Base de Datos I

Herramientas a utilizar

- **Bases de Datos MySql**: Es un sistema de gestión de bases de datos relacional y está considerada como la base datos open source más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web.

- **PhpMyAdmin**: Crea y maneja bases de datos MySQL desde cualquier navegador. Con phpMyAdmin se pueden realizar todas las tareas de administración necesarias en cualquier base de datos: crear y eliminar bases y tablas, gestionarlas, añadir, eliminar o modificar campos, ejecutar secuencias de comandos SQL, etc.

- **MySqlWorkbench**: Es una herramienta que permite modelar diagramas de entidad-relación para bases de datos MySQL. Puede ser utilizada para diseñar el esquema de una base de datos nueva, documentar una ya existente o realizar una migración.

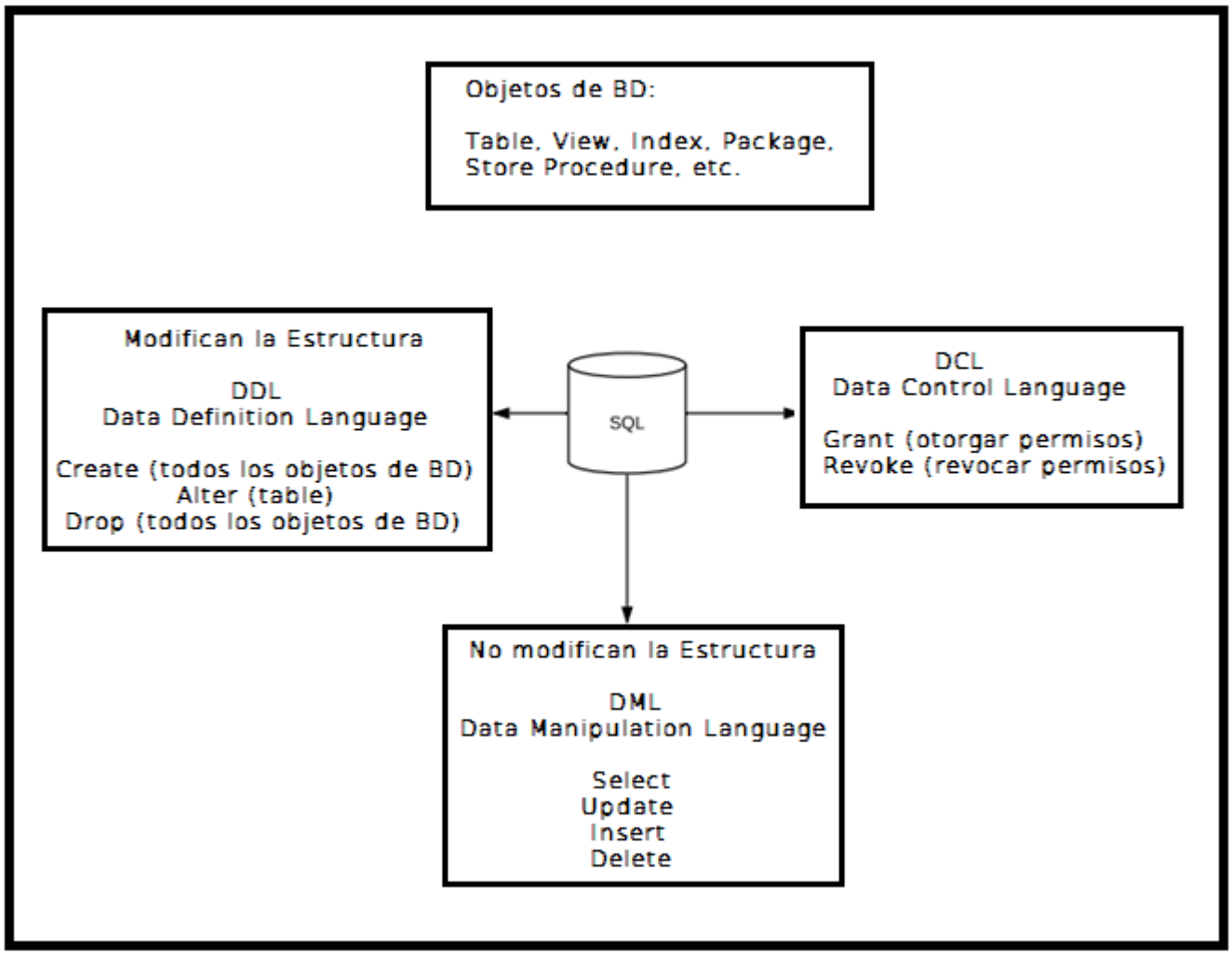
Unidad 1

DEFINICION DE UN SISTEMA DE BASES DE DATOS

Un sistema de base de datos es un sistema computarizado cuyo propósito general es mantener la información centralizada y hacer que esté disponible cuando se la solicite.

Existen dos aspectos del diseño de la base de datos que son importantes para lograr la flexibilidad de su uso:

- Los datos deben ser independientes de los programas que los utilizan, de modo que cualquier cambio no implique una modificación de los programas.
- Debe ser posible explorar la base de datos sin necesidad de programar con lenguajes convencionales de programación. Para esto se utilizarán lenguajes de consultas. (SQL – Structured Query Language).



COMPONENTES DE UN SISTEMA DE BASE DE DATOS.

Los cuatro componentes principales de un sistema de base de datos son:

I. Información:

En general, la información en la base de datos está integrada y es compartida.

- Integrada significa que la base de datos puede considerarse como una unificación de varios archivos de datos, eliminando total o parcialmente cualquier redundancia, o sea que los datos no se repitan.
- Compartida significa que los elementos individuales de información en la base de datos pude compartirse entre varios usuarios distintos.

Además de los datos almacenados, se maneja la definición de la base de datos almacenada, llamado "catálogo del sistema", que funcionará como diccionario de datos e incluirá todos los detalles descriptivos de la base.

II. Equipos:

Consiste en los servidores de la base de datos: volúmenes de almacenamiento, los dispositivos de E/S, controladores de dispositivos, procesadores y memorias principales.

