Asegura que los datos sean consistentes y exactos.
- **Facilidad de Mantenimiento**: Facilita la actualización y modificación de los datos.
- **Mejora del Rendimiento**: Optimiza las consultas al reducir la cantidad de datos que se deben procesar.

4.2 primera forma normal

La Primera Forma Normal (1NF) es la base del proceso de normalización en bases de datos y asegura que una tabla cumpla con ciertos criterios fundamentales. El objetivo principal de la 1NF es garantizar que los datos en una tabla estén organizados de manera que cada valor sea atómico y que no haya grupos repetitivos. A continuación, se describen los conceptos y requisitos de la 1NF:

Valores Atómicos:

- Cada campo (o celda) en una tabla debe contener un solo valor indivisible. No deben existir conjuntos de valores o listas en una celda.

ID	Nombre	Cursos
1	Juan	Matematicas, fisica
2	Maria	Quimica, Biologia

ID	Nombre	Cursos
1	Juan	Matematicas
1	Juan	Fisica
2	Maria	Quimica
3	Maria	Biologia

Dominio de Atributos:

- Todos los valores de una columna deben ser del mismo tipo de datos.

Columnas Uniquas:

- Cada columna en la tabla debe tener un nombre único.

Filas Uniquas:

- Cada fila en la tabla debe ser única y distinguible. Esto generalmente se logra mediante el uso de una clave primaria.

4.3 Dependencias funcionales y transitivas

Dependencias Funcionales

Una dependencia funcional describe una relación entre dos conjuntos de atributos en una relación (tabla) de una base de datos. Se dice que un conjunto de atributos AA determina funcionalmente a otro conjunto de atributos BB (denotado como $A \rightarrow B$A→B) si para cualquier par de tuplas (filas) en la relación, si ambas tuplas coinciden en los valores de los atributos de AA, también deben coincidir en los valores de los atributos de BB.

Formalmente:

- Sea RR una relación con atributos AA y BB.

- $A \rightarrow B$A→B si, para cualquier tupla t1t1 y t2t2 en RR, si $t1[A]=t2[A]$t1[A]=t2[A], entonces $t1[B]=t2[B]$t1[B]=t2[B].

Ejemplo: Consideremos una tabla Estudiantes con los atributos ID_Alumno, Nombre y Fecha_Nacimiento. Si cada ID_Alumno es único para cada estudiante, entonces podemos decir que:

- $ID_Alumno \rightarrow Nombre, Fecha_Nacimiento$ID_Alumno→Nombre,Fecha_Nacimiento

Esto significa que el ID_Alumno determina funcionalmente los atributos Nombre y Fecha_Nacimiento.

Dependencias Transitivas

Una dependencia transitiva es un tipo específico de dependencia funcional que ocurre cuando un atributo está funcionalmente determinado por otro atributo, el cual a su vez está funcionalmente determinado por un tercer atributo. En otras palabras, si $A \rightarrow B$ y $B \rightarrow C$, entonces $A \rightarrow C$ es una dependencia transitiva.

Formalmente:

- Sea A, B y C conjuntos de atributos en una relación R.
- Si $A \rightarrow B$ y $B \rightarrow C$(donde A, B y C son conjuntos de atributos y A no es un superconjunto de B), entonces $A \rightarrow C$ es una dependencia transitiva.

Ejemplo: Consideremos una tabla Pedidos con los atributos ID_Pedido, ID_Cliente y Nombre_Cliente. Supongamos las siguientes dependencias funcionales:

- $ID\_Pedido \rightarrow ID\_Cliente$
- $ID\_Cliente \rightarrow Nombre\_Cliente$

Podemos concluir que:

- $ID\_Pedido \rightarrow Nombre\_Cliente$

Esta es una dependencia transitiva porque la determinación de Nombre_Cliente por ID_Pedido pasa a través de ID_Cliente.

