

# Práctica 4

## Explicación de Práctica: SQL (*Structured Query Language*)

Es el *lenguaje de consultas* de BD, compuesto por dos *submódulos*:

- **DDL (Data Definition Language)** → Módulo para *definición del modelo de datos*.
  - **CREATE / DROP SCHEMA** → Crea o elimina *BD*.
  - **CREATE / ALTER / DROP DOMAIN** → Crea, altera o elimina *dominios*.
  - **CREATE / ALTER / DROP TABLE** → Crea, altera o elimina *tablas*.
  - **CREATE / DROP VIEW** → Crea o elimina *vistas*.
  - **CREATE / DROP INDEX** → Crea o elimina *índices* en algunos DBMS.
- **DML (Data Manipulation Language)** → Módulo para la *operatoria normal de la BD*.
  - **SELECT** → Sentencia utilizada para *listar contenido de una o varias tablas*.
  - **INSERT** → Sentencia utilizada para *agregar contenido* en una tabla.
  - **UPDATE** → Sentencia utilizada para *actualizar contenido* de una tabla.
  - **DELETE** → Sentencia utilizada para *eliminar contenido* de una tabla.

### Operaciones DML:

- **SELECT campo/s**

**FROM tabla/s**

Ese es el formato básico de la instrucción, por ejemplo si tenemos la tabla *Alumno = (dni, nombre, apellido)*:

**SELECT nombre**

**FROM alumno**

Esto genera como resultado una tabla con los nombres de la tabla alumno. En el SELECT puedo utilizar **\***, que mostrará todos los campos de las

tablas involucradas sin necesidad de aclarar/escribir uno por uno. También se puede agregar la palabra **DISTINCT** que se aplicará a campos del SELECT y eliminará las tuplas repetidas, por ejemplo:

***SELECT DISTINCT nombre***

***FROM alumno***

En este caso, devuelve una tabla con todos los nombres de los alumnos pero sin repetición de nombre (si hay más de un "Juan" solo pone una existencia del mismo).

Los atributos utilizados en el SELECT de una consulta SQL pueden tener asociados operaciones válidas para sus dominios (el tipo de dato). Por ejemplo:

```
SELECT apellido, dni + 5000  
FROM alumno
```

Se muestran el apellido y el valor de dni de cada alumno sumándole 5000 al mismo.

- ***INSERT INTO ... VALUES***

Agrega tuplas/registros a una tabla. Si seguimos con el ejemplo de la tabla alumno:

***INSERT INTO alumno VALUES (2827893, 'Raul', 'Perez');***

Esta operación inserta una nueva tupla con los valores indicados luego de VALUES, que tienen que estar en el mismo orden en que son declarados en la tabla, en la tabla alumno.

- ***DELETE FROM***

Borra una tupla/registro o un conjunto de ellas de una tabla determinada, por ejemplo:

***DELETE FROM alumno WHERE dni = 22222222;***

Esto genera que se elimine de la tabla alumno toda aquella tupla que tenga el valor 22222222 en el campo dni.

- ***UPDATE ... SET***

Modifica el contenido de uno o varios atributos de una tabla, por ejemplo:

***UPDATE alumno SET nombre = 'Lorenzo' WHERE dni = 22232425;***

Esto modifica en la tabla alumno, la tupla que coincide con el dni 22232425, cambiándole el nombre a Lorenzo.

### **Operadores DML en SELECT:**

- → Hace comparaciones por "*igual a*", por ejemplo:

```
SELECT nombre, apellido  
FROM alumno  
WHERE (nombre = "Luciana")
```

- → Hace comparaciones por "*mayor a*", por ejemplo:

```
SELECT nombre  
FROM alumno  
WHERE (dni > 24564321)
```

- → Hace comparaciones por "*menor a*", por ejemplo:

```
SELECT nombre  
FROM alumno  
WHERE (dni < 24564321)
```

- → Hace comparaciones por "*mayor o igual*", por ejemplo:

```
SELECT nombre  
FROM alumno  
WHERE (dni ≥ 24564321)
```

- → Hace comparaciones por "*menor o igual*", por ejemplo:

```
SELECT nombre  
FROM alumno  
WHERE (dni ≤ 24564321)
```

- → Hace comparaciones por "*distinto a*", por ejemplo:

```
SELECT nombre  
FROM alumno  
WHERE (dni <> 24564321)
```

- → Toma *valores en un cierto rango* indicado, *incluyendo los extremos*, por ejemplo:

```
SELECT nombre  
FROM alumno  
WHERE (dni BETWEEN 30000000 and 40000000)
```

- **LIKE** → Brinda gran potencia para aquellas *consultas que requieren manejo de Strings*, comparando ciertos caracteres que puedan tener. Se puede combinar con:
  - **%** → Representa *cualquier cadena de caracteres*, inclusive la *cadena vacía*.
  - **\_** → Sustituye solo el carácter del lugar donde aparece.

Por ejemplo:

```
SELECT nombre, apellido
FROM alumno
WHERE (apellido LIKE '%or%')
```

En este caso, se seleccionan los nombres y apellidos de la tabla alumno, donde el apellido sea como aquello entre comillas, que contenga 'or'.

Otro ejemplo:

```
SELECT nombre, apellido
FROM alumno
WHERE (nombre LIKE '___')
```

En este caso, se seleccionan los nombres y apellidos de la tabla alumno, donde el nombre sea como aquello entre comillas, que tenga solo 3 caracteres.

- **IS NULL / IS NOT NULL** → Verifica si un **atributo contiene el valor de NULL**, valor que *se almacena por defecto si el usuario no define otro*. NULL es distinto de 0, no es un valor numérico. Por ejemplo:

```
SELECT nombre, apellido
FROM alumno
WHERE (nombre IS NOT NULL)
```

Se filtran las tuplas donde el nombre no es nulo, es decir donde ese atributo tiene un valor asignado.

- **PRODUCTO CARTESIANO ( , )** → Se utiliza una coma para indicar este producto, basta con poner en la cláusula FROM dos o más tablas separadas por comas. Por ejemplo:

```
SELECT *
FROM alumno, localidad
```

Se mostrará como resultado una tabla con todas las combinaciones de tuplas entre las tablas alumno y localidad.

- **RENOMBRE** → Se pueden renombrar atributos y tablas, por ejemplo:

```
SELECT a.nombre as 'Nombre alumno', l.nombre as 'Nombre localidad'  
FROM alumno a, localidad l
```

El **AS** se utiliza para el *renombre de los atributos* de la manera '*nombre atributo AS renombre*'. Para *renombrar las tablas* simplemente se pone en el FROM el *nombre original de la tabla seguido de su renombre*.

- **PRODUCTO NATURAL** → Se pueden realizar distintos tipos de producto según la combinación de tuplas que se realice entre tablas.