III. Programas:

- Programas de aplicación: permiten la interacción entre los usuarios y la base de datos.
- Programas de administración: el sistema de administración (SGBD), maneja las solicitudes de acceso a la base de datos, la obtención y puesta al día de los datos. El SGBD está compuesto por software para procesar consultas y software para tener acceso a los datos almacenados.

IV. Usuarios:

Existen cuatro tipos de usuarios de las BD:

- **Administrador de Bases de Datos:** El DBA es la persona que toma las decisiones estratégicas y de política con respecto a la información de la empresa. Es el encargado del control general del sistema a nivel técnico.

  Sus funciones son:
  - La definición del esquema conceptual.
    Identificar las entidades y la información que debe registrarse acerca de esas entidades (diseño Lógico).
  - La definición del esquema interno.
    Cómo se representará la información en la BD (diseño Físico). Se utiliza DDL (data definitionlanguage).
  - La vinculación con los usuarios.
    El DBA se encargará de garantizar la disponibilidad de los datos que los usuarios necesiten y ayudarlos con el esquema externo.

- La definición de las verificaciones de seguridad e integridad.
- La puesta en práctica de procedimientos de respaldo y recuperación.
- La supervisión del desempeño y responder a cambios en los requerimientos.

- **Analista de sistemas y programador de aplicaciones:** los analistas determinan los requerimientos de los usuarios finales. Los programadores escriben los programas que utilizan la BD, para luego probar, depurar, documentar y mantener estas transacciones programadas.

- **Usuarios Finales:** interactúan con el sistema a través de las aplicaciones que escriben los programadores. Son las personas que necesitan tener acceso a la base de datos para consultarla, actualizarla, generar informes, etc.

CARACTERISTICAS DE UNA BASE DE DATOS

Una BD está constituida por un conjunto de datos persistentes, disponibles para muchos usuarios, los cuales pueden acceder en forma simultánea.

Características:

- **Independencia de datos respecto de los sistemas:**
  Los programas de acceso a la BD se escriben de modo independiente a los datos. En caso que la estructura de los datos sufra modificaciones no tiene porqué incidir en la estructura de la codificación de los programas que utilizan dichos datos.

- **Posibilidad de disminuir la redundancia:**
  El control de la redundancia en la BD permite tener los datos en forma única, y si existe una repetición, que sea reducida al mínimo. Se ahorrará espacio y se mantendrán mejor los datos.

- **Naturaleza auto descriptiva de los sistemas de las Bases de Datos:**
  El sistema no sólo contiene la base de datos misma, sino también una definición completa con información de estructuras, tipo y formato de almacenamiento de cada elemento y diversas restricciones que se aplican a los datos.

- **Manejo de múltiples vistas de los datos:**
  Una BD suele ser vista por varios usuarios, en diferentes momentos y de diferentes maneras. Se tiene la posibilidad de que cada usuario vea sólo lo que necesite.

- **Posibilidad de compartir los datos:**
  La información está disponible para varios usuarios y por distintas vías de acceso. Pueden acceder varios usuarios y requerir diferentes vistas de la misma.

- **Compacto, rápido y actual:**
  Se dispone en cualquier momento de la información precisa y al día.

- **Posibilidad de aplicar restricciones de seguridad:**
  Se definen maneras de asegurar los accesos a la BD, ya sea con palabras claves o vistas parciales. Se conceden permisos de accesos para consultas o modificaciones.

- **Auditoria:**
  Proveen la facilidad para obtener estados de la BD, accesos realizados y registros borrados.

- **Recupero:**
  La recuperación y restauración ante algún problema de la base está considerada. Se las actualiza pero al mismo tiempo que va haciendo la misma, se guarda.

- **Relacionabilidad:**
  Los datos podrán ser utilizados o explorados de manera flexible, con diferentes caminos de acceso, gracias a su relación incluida de datos.

Unidad 2

BASE DE DATOS RELACIONAL:

Una BD relacional es aquella cuyos usuarios la perciben como un conjunto de tablas. Se visualiza con un diseño de tabla de doble entrada.

Ejemplo:

Tabla 1: Proveedores.

NUMERO	NOMBRE	DOMICILIO	LOCALIDAD
101	Gómez	Nazca 920	Capital Federal
102	Pérez	Argerich 1030	Capital Federal
103	Vázquez	Sarmiento 450	Ramos Mejía
104	López	Alsina 720	Avellaneda

Tabla 2: Productos.

PNRO	PNOMBRE	PRECIO	TAMAÑO	LOCALIDAD
001	Talco	500	Chico	Capital Federal
002	Talco	700	Mediano	Capital Federal
003	Crema	3500	Grande	Ramos Mejía
004	Cepillo	2500	Grande	Avellaneda
005	Esmalte	1200	Normal	Chacarita

Tabla 3: Prov-Prod.

NUMERO	PNRO	CANTIDAD
101	001	300
101	002	200
101	003	400
101	004	200
101	005	100
102	001	300
102	002	400
103	002	200
104	002	200
104	004	300

Propiedades de las tablas relacionales:

- Cada entrada de la tabla representa un ítem de datos, no hay grupos repetitivos. Ej.: significa que ninguna tabla tendrá dos columnas con los mismos datos.
- Son homogéneas por columna. Todos los ítems de una columna son de la misma clase. En cada intersección de fila y columna sólo hay un valor.
- Cada columna tiene su propio nombre, único en esa tabla.
- Todas las filas son distintas; no se permiten filas duplicadas.
- Las tablas tienen una clave primaria única para esa tabla. La clave primaria es un identificador único para la tabla. Puede ser una PK simple, si está representada por sólo una columna, o compuesta, si la PK está representada por más de una columna.
- Las filas y columnas pueden considerarse en cualquier secuencia y momento. No están ordenadas.