Importancia en la Normalización

Identificar y gestionar adecuadamente las dependencias funcionales y transitivas es esencial para el proceso de normalización de bases de datos, cuyo objetivo es reducir la redundancia y mejorar la integridad de los datos. En particular:

- **Primera Forma Normal (1NF):** Elimina los grupos repetitivos asegurando que cada campo contenga un solo valor.
- **Segunda Forma Normal (2NF):** Elimina las dependencias parciales asegurando que todos los atributos no clave dependan funcionalmente de toda la clave primaria.
- **Tercera Forma Normal (3NF):** Elimina las dependencias transitivas asegurando que los atributos no clave dependan directamente de la clave primaria y no de otros atributos no clave.

Ejemplo de Normalización:

Supongamos una tabla inicial con las siguientes dependencias:

Pedidos (ID_Pedido, ID_Cliente, Nombre_Cliente, Fecha_Pedido)

- Dependencias:
  - $ID\_Pedido \rightarrow ID\_Cliente, Fecha\_Pedido$
  - $ID\_Cliente \rightarrow Nombre\_Cliente$

Para normalizar esta tabla a 3NF:

I. **Identificar y eliminar dependencias parciales (2NF):**
   - En este caso, no hay dependencias parciales ya que ID_Pedido es la clave primaria completa.

II. **Identificar y eliminar dependencias transitivas (3NF):**
   - ID_Cliente \rightarrow Nombre_Cliente es una dependencia transitiva a través de ID_Pedido.

Separar en dos tablas:

Pedidos (ID_Pedido, ID_Cliente, Fecha_Pedido)
Clientes (ID_Cliente, Nombre_Cliente)

Ahora, las tablas están en 3NF ya que se han eliminado las dependencias transitivas.

4.4 Segunda forma normales

La Segunda Forma Normal (2NF) es un paso importante en el proceso de normalización de bases de datos. Su objetivo principal es eliminar las dependencias funcionales parciales, es decir, asegurarse de que todos los atributos no clave dependan completamente de toda la clave primaria y no solo de una parte de ella.

Requisitos para la Segunda Forma Normal (2NF)

Una tabla está en 2NF si cumple con los siguientes criterios:

I. **Debe estar en Primera Forma Normal (1NF)**: Todos los valores en las columnas deben ser atómicos, y la tabla no debe tener duplicación de datos.
II. **Eliminación de dependencias funcionales parciales**: Todos los atributos no clave deben depender completamente de la clave primaria.

Dependencia Funcional Parcial

Una dependencia funcional parcial ocurre cuando un atributo no clave depende solo de una parte de una clave primaria compuesta, en lugar de depender de toda la clave primaria.

4.5 Tercera Forma normal

La Tercera Forma Normal (3NF) es un nivel de normalización en bases de datos cuyo objetivo principal es eliminar las dependencias transitivas, asegurando que todos los atributos no clave dependan únicamente de la clave primaria. Este paso es crucial para mejorar la integridad y la eficiencia de una base de datos.

Requisitos para la Tercera Forma Normal (3NF)

Una tabla está en 3NF si cumple con los siguientes criterios:

I. **Debe estar en Segunda Forma Normal (2NF)**: Esto significa que la tabla ya ha eliminado las dependencias funcionales parciales.
II. **Eliminación de dependencias transitivas**: Ningún atributo no clave debe depender transitivamente de la clave primaria.

Dependencia Transitiva

Una dependencia transitiva ocurre cuando un atributo no clave depende de otro atributo no clave a través de una relación indirecta. Formalmente, si $A \rightarrow B$A→B y $B \rightarrow C$B→C, entonces $A \rightarrow C$A→C es una dependencia transitiva.

4.6 Forma normal Boyce-codd.

La Forma Normal de Boyce-Codd (BCNF) es una versión más estricta de la Tercera Forma Normal (3NF). BCNF elimina ciertas anomalías de actualización que la 3NF no puede manejar adecuadamente. Una tabla está en BCNF si cumple con las siguientes condiciones:

Requisitos para la Forma Normal de Boyce-Codd (BCNF)

I. **Debe estar en Tercera Forma Normal (3NF)**: Esto significa que ya se han eliminado las dependencias transitivas.
II. **Para cada dependencia funcional $X \rightarrow Y$X→Y en la tabla, XX debe ser una superclave**: Es decir, XX debe ser una clave candidata (un conjunto de atributos que puede actuar como una clave primaria).

