Apuntes

ALGEBRA RELACIONAL:

1. **Select (σ):** El operador Select se utiliza para seleccionar tuplas de una relación que satisfacen una condición dada. La condición se puede especificar utilizando operadores de comparación como =, <>, <, <=, >, >=.
2. **Project (π):** El operador Project se utiliza para seleccionar ciertos atributos (columnas) de una relación.
3. **Union (∪):** El operador Union combina las tuplas de dos relaciones y elimina las duplicadas. Las dos relaciones deben ser compatibles con la unión, es decir, deben tener el mismo número de atributos y los atributos correspondientes deben tener el mismo dominio.
4. **Set Difference (-):** El operador Set Difference elimina las tuplas de una relación que también aparecen en otra.
5. **Cartesian Product (x):** El operador Cartesian Product combina tuplas de dos relaciones.
6. **Rename (ρ):** El operador Rename cambia los nombres de las relaciones o los atributos.
7. **Join (⋈):** El operador Join combina tuplas de dos relaciones basándose en una condición de igualdad. Hay varios tipos de Join, incluyendo Inner Join, Left Outer Join, Right Outer Join y Full Outer Join.

Si tenemos dos relaciones R1(A, B) y R2(B, C) con los siguientes datos:

R1		R2	
A	B	B	C
1	2	2	20
3	4	4	40
5	6	6	60
7	8	8	80
9	10	10	100
		12	120

Si queremos combinar las dos relaciones basándonos en una condición de igualdad en el atributo B, usaríamos el siguiente operador:

R1 ⋈{R1.B = R2.B} R2

Esto nos devolvería la siguiente relación:

A	B	C
1	2	20
3	4	40
5	6	60
7	8	80
9	10	100

SQL 1:

1. Crear Tabla:

```
CREATE TABLE Personas (
    id int,
    nombre varchar(255),
    edad int,
    PRIMARY KEY (id)
);
```

ID	Nombre	Edad
1	Esteban	21
2	Monse	22
3	Mateo	19

1.1 Llaves Foráneas:

Una llave foránea establece un enlace entre los datos en dos tablas, y lo hace con el propósito de asociar los datos que se encuentran en las tablas.

```
CREATE TABLE Autores (
    id int,
    nombre varchar(255),
    edad int,
    PRIMARY KEY (id)
);

CREATE TABLE Libros (
    id int,
    titulo varchar(255),
    id_autor int,
    PRIMARY KEY (id),
    FOREIGN KEY (id_autor) REFERENCES Autores(id)
);
```

Autores			Libros
id	nombre	edad	id
1	Gabriel Garcia Marquez	87	1
2	Julio Cortázar	69	2
3	Carlos Ruiz Zafón	55	3
			4

En este ejemplo, la tabla 'Libros' tiene una llave foránea 'id_autor' que referencia a la columna 'id' de la tabla 'Autores'. Esto significa que cada libro en la tabla 'Libros' está asociado con un autor en la tabla 'Autores' a través de la llave foránea 'id_autor'.

2. Modificar Tabla:

a. **DROP TABLE** <Nombre_tabla>: Eliminar la Tabla

b. **ALTER TABL**E <Nombre_tabla> **DROP COLUMN** <Atributo>: Eliminar Atributo de la Tabla

c. **ALTER TABL**E <Nombre_tabla> **ADD COLUMN** <Atributo> <varchar(255)>: Añadir Atributo a la Tabla

3. Insertar Datos:

a. **INSERT INTO** Tabla(at_1,...,at_n) **VALUES** (v_1,...,v_n): Insertar una Tupla a la tabla

b. **INSERT INTO <Nombre_tabla> VALUES** (v_1,...,v_n): Manera Resumida de insertar (Se asume el orden)

- Observacion: El siguiente codigo dará error: (NO PODEMOS CREAR UNA FILA EN QUE LA LLAVE FORANEA NO EXISTA EN LA OTRA TABLA)

```
CREATE TABLE R(a int, b int, PRIMARY KEY(a));
CREATE TABLE S(a int, c int, FOREIGN KEY(a) REFERENCES R, ...);
INSERT INTO R VALUES(1, 1);
INSERT INTO S VALUES(1, 2);
INSERT INTO S VALUES(2, 3); -- Esto provocará un error  ya que no existe un registro con a =2 en la tabla R.
```

Tabla R		Tabla S	
a	b	a	c
1	1	1	2
		2	3

En este ejemplo, la tabla S tiene una llave foránea que referencia a la columna 'a' de la tabla R. Como se puede observar, en la tabla S se intenta insertar un valor '2' en la columna 'a', pero no existe un valor correspondiente en la tabla R. Por lo tanto, este intento de inserción provocará un error.