Definiciones básicas de BD:

- **Fila:** En general se la denomina tupla, representa un registro de datos.
- **Columna:** La información que se encuentra dentro de cada una de ellas se denomina atributo, representa un ítem de datos.
- **Grado de la relación:** Es la cantidad de columnas que tiene una tabla.
- **Cardinalidad:** Es el número de filas que contiene una tabla.
- **Dominio:** Es un conjunto de valores atómicos. Se trata de especificar un tipo de datos al cual pertenecen los valores que conforman el dominio. Por ejemplo, en la tabla Proveedores, el atributo NUMERO, se pude considerar que se ha definido para dicho campo una longitud de 3 dígitos numéricos. Por lo tanto podemos definir el dominio con un rango de 001 a 999.

Tipos de Relaciones:

- **Relaciones base o reales:** Son aquellas relaciones con nombre. Son las tablas o entidades.

- **Vistas (relaciones virtuales):** Es una relación derivada con nombre, representada dentro del sistema mediante su definición en términos de otras relaciones con nombre. No posee datos almacenados propios, separables y distinguibles.

- **Instantáneas (snapshot):** También es una relación derivada con nombre, pero son reales; están representadas no sólo por su definición en términos de otras relaciones con nombre sino también por sus datos propios almacenados.

- **Resultado de consultas:** Pueden o no tener nombre. No tienen existencia persistente.

- **Relaciones temporales:** Es una relación con nombre que se destruye, por ejemplo, al terminar una sesión activa.

EJERCITACIÓN:

1) Tabla: ALUMNOS

LEGAJO	NOMBRE	DOMICILIO	LOCALIDAD	EDAD
6097	Perez, Juan	Nazca 920	Capital Federal	19
6102	Gomez, Alex	Artigas 1230	Capital Federal	20
6110	Pari, Oscar	Puán 760	Capital Federal	21
6115	Nero, Franco	Mitre 545	Avellaneda	19
6123	Menen, Ana	Mitre 670	Avellaneda	18
6130	Roca, Alicia	Alsina 54	Ramos Mejía	22

A) Qué cardinalidad tiene esta relación?
B) Cuál es el grado de la relación?
C) Dicha tabla representa una entidad o una interrelación? Justificar.

2) Tabla1

LEGAJO	MATERIA	NOTA
6097	Matemática	7
6097	Programación I	8
6102	Programación I	5
6102	Programación II	7
6115	Matemática	4
6123	Laboratorio I	9
6123	Bases de Datos	9

A) Definir el dominio para la columna NOTA.
B) Colocar un nombre técnico a la relación.
C) Indicar cuál o cuáles campos pueden identificar a la fila unívocamente.

Unidad 3
DEFINICION DE DATOS (DDL)

•CREATE TABLE

Sintaxis:

CREATE TABLE tabla
(definición-de-columna [, definición-de-columna]…
[, definición-de-clave-primaria]
[, definición-de-clave-ajena] [, definición-de-clave-ajena]…]);

Se entiende como "definición de columna":

Nombre-columna tipo-de-datos [NOT NULL]

TIPOS DE DATOS:

- Datos numéricos
  - INTEGER Entero binario de palabra completa (INT)
  - DECIMAL Número decimal empaquetado (DEC)
  - FLOAT Número de punto flotante

- Datos de cadena
  - CHARACTER(n) Cadena de longitud fija con exactamente n caracteres de 8 bits (CHAR)
  - VARCHAR(n) Cadena de longitud variable con hasta n caracteres de 8 bits

- Datos de Fecha / Hora
  - DATE Fecha (AAAAMMDD)
  - TIME Hora (HHMMSS)

Ejemplo:

```
CREATE TABLE PROVEEDORES
(NUMERO       INT    NOT NULL,
 NOMBRE       VARCHAR(20)   NOT NULL,
 DOMICILIO    VARCHAR(15)   NOT NULL,
 LOCALIDAD    VARCHAR(12),
 PRIMARY KEY (NUMERO));
```

```
CREATE TABLE PRODUCTOS
(PNRO                    INT   NOT NULL,
 PNOMBRE      VARCHAR(20)   NOT NULL,
 PRECIO          INT    NOT NULL,
 TAMAÑO          VARCHAR(20),
 LOCALIDAD     VARCHAR(12)  NOT NULL,
 PRIMARY KEY (PNRO));

CREATE TABLE PROV-PROD
(NUMERO       INT   NOT NULL,
 PNRO          INT   NOT NULL,
 CANTIDAD      INT    NOT NULL,
 PRIMARY KEY (NUMERO, PNRO),
 FOREIGN KEY (NUMERO) REFERENCES PROVEEDORES (NUMERO),
FOREIGN KEY (PNRO) REFERENCES PRODUCTOS (PNRO));
```

- **ALTER TABLE**

Sintaxis:

```
ALTER TABLE tabla ADD columna        tipo-de-datos;
```

Ejemplo:

```
ALTER TABLE PRODUCTOS ADD COLOR VARCHAR (10);

ALTER TABLE PRODUCTOS MODIFY (COLOR DEFAULT 'ROJO' NOT NULL);
```

- **DROP TABLE**

Sintaxis:

```
DROP TABLE tabla;
```

Ejemplo:

```
DROP TABLE PRODUCTOS;
```

•INDICES

Los índices son estructuras de acceso que sirven para obtener los registros prontamente, en respuesta a ciertas condiciones de búsqueda.

•CREATE INDEX

Sintaxis:

CREATE INDEX índice
ON tabla (columna [, columna…)]

Ejemplo:

CREATE INDEX I_XN ON PROVEEDORES (NUMERO);
CREATE INDEX I_XP ON PRODUCTOS (PNRO);
CREATE INDEX I_XNP ON PROV-PROD (NUMERO,PNRO);

•DROP INDEX

Sintaxis:

DROP INDEX índice;

Ejemplo:

DROP INDEX I_XNP;

•VISTAS

Las vistas son relaciones virtuales derivadas con nombre.