Dependencia Funcional y Superclave

I. **Dependencia Funcional**: Como ya se explicó, si AA y BB son conjuntos de atributos, $A \rightarrow B$A→B significa que AA determina BB.
II. **Superclave**: Un conjunto de uno o más atributos que, tomado colectivamente, permite identificar de manera única una tupla en una tabla.

4.7 Otras formas normales

Además de las primeras tres formas normales (1NF, 2NF, 3NF) y la Forma Normal de Boyce-Codd (BCNF), existen otras formas normales más avanzadas que se utilizan para abordar problemas más complejos de redundancia y dependencia en bases de datos. Estas formas incluyen la Cuarta Forma Normal (4NF) y la Quinta Forma Normal (5NF). A continuación, se describen estos niveles de normalización y sus requisitos:

Cuarta Forma Normal (4NF)

La Cuarta Forma Normal aborda el problema de las dependencias multivaluadas. Una dependencia multivaluada ocurre cuando un atributo puede tener múltiples valores independientes de otros atributos no clave.

Requisitos para la Cuarta Forma Normal (4NF)

I. **Debe estar en BCNF**: La tabla debe cumplir con los criterios de la Forma Normal de Boyce-Codd.
II. **No debe haber dependencias multivaluadas no triviales**: Una tabla está en 4NF si, para cada dependencia multivaluada $A \rightarrow B$A→B, AA es una superclave.

Quinta Forma Normal (5NF)

La Quinta Forma Normal, también conocida como la Forma Normal de Proyección-Unión (PJNF), aborda el problema de las dependencias de unión. Una dependencia de unión ocurre cuando una tabla puede ser descompuesta en tablas más pequeñas sin pérdida de información.

Requisitos para la Quinta Forma Normal (5NF)

1. **Debe estar en 4NF**: La tabla debe cumplir con los criterios de la Cuarta Forma Normal.
2. **No debe haber dependencias de unión no triviales**: Una tabla está en 5NF si cada unión no trivial es implícitamente derivada de la clave.

Normalización de una tabla de ejemplo

Estos pasos demuestran el proceso de normalización de una tabla de alumnos ficticia.

1. Tabla sin normalizar:

Expandir tabla

Nº alumno	Tutor	Despacho-Tut	Clase1	Clase2	Clase3
1022	García	412	101-07	143-01	159-02
4123	Díaz	216	101-07	143-01	179-04

2. Primera forma normal: sin grupos repetidos

Las tablas sólo deben tener dos dimensiones. Puesto que un alumno tiene varias clases, estas clases deben aparecer en una tabla independiente. Los campos Clase1, Clase2 y Clase3 de los registros anteriores son indicativos de un problema de diseño.

Las hojas de cálculo suelen usar la tercera dimensión, pero las tablas no deberían hacerlo. Otra forma de considerar este problema es con una relación de uno a varios y poner el lado de uno y el lado de varios en la misma tabla. En su lugar, cree otra tabla en la primera forma normal eliminando el grupo repetido (N.º clase), según se muestra a continuación:

Expandir tabla

Nº alumno	Tutor	Despacho-Tut	Nº clase
1022	García	412	101-07
1022	García	412	143-01
1022	García	412	159-02
4123	Díaz	216	101-07
4123	Díaz	216	143-01
4123	Díaz	216	179-04

3. Segunda forma normal: eliminar los datos redundantes

Observe los diversos valores de **N.º clase** para cada valor de **N.º** alumno en la tabla anterior. El N.º clase no depende funcionalmente de N.º alumno (la clave principal), de modo que la relación no cumple la segunda forma normal.