- **INNER JOIN** → Producto natural, reúne las *tuplas de las relaciones que tienen sentido*. El producto natural se realiza en la cláusula FROM indicando las *tablas involucradas* en dicho producto, y luego de la sentencia ON la condición que debe cumplirse.

```
SELECT a.nombre, a.apellido, e.nota  
FROM alumno a INNER JOIN examen e ON (a.dni = e.dni)
```

Se reúnen las tuplas de las tablas alumno y examen que coinciden en el valor de atributo dni.

- **NATURAL JOIN** → Análogo al producto natural de AR, trabaja por equicombinación.
- **LEFT JOIN** → Contiene *todos los registros de la tabla de la izquierda, aún cuando no exista un registro correspondiente en la tabla de la derecha*, para uno de la izquierda. Retorna un *valor nulo (NULL) en caso de no correspondencia*.

```
SELECT *  
FROM alumno a LEFT JOIN examen e ON (a.dni = e.dni)
```

Se reúnen todas las tuplas que tienen correspondencia con el valor dni en las tablas alumno y examen, pero agrega todas las tuplas que estén en la tabla alumno aunque no tenga correspondencia en la tabla examen.

- **RIGHT JOIN** → Es la inversa del LEFT JOIN (contiene todos los *registros de la tabla de la derecha, aún cuando no exista registro correspondiente en la tabla de la izquierda*, para uno de la derecha).

```
SELECT *  
FROM alumno a RIGHT JOIN examen e ON (a.dni = e.dni)
```

Se reúnen todas las tuplas que tienen correspondencia con el valor dni en las tablas alumno y examen, pero agrega todas las tuplas que estén en la tabla examen aunque no tengan correspondencia en la tabla alumno.

- **UNION** → Misma interpretación que en AR. *No retorna tuplas duplicadas.*  
IMPORTANTE: Las consultas a unir *deben tener esquemas compatibles*.

```
SELECT nombre  
FROM alumno  
WHERE (dni > 25000000)
```

#### **UNION**

```
SELECT nombre  
FROM materia
```

Se unen los nombres de alumno que tengan dni 25000000 con los nombres de las materias (sin repetir tuplas).

- **UNION ALL** → Misma interpretación que la UNION pero *retorna las tuplas duplicadas*. IMPORTANTE: Las consultas a unir *deben tener esquemas compatibles*.
- **EXCEPT** → Cláusula definida para la *diferencia de conjuntos*. IMPORTANTE: Ambas consultas *deben tener esquemas compatibles*.

```
SELECT dni  
FROM alumno  
WHERE (dni > 25000000)  
EXCEPT  
(SELECT dni  
FROM examen)
```

En este caso, nos quedamos con los dni de los alumnos que sean mayores a 25000000, exceptuando aquellos que aparezcan en los dni de la tabla examen.

- **INTERSECT** → Cláusula para la *operación de intersección* (se intersectan dos o más tablas y se obtienen las tuplas que tienen en común, que se encuentran en ambas).
- **ORDER BY** → Permite *ordenar las tuplas resultantes por el atributo que se le indique*. Por defecto ordena de *menor a mayor* (operador ASC). Si se desea

ordenar de *mayor a menor*, se utiliza el operador **DESC**.

```
SELECT nombre, apellido, dni  
FROM alumno  
WHERE (dni > 23000000) and (nombre = 'Luciana')  
ORDER BY apellido, dni DESC
```

Dentro de la cláusula ORDER BY **se pueden indicar más de un criterio de ordenación**. El segundo criterio se *aplica en caso de empate en el primero y así sucesivamente*.

- **Funciones de Agregación** → Se utilizan en el **SELECT** y *operan sobre un conjunto de tuplas de entrada produciendo un único valor de salida*.
  - **AVG** → *Promedio del atributo indicado para todas las tuplas del conjunto.*
  - **COUNT** → *Cantidad de tuplas involucradas en el conjunto de entrada.*
  - **MAX** → *Valor más grande dentro del conjunto de tuplas para el atributo indicado.*
  - **MIN** → *Valor más pequeño dentro del conjunto de tuplas para el atributo indicado.*
  - **SUM** → *Suma del valor del atributo indicado para todas las tuplas del conjunto.*
- **Agrupamiento**
  - **GROUP BY** → *Agrupa las tuplas de una consulta por algún criterio con el objetivo de aplicar alguna función de agregación.*

```
SELECT nombre, apellido, AVG (nota) as promedio  
FROM alumno a INNER JOIN examen e ON (a.dni = e.dni)  
GROUP BY e.dni, nombre, apellido
```

Qué información se puede mostrar cuando se realizó un agrupamiento?  
Porque es importante agrupar además por PK?
  - **HAVING** → La cláusula **HAVING** se usa con la cláusula **GROUP BY** para *restringir los grupos que aparecen en la tabla de resultados mediante alguna condición que deben cumplir los grupos.*

```
SELECT nombre, apellido, AVG (nota) as promedio  
FROM alumno a INNER JOIN examen e ON (a.dni = e.dni)
```

GROUP BY e.dni, nombre, apellido  
HAVING AVG (nota)  $\geq$  6

- **Subconsultas**

Consiste en *ubicar una consulta SQL dentro de otra*. SQL define *operadores de comparación para subconsultas*:

- = (igualdad) → Cuando una *subconsulta retorna un único resultado*, es posible *compararlo contra un valor simple*.

- IN (pertenencia) → Comprueba *si un elemento es parte o no de un conjunto*. Negación ( NOT IN ).

```
SELECT nombre, apellido  
FROM alumno  
WHERE dni IN (SELECT dni  
               FROM examen  
               WHERE nota < 4)
```

- = SOME → *Igual a alguno.*
- > ALL → *Mayor que todos.*
- ≤ SOME → *Menor o igual que alguno.*

- EXISTS → Permite *comprobar si una subconsulta generó o no alguna tupla como respuesta*. El resultado de la cláusula EXISTS es *verdadero si la subconsulta tiene al menos una tupla, y falso en caso contrario*. Negación ( NOT EXISTS ).

```
SELECT dni, nombre, apellido  
FROM alumno a  
WHERE EXISTS (SELECT *  
               FROM examen e  
               WHERE (a.dni = e.dni) and (e.nota = 10))
```

### Equivalencias AR - SQL:

- **T1 |x| T2 ≡ SELECT \* FROM T1 NATURAL JOIN T2** (equicombinación de tuplas de T1 y T2 *sobre la foreign key de una que refiere a la primary key de la otra*).
- **T1 |x| T2 ≡ SELECT \* FROM T1 INNER JOIN T2 ON (...)** (equicombinación de tablas *sobre los campos que se especifiquen*, cuando se quieren igualar

*campos que no coincidan con la primary key de una y la foreign key de la otra).*

- $T1 | \times | \theta T2 \equiv \text{SELECT * FROM } T1 \text{ NATURAL JOIN } T2 \text{ WHERE } .$
- $T1 | \times | T2 \equiv \text{SELECT * FROM } T1 \text{ LEFT JOIN } T2 \text{ ON } (. . .).$
- $T1 - T2 \equiv T1 \text{ EXCEPT } T2.$

