AR

1. Dada la operación R1 ← A - B, "siempre" es posible realizarla cuando:
   a) A y B posean la misma cantidad de atributos
   b) A y B posean al menos algún atributo en común
   c) A y B tengan tuplas en común
   d) Si A y B poseen distinta cantidad de atributos, siempre A debe tener más atributos que B
   e) Ninguna de las anteriores

2. Indique V ó F y Justifique su respuesta:
"Un operador de Junta Natural, podría comportarse como un producto cartesiano, aunque se haya indicado que es junta natural"

SQL

3. Cuando se crea una función de **tipo escalar:**
   a) Puede devolver en su nombre una tabla, pero sólo con una variable de tipo table
   b) Podría realizar actualizaciones de tablas físicas existentes
   c) Se pueden declarar variable como parte del código de la función
   d) Si una tabla de su código posee un trigger, la función lo dispararía al realizar la operación en la función
   e) Ninguna de las anteriores

4. En SQL, ¿qué operador se utiliza para combinar los resultados de dos consultas, devolviendo todas las filas de amb eliminando las filas duplicadas?
   a) INNER JOIN
   b) UNION
   c) INTERSECT
   d) EXCEPT
   e) Ninguna de las anteriores

5. ¿Cuál es uno de los posibles usos de los triggers en SQL?
   a) Realizar consultas complejas para generar informes
   b) Garantizar la integridad referencial entre tablas relacionadas
   c) Controlar el acceso de los usuarios a la base de datos
   d) Optimizar el rendimiento de las consultas SELECT en una tabla
   e) Ninguna de las anteriores

6. En una vista:
   a) No se puede utilizar el operador except uniendo varias consultas
   b) No se puede utilizar full join, pero sí el resto de los tipos de joins
   c) No se puede utilizar el exists dentro de un where
   d) No se puede utilizar sentencias de tipo delete
   e) Todas las anteriores

PRÁCTICA **TEMA 1**

Referencia: PK / FK / PK + FK

AR

Dada la BD Empresa, con el siguiente Modelo Relacional:

Empleado(id_empleado, nombre, **id_departamento**, fingreso)
Departamento(id_departamento, nombre, ubicación)
Trabaja(**id_empleado**, **id_departamento**, salario)
Venta(id_venta, **id_empleado**, monto)

7. Realice la consulta en AR que devuelva los nombres de los empleados que trabajan en el departamento de "Ventas" y que hayan realizado al menos tres ventas.

EmpVent a←Prod(e.id,e.nombre)(Sel d.nombre="Ventas"(Empleado |X| Departamento))

Prod (1,2)(Sel(1=4 And 1=7 and 1=10 and 3<>6 and 3<>9 and 6<>9))(EmpVent X Venta X Venta X Venta))

8. Realice una consulta en AR que permita listar el empleado más antiguo de cada departamento. Listar nombre y salario del emplea y el nombre del departamento.

8. Realice una consulta en AR que permita listar el empleado más antiguo de cada departamento. Listar nombre y sal y el nombre del departamento.

```
MasReciente ← Prod( 1,3)(SEL(3=8 and 3>9) Empleado X Empleado)
Antiguos ←Prod (1,3) Empleado - MasReciente
Prod(1,2,9) Antiguos|X|Empleado|X|Trabaja
```

SQL

9. Dada una tabla llamada "Pedido" y otra tabla llamada "Historial_Pedido" que se utiliza para mantener un registro de los cambios realizados en los pedidos. Queremos crear un trigger que, cada vez que se inserte, actualice o elimine un registro en la tabla "Pedido", registre la acción realizada en la tabla "Historial_Pedido":

- Pedido (PedidoID,ClienteID, ProductoID, Cantidad, Fecha)
- Historial_Pedido (RegistroID, Tipo_Accion, **PedidoID**, Fecha_Accion)

```
CREATE TRIGGER Registro_Historial_Pedido ON Pedido
AFTER INSERT, UPDATE, DELETE
AS
BEGIN DECLARE @accion VARCHAR(10);
 IF EXISTS(SELECT * FROM INSERTED) AND EXISTS(SELECT * FROM DELETED)
SET @accion = 'UPDATE';
 ELSE IF EXISTS(SELECT * FROM INSERTED) SET @accion = 'INSERT';
 ELSE IF EXISTS(SELECT * FROM DELETED) SET @accion = 'DELETE';
INSERT INTO Historial_Pedido (Tipo_Accion, PedidoID, Fecha_Accion) SELECT @accion, COALESCE(i.PedidoID, d.Pedic
      FROM INSERTED i FULL OUTER JOIN DELETED d ON i.PedidoID = d.PedidoID;
END;
```