•CREATE VIEW

Sintaxis:

CREATE VIEW vista [(columna [, columna…)]

ASsubconsulta;

Ejemplo:

```
CREATE VIEW V_ALGUNOS_PROVEEDORES  AS
  SELECT NUMERO, LOCALIDAD
FROM PROVEEDORES
WHERE LOCALIDAD = 'CAPITAL';
```

- **DROP VIEW**

Sintaxis:

```
DROP VIEW vista;
```

Ejemplo:

```
DROP VIEW V_ALGUNOS_PROVEEDORES;
```

EJERCICIOS DE DDL:

1)Crear una vista formada por los números de proveedores y números de productos situados en diferentes localidades.
2)Agregar la columna IMPORTADOR a la tabla PRODUCTOS.
3)Crear una vista formada por los registros de los proveedores que viven en Wilde.
4)Crear las tablas DEPARTAMENTOS y EMPLEADOS con sus relaciones, y las tablas PACIENTES y MEDICAMENTOS con sus relaciones.

MANIPULACION DE DATOS (DML)

SELECT (consultas simples)

Sintaxis:

SELECT nombre-columna [, nombre-columna]
FROM tabla
[WHERE condición];

SELECT: "Seleccionar" los campos especificados.
FROM: "desde" la tabla especificada.
WHERE: "donde" se cumple la condición especificada.

Ejemplo 1:

SELECT NUMERO, NOMBRE
FROM PROVEEDORES
WHERE LOCALIDAD = 'CAPITAL';

Ejemplo 2:

SELECT PNRO
FROM PROV-PROD

Resultado: 001 – 002 – 003 – 004 – 005 – 001 – 002 – 002 – 002 - 004

SELECT DISTINCT PNRO
FROM PROV-PROD

Resultado: 001 – 002 – 003 – 004 – 005

Ejemplo 3:

SELECT *
FROM PRODUCTOS;

Ejemplo 4:

SELECT PNOMBRE
FROM PRODUCTOS
WHERE TAMAÑO = 'GRANDE'
AND PRECIO > 500;

La condición WHERE puede combinarse con operadores de comparación =, <> (distinto, o !=), >, >=, <, <= y operadores boléanos AND, OR y NOT.

Ejemplo 5:

```
SELECT  PNOMBRE,  PRECIO
FROM PRODUCTOS
WHERE TAMAÑO = 'GRANDE'
ORDER BY PRECIO DESC;
```

Resultado:

PNOMBRE	PRECIO
Crema	3500
Cepillo	2500

- **SELECT (consultas de reunión)**

Ejemplo 1:

```
SELECT  PROVEEDORES.*,  PRODUCTOS.*
FROM PROVEEDORES , PRODUCTOS
WHERE PROVEEDORES.LOCALIDAD = PRODUCTOS.LOCALIDAD;
```

Ejemplo 2:

```
SELECT  P.*,  PR.*
FROM PROVEEDORES AS P , PRODUCTOS AS PR
WHERE P.LOCALIDAD = PR.LOCALIDAD
AND PR.PRECIO> 600;
```

- **SELECT (funciones de agregados)**

COUNT: Número de valores en la columna
SUM: Suma de los valores de la columna
AVG: Promedio de los valores de la columna
MIN: Valor más chico en la columna
MAX: Valor más grande en la columna

Ejemplo 1:

Obtener el número total de productos:

```
SELECT  COUNT(*)
FROM  PRODUCTOS;
```

El resultado en este caso será: 5

Ejemplo 2:

Obtener el número total de proveedores que entregan productos actualmente:

```
SELECT  COUNT(DISTINCT NUMERO)
FROM  PROV-PROD;
```

El resultado en este caso será: 4

Ejemplo 3:

Obtener el número DE ENVÍOS DEL PRODUCTO 002:

```
SELECT  COUNT (*)
FROM  PROV-PROD
WHERE PNRO = 002;
```

El resultado en este caso será: 4

Ejemplo 4:

Obtener la cantidad total suministrada de cada producto:

```
SELECT  PNRO, SUM (CANTIDAD) AS CANTIDAD
FROM  PROV-PROD
GROUP BY PNRO;
```

El resultado en este caso será:

PNRO	CANTIDAD
001	600
002	1000
003	400
004	500
005	100

El operador GROUP BY (agrupar por) reorganiza en sentido lógico la tabla en cuestión formando grupos, en este caso de igual número de producto.
Lo que figura en el GROUP BY **DEBE** estar incluido en el SELECT.

Ejemplo 5:

Obtener los números de todos los productos suministrados por más de un proveedor:

```
SELECT  PNRO
FROM  PROV-PROD
GROUP BY PNRO
HAVING COUNT (*) > 1;
```

El resultado en este caso será:

PNRO
001
002
004

El operador HAVING (con) sirve para eliminar grupos, de la misma manera como WHERE sirve para eliminar filas.

- **SELECT (Características avanzadas)**

Ejemplo 1:

Obtener todos los productos cuyos nombres comiencen con la letra A:

```
SELECT  *
FROM  PRODUCTOS
WHERE PNOMBRE LIKE 'A%';
```

El carácter % representa cualquier secuencia de n caracteres (donde n puede ser cero).
También podría encontrarse la siguiente sintaxis:

LIKE 'A_' donde el carácter de subrayado representa un carácter individual.

En este otro caso: LOCALIDAD NOT LIKE '%E%' se cumplirá si LOCALIDAD no contiene la letra E.

Ejemplo 2:

Obtener los nombres de los proveedores que suministran el producto 004:

```
SELECT  NOMBRE
FROM  PROVEEDORES
WHERE NUMERO IN
        (SELECT NUMERO
        FROM PROV-PROD
         WHERE PNRO = 004);
```

El resultado en este caso de recuperación de datos con subconsulta será:

NOMBRE
Gómez
López

Ejemplo 3:

Obtener los números de los productos cuyo precio sea menor que el valor máximo actual de precio en la tabla PRODUCTOS:

```
SELECT  PNRO
FROM  PRODUCTOS
WHERE PRECIO <
        (SELECT MAX(PRECIO)
          FROM PRODUCTOS);
```

El resultado en este caso de recuperación de datos con subconsulta será:

PNRO
001
002
004
005

Ejemplo 4:

Obtener los números de los productos cuyo precio sea menor que 600, o sean suministrados por el proveedor 102.

```
SELECT  PNRO
FROM  PRODUCTOS
WHERE PRECIO < 600
UNION
SELECT PNRO
FROM PROV-PROD
WHERE NUMERO = 102;
```