Las tablas siguientes demuestran la segunda forma normal:

Alumnos:

Expandir tabla

Nº alumno	Tutor	Despacho-Tut
1022	García	412
4123	Díaz	216

Registro:

Expandir tabla

Nº alumno	Nº clase
1022	101-07
1022	143-01
1022	159-02
4123	101-07
4123	143-01
4123	179-04

4. Tercera forma normal: eliminar los datos que no dependen de la clave

En el último ejemplo, Despacho-Tut (el número de despacho del tutor) es funcionalmente dependiente del atributo Tutor. La solución es pasar ese atributo de la tabla Alumnos a la tabla Personal, según se muestra a continuación:

Alumnos:

Expandir tabla

Nº alumno	Tutor
1022	García
4123	Díaz

Personal:

Expandir tabla

Nombre	Sala	Dept.
García	412	42
Díaz	216	42

5.- Algebra relacional

El álgebra relacional es un conjunto de operaciones matemáticas aplicables a bases de datos relacionales para manipular y transformar datos. Este sistema permite definir consultas de manera estructurada y eficiente, facilitando la realización de operaciones como selecciones, proyecciones, uniones, intersecciones, diferencias, y productos cartesianos, entre otras. A continuación, se detallan algunos de los conceptos clave y operaciones fundamentales del álgebra relacional:

Operaciones Básicas

- **Selección (σ)**: Filtra registros de una relación basándose en una condición específica.
- **Proyección (Π)**: Selecciona ciertos atributos de una relación, eliminando duplicados y posiblemente reduciendo el número total de registros.
- **Unión (∪)**: Combina registros de dos relaciones que tienen el mismo esquema, manteniendo solo aquellos registros únicos.
- **Intersección (∩)**: Combina registros de dos relaciones que tienen el mismo esquema, manteniendo solo aquellos registros que aparecen en ambas relaciones.
- **Diferencia (-)**: Elimina registros de una relación que también están presentes en otra relación con el mismo esquema.
- **Producto Cartesiano (×)**: Crea una nueva relación combinando todos los registros de dos relaciones dadas, sin tener en cuenta si los registros correspondientes son relevantes o no.

Operaciones Derivadas

- **Join**: Combina registros de dos relaciones basándose en una condición de igualdad entre atributos clave. Es equivalente a una selección sobre el producto cartesiano de las relaciones.
- **División**: Similar al join, pero se utiliza para encontrar registros en una relación que están asociados con todos los registros de otra relación mediante una condición de igualdad.

5.1 operaciones funcionales del álgebra relacional

La álgebra relacional es un lenguaje formal utilizado para consultar y manipular bases de datos relacionales. Fue introducida por Edgar F. Codd y proporciona una serie de operaciones para trabajar con datos almacenados en tablas (relaciones). Las principales operaciones de la álgebra relacional son:

1. **Selección (σ)**: Filtra filas de una tabla que cumplen con una condición específica.
    - Notación: σcondicio´n(R)
    - Ejemplo: σedad>30(Empleados) selecciona los empleados con edad mayor a 30.
2. **Proyección (π)**: Extrae columnas específicas de una tabla.
    - Notación: πcolumnas(R)
    - Ejemplo: ππnombre,salario(Empleados),salario(Empleados) selecciona las columnas de nombre y salario de la tabla Empleados.
3. **Unión (∪)**: Combina las filas de dos tablas que tienen el mismo esquema.
    - Notación: R∪S
    - Ejemplo: Empleados ∪ DirectoresEmpleados combina las filas de las tablas Empleados y Directores.
4. **Intersección (∩)**: Devuelve las filas que son comunes a dos tablas.
    - Notación: R∩S
    - Ejemplo: Empleados ∩ Directores devuelve las filas que están tanto en Empleados como en Directores.
5. **Diferencia (\)**: Devuelve las filas de una tabla que no están en otra tabla.
    - Notación: R\S
    - Ejemplo: Empleados\Directores devuelve las filas de Empleados que no están en Directores.
6. **Producto Cartesiano (×)**: Combina todas las filas de una tabla con todas las filas de otra tabla.

- Notación: R×S
- Ejemplo: Empleados×Departamentos

  devuelve todas las combinaciones posibles de filas de Empleados y Departamentos.

7. **Renombrar (ρ)**: Cambia el nombre de una tabla o de sus columnas.
   - Notación: ρnuevo_nombre(R) o ρ(A1,A2,…,An)(R)
   - Ejemplo: ρE(Empleados) renombra la tabla Empleados a E.
8. **Join (⋈)**: Combina filas de dos tablas basándose en una condición.
   - Notación: R⋈condicio´nS
   - Ejemplo: Empleados⋈Empleados.id_departamento = Departamentos.id Departamentos

     combina las filas de Empleados y Departamentos donde los IDs de departamento coinciden.