**Ejemplos:**

**Sucursal** = (nombreSucursal, ciudadSucursal, activo).

**Cliente** = (codCliente, dni, nombCliente, dirCliente, viveCliente).

**Prestamo** = (nroPrestamo, importe, nombreSucursal(fk)).

**PropietarioPrestamo** = (codCliente(fk), nroPrestamo(fk)).

**Cuenta** = (nroCuenta, saldo, nombreSucursal(fk)).

**PropietarioCuenta** = (nroCuenta(fk), codCliente(fk)).

1. Clientes con cuentas o préstamos en cualquier sucursal.

```
(Select dni, nombCliente  
From PropietarioCuenta natural join cliente)  
Union  
(Select dni, nombCliente  
From PropietarioPrestamo natural join cliente)
```

2. Clientes con cuentas y préstamos en cualquier sucursal.

```
(Select dni, nombCliente  
From PropietarioCuenta natural join cliente)  
Intersect  
(Select dni, nombCliente  
From PropietarioPrestamo natural join cliente)
```

---

*Alternativa a  $\cap$ :*

```
Select distinct nombCliente  
From propietarioprestamo inner join cliente on  
(propietarioprestamo.codcliente = cliente.codcliente)  
Where codcliente in (Select codcliente  
From propietariocuenta)
```

3. Clientes con cuentas y sin préstamos.

```
(Select dni, nombCliente  
From PropietarioCuenta natural join cliente)
```

**Except**

**(Select dni, nombCliente  
From PropietarioPrestamo natural join cliente)**

4. Cantidad de cuentas con saldo mayor a \$50000.

**Select COUNT(nroCuenta) as cantCuentas** ; podria contar cualquier campo  
**From Cuenta**  
**Where saldo > 50000**

5. Saldo promedio de las cuentas de la sucursal 'X'.

**Select AVG(saldo) as Promedio  
From Cuenta  
Where nombreSucursal = 'X'**

6. Importe del mayor préstamo otorgado por la sucursal 'Y'.

**Select MAX(importe) as Maximo  
From Prestamo  
Where nombreSucursal = 'Y'**

7. Importe total asignado a préstamos.

**Select Sum(importe) as importe  
From Prestamo**

8. Obtener saldo promedio de las cuentas de c/ sucursal.

**Select nombreSucursal, avg(saldo) as saldoProm  
From cuenta  
Group by nombreSucursal**

9. Presentar las sucursales y su saldo promedio siempre y cuando supere los \$200.000.

**Select nombreSucursal, avg(saldo)  
From cuenta  
Group by nombreSucursal  
Having avg(saldo) > 200000**

10. Contar el N° de clientes que tienen cuentas de cada sucursal.

**Select nombreSucursal, count(distinct codCliente)  
From cuenta inner join propietariocuenta on (cuenta.nroCuenta =**

propietariocuenta.nroCuenta)

**Group by** nombreSucursal

11. Saldo promedio de las cuentas de c/ cliente que vive en La Plata y tiene al menos 3 cuentas.

**Select** nombCliente, **avg**(saldo)

**From** propietariocuenta **inner join** cuenta **on** (propietariocuenta.nroCuenta = cuenta.nroCuenta) **Inner join** cliente **on** (propietariocuenta.codcliente = cliente.codCliente)

**Where** viveCliente = "La Plata"

**Group by** propietariocuenta.codcliente, cliente.nombCliente

**Having count** (propietariocuenta.nroCuenta)  $\geq 3$

12. Clientes con préstamos y cuentas en una misma sucursal de "La Plata".

**Select distinct** nombCliente

**From** propietarioprestamo pp **inner join** prestamo p **on** (pp.nroPréstamo = p.nroPrestamo) **inner join** sucursal s **on** (p.nombreSucursal = s.nombreSucursal) **inner join** cliente c **on** (pp.codCliente = c.codCliente)

**Where** s.ciudadSucursal = "La Plata" **and** c.codCliente **in** (**Select** codCliente

**From**

propietariocuenta pc **inner**

**join** cuenta c **on** (pc.nroCuenta = c.nroCuenta)

**Where**

c.nombreSucursal =

s.nombreSucursal)  $\rightarrow$  s

corresponde a la consulta ppal.

13. Presentar las sucursales que tengan activo mayor que alguna otra.

**Select** nombreSucursal

**From** sucursal

**Where** activo > **some** (**Select** activo

**From** sucursal)

14. Presentar la sucursal que tenga activo superior a todas.

**Select** nombreSucursal

**From** sucursal

**Where** activo  $\geq$  **all** (**Select** activo

**From** sucursal)

15. Encontrar la sucursal que tiene el mayor saldo promedio.

```
Select nombreSucursal  
From cuenta  
Group by nombreSucursal  
Having avg (saldo)  $\geq$  all (Select avg (saldo)  
                 From cuenta  
                 Group by nombreSucursal)
```

16. Cliente con cuentas en todas las sucursales.

```
Select nombCliente  
From cliente  
Where not exists (Select *  
                 From sucursal s  
                 Where not exists (Select *  
                    From propietariocuenta pc inner join  
                    cuenta c on  
                    c.nroCuenta)  
                    Where c.nombreSucursal =  
                    s.nombreSucursal and  
                    pc.codCliente  
                    = cliente.codCliente)) → corresponden  
                    a las consultas de nivel superior.
```

## Ejercicios

Dadas las siguientes relaciones, resolver utilizando SQL las consultas planteadas.

1. **Cliente** = (idCliente, nombre, apellido, DNI, telefono, direccion).

**Factura** = (nroTicket, total, fecha, hora, idCliente (fk)).

**Detalle** = (nroTicket (fk), idProducto (fk), cantidad, preciounitario).

**Producto** = (idProducto, nombreP, descripcion, precio, stock).

1. Listar datos personales de clientes cuyo apellido comience con el string 'Pe'. Ordenar por DNI.

```
SELECT nombre, apellido, DNI, teléfono, dirección
```

```
FROM Cliente  
WHERE (apellido LIKE 'Pe%')  
ORDER BY DNI
```