El resultado en este caso de recuperación de datos con UNION será:

PNRO
001
002
004
005

- **SELECT (INNER JOIN – OUTER JOIN)**

```
SQL>select * from tabla1;

  CODIGO         FECHA
  ---------      ----------
     1           10.03.2024
     2           10.04.2024
     3           10.05.2024

SQL>select * from tabla2;

  CODIGO       FECHA_ST        FECHA_END
  ---------    ---------------    ---------------
1              09.05.2024         09.06.2024
2              10.05.2024         09.06.2025
```

Inner join:

```
select a.*, b.*
from tabla1 a, tabla2 b
where a.fecha = b.fecha_st;
```

CODIGO	FECHA	CODIGO	FECHA_ST	FECHA_END
3	10.05.2024	2	10.05.2024	09.06.2025

Outer join:

```
select a.*, b.*
from tabla1 a, tabla2 b
where a.fecha(+) = b.fecha_st
```

CODIGO	FECHA	CODIGO	FECHA_ST	FECHA_END
3	10.05.2024	1	09.05.2024	09.06.2024
		2	10.05.2024	09.06.2025

--Trae todo lo de la tabla2 (b) y de la tabla1 (a) trae el o los registros que cumplen con la condición. **(LEFT JOIN)**

```
select a.*, b.*
from tabla1 a, tabla2 b
where a.fecha = b.fecha_st(+)
```

CODIGO	FECHA	CODIGO	FECHA_ST	FECHA_END
1	10.03.2024			
2	10.04.2024			
3	10.05.2024	2	10.05.2024	09.06.2025

--Trae todo lo de tabla1(a), y de la tabla2(b) trae el o los registros que cumplen con la condición. **(RIGHT JOIN)**

- **UPDATE**

Sintaxis:

```
UPDATE tabla
SET campo = valor
[, campo = valor]
[WHERE condición];
```

Ejemplo:

Cambiar el tamaño a "chico", disminuir el precio del producto 003 de 8000 a 7000 pesos e indicar que su localidad es "desconocida".

```
UPDATE PRODUCTOS
SET TAMAÑO = 'CHICO',
PRECIO = PRECIO – 1000,
LOCALIDAD = NULL
WHERE PNRO = 003;
```

- **DELETE**

Sintaxis:

```
DELETE FROM tabla
[WHERE condición];
```

Ejemplo 1:

Eliminar el producto 004 (elimino un solo registro)

```
DELETE FROM PRODUCTOS
WHERE PNRO = 004;
```

Ejemplo 2:

Eliminar todos los productos realizados en Capital (elimino varios registros)

```
DELETE FROM PRODUCTOS
WHERE LOCALIDAD = 'CAPITAL';
```

- **INSERT**

Sintaxis:

```
INSERT INTO tabla [ (campo [, campo]…) ]
VALUES ([ (valor [, valor]…) ]
```

Ejemplo 1:

```
INSERT INTO PRODUCTOS (PNRO, PNOMBRE, PRECIO, LOCALIDAD)
VALUES (004, 'jabón', 1000, 'Avellaneda');
```

Ejemplo 2:

INSERT INTO PRODUCTOS
VALUES (004, 'jabón', 1000, 'chico', 'Avellaneda');

TRANSACCIONES:

Una transacción es una unidad lógica de trabajo, consiste en una secuencia de instrucciones de consulta y actualizaciones. Una transacción entra en un estado activo inmediatamente después de que inicia su ejecución (al ejecutar una instrucción SQL). Cuando termina, pasa al estado parcialmente confirmado.

Commit : Compromete la transacción actual; es decir, hace que los cambios realizados por la transacción sean permanentes en la base de datos. Señala el término exitoso de la transacción, le dice al manejador que se ha finalizado con éxito una unidad lógica de trabajo. O sea, las modificaciones quedan "comprometidas".

Rollback : Causa el retroceso de la transacción actual; es decir, deshace todas las actualizaciones realizadas; así, el estado de la base de datos se restaura al que existía previo a la ejecución de la transacción.
El retroceso de transacciones es útil si se detecta alguna condición de error durante la ejecución de una transacción.

EJERCICIOS DE DML:

Realizar las siguientes proposiciones SQL con la siguiente estructura:

PROVEEDORES (NUMERO, NOMBRE, DOMICILIO, LOCALIDAD)
PRODUCTOS (PNRO, PNOMBRE, PRECIO, TAMAÑO, LOCALIDAD)
PROV-PROD (NUMERO, PNRO, CANTIDAD)

1) Obtener los detalles completos de todos los productos.
2) Obtener los detalles completos de todos los proveedores de Capital.
3) Obtener todos los envíos en los cuales la cantidad está entre 200 y 300 inclusive.
4) Obtener los números de los productos suministrados por algún proveedor de Avellaneda.
5) Obtener la cantidad total del producto 001 enviado por el proveedor 103.
6) Obtener los números de los productos y localidades en los cuales la segunda letra del nombre de la localidad sea A.
7) Obtener los precios de los productos enviados por el proveedor 102.
8) Construir una lista de todas las localidades en las cuales esté situado por lo menos un proveedor o un producto.
9) Cambiar a "Chico" el tamaño de todos los productos medianos.
10) Eliminar todos lo sproductos para los cuales no haya envíos.

11) Insertar un nuevo proveedor (107) en la tabla PROVEEDORES. El nombre y la localidad son Rosales y Wilde respectivamente; el domicilio no se conoce todavía.

Dadas las siguientes tablas:

CLIENTES (código_cliente, nombre, domicilio, provincia)
PRODUCTOS (código_producto, nombre_producto)
ITEM_VENTAS (número_factura, coódigo_producto, cantidad, precio)
VENTAS (número_factura, código_cliente, fecha)

Resolver las siguientes consultas:

1. Obtener la cantidad de unidades máxima.
2. Obtener la cantidad total de unidades vendidas del producto c.
3. Cantidad de unidades vendidas por producto, indicando la descripción del producto, ordenado de mayor a menor por las cantidades vendidas.
4. Cantidad de unidades vendidas por producto, indicando la descripción del producto, ordenado alfabéticamente por nombre de producto para los productos que vendieron más de 30 unidades.
5. Obtener cuantas compras (1 factura = 1 compra) realizó cada cliente indicando el código y nombre del cliente ordenado de mayor a menor.
6. Promedio de unidades vendidas por producto, indicando el código del producto para el cliente 1.