Ejemplo en SQL

1.- Crar base de datos

```
CREATE DATABASE Empresa;
USE Empresa;

CREATE TABLE Empleados (
    id INT PRIMARY KEY,
    nombre VARCHAR(50),
    id_departamento INT,
    salario DECIMAL(10, 2)
);

CREATE TABLE Departamentos (
    id INT PRIMARY KEY,
    nombre VARCHAR(50)
);

INSERT INTO Empleados (id, nombre, id_departamento, salario) VALUES
(1, 'Juan', 10, 3000.00),
(2, 'Ana', 20, 3500.00),
(3, 'Luis', 10, 4000.00);

INSERT INTO Departamentos (id, nombre) VALUES
(10, 'Recursos Humanos'),
(20, 'IT');
```

2.- Realizar consultas

2.1.- Seleccion y proyeccion

Queremos encontrar los nombres de los empleados del departamento de IT.

```
SELECT nombre
FROM Empleados
WHERE id_departamento = 20;
```

2.2.- Join

Queremos encontrar los nombres de los empleados junto con los nombres de sus departamentos.

```
SELECT Empleados.nombre AS nombre_empleado, Departamentos.nombre AS nombre_departamento
FROM Empleados
JOIN Departamentos ON Empleados.id_departamento = Departamentos.id;
```

2.3.- Unión

Si tuviéramos otra tabla Directores con una estructura similar a Empleados, podríamos unir las dos tablas. Primero, creamos la tabla Directores y luego realizamos la unión.

```
CREATE TABLE Directores (
    id INT PRIMARY KEY,
    nombre VARCHAR(50),
    id_departamento INT,
    salario DECIMAL(10, 2)
);

INSERT INTO Directores (id, nombre, id_departamento, salario) VALUES
(4, 'Maria', 10, 5000.00),
(5, 'Carlos', 20, 5500.00);

SELECT id, nombre, id_departamento, salario
FROM Empleados
UNION
SELECT id, nombre, id_departamento, salario
FROM Directores;
```

5.2 Álgebra Relacional Extendida

La álgebra relacional extendida es una extensión de la álgebra relacional clásica utilizada en bases de datos relacionales. Esta extensión incluye operaciones adicionales que no están presentes en la álgebra relacional estándar, lo que permite una mayor flexibilidad y potencia

en las consultas. A continuación, se describen algunas de las operaciones extendidas comunes en la álgebra relacional:

Agrupamiento y Agregación (GROUP BY y funciones de agregación).

Permite agrupar tuplas en conjuntos y aplicar funciones de agregación como SUM, AVG, COUNT, MIN, y MAX.

- **Ejemplo**: Calcular el salario promedio por departamento.

π departamento,AVG(salario)(γdepartamento,AVG(salario)(Empleados))

Unión Generalizada (OUTER JOIN)

Incluye LEFT OUTER JOIN, RIGHT OUTER JOIN y FULL OUTER JOIN, que permiten unir tablas manteniendo las tuplas de una o ambas tablas que no tienen coincidencias.

- **Ejemplo**: Unir empleados con sus departamentos, incluyendo empleados sin departamento asignado.

Empleados LEFT OUTER JOIN Departamentos ON Empleados.depto_id=Departamentos.id

División (DIVISION)

Utilizada para encontrar tuplas en una relación que están relacionadas con todas las tuplas de otra relación.

- **Ejemplo**: Encontrar los empleados que han trabajado en todos los proyectos.

Empleados÷Proyectos

Operaciones Recursivas

Permite definir consultas recursivas, útiles para trabajar con jerarquías y grafos.

- **Ejemplo**: Encontrar todos los subordinados directos e indirectos de un gerente.

WITH RECURSIVE Subordinados AS (πgerente,subordinado(Relacion)∪πgerente,subordinado(σSubordinados.subordinado=Relacion.gerente(Relacion)))

5. Proyección Extendida

Permite la creación de nuevas columnas en el resultado de una consulta.