2. Listar nombre, apellido, DNI, teléfono y dirección de clientes que realizaron compras solamente durante 2024.  

```
(SELECT nombre, apellido, DNI, teléfono, dirección  
FROM Cliente natural join Factura  
WHERE (YEAR (fecha) = 2024))  
EXCEPT  
(SELECT nombre, apellido, DNI, teléfono, dirección  
FROM Cliente natural join Factura  
WHERE (YEAR (fecha) <> 2024))
```
3. Listar nombre, descripción, precio y stock de productos vendidos al cliente con DNI 45789456, pero que no fueron vendidos a clientes de apellido 'García'.  

```
(SELECT p.nombreP, p.descripcion, p.precio, p.stock  
FROM Cliente c INNER JOIN Factura f ON (c.idCliente = f.idCliente)  
INNER JOIN Detalle d ON (f.nroTicket = d.nroTicket) INNER JOIN  
Producto p ON (d.idProducto = p.idProducto)  
WHERE (c.DNI = 45789456))  
EXCEPT  
(SELECT p.nombreP, p.descripcion, p.precio, p.stock  
FROM Cliente c INNER JOIN Factura f ON (c.idCliente = f.idCliente)  
INNER JOIN Detalle d ON (f.nroTicket = d.nroTicket) INNER JOIN  
Producto p ON (d.idProducto = p.idProducto)  
WHERE (c.apellido = 'García'))
```
4. Listar nombre, descripción, precio y stock de productos no vendidos a clientes que tengan teléfono con característica 221 (la característica está al comienzo del teléfono). Ordenar por nombre.  

```
SELECT nombreP, descripción, precio, stock  
FROM Producto p  
WHERE NOT EXISTS (SELECT *  
FROM Cliente c INNER JOIN Factura f ON  
(c.idCliente = f.idCliente) INNER JOIN Detalle d ON  
(f.nroTicket = d.nroTicket)  
WHERE (c.teléfono LIKE '221%') AND (d.idProducto
```

```
= p.idProducto))  
ORDER BY nombreP
```

5. Listar para cada producto nombre, descripción, precio y cuantas veces fue vendido. Tenga en cuenta que puede no haberse vendido nunca el producto.

```
SELECT p.nombreP, p.descripcion, p.precio, COUNT (d.idProducto) AS  
vecesVendidas  
FROM Producto p LEFT JOIN Detalle d ON (p.idProducto =  
d.idProducto)  
GROUP BY p.idProducto, p.nombreP, p.descripcion, p.precio
```

6. Listar nombre, apellido, DNI, teléfono y dirección de clientes que compraron los productos con nombre 'prod1' y 'prod2' pero nunca compraron el producto con nombre 'prod3'.

```
((SELECT c.nombre, c.apellido, c.DNI, c.telefono, c.direccion  
FROM Cliente c INNER JOIN Factura f ON (c.idCliente = f.idCliente)  
INNER JOIN Detalle d ON (f.nroTicket = d.nroTicket) INNER JOIN  
Producto p ON (d.idProducto = p.idProducto)  
WHERE (p.nombreP = 'prod1'))
```

INTERSECT

```
(SELECT c.nombre, c.apellido, c.DNI, c.telefono, c.direccion  
FROM Cliente c INNER JOIN Factura f ON (c.idCliente = f.idCliente)  
INNER JOIN Detalle d ON (f.nroTicket = d.nroTicket) INNER JOIN  
Producto p ON (d.idProducto = p.idProducto)  
WHERE (p.nombreP = 'prod2'))
```

EXCEPT

```
(SELECT c.nombre, c.apellido, c.DNI, c.telefono, c.direccion  
FROM Cliente c INNER JOIN Factura f ON (c.idCliente = f.idCliente)  
INNER JOIN Detalle d ON (f.nroTicket = d.nroTicket) INNER JOIN  
Producto p ON (d.idProducto = p.idProducto)  
WHERE (p.nombreP = 'prod3'))
```

7. Listar nroTicket, total, fecha, hora y DNI del cliente, de aquellas facturas donde se haya comprado el producto 'prod38' o la factura tenga fecha de 2023.

```
SELECT f.nroTicket, f.total, f.fecha, f.hora, c.DNI  
FROM Cliente c INNER JOIN Factura f ON (c.idCliente = f.idCliente)  
WHERE (YEAR (f.fecha) = 2023) OR (f.nroTicket IN (SELECT d.nroTicket
```

```

        FROM Detalle d INNER
JOIN                                         Producto
p ON (d.idProducto =
p.idProducto)
WHERE (p.nombreP =
'prod38'))

```

8. Agregar un cliente con los siguientes datos: nombre: 'Jorge Luis', apellido: 'Castor', DNI: 40578999, teléfono: '221-4400789', dirección: '11 entre 500 y 501' nro: '2587' y el id de cliente: 500002. Se supone que el idCliente 500002 no existe.

```

INSERT INTO Cliente VALUES (500002, 'Jorge Luis', 'Castor', 40578999,
'221-4400789', '11 2587, entre 500 y 501')

```

9. Listar nroTicket, total, fecha, hora para las facturas del cliente 'Jorge Pérez' donde no haya comprado el producto 'Z'.

```

SELECT f.nroTicket, f.total, f.fecha, f.hora
FROM Factura f INNER JOIN Cliente c ON (f.idCliente = c.idCliente)
WHERE ((c.nombre = 'Jorge') AND (c.apellido = 'Pérez'))
      AND NOT EXISTS (SELECT *
                        FROM Detalle d INNER
JOIN Producto p
ON
(d.idProducto = p.idProducto)
WHERE ((d.nroTicket =
f.nroTicket)
      AND
(p.nombreP = 'Z'))))

```

10. Listar DNI, apellido y nombre de clientes donde el monto total comprado, teniendo en cuenta todas sus facturas, supere \$100000.

```

SELECT c.DNI, c.apellido, c.nombre
FROM Cliente c INNER JOIN Factura f ON (c.idCliente = f.idCliente)
GROUP BY c.idCliente, c.DNI, c.apellido, c.nombre
HAVING (SUM (f.total) > 100000)

```

2. ***Localidad*** = (codigoPostal, nombreL, descripcion, nroHabitantes).

***Arbol*** = (nroArbol, especie, anios, calle, nro, codigoPostal(fk)).

***Podador*** = (DNI, nombre, apellido, telefono, fnac, codigoPostalVive(fk)).

***Poda*** = (codPoda, fecha, DNI(fk), nroArbol(fk)).

1. Listar especie, años, calle, nro y localidad de árboles podados por el podador 'Juan Perez' y por el podador 'Jose Garcia'.

```
(SELECT a.especie, a.anios, a.calle, a.nro, l.nombreL  
FROM Árbol a INNER JOIN Localidad l ON (a.códigoPostal =  
l.códigoPostal) INNER JOIN Poda p ON (a.nroArbol = p.nroArbol) INNER  
JOIN Podador pod ON (p.DNI = pod.DNI)  
WHERE ((pod.nombre = 'Juan') AND (pod.apellido = 'Perez')))  
INTERSECT  
(SELECT a.especie, a.anios, a.calle, a.nro, l.nombreL  
FROM Árbol a INNER JOIN Localidad l ON (a.códigoPostal =  
l.códigoPostal) INNER JOIN Poda p ON (a.nroArbol = p.nroArbol) INNER  
JOIN Podador pod ON (p.DNI = pod.DNI)  
WHERE ((pod.nombre = 'José') AND (pod.apellido = 'García')))
```