otra manera:

```
--
CREATE TRIGGER Registro_Historial_Pedido AFTER INSERT, UPDATE, DELETE ON Pedido
FOR EACH ROW BEGIN DECLARE accion VARCHAR(10);
IF INSERTING THEN SET accion = 'INSERT';
ELSEIF UPDATING THEN SET accion = 'UPDATE';
ELSEIF DELETING THEN SET accion = 'DELETE';
END IF;
INSERT INTO Historial_Pedido (Tipo_Accion, PedidoID, Fecha_Accion) VALUES (accion, NEW.PedidoID, NOW()); END;
```

10. Dada la tabla llamada Empleado, ¿cuál de las siguientes consultas devolverá la cantidad total de empleados por departamento, incluido el departamento llamado "Ventas" incluso si no tiene empleados asignados?

```
Empleado (EmpleadoID, Nombre, Departamento)
```

a) SELECT Departamento, COUNT(EmpleadoID) AS TotalEmpleados FROM Empleado GROUP BY Departamento;
b) SELECT Departamento, COUNT(*) AS TotalEmpleado FROM Empleado RIGHT JOIN (SELECT 'Ventas' AS Departamento UNION ALL SELECT DISTINCT Departamento FROM Empleados) AS Departamento ON Empleado.Departamento = Departamento.Departamento GROUP BY Departamento;
c) SELECT Departamento, COUNT(EmpleadoID) AS TotalEmpleados FROM Empleado LEFT JOIN (SELECT DISTINCT Departamento FROM Empleado) AS Departamento ON Empleado.Departamento = Departamento.Departamento GROUP BY Departamento;
d) SELECT Departamento, COUNT(*) AS TotalEmpleados FROM Empleado LEFT JOIN (SELECT DISTINCT Departamento FROM Empleado) AS Departamento USING (Departamento) GROUP BY Departamento;
e) Ninguna de las anteriores

10. Dada la tabla llamada Empleado, ¿cuál de las siguientes consultas devolverá la cantidad total de empleados por departamento, incluido el departamento llamado "Ventas" incluso si no tiene empleados asignados?

```
Empleado (EmpleadoID, Nombre, Departamento)
```

a) SELECT Departamento, COUNT(EmpleadoID) AS TotalEmpleados FROM Empleado GROUP BY Departamento;
Esta consulta devuelve la cantidad de empleados por departamento, pero no garantiza la inclusión del departamento "Ventas" si no tiene empleados.

b)
```
SELECT Departamento, COUNT(*) AS TotalEmpleado
FROM Empleado RIGHT JOIN
    (SELECT 'Ventas' AS Departamento
    UNION ALL
    SELECT DISTINCT Departamento FROM Empleados)
    AS Departamento
ON Empleado.Departamento = Departamento.Departamento
GROUP BY Departamento;
```

c) SELECT Departamento, COUNT(EmpleadoID) AS TotalEmpleados FROM Empleado LEFT JOIN (SELECT DISTINCT Departamento
Esta consulta utiliza un LEFT JOIN con una subconsulta que lista todos los departamentos distintos. Aunque puede devolver los departamentos con cero empleados, no garantiza explícitamente la inclusión del departamento "Ventas" si no tiene empleados asignados.

d) SELECT Departamento, COUNT(*) AS TotalEmpleados FROM Empleado LEFT JOIN (SELECT DISTINCT Departamento FROM Empleado) AS Departamento USING (Departamento) GROUP BY Departamento;
e) Ninguna de las anteriores

11) Dada la siguiente consulta:

```
SELECT e.nombre
FROM Empleado e INNER JOIN ( SELECT departamento, AVG(salario) AS avg_salario FROM Empleado GROUP BY
    departamento ) AS dept_avg ON e.departamento = dept_avg.departamento WHERE e.salario >
    dept_avg.avg_salario AND e.id_empleado NOT IN ( SELECT id_empleado FROM Empleado WHERE
    departamento = e.departamento);
```