- **Ejemplo**: Calcular el salario anual de los empleados.

πnombre,salario_anual=salario×12(Empleados)

6. Asociación Natural (NATURAL JOIN)

Una variante del JOIN que se basa en todas las columnas con el mismo nombre en ambas relaciones.

- **Ejemplo**: Unir empleados con sus departamentos automáticamente por el campo depto_id.

Empleados NATURAL JOIN Departamentos

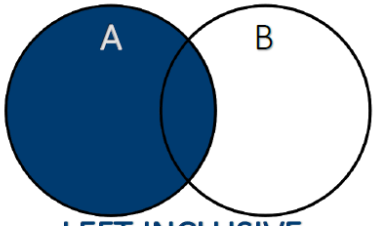
7. Renombramiento de Atributos (RENAME)

Permite cambiar el nombre de las columnas en una relación para evitar conflictos o hacer el resultado más claro.

- **Ejemplo**: Renombrar la columna salario a salario_mensual.

ρsalario_mensual/salario(Empleados)

Estas operaciones extendidas permiten realizar consultas más complejas y detalladas, mejorando la capacidad de los sistemas de bases de datos para manejar y extraer información valiosa de los datos almacenados.



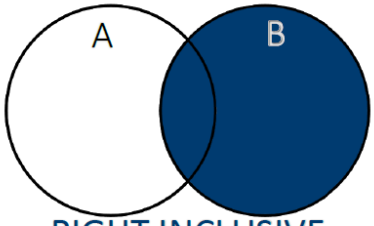
LEFT INCLUSIVE

SQL JOINS

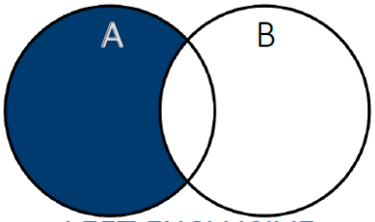
LEFT INCLUSIVE	RIGHT INCLUSIVE
SELECT *[Select List]* FROM TableA A LEFT OUTER JOIN TableB B ON A.Key= B.Key	SELECT *[Select List]* FROM TableA A RIGHT OUTER JOIN TableB B ON A.Key= B.Key
LEFT EXCLUSIVE	**RIGHT EXCLUSIVE**
SELECT *[Select List]* FROM TableA A LEFT OUTER JOIN TableB B ON A.Key= B.Key WHERE B.Key IS NULL	SELECT *[Select List]* FROM TableA A LEFT OUTER JOIN TableB B ON A.Key= B.Key WHERE A.Key IS NULL
FULL OUTER INCLUSIVE	**FULL OUTER EXCLUSIVE**
SELECT *[Select List]* FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key	SELECT *[Select List]* FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key WHERE A.Key IS NULL OR B.Key IS NULL

INNER JOIN

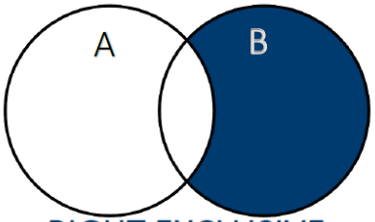
SELECT *[Select List]*
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key