2. Reportar DNI, nombre, apellido, fecha de nacimiento y localidad donde viven de aquellos podadores que tengan podas realizadas durante 2023.

```
SELECT DISTINCT pod.DNI, pod.nombre, pod.apellido, pod.fnac,  
l.nombreL  
FROM Poda p NATURAL JOIN Podador pod INNER JOIN Localidad l ON  
(pod.códigoPostalVive = l.códigoPostal)  
WHERE (YEAR (p.fecha) = 2023)
```

3. Listar especie, años, calle, nro y localidad de árboles que no fueron podados nunca.

```
SELECT a.especie, a.anios, a.calle, a.nro, l.nombreL  
FROM Árbol a NATURAL JOIN Localidad l  
WHERE (a.nroÁrbol NOT IN (SELECT DISTINCT p.nroÁrbol  
FROM Poda p))
```

4. Reportar especie, años, calle, nro y localidad de árboles que fueron podados durante 2022 y no fueron podados durante 2023.

```
(SELECT a.especie, a.anios, a.calle, a.nro, l.nombreL  
FROM Árbol a INNER JOIN Localidad l ON (a.códigoPostal =  
l.códigoPostal) INNER JOIN Poda p ON (a.nroÁrbol = p.nroÁrbol)  
WHERE (YEAR (p.fecha) = 2022))  
EXCEPT  
(SELECT a.especie, a.anios, a.calle, a.nro, l.nombreL  
FROM Árbol a INNER JOIN Localidad l ON (a.códigoPostal =
```

```
I.códigoPostal) INNER JOIN Poda p ON (a.nroÁrbol = p.nroÁrbol)
WHERE (YEAR (p.fecha) = 2023))
```

5. Reportar DNI, nombre, apellido, fecha de nacimiento y localidad donde viven de aquellos podadores con apellido terminado con el string 'ata' y que tengan al menos una poda durante 2024. Ordenar por apellido y nombre.

```
SELECT DISTINCT pod.DNI, pod.nombre, pod.apellido, pod.fnac,
l.nombreL
FROM Podador pod INNER JOIN Localidad l ON (pod.códigoPostalVive
= l.códigoPostal)
WHERE (pod.apellido LIKE '%ata') AND EXISTS (SELECT *
                                              FROM Poda p
                                              WHERE ((p.DNI = pod.DNI)
AND
                                              (YEAR
(p.fecha) = 2024)))
ORDER BY pod.apellido, pod.nombre
```

6. Listar DNI, apellido, nombre, teléfono y fecha de nacimiento de podadores que solo podaron árboles de especie 'Coníferas'.  
(SELECT pod.DNI, pod.apellido, pod.nombre, pod.teléfono, pod.fnac  
FROM Podador pod INNER JOIN Poda p ON (pod.DNI = p.DNI) INNER  
JOIN Árbol a ON (p.nroÁrbol = a.nroÁrbol)  
WHERE (a.especie = 'Coníferas'))  
EXCEPT  
(SELECT pod.DNI, pod.apellido, pod.nombre, pod.teléfono, pod.fnac  
FROM Podador pod INNER JOIN Poda p ON (pod.DNI = p.DNI) INNER  
JOIN Árbol a ON (p.nroÁrbol = a.nroÁrbol)  
WHERE (a.especie <> 'Coníferas'))

7. Listar especies de árboles que se encuentren en la localidad de 'La Plata' y también en la localidad de 'Salta'.  
(SELECT a.especie  
FROM Árbol a NATURAL JOIN Localidad l  
WHERE (l.nombreL = 'La Plata'))
INTERSECT  
(SELECT a.especie  
FROM Árbol a NATURAL JOIN Localidad l  
WHERE (l.nombreL = 'Salta'))

8. Eliminar el podador con DNI 22234566.

```
DELETE FROM Podador  
WHERE (DNI = 22234566)
```

9. Reportar nombre, descripción y cantidad de habitantes de localidades que tengan menos de 5 árboles.

```
SELECT l.nombreL, l.descripcion, l.nroHabitantes  
FROM Localidad l LEFT JOIN Arbol a ON (l.codigoPostal =  
a.codigoPostal)  
GROUP BY l.codigoPostal, l.nombreL, l.descripcion, l.nroHabitantes  
HAVING (COUNT(a.nroArbol) < 5)
```

3. **Banda** = (codigoB, nombreBanda, genero\_musical, anio\_creacion).

**Integrante** = (DNI, nombre, apellido, dirección, email, fecha\_nacimiento, codigoB(fk)).

**Escenario** = (nroEscenario, nombre\_escenario, ubicación, cubierto, m2, descripción).

**Recital** = (fecha, hora, nroEscenario (fk), codigoB(fk)).

1. Listar DNI, nombre, apellido, dirección y email de integrantes nacidos entre 1980 y 1990 y que hayan realizado algún recital durante 2023.

```
SELECT DISTINCT DNI, nombre, apellido, dirección, email  
FROM Integrante i NATURAL JOIN Recital r  
WHERE ((YEAR (fecha_nacimiento) BETWEEN 1980 AND 1990) AND  
(YEAR (fecha) = 2023))
```

2. Reportar nombre, género musical y año de creación de bandas que hayan realizado recitales durante 2023, pero no hayan tocado durante 2022.

```
(SELECT nombreBanda, genero_musical, anio_creacion  
FROM Banda NATURAL JOIN Recital  
WHERE (YEAR (fecha) = 2023))  
EXCEPT  
(SELECT nombreBanda, genero_musical, anio_creacion  
FROM Banda NATURAL JOIN Recital  
WHERE (YEAR (fecha) = 2022))
```

3. Listar el cronograma de recitales del día 04/12/2023. Se deberá listar nombre de la banda que ejecutará el recital, fecha, hora, y el nombre y

ubicación del escenario correspondiente.

```
SELECT b.nombreBanda, r.fecha, r.hora, e.nombre_escenario,  
e.ubicación  
FROM Banda b INNER JOIN Recital r ON (b.códigoB = r.códigoB) INNER  
JOIN Escenario e ON (r.nroEscenario = e.nroEscenario)  
WHERE (r.fecha = '04/12/2023')
```

4. Listar DNI, nombre, apellido, email de integrantes que hayan tocado en el escenario con nombre 'Gustavo Cerati' y en el escenario con nombre 'Carlos Gardel'.

```
(SELECT i.DNI, i.nombre, i.apellido, i.email  
FROM Integrante i INNER JOIN Recital r ON (i.códigoB = r.códigoB)  
INNER JOIN Escenario e ON (r.nroEscenario = e.nroEscenario)  
WHERE (e.nombre_escenario = 'Gustavo Cerati'))  
INTERSECT  
(SELECT i.DNI, i.nombre, i.apellido, i.email  
FROM Integrante i INNER JOIN Recital r ON (i.códigoB = r.códigoB)  
INNER JOIN Escenario e ON (r.nroEscenario = e.nroEscenario)  
WHERE (e.nombre_escenario = 'Carlos Gardel'))
```