Se tiene la siguiente base de datos relacional:

Documentos (cod_documento, descripción)
Oficinas (cod_oficina, codigo_director, descripcion)
Empleados (cod_empleado, apellido, nombre, fecha_nacimiento, num_doc, cod_jefe, cod_oficina, cod_documento)
Datos_contratos (cod_empleado, fecha_contrato, cuota, ventas)
Fabricantes (cod_fabricante, razón_social)
Listas (cod_lista, descripción, ganancia)
Productos (cod_producto, descripcion, precio, cantidad_stock, punto_reposición, cod_fabricante)
Precios (cod_producto, cod_lista, precio)
Clientes (cod_cliente, cod_lista, razón_social)
Pedidos (cod_pedido, fecha_pedido, cod_empleado, cod_cliente)
Detalle_pedidos (cod_pedido, numero_linea, cod_producto, cantidad)

Resolver las siguientes consultas utilizando sentencias SQL:

Consultas simples (una sola tabla)

1) Obtener una lista con los nombres de las distintas oficinas de la empresa.
2) Obtener una lista de todos los productos indicando descripción del producto, su precio de costo y su precio de costo IVA incluído (tomar el IVA como 21%).
3) Obtener una lista indicando para cada empleado apellido, nombre, fecha de cumpleaños y edad.
4) Listar todos los empleados que tiene un jefe asignado.
5) Listar los empleados de nombre "María" ordenado por apellido.
6) Listar los clientes cuya razón social comience con "L" ordenado por código de cliente.
7) Listar toda la información de los pedidos de Marzo ordenado por fecha de pedido.
8) Listar las oficinas que no tienen asignado director.
9) Listar los 4 productos de menor precio de costo.
10) Listar los códigos de empleados de los tres que tengan la mayor cuota.

Consultas multitablas

1) De cada producto listar descripción, razón social del fabricante y stock ordenado por razón social y descripción.
2) De cada pedido listar código de pedido, fecha de pedido, apellido del empleado y razón social del cliente.
3) Listar por cada empleado apellido, cuota asignada, oficina a la que pertenece ordenado en forma descendente por cuota.
4) Listar sin repetir la razón social de todos aquellos clientes que hicieron pedidos en Abril.
5) Listar sin repetir los productos que fueron pedidos en Marzo.
6) Listar aquellos empleados que están contratados por más de 10 años ordenado por cantidad de años en forma descendente.
7) Obtener una lista de los clientes mayoristas ordenada por razón social.
8) Obtener una lista sin repetir que indique qué productos compró cada cliente, ordenada por razón social y descripción.
9) Obtener una lista con la descripción de aquellos productos cuyo stock está por debajo del punto de reposición indicando cantidad a comprar y razón social del fabricante ordenada por razón social y descripción.
10) Listar aquellos empleados cuya cuota es menor a 50000 o mayor a 100000.

Unidad 4
CATALOGO DEL SISTEMA

Las aplicaciones dinámicas que no están codificadas de forma rígida para funcionar con un conjunto específico de tablas y vistas deben disponer de un mecanismo para determinar la estructura y los atributos de los objetos de cualquier base de datos a la que se conectan. Las aplicaciones pueden necesitar información como la siguiente:

- El número y nombre de las tablas y vistas de una base de datos.
- El número de columnas de una tabla o vista, junto con el nombre, el tipo de datos, la escala y la precisión de cada columna.
- Las restricciones definidas en una tabla.
- Los índices y las claves definidas para una tabla.

El catálogo del sistema proporciona esta información para las bases de datos. El núcleo de los catálogos del sistema de gestión de base de datos (SGBD) es un conjunto de vistas que muestran metadatos que describen los objetos. Los metadatos son datos que describen los atributos de los objetos de un sistema.
Las vistas de catálogo proporcionan acceso a los metadatos almacenados en cada base de datos.
Si queremos encontrar una similitud con algo conocido y común, se puede asemejar a un diccionario de datos.
El catálogo proporciona información a los usuarios y al DBA.

La información almacenada incluye descripciones de los nombres de las relaciones, nombres de los atributos, dominios de los atributos, claves, etc.
Maneja descripciones de nivel externo de las vistas y de nivel interno de las estructuras de almacenamiento e índices.
También incluye normas de seguridad y autorización, que permiten a los usuarios acceder a las relaciones y vistas de las bases de datos bajo un orden determinado.

Cabe señalar que el catálogo no es igual en los diferentes SGBD.

Algunas vistas del catálogo son:

SYSTABLES: Esta vista proporciona una fila por cada tabla en la base de datos. Para cada una de estas tablas indica:

NAME	Nombre de la tabla
CREATOR	Nombre del usuario dueño de la tabla
COLCOUNT	Número de columnas de la tabla, etc.

SYSCOLUMNS: Esta vista proporciona una fila por cada columna de cada tabla nombrada en SYSTABLES. Para cada una de estas columnas indica:

NAME	Nombre de la columna
TBNAME	Nombre de la tabla de la cual forma parte
COLTYPE	Tipo de datos de la columna, etc.

SYSINDEXES: Esta vista proporciona una fila por cada índice en el sistema. Para cada una de estos índices indica:

NAME	Nombre del índice
TBNAME	Nombre de la tabla indexada
CREATOR	Nombre del usuario dueño del índice, etc.

SYSVIEWS: Esta vista proporciona una fila por cada vista en el sistema. Para cada una de estas vistas indica:

NAME	Nombre de la vista
CREATOR	Nombre del usuario dueño de la vista, etc.

NOTA: Solamente se pueden realizar operaciones de SELECT sobre los objetos del catálogo.