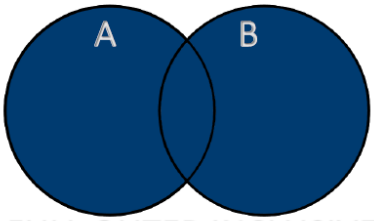
RIGHT INCLUSIVE



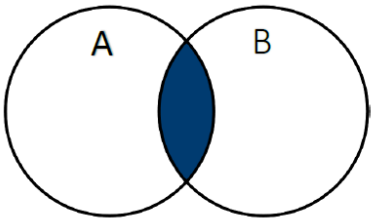
LEFT EXCLUSIVE



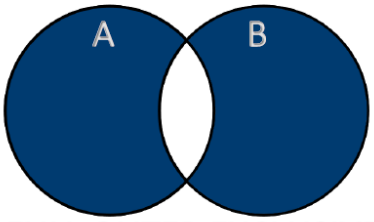
RIGHT EXCLUSIVE



FULL OUTER INCLUSIVE



INNER JOIN



FULL OUTER EXCLUSIVE

Ejemplo en SQL

1.- Creear bd e inserta datos

```
CREATE TABLE Empleados (
    id INT PRIMARY KEY,
    nombre VARCHAR(50),
    salario DECIMAL(10, 2),
    depto_id INT
);

CREATE TABLE Departamentos (
    id INT PRIMARY KEY,
    nombre VARCHAR(50)
);

CREATE TABLE Proyectos (
    id INT PRIMARY KEY,
    nombre VARCHAR(50)
);
```

```
CREATE TABLE Trabaja_en (
    empleado_id INT,
    proyecto_id INT,
    FOREIGN KEY (empleado_id) REFERENCES Empleados(id),
    FOREIGN KEY (proyecto_id) REFERENCES Proyectos(id)
);

INSERT INTO Empleados (id, nombre, salario, depto_id) VALUES
(1, 'Juan', 3000.00, 1),
(2, 'Ana', 3500.00, 1),
(3, 'Luis', 2500.00, 2),
(4, 'Marta', 4000.00, NULL);

INSERT INTO Departamentos (id, nombre) VALUES
(1, 'Recursos Humanos'),
(2, 'Finanzas');

INSERT INTO Proyectos (id, nombre) VALUES
(1, 'Proyecto A'),
(2, 'Proyecto B');

INSERT INTO Trabaja_en (empleado_id, proyecto_id) VALUES
(1, 1),
(2, 1),
(2, 2),
(3, 1);
```

2.- Agrupamiento y Agregación.

2. Calcular el salario promedio por departamento:

```
SELECT depto_id, AVG(salario) AS salario_promedio
FROM Empleados
GROUP BY depto_id;
```

3. Unión Generalizada (LEFT OUTER JOIN)

Unir empleados con sus departamentos, incluyendo empleados sin departamento asignado:

```
SELECT Empleados.nombre, Departamentos.nombre AS departamento
FROM Empleados
LEFT OUTER JOIN Departamentos ON Empleados.depto_id = Departamentos.id;
```

4.- División

Encontrar los empleados que han trabajado en todos los proyectos:

```
SELECT e.nombre
FROM Empleados e
WHERE NOT EXISTS (
    SELECT p.id
    FROM Proyectos p
    WHERE NOT EXISTS (
        SELECT t.proyecto_id
        FROM Trabaja_en t
        WHERE t.empleado_id = e.id AND t.proyecto_id = p.id
    )
);
```

5.- Operaciones Recursivas

Encontrar todos los subordinados directos e indirectos de un gerente (suponiendo una tabla de relación de subordinación):

```
CREATE TABLE Relacion (
    gerente INT,
    subordinado INT
);

INSERT INTO Relacion (gerente, subordinado) VALUES
(1, 2),
(1, 3),
(2, 4),
(3, 5);

WITH RECURSIVE Subordinados AS (
    SELECT gerente, subordinado
    FROM Relacion
    WHERE gerente = 1
    UNION
    SELECT r.gerente, r.subordinado
    FROM Relacion r
    INNER JOIN Subordinados s ON s.subordinado = r.gerente
)
SELECT subordinado
FROM Subordinados;
```

6.- Proyección Extendida

Calcular el salario anual de los empleados:

```
SELECT nombre, salario * 12 AS salario_anual
FROM Empleados;
```

7.- Renombramiento de Atributos

Renombrar la columna salario a salario_mensual:

```
SELECT nombre, salario AS salario_mensual
FROM Empleados;
```

6.- Introduccion al lenguaje sql

SQL (Structured Query Language) es un lenguaje de programación utilizado para gestionar y manipular bases de datos relacionales. Es esencial para realizar consultas, insertar, actualizar y eliminar datos, así como para administrar y estructurar esquemas de bases de datos.

Historia de SQL

SQL fue desarrollado en los años 70 por IBM y fue adoptado como estándar por el American National Standards Institute (ANSI) y la International Organization for Standardization (ISO) en los años 80.