5. Reportar nombre, género musical y año de creación de bandas que tengan más de 5 integrantes.

```
SELECT nombreBanda, género_musical, anio_creación  
FROM Banda NATURAL JOIN Integrante  
GROUP BY códigoB, nombreBanda, género_musical, anio_creación  
HAVING (COUNT (*) > 5)
```

6. Listar nombre de escenario, ubicación y descripción de escenarios que solo tuvieron recitales con el género musical rock and roll. Ordenar por nombre de escenario.

```
SELECT e.nombre_escenario, e.ubicación, e.descripcion  
FROM Escenario e  
WHERE NOT EXISTS (SELECT *  
FROM Banda b INNER JOIN Recital r ON (b.códigoB  
= r.códigoB)  
WHERE ((r.nroEscenario = e.nroEscenario) AND  
(género_musical  
<> 'rock and roll')))  
ORDER BY e.nombre_escenario
```

7. Listar nombre, género musical y año de creación de bandas que hayan realizado recitales en escenarios cubiertos durante 2023. //cubierto es

*true, false según corresponda.*

```
SELECT b.nombreBanda, b.género_musical, b.anio_creación  
FROM Banda b INNER JOIN Recital r ON (b.códigoB = r.códigoB) INNER  
JOIN Escenario e ON (r.nroEscenario = e.nroEscenario)  
WHERE ((YEAR (r.fecha) = 2023) AND (e.cubierto = true))
```

8. Reportar para cada escenario, nombre del escenario y cantidad de recitales durante 2024.

```
SELECT e.nombre_escenario, COUNT (r.fecha) AS cantRecitales2024  
FROM Escenario e LEFT JOIN Recital r ON (e.nroEscenario =  
r.nroEscenario AND YEAR (r.fecha) = 2024)  
GROUP BY e.nroEscenario, e.nombre_escenario
```

9. Modificar el nombre de la banda 'Mempis la Blusera' a: 'Memphis la Blusera'.

```
UPDATE Banda  
SET nombreBanda = 'Memphis la Blusera'  
WHERE (nombreBanda = 'Mempis la Blusera')
```

4. **Persona** = (DNI, Apellido, Nombre, Fecha\_Nacimiento, Estado\_Civil, Genero).

**Alumno** = (DNI (fk), Legajo, Anio\_Ingreso).

**Profesor** = (DNI (fk), Matricula, Nro\_Expediente).

**Titulo** = (Cod\_Titulo, Nombre, Descripcion).

**Titulo-Profesor** = (Cod\_Titulo (fk), DNI (fk), Fecha).

**Curso** = (Cod\_Curso, Nombre, Descripcion, Fecha\_Creacion, Duracion).

**Alumno-Curso** = (DNI (fk), Cod\_Curso (fk), Anio, Desempenio, Calificacion).

**Profesor-Curso** = (DNI (fk), Cod\_Curso (fk), Fecha\_Desde, Fecha\_Hasta?).

1. Listar DNI, legajo y apellido y nombre de todos los alumnos que tengan año de ingreso inferior a 2014.

```
SELECT DNI, Legajo, Apellido, Nombre  
FROM Alumno NATURAL JOIN Persona  
WHERE (Anio_Ingreso < 2014)
```

2. Listar DNI, matrícula, apellido y nombre de los profesores que dictan cursos que tengan más de 100 horas de duración. Ordenar por DNI.

```
SELECT DISTINCT p.DNI, p.Matrícula, per.Apellido, per.Nombre  
FROM Profesor p INNER JOIN Persona per ON (p.DNI = per.DNI) INNER  
JOIN Profesor-Curso prof ON (p.DNI = prof.DNI) INNER JOIN Curso c  
ON (prof.Cod_Curso = c.Cod_Curso)  
WHERE (c.Duración > 100)  
ORDER BY p.DNI
```

3. Listar el DNI, Apellido, Nombre, Género y Fecha de nacimiento de los alumnos inscriptos al curso con nombre "Diseño de Bases de Datos" en 2023.

```
SELECT DISTINCT p.DNI, p.Apellido, p.Nombre, p.Género,  
p.Fecha_Nacimiento  
FROM Persona p INNER JOIN Alumno-Curso ac ON (p.DNI = ac.DNI)  
INNER JOIN Curso c ON (ac.Cod_Curso = c.Cod_Curso)  
WHERE ((c.Nombre = 'Diseño de Bases de Datos') AND (ac.Año =  
2023))
```

4. Listar el DNI, Apellido, Nombre y Calificación de aquellos alumnos que obtuvieron una calificación superior a 8 en algún curso que dicta el profesor "Juan Garcia". Dicho listado deberá estar ordenado por Apellido y nombre.

```
SELECT DISTINCT alu.DNI, alu.Apellido, alu.Nombre, ac.Calificación  
FROM Persona alu INNER JOIN Alumno-Curso ac ON (alu.DNI = ac.DNI)  
INNER JOIN Profesor-Curso pc ON (ac.Cod_Curso = pc.Cod_Curso)  
INNER JOIN Persona prof ON (pc.DNI = prof.DNI)  
WHERE ((ac.Calificación > 8) AND (prof.Nombre = 'Juan') AND  
(prof.Apellido = 'García'))  
ORDER BY alu.Apellido, alu.Nombre
```

5. Listar el DNI, Apellido, Nombre y Matrícula de aquellos profesores que posean más de 3 títulos. Dicho listado deberá estar ordenado por Apellido y Nombre.

```
SELECT p.DNI, per.Apellido, per.Nombre, p.Matrícula  
FROM Persona per INNER JOIN Profesor p ON (per.DNI = p.DNI) INNER  
JOIN Título-Profesor tp ON (p.DNI = tp.DNI)  
GROUP BY p.DNI, per.Apellido, per.Nombre, p.Matrícula  
HAVING (COUNT (tp.Cod_Título) > 3)  
ORDER BY per.Apellido, per.Nombre
```

6. Listar el DNI, Apellido, Nombre, Cantidad de horas y Promedio de horas que dicta cada profesor. La cantidad de horas se calcula como la suma

de la duración de todos los cursos que dicta.

```
SELECT p.DNI, p.Apellido, p.Nombre, SUM (c.Duración) AS  
Cantidad_horas, AVG (c.Duración) AS Promedio_horas  
FROM Persona p INNER JOIN Profesor-Curso pc ON (p.DNI = pc.DNI)  
INNER JOIN Curso c ON (pc.Cod_Curso = c.Cod_Curso)  
GROUP BY p.DNI, p.Apellido, p.Nombre
```

7. Listar Nombre y Descripción del curso que posea más alumnos inscriptos y del que posea menos alumnos inscriptos durante 2024.

```
SELECT c.Nombre, c.Descripción  
FROM Curso c LEFT JOIN Alumno-Curso ac ON (c.Cod_Curso =  
ac.Cod_Curso AND ac.Año = 2024)  
GROUP BY c.Cod_Curso, c.Nombre, c.Descripción  
HAVING (COUNT (ac.DNI) ≥ ALL (SELECT COUNT (DNI)  
FROM Alumno-Curso  
WHERE (Año = 2024)  
GROUP BY Cod_Curso)  
OR COUNT (ac.DNI) ≤ ALL (SELECT COUNT (DNI)  
FROM Alumno-Curso  
WHERE (Año = 2024)  
GROUP BY Cod_Curso)))
```