Ejercicios:

1)Cuáles tablas contienen la columna LOCALIDAD?
2)Cuántas columnas tiene la tabla PRODUCTOS?
3)Obtener una lista de todos los usuarios dueños de por lo menos una tabla, junto con el número de tablas que le pertenecen a cada uno.
4)Obtener una lista de los nombres de todas las tablas que tienen por lo menos un índice.

REGLAS DE INTEGRIDAD RELACIONAL

Las restricciones o reglas de integridad proporcionan un medio de asegurar que las modificaciones hechas a la base de datos no provoquen la pérdida de la consistencia de los datos. Por tanto, las restricciones de integridad protegen a la base de datos contra los daños accidentales.

La integridad de los datos está garantizada por un conjunto predefinido de reglas de negocio o constraints. La definición de constraints garantiza:

- Permitir o no valores nulos en las columnas.
- Valores únicos para las columnas.
- Un valor primario identificando cada una de las filas en una tabla.
- Existencia de valores referenciales para cada columna que se lo indique.
- Garantizar que las columnas cumplan reglas de negocio.

Ventajas:

- Mejoramiento de la eficiencia.
- Centralización de reglas.
- Fácilmente modificables.
- Flexibilidad (Enable/Disable)
- Documentación completa en el diccionario de datos.

Tipos de constraints:

- Notnull
- Unique
- Check
- Primary Key
- Foreign Key

Las constraints pueden ser declaradas cuando se crea la tabla. Pueden definirse como parte de la columna imponiendo reglas sobre ella. O puede definirse como parte de la tabla, en cuyo caso impone reglas que alcanzan cualquier columna o conjunto de columnas de la tabla.

Las constraints deben ser nombradas cuando son definidas. El nombre debe ser único en el esquema. Dicho nombre aparecerá en los mensajes cuando la constraint es violada y se utilizará en las vistas del diccionario de datos. Si una constraint no es nombrada, el motor de base de datos le asignará un nombre.

NotNullConstraint

Cuando una columna debe contener siempre valores, debe usarse la *notnull*constraint, la cual se controlará en las inserciones y actualizaciones. Es la única constraint que se recomienda que no tenga nombre.
Ejemplo:

```
CREATE TABLE empleado(
        Id_empleado INT NOT NULL,
        ...
        );
```

Unique Constraint

Garantiza que dos filas en la tabla no tengan valores duplicados para la columna o columnas sobre las cuales se define la uniqueconstraint. Los valores de una constraint de unicidad deben ser distintos o nulos.
Ejemplo:

```
CREATE TABLE empleado(
        Id_empleadoINT NOT NULL,
DocumentoINT CONSTRAINT uk_documento   UNIQUE
        ...
        );
```

Check Constraint

Obliga a que una condición especificada sea evaluada a TRUE para la columna o grupo de columnas que se aplica la checkconstraint.
Ejemplo:

```
CREATE TABLE empleado(
        Edad_empINT CONSTRAINT ck_edad_emp
        CHECK (edad_emp> 17 and edad_emp< 65),
...)

CONSTRAINT ck_est_civil CHECK
        ((est_civil in ('CASADO', 'PAREJA') AND
dni_conyuge IS NOT NULL) OR
        (est_civil in ('SOLTERO', 'SEPARADO', 'VIUDO') AND
        dni_conyuge IS NULL));
```

Primary Key Constraint (regla de integridad de las entidades)

Especifica un simple o compuesto conjunto de columnas que identifican cada una de las filas de la tabla. Una tabla puede tener una sola clave primaria y las columnas que la componen no pueden todas tener valores nulos.
Ejemplo:

```
CREATE TABLE gerentes(
Id_sucursalINT NOT NULL,
Id_gerenteINT NOT NULL,
....
CONSTRAINT pk_gerentes PRIMARY KEY (id_sucursal,
Id_gerente));
```

Foreign Key Constraint (regla de integridadreferencial)

Asegura que los valores en las columnas de la constraint existan previamente en la definición de una clave primaria o única de otra tabla o en la misma tabla. No garantiza que los valores de las columnas sean no nulos (para esto usar una constraint NOT NULL). Es la más usada de las constraints para definir una base de datos relacional.

Ejemplo:

Creación de una tabla con una PK y otra con una FK.

```
CREATE TABLE s_dept(Id INT
CONSTRAINT s_dept_pk_id PRIMARY KEY (id),.....);

CREATE TABLE s_emp(....,
        Dep_idINT
CONSTRAINT  s_emp_fk_dept_id  FOREIGN KEY      REFERENCES  s_dept(id);
```

Supongamos que los únicos registros insertados en la tabla s_dept son:

```
INSERT INTO s_dept VALUES (26, ... , 10);
INSERT INTO s_dept VALUES (28, ... , 15);
```

Al insertar un registro como sigue en la tabla s_emp

```
INSERT INTO s_emp VALUES (1, ... ,  80);
```

Se obtendrá el error: Integrity Constraint (S_EMP_FK_DEPT_ID) violated. Parent Key not found.

Borrado en cascada

Las filas que son referenciadas desde una FK, por defecto no pueden ser borradas si existen filas referenciándolas. Excepto que las constraints que las referencian sean definidas con la opción de DELETE CASCADE. Lo cual es peligroso, porque puede propagar un borrado de un único registro al borrado de miles de registros en diversas tablas.
Ejemplo:
Supongamos que la tabla empleados se creó como:

```
CREATE TABLE EMPLEADOS (
        Id     INT not null, ...
        OficINT CONSTRAINT oficinas_fk_ofic FOREIGN KEY
                REFERENCES oficinas (ofic) ON DELETE CASCADE);
```

Lo que gráficamente sería:

EMPLEADOS		
1	.....	10
2	.....	15
3	.....	10
4	.....	10
5	.....	20

ID ..... OFIC

OFICINAS	
10	Operaciones
15	Sistemas
20	Rec. Humanos

OFIC DESC_OFIC

Si se ejecuta el comando

```
DELETE FROM oficinas WHERE ofic = 10;
```

No sólo se borrará un registro de la tabla de oficinas, sino que también se borrarán tres registros de la tabla empleados, y así sucesivamente si todas las FK que referencien a la tabla *oficinas* en otras tablas están definidas como ON DELETE CASCADE, y lo mismo para la tabla empleados, etc.