Indique la opción correcta:

a) Busca empleados que tengan un salario superior al salario promedio de todos los empleados en su departamento sin contar algún empleado que sea de su departamento.
No excluye a los empleados del mismo departamento en el cálculo del promedio.

b) Nos permite encontrar empleados que tienen un salario superior al salario promedio de algun departamento, excluyendo al propio empleado en el cálculo del promedio
No busca el promedio de algún departamento en específico, sino el promedio del departamento al que pertenece cada empleado.

c) Nos devuelve los empleados con mayor salario al promedio de los otros departamentos
No compara el salario de los empleados con el promedio de otros departamentos

d) Nos permite encontrar el nombre de los empleados que tienen un salario superior al salario promedio de todos los empleados en su departamento, excluyendo al empleado en el cálculo del promedio

12. Dada la siguiente estructura de tablas:

```
Usuario (idusuario, nya, fechaultacceso, clave)
Session (idusuario, fechacreacion, fechultmodificacion, tipooperacion)
```

p_loginusuario (....): Crear un procedimiento que permita verificar si un usuario puede o no acceder a la aplicación. Cada vez que un usuario accede, se solicita el id de usuario y su clave. En el caso que dichos datos sean correctos, se creará una nueva sesión del usuario, de lo contrario se denegará el acceso y la aplicación debe saber que ese acceso fue denegado, para que esta cuenta no pueda ingresar. Se debe tener en cuenta que sólo guardaremos la última sesión activa del usuario, el resto no las tendremos en línea. Un usuario podrá entrar y salir cientos de veces a la aplicación, siempre y cuando sus datos sean correctos. Si puede ingresar, tendrá una sesión activa, pero sólo una.

```
CREATE PROCEDURE p_loginusuario
  @idusuario INT,
  @clave VARCHAR(50),
  @accesoPermitido BIT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  DECLARE @claveCorrecta VARCHAR(50);
  DECLARE @ultimaSesionExistente BIT;

  -- Verificar credenciales
  SELECT @claveCorrecta = clave
  FROM Usuario
  WHERE idusuario = @idusuario;
```

```
IF @claveCorrecta IS NULL
        BEGIN
            -- El usuario no existe
            SET @accesoPermitido = 0;
            RETURN;
        END
```

```
        IF @clave = @claveCorrecta
        BEGIN
            -- Credenciales OK
            -- Verificar si ya existe una sesión activa para este usuario
            SELECT @ultimaSesionExistente = CASE WHEN EXISTS (
                SELECT 1
                FROM Session
                WHERE idusuario = @idusuario
            ) THEN 1 ELSE 0 END;

            -- Eliminar  sesiones activas anteriores
            IF @ultimaSesionExistente = 1
            BEGIN
                DELETE FROM Session
                WHERE idusuario = @idusuario;
            END
```

```
            -- Crear una nueva sesión para el usuario
            INSERT INTO Session (idusuario, fechacreacion, fechultmodificacion, tipooperacion)
            VALUES (@idusuario, GETDATE(), GETDATE(), 'Inicio de sesión');

            SET @accesoPermitido = 1;
        END
    ELSE
        BEGIN
            -- Las credenciales son incorrectas
            SET @accesoPermitido = 0;
        END
END;
```

PARCIAL TEMA:2

TEMA 2
TEORÍA
AR

1. Dada la operación R1 ← A |X| B, "siempre" es posible realizarla cuando:
   a) A y B posean la misma cantidad de atributos
   b) A y B posean al menos algún atributo en común con el mismo nombre
   c) A y B tengan tuplas en común
   d) Si A y B poseen distinta cantidad de atributos, siempre A debe tener más atributos que B
   e) Ninguna de las anteriores

2. Indique V ó F:
   "Es posible convertir una operación (R⌈X⌋T) utilizando el operador de intersección ∩."
   VERDADERO R(A B) T(B C) → Pr(A,B',C) (R∩T)

SQL

3. Indique qué tipos de sentencias son admitidas dentro de un procedimiento almacenado (*se puede elegir una o varias opciones*):
   a) SELECT
   b) DELETE
   c) CREATE TABLE
   d) IF
   e) DROP FUNCTION
   f) DECLARE @i int