8. Listar el DNI, Apellido, Nombre y Legajo de alumnos que realizaron cursos con nombre conteniendo el string 'BD' durante 2022 pero no realizaron ningún curso durante 2023.

```
(SELECT a.DNI, p.Apellido, p.Nombre, a.Legajo  
FROM Alumno a INNER JOIN Persona p ON (a.DNI = p.DNI) INNER JOIN  
Alumno-Curso ac ON (a.DNI = ac.DNI) INNER JOIN Curso c ON  
(ac.Cod_Curso = c.Cod_Curso)  
WHERE ((c.Nombre LIKE '%BD%') AND (ac.Año = 2022)))  
EXCEPT  
(SELECT a.DNI, p.Apellido, p.Nombre, a.Legajo  
FROM Alumno a INNER JOIN Persona p ON (a.DNI = p.DNI) INNER JOIN  
Alumno-Curso ac ON (a.DNI = ac.DNI) INNER JOIN Curso c ON  
(ac.Cod_Curso = c.Cod_Curso)  
WHERE (ac.Año = 2023))
```

9. Agregar un profesor con los datos que prefiera y agregarle el título con código: 25.

```
INSERT INTO Persona VALUES (12345678, 'Perez', 'Lucas',
```

```
'09/08/1980', 'Soltero', 'Masculino')  
INSERT INTO Profesor VALUES (12345678, 1234, 2345)  
INSERT INTO Título VALUES (25, 'xxx', 'xxxxx')  
INSERT INTO Título-Profesor VALUES (25, 12345678, '10/04/2004')
```

10. Modificar el estado civil del alumno cuyo legajo es '2020/09', el nuevo estado civil es divorciado.

```
UPDATE Persona  
SET Estado_Civil = 'Divorciado'  
WHERE (DNI = (SELECT DNI  
FROM Alumno  
WHERE (Legajo = '2020/09')))
```

11. Dar de baja el alumno con DNI 30568989. Realizar todas las bajas necesarias para no dejar el conjunto de relaciones en un estado inconsistente.

```
DELETE FROM Alumno-Curso  
WHERE (DNI = 30568989)  
DELETE FROM Alumno  
WHERE (DNI = 30568989)  
DELETE FROM Persona  
WHERE (DNI = 30568989)
```

5. **Agencia** = (razon\_social, dirección, telef, email).

**Ciudad** = (codigo\_postal, nombreCiudad, anioCreacion).

**Cliente** = (dni, nombre, apellido, telefono, direccion).

**Viaje** = (fecha, hora, dni (fk), cpOrigen(fk), cpDestino(fk), razon\_social(fk), descripcion). //cpOrigen y cpDestino corresponden a la ciudades origen y destino del viaje.

1. Listar razón social, dirección y teléfono de agencias que realizaron viajes desde la ciudad de 'La Plata' (ciudad origen) y que el cliente tenga apellido 'Roma'. Ordenar por razón social y luego por teléfono.

```
SELECT DISTINCT a.razón_social, a.dirección, a.telef  
FROM Agencia a INNER JOIN Viaje v ON (a.razón_social =  
v.razón_social) INNER JOIN Ciudad c ON (v.cpOrigen =  
c.código_postal) INNER JOIN Cliente cli ON (v.dni = cli.dni)
```

```
WHERE ((c.nombreCiudad = 'La Plata') AND (cli.apellido = 'Roma'))  
ORDER BY a.razón_social, a.telef
```

2. Listar fecha, hora, datos personales del cliente, nombres de ciudades origen y destino de viajes realizados en enero de 2019 donde la descripción del viaje contenga el String 'demorado'.

```
SELECT v.fecha, v.hora, cli.dni, cli.nombre, cli.apellido, cli.teléfono,  
cli.dirección, cOrigen.nombreCiudad, cDestino.nombreCiudad  
FROM Viaje v INNER JOIN Cliente cli ON (v.dni = cli.dni) INNER JOIN  
Ciudad cOrigen ON (v.cpOrigen = cOrigen.código_postal) INNER JOIN  
Ciudad cDestino ON (v.cpDestino = cDestino.código_postal)  
WHERE ((MONTH(v.fecha) = 01) AND (YEAR (v.fecha) = 2019) AND  
(v.descripcion LIKE '%demorado%'))
```

3. Reportar información de agencias que realizaron viajes durante 2019 o que tengan dirección de mail que termine con '@jmail.com'.

```
SELECT DISTINCT a.razón_social, a.dirección, a.telef, a.email  
FROM Agencia a LEFT JOIN Viaje v ON (a.razón_social = v.razón_social)  
WHERE ((YEAR (v.fecha) = 2019) OR (a.email LIKE '%@jmail.com'))
```

4. Listar datos personales de clientes que viajaron solo con destino a la ciudad de 'Coronel Brandsen'.

```
SELECT cli.dni, cli.nombre, cli.apellido, cli.teléfono, cli.dirección  
FROM Cliente cli  
WHERE NOT EXISTS (SELECT *  
                  FROM Viaje v INNER JOIN Ciudad c ON  
(v.cpDestino = c.código_postal)  
                  WHERE ((v.dni = cli.dni) AND (c.nombreCiudad <>  
'Coronel' Brandsen')))  
      AND EXISTS (SELECT *  
                  FROM Viaje v INNER JOIN Ciudad c ON (v.cpDestino  
= c.código_postal)  
                  WHERE ((v.dni = cli.dni) AND (c.nombreCiudad =  
'Coronel' Brandsen')))
```

5. Informar cantidad de viajes de la agencia con razón social 'TAXI Y' realizados a 'Villa Elisa'.

```
SELECT COUNT (*) AS cantidadDeViajes  
FROM Viaje v INNER JOIN Ciudad c ON (v.cpDestino = c.código_postal)  
WHERE ((v.razón_social = 'TAXI Y') AND (c.nombreCiudad = 'Villa  
Elisa'))
```

6. Listar nombre, apellido, dirección y teléfono de clientes que viajaron con todas las agencias.

```
SELECT cli.nombre, cli.apellido, cli.dirección, cli.teléfono  
FROM Cliente cli  
WHERE NOT EXISTS (SELECT *  
                   FROM Agencia a  
                   WHERE NOT EXISTS (SELECT *  
                                     FROM Viaje v  
                                     WHERE ((cli.dni = v.dni) AND  
(v.razón_social  
=  
a.razón_social))))
```

7. Modificar el cliente con DNI 38495444 actualizando el teléfono a '221-4400897'.

```
UPDATE Cliente  
SET teléfono = '221-4400897'  
WHERE (DNI = 38495444)
```