Agregando Constraints

Una constraint puede ser agregada a una tabla que ya está creada, para ello se debe usar el comando ALTER TABLE.
Ejemplo:

```
ALTER TABLE s_emp
ADD (CONSTRAINT s_emp_pk_id PRIMARY KEY (id),
    CONSTRAINT s_emp_ck_salario CHECK (salario> 500));
```

Deshabilitando / Habilitando constraints

El control establecido por las constraints puede ser habilitado o deshabilitado.

Constraints deshabilitadas

- Cuando una constraint es deshabilitada (disable), la constraint no seguirá siendo controlada.
- Una constraint puede ser deshabilitada para cargar grandes volúmenes de filas, o para exportar/importar una tabla más rápidamente.

La sintáxis para deshabilitar una constraint es:
ALTER TABLE tablaDISABLE CONSTRAINTnombre_constraint;

Cuando una constraint es habilitada (enable):

- La constraint empezará a ser controlada para los futuros INSERT/UPDATE.
- Todos los registros actuales de la tabla se controlarán para ver si cumplen con la constraint.

- La tabla es bloqueada hasta que el control sea realizado.
- Si una excepción es encontrada, la constraint no es habilitada.

La sintáxis para habilitar una constraint es:
ALTER TABLE tablaENABLE CONSTRAINTnombre_constraint;

Borrando constraints

Para borrar una constraint se debe utilizar el comando ALTER TABLE

ALTER TABLE tablaDROP CONSTRAINTnombre_constraint;

LOS DISPARADORES (TRIGGERS)

Los disparadores, son instrucciones que el sistema ejecuta automáticamente como efecto colateral de una modificación de la base de datos. Los disparadores se usan para asegurar algunos tipos de integridad.

Para diseñar un mecanismo disparador hay que cumplir dos requisitos:

1. Especificar las condiciones en las que se va a ejecutar el disparador. Esto se descompone en un evento que causa la comprobación del disparador y una condición que se debe cumplir para ejecutar el disparador.
2. Especificar las acciones que se van a realizar cuando se ejecute el disparador.
   Este modelo de disparadores se denomina
   modelo **evento- condición-acción**.
   La base de datos almacena disparadores como si fuesen datos normales, por lo que son persistentes y accesibles para todas las operaciones de la base de datos.
   Una vez que se almacena un disparador en la base de datos, el sistema de base de datos asume la responsabilidad de ejecutarlo cada vez que ocurra el evento especificado y se satisfaga la condición correspondiente.

Los disparadores son mecanismos útiles para alertar a los usuarios o para realizar de manera automática ciertas tareas cuando se cumplen determinadas condiciones.

Ejemplo: Sobre tabla TOPERACION, informar cuándo los campos FH_RECEPCION e I_OPERACION son MODIFICADOS.

```
CREATE TABLE TOPERACION_AUDITORIA
(
  C_FORMULARIO VARCHAR(12)      NOT NULL,
  O_OPERACION  INT(10)        NOT NULL,
  FH_RECEPCION DATE,
  I_OPERACION  INT(10),
  USUARIO      VARCHAR(30),
  FECHA        DATE,
  ACCION       VARCHAR(50)
)

CREATE OR REPLACE TRIGGER TOPERACION_AUD
BEFORE INSERT  OR DELETE  OR UPDATE  ON TOPERACION
REFERENCING OLD AS OLD NEW AS NEW FOR EACH ROW
DECLARE
BEGIN
IF UPDATING AND :NEW.FH_RECEPCION <> :OLD.FH_RECEPCION AND :NEW.I_OPERACION <>
:OLD.I_OPERACION THEN
INSERT INTO TOPERACION_AUDITORIA
 (
C_FORMULARIO ,
  O_OPERACION  ,
  FH_RECEPCION ,
  I_OPERACION  ,
  USUARIO      ,
  FECHA        ,
  ACCION       )
VALUES (
  :NEW.C_FORMULARIO ,
  :NEW.O_OPERACION  ,
  :NEW.FH_RECEPCION ,
  :NEW.I_OPERACION  ,
  USER,
  SYSDATE,
  'FH_RECEPCION Y I_OPERACION MODIFICADOS');
```

El evento del disparador puede tener varias formas:
-El evento del disparador puede ser **insert** o **delete** en lugar de **update**.
-La cláusula **referencing old row as** se puede usar para crear una variable que almacene el valor anterior de una fila actualizada o borrada.

-Los disparadores se pueden activar antes (**before**) del evento (**insert/delete/update**) en lugar de después (**after**) del evento.

Ejercicios:

1) Una base de datos hospitalaria contiene las siguientes relaciones:
PACIENTES (CODIGO-PAC, APELLIDO-PAC, EDAD)
MEDICAMENTOS (CODIGO-MED, PRECIO-UNITARIO)
GASTOS (CODIGO-PAC, CODIGO-MED)
Indicar las claves primarias y claves ajenas de cada relación.

2) Dadas las siguientes relaciones:
CURSOS (NUMCURSO, TITULO)
OFRECIMIENTOS (NUMCURSO, NUMOFR, FECHA, AULA)
PROFESORES (NUMCURSO, NUMOFR, NUMEMP)
ESTUDIANTES (NUMCURSO, NUMOFR, NUMEMP, CALIFICACION)
EMPLEADOS (NUMEMP, EMPLE-NOMBRE, PUESTO)
Indicar las claves primarias y claves ajenas de cada relación.

3) Dadas las siguientes tablas:
Empleados (cod_emp, nombre, apellido, tipo_doc, num_doc, categoria, cod_ofic)
Oficinas (cod_ofic, descripción)

Crear las siguientes reglas de integridad:

- La columna cod_emp debe ser clave primaria.
- La columna cod_emp debe tener valores entre 100 y 1000.
- Las columnas tipo_doc y num_doc deben contener valores no repetidos (únicos).
- La columna Categoria debe contener algunos de los siguientes valores: Senior, Semi Senior, Junior.
- La columna cod_ofic debe tener valores que existan en la tabla Oficinas.