4. ¿Cuál de las siguientes afirmaciones describe mejor el propósito de la cláusula GROUP BY en una consulta SQL?
   a) La cláusula GROUP BY se utiliza para filtrar filas específicas de una tabla en función de una condición específica.
   b) La cláusula GROUP BY se utiliza para combinar filas de una tabla que tienen valores idénticos en una o más columnas y realizar operaciones de agregación en los grupos resultantes.
   c) La cláusula GROUP BY se utiliza para ordenar los resultados de una consulta en función de una o más columnas.
   d) La cláusula GROUP BY se utiliza para unir filas de dos o más tablas en una sola tabla de resultados.
   e) Ninguna de las anteriores

5. Indique cuál de las siguientes afirmaciones sobre la restricción/constraint PRIMARY KEY es correcta:
   a) Una tabla sólo puede tener un atributo con la restricción de PRIMARY KEY
   b) La restricción PRIMARY KEY puede contener valores NULL
   c) La restricción PRIMARY KEY garantiza que los valores en una columna sean únicos y no nulos.
   d) La restricción PRIMARY KEY solo se puede aplicar a columnas numéricas.
   e) Ninguna de las anteriores

6- Indique qué operaciones se pueden realizar en una vista:

a) Consulta de Select con joins entre distintas tablas

b) Cláusula IF para poder ejecutar una consulta u otra, según se cumpla o no la condición

c) Declaración de variables

d) Create table para poder crear una tabla temporal

e) Todas las anteriores

PRACTICA

AR Referencia: PK / FK / PK + FK

Dado el siguiente Modelo Relacional:

```
Estudiante (id_estudiante, nombre, edad, carrera)
Materia (id_mat, nombre)
Final (id_estudiante, id_mat, año, calificación)
Cursa (id_estudiante, id_mat)
```

7. Realizar la consulta en AR para obtener el nombre de los estudiantes que están cursando al menos dos materias diferentes y que hayan obtenido una calificación de al menos 7 en todas las materias ya finalizadas

```
Cursa2 ← Prod(1) SEL(1=3 and 2<>4)(CursaXCursa)
Desaprobo ←Prod (1) (SEL (calificacion<7)Final)
Aprobo1 ←Prod (1) (SEL (calificacion>=7)Final)
RTA←(Cursa ∩Aprobo1) - Desaprobo
```

8. Realice una consulta en AR que permita listar el estudiante con la mejor calificación en cada materia rendida este año de la carrera "Ingeniería en Sistemas". Listar el nombre del estudiante, el nombre de la materia y su calificación.

```
Mat2024 ←SEL(año=2024)Final
Menores ←Prod(idMat,calificacion)SEL(2=6 and 4<8) Mat2024 X Mat2024
MejorCalif2024 ←Prod(id_Mat,calificacion)Mat2024-Menores
Prod(1,2,3) ← MejorCalif2024 |X| Mat2024
```

SQL

9. Dada la tabla Producto(Producto_ID , Nombre, Precio, categoría, Fecha_Creacion), que contiene información sobre productos que puede vender una tienda en línea. Queremos encontrar el nombre y el precio de los productos más recientes para cada categoría. Sin embargo, no queremos simplemente obtener el producto más reciente por fecha para cada categoría, sino que queremos encontrar el producto más reciente antes de una fecha específica ("2023-12-31").

```
SELECT Nombre, Precio
        FROM Producto P1
        WHERE Fecha_Creacion < '2023-12-31' AND NOT EXISTS
                (SELECT *
                        FROM Producto P2
                        WHERE P2.Categoria = P1.Categoria AND P2.Fecha_Creacion > P1.Fecha_Creacion);
```