8. Listar razón social, dirección y teléfono de la/s agencias que tengan mayor cantidad de viajes realizados.

```
SELECT a.razón_social, a.dirección, a.telef  
FROM Agencia a INNER JOIN Viaje v ON (a.razón_social =  
v.razón_social)  
GROUP BY a.razón_social, a.dirección, a.telef  
HAVING (COUNT (*) ≥ ALL (SELECT COUNT (*)  
                           FROM Viaje v  
                           GROUP BY razón_social))
```

9. Reportar nombre, apellido, dirección y teléfono de clientes con al menos 5 viajes.

```
SELECT c.nombre, c.apellido, c.dirección, c.teléfono  
FROM Cliente c INNER JOIN Viaje v ON (c.dni = v.dni)  
GROUP BY c.dni, c.nombre, c.apellido, c.dirección, c.teléfono  
HAVING (COUNT * ≥ 5)
```

10. Borrar al cliente con DNI 40325692.

```
DELETE FROM Viaje  
WHERE (DNI = 40325692)  
DELETE FROM Cliente  
WHERE (DNI = 40325692)
```

6. **Técnico** = (codTec, nombre, especialidad) // técnicos.

**Repuesto** = (codRep, nombre, stock, precio) // repuestos.

**RepuestoReparacion** = (nroReparac (fk), codRep (fk), cantidad, precio) //repuestos utilizados en reparaciones.

**Reparación** = (nroReparac, codTec (fk), precio\_total, fecha) //reparaciones realizadas.

1. Listar los repuestos, informando el nombre, stock y precio. Ordenar el resultado por precio.

```
SELECT nombre, stock, precio  
FROM Repuesto  
ORDER BY precio
```

2. Listar nombre, stock y precio de repuestos que se usaron en reparaciones durante 2023 y que no se usaron en reparaciones del técnico 'José Gonzalez'.

```
SELECT r.nombre, r.stock, r.precio  
FROM Repuesto r INNER JOIN RepuestoReparación rr ON (r.codRep =  
rr.codRep) INNER JOIN Reparación repa ON (rr.nroReparac =  
repa.nroReparac)  
WHERE (YEAR (repa.fecha) = 2023) AND NOT EXISTS (SELECT *  
FROM Reparación r1  
INNER JOIN  
RepuestoReparación rr1 ON  
(r1.nroReparac = rr1.nroReparac)  
INNER JOIN Técnico t1 ON  
(r1.codTec = t1.codTec)  
WHERE (t1.nombre  
= 'José  
Gonzalez')  
AND (r.codRep = rr1.codRep))
```

3. Listar el nombre y especialidad de técnicos que no participaron en ninguna reparación. Ordenar por nombre ascendentemente.

```
SELECT t.nombre, t.especialidad  
FROM Técnico t  
WHERE NOT EXISTS (SELECT *  
FROM Reparación r)
```

```
        WHERE (t.codTec = r.codTec))  
ORDER BY t.nombre
```

4. Listar el nombre y especialidad de los técnicos que solamente participaron en reparaciones durante 2022.

```
SELECT t.nombre, t.especialidad  
FROM Técnico t INNER JOIN Reparación r ON (t.codTec = r.codTec)  
WHERE (YEAR (r.fecha) = 2022) AND NOT EXISTS (SELECT *  
                                              FROM Reparación  
                                              WHERE (codTec =  
t.codTec) AND  
(YEAR (fecha) <> 2022))
```

5. Listar para cada repuesto nombre, stock y cantidad de técnicos distintos que lo utilizaron. Si un repuesto no participó en alguna reparación igual debe aparecer en dicho listado.

```
SELECT r.nombre, r.stock, COUNT (DISTINCT repa.codTec) AS  
cantTécnicos  
FROM Repuesto r LEFT JOIN RepuestoReparación rr ON (r.codRep =  
rr.codRep) LEFT JOIN Reparación repa (rr.nroReparac =  
repa.nroReparac)  
GROUP BY r.codRep, r.nombre, r.stock
```

6. Listar nombre y especialidad del técnico con mayor cantidad de reparaciones realizadas y el técnico con menor cantidad de reparaciones.

```
SELECT t.nombre, t.especialidad  
FROM Técnico t LEFT JOIN Reparación r ON (t.codTec = r.codTec)  
GROUP BY t.codTec, t.nombre, t.especialidad  
HAVING (COUNT (r.nroReparac) ≥ ALL (SELECT COUNT (*)  
                                         FROM Reparación  
                                         GROUP BY codTec))  
OR COUNT (r.nroReparac) ≤ ALL (SELECT COUNT (*)  
                                         FROM Reparación  
                                         GROUP BY codTec))
```

7. Listar nombre, stock y precio de todos los repuestos con stock mayor a 0 y que dicho repuesto no haya estado en reparaciones con un precio total superior a \$10000.

```
SELECT r.nombre, r.stock, r.precio  
FROM Repuesto r
```

```

WHERE ((r.stock > 0) AND NOT EXISTS (SELECT *
                                         FROM Reparación rep INNER
                                         JOIN
                                         RepuestoReparación rr ON (rep.nroReparac =
                                         rr.nroReparac)
                                         WHERE ((precio_total > 10000)
                                         AND
                                         (rr.codRep =
                                         r.codRep)))

```

8. Proyectar número, fecha y precio total de aquellas reparaciones donde se utilizó algún repuesto con precio en el momento de la reparación mayor a \$10000 y menor a \$15000.

```

SELECT DISTINCT r.nroReparac, r.fecha, r.precio_total
FROM Reparación r INNER JOIN RepuestoReparación rr ON
(r.nroReparac = rr.nroReparac)
WHERE (rr.precio BETWEEN 10000 AND 15000)

```

9. Listar nombre, stock y precio de repuestos que hayan sido utilizados por todos los técnicos.

```

SELECT r.nombre, r.stock, r.precio
FROM Repuesto r
WHERE NOT EXISTS (SELECT *
                           FROM Técnico t
                           WHERE NOT EXISTS (SELECT *
                                             FROM RepuestoReparación rr
                                             INNER JOIN
                                             Reparación
                                             repa ON (rr.nroReparac =
                                             repa.nroReparac)
                                             WHERE ((repa.codTec =
                                             t.codTec) AND
                                             (rr.codRep = r.codRep))))

```

10. Listar fecha, técnico y precio total de aquellas reparaciones que necesitaron al menos 4 repuestos distintos.

```

SELECT repa.fecha, t.nombre, repa.precio_total
FROM Reparación repa INNER JOIN RepuestoReparación rr ON
(repa.nroReparac = rr.nroReparac) INNER JOIN Técnico t ON
(repa.codTec = t.codTec)
GROUP BY repa.nroReparac, repa.fecha, t.nombre, repa.precio_total
HAVING (COUNT (DISTINCT rr.codRep) ≥ 4)

```