Características Principales de SQL

1. **Lenguaje Declarativo**: SQL permite especificar qué datos se requieren sin especificar cómo obtenerlos.
2. **Estandarizado**: SQL es un estándar reconocido internacionalmente, lo que asegura la portabilidad entre diferentes sistemas de bases de datos.
3. **Potente**: Puede manejar grandes volúmenes de datos y realizar consultas complejas.

Componentes de SQL

SQL se compone de varios subconjuntos que se encargan de diferentes tareas dentro de la gestión de bases de datos:

1. **DDL (Data Definition Language)**: Utilizado para definir la estructura de la base de datos.
    - CREATE: Crea una nueva tabla, vista, índice, etc.
    - ALTER: Modifica una tabla existente.
    - DROP: Elimina una tabla, vista, índice, etc.

2. **DML (Data Manipulation Language)**: Utilizado para manipular los datos dentro de las tablas.
    - SELECT: Recupera datos de una o más tablas.
    - INSERT: Inserta datos en una tabla.
    - UPDATE: Actualiza datos existentes en una tabla.
    - DELETE: Elimina datos de una tabla.
3. **DCL (Data Control Language)**: Utilizado para controlar el acceso a los datos.
    - GRANT: Otorga permisos a los usuarios.
    - REVOKE: Revoca permisos a los usuarios.
4. **TCL (Transaction Control Language)**: Utilizado para gestionar transacciones en la base de datos.
    - COMMIT: Guarda las transacciones realizadas.
    - ROLLBACK: Deshace las transacciones no guardadas.
    - SAVEPOINT: Establece puntos de control dentro de una transacción.

6.1 Lenguaje de Definición de Datos (DDL)

El Lenguaje de Definición de Datos (DDL, por sus siglas en inglés) es una parte fundamental de SQL que se utiliza para definir, alterar y eliminar la estructura de los objetos en una base de datos. Los comandos DDL no manipulan datos, sino que se enfocan en la estructura de la base de datos misma, como tablas, índices, vistas y esquemas.

Comandos Principales de DDL

1. **CREATE**: Este comando se utiliza para crear objetos en la base de datos, como tablas, índices y vistas.
2. **ALTER**: Se usa para modificar la estructura de los objetos existentes en la base de datos.
3. **DROP**: Este comando elimina objetos de la base de datos.
4. **TRUNCATE**: Se utiliza para eliminar todos los registros de una tabla sin eliminar la tabla en sí.
5. **COMMENT**: Añade comentarios descriptivos a los objetos en la base de datos.
6. **RENAME**: Cambia el nombre de un objeto existente.

CREATE

Creacion de una tabla

```
CREATE TABLE empleados (
    id INT PRIMARY KEY,
    nombre VARCHAR(50),
    puesto VARCHAR(50),
    salario DECIMAL(10, 2)
);
```

Creación de índice

```
CREATE INDEX idx_empleados_nombre ON empleados (nombre);
```

Crear una vista

```
CREATE VIEW vista_altos_salarios AS
SELECT nombre, salario
FROM empleados
WHERE salario > 70000;
```

ALTER

Añadir columna

```
ALTER TABLE empleados ADD fecha_contratacion DATE;
```

Modificar columnas

```
ALTER TABLE empleados ALTER COLUMN salario DECIMAL(12, 2);
```

Eliminar columnas

```
ALTER TABLE empleados DROP COLUMN fecha_contratacion;
```

Drop

Eliminar una tabla

```
DROP TABLE empleados;
```

Eliminar indice

```
DROP INDEX idx_empleados_nombre;
```

Eliminar vista

```
DROP VIEW vista_altos_salarios;
```

Truncate

Añadir Comentarios a una Tabla

```
COMMENT ON TABLE empleados IS 'Tabla que almacena información sobre los empleados';
```

Añadir Comentarios a una Column

```
COMMENT ON COLUMN empleados.salario IS 'Salario anual del empleado';
```

Rename

renombrar una tabla

```
ALTER TABLE empleados RENAME TO trabajadores;
```

Renombrar una Columna

```
ALTER TABLE empleados RENAME COLUMN nombre TO nombre_completo;
```