10. Dadas las siguientes tablas en una base de datos de una biblioteca, queremos encontrar el título del libro que ha sido prestado más veces en el año 2023.

```
Libro(id_libro, titulo)
Prestamo(id_prestamo, id_libro, fecha_prestamo)
```

```
SELECT libros.titulo
        FROM libro INNER JOIN prestamo  pr ON libros.id_libro = prestamo.id_libro
        WHERE YEAR(pr.fecha_prestamo) = 2023
        GROUP BY libro.titulo
        ORDER BY COUNT(pr.id_prestamo) DESC LIMIT 1;
SELECT l.titulo
        FROM Libro l
        JOIN (
          SELECT id_libro
          FROM Prestamo
          WHERE YEAR(fecha_prestamo) = 2023
          GROUP BY id_libro
          HAVING COUNT(*) = (
            SELECT MAX(num_prestamos)
            FROM (
              SELECT COUNT(*) as num_prestamos
              FROM Prestamo
              WHERE YEAR(fecha_prestamo) = 2023
              GROUP BY id_libro
            ) AS subconsulta
          )
```

```
SELECT libros.titulo
        FROM libro INNER JOIN prestamo  pr ON libros.id_libro = prestamo.id_libro
        WHERE YEAR(pr.fecha_prestamo) = 2023
        GROUP BY libro.titulo
        ORDER BY COUNT(pr.id_prestamo) DESC LIMIT 1;
SELECT l.titulo
        FROM Libro l
        JOIN (
          SELECT id_libro
          FROM Prestamo
          WHERE YEAR(fecha_prestamo) = 2023
          GROUP BY id_libro
          HAVING COUNT(*) = (
            SELECT MAX(num_prestamos)
            FROM (
              SELECT COUNT(*) as num_prestamos
              FROM Prestamo
              WHERE YEAR(fecha_prestamo) = 2023
              GROUP BY id_libro
            ) AS subconsulta
          )
        ) AS max_prestamos ON l.id_libro = max_prestamos.id_libro;
```

```
SELECT TOP 1 l.titulo
        FROM Libro l
        JOIN Prestamo p ON l.id_libro = p.id_libro
        WHERE YEAR(p.fecha_prestamo) = 2023
        GROUP BY l.titulo
        ORDER BY COUNT(*) DESC;
```

11. Dada la BD "Comercio" con las tablas:

```
Producto (idProducto, nombre, precio)
Venta(NroFac ,idProducto, cantidad)
```

Escribir un stored procedure que reciba como parámetro el ID de un producto y devuelva la cantidad total vendida de ese producto en todas las ventas, en una variable de salida.

```
CREATE PROCEDURE sp_CantidadTotalVendida
  @idProducto INT,
  @cantidadTotal INT OUTPUT
AS
BEGIN
  -- Inicializar la cantidad total
  SET @cantidadTotal = 0;
  -- Calcular la cantidad total vendida del producto
  SELECT @cantidadTotal = SUM(cantidad)
  FROM Venta
  WHERE idProducto = @idProducto;
  -- Manejar el caso en que no se encontraron ventas para el producto
  IF @cantidadTotal IS NULL
  BEGIN
    SET @cantidadTotal = 0;
  END
END;
```

12. Dada la siguiente estructura de tablas:

```
Usuario (idusuario, nya, fecha_ult_acceso, clave)
HistorialUsuario (id, idusuario, fecha, tipooperacion)
```

Se desea realizar lo siguiente: cada vez que un usuario acceda y cambie su clave, se debe registrar el movimiento realizado del cambio. En el caso que la clave sea igual a la anterior, se deberá denegar el cambio de clave. Sólo se podrá cambiar la clave, si transcurrieron 30 días del último cambio de clave. (tip! datediff(dd,fecha1,fecha2)= cantidad de días transcurridos entre 2 fechas)

```
CREATE TRIGGER TR_Usuario_clave ON Usuario AFTER UPDATE
AS
BEGIN
  DECLARE @idusuario INT, @claveAnterior NVARCHAR(50), @fechaUltimoCambio DATE, @nuevaClave NVARCHAR(50);
  SELECT @idusuario = idusuario, @claveAnterior = clave, @fechaUltimoCambio = fecha_ult_acceso FROM deleted;
  SELECT @nuevaClave = clave FROM inserted;
  IF @claveAnterior <> @nuevaClave
  BEGIN
    DECLARE @diasTranscurridos INT;
    SELECT @diasTranscurridos = DATEDIFF(dd, @fechaUltimoCambio, GETDATE());
    IF @diasTranscurridos >= 30
    BEGIN
      INSERT INTO HistorialUsuario (idusuario, fecha, tipooperacion)
      VALUES (@idusuario, GETDATE(), 'Cambio de clave');
    END
    ELSE
    BEGIN
      RAISERROR('No puede cambiar la clave. Deben pasar al menos 30 días desde el cambio anterior.', 16, 1);
      ROLLBACK TRANSACTION;
      RETURN;
    END
  END
END;
```