4. Consultas Sql (Query) + CRUD

1. **SELECT**: Selecciona datos de una base de datos. Los datos devueltos se almacenan en una tabla de resultados, llamada conjunto de resultados. Por ejemplo, `SELECT nombre FROM Personas;` seleccionará todos los nombres de la tabla Personas.
2. **FROM**: Se utiliza para especificar la tabla de la que se obtendrán los datos. Por ejemplo, en la consulta anterior, `Personas` es la tabla especificada con FROM.
3. **WHERE**: Filtra registros. Extrae solo aquellos registros que cumplen una condición especificada. Por ejemplo, `SELECT * FROM Personas WHERE edad > 20;`
   - **WHERE** en SQL permite el uso de los siguientes operadores:
     - **=**: Igual que.
     - **<>** o **!=**: Diferente que.
     - **<=**: Menor o igual que.
     - **>=**: Mayor o igual que.
     - **AND**: Operador lógico AND. Selecciona registros donde dos o más condiciones deben ser verdaderas.
     - **OR**: Operador lógico OR. Selecciona registros donde al menos una de las condiciones debe ser verdadera.
     - **NOT**: Operador lógico NOT. Selecciona registros donde una condición debe ser falsa.
     - **IN**: Permite especificar múltiples valores en una cláusula WHERE.
     - **BETWEEN**: Selecciona valores dentro de un rango. Los valores pueden ser números, texto o fechas.
4. **UPDATE**: Modifica los datos existentes en una tabla. Por ejemplo, `UPDATE Personas SET edad = 30 WHERE nombre = 'Esteban';` actualizará la edad de Esteban a 30 en la tabla Personas.

5. **DELETE**: Elimina registros existentes en una tabla. Por ejemplo, `DELETE FROM Personas WHERE nombre = 'Esteban';` eliminará todos los registros de la tabla Personas donde el nombre sea 'Esteban'.
    - Es importante tener cuidado al usar el comando DELETE, ya que si se omite la cláusula WHERE, se eliminarán todos los registros de la tabla.

---

6. Producto Cruz:
7.

- **SELECT**:

```
SELECT nombre FROM Personas;
```

Esto seleccionaría todos los nombres de la tabla Personas y los mostraría como sigue:

Nombre	
Esteban	
Monse	
Mateo	

- **FROM**:

```
SELECT nombre, edad FROM Personas;
```

Esta consulta seleccionaría los nombres y las edades de la tabla Personas y los mostraría así:

Nombre	Edad
Esteban	21
Monse	22
Mateo	19

- **WHERE**:

```
SELECT * FROM Personas WHERE edad > 20;
```

Esta consulta seleccionaría todos los registros de la tabla Personas donde la edad sea mayor a 20 y los mostraría de esta forma:

ID	Nombre	Edad
1	Esteban	21
2	Monse	22

- **UPDATE**:

```
UPDATE Personas SET edad = 30 WHERE nombre = 'Esteban';
```

Esta consulta actualizaría la edad de Esteban a 30 en la tabla Personas. La tabla se vería así:

ID	Nombre	Edad
1	Esteban	30
2	Monse	22
3	Mateo	19

- **DELETE**:

```
DELETE FROM Personas WHERE nombre = 'Esteban';
```

Esta consulta eliminaría todos los registros de la tabla Personas donde el nombre sea 'Esteban'. La tabla quedaría así:

ID	Nombre	Edad
2	Monse	22
3	Mateo	19

* Eliminación con Llaves Foráneas:

Supongamos que tenemos las siguientes tablas de 'Autores' y 'Libros':

```
CREATE TABLE Autores (
    id int,
    nombre varchar(255),
    edad int,
    PRIMARY KEY (id)
);

CREATE TABLE Libros (
    id int,
    titulo varchar(255),
    id_autor int,
    PRIMARY KEY (id),
    FOREIGN KEY (id_autor) REFERENCES Autores(id)
);
```

Autores			Libros		
id	nombre	edad	id	titulo	id_autor
1	Gabriel Garcia Marquez	87	1	Cien años de soledad	1
2	Julio Cortázar	69	2	Rayuela	2
3	Carlos Ruiz Zafón	55	3	La sombra del viento	3
			4	El amor en los tiempos del cólera	1

Si intentamos ejecutar la siguiente consulta para eliminar al autor 'Gabriel Garcia Marquez':

```
DELETE FROM Autores WHERE nombre = 'Gabriel Garcia Marquez';
```

Recibiríamos un error, ya que existen libros en la tabla 'Libros' que están haciendo referencia al 'id' del autor 'Gabriel Garcia Marquez' a través de la llave foránea 'id_autor'.

Para poder eliminar a este autor, primero tendríamos que eliminar todos los libros asociados a él en la tabla 'Libros'.

Existen varias opciones para manejar este problema:

1. **Propagar la eliminación (Cascada de eliminaciones):** En este caso, al eliminar una tupla en la tabla 'Autores', también se eliminarán todas las tuplas en la tabla 'Libros' que hacen referencia a ella. Esto se puede hacer añadiendo la cláusula `ON DELETE CASCADE` a la definición de la llave foránea en la tabla 'Libros'. De esta manera, si borramos un autor de la tabla 'Autores', todos sus libros correspondientes en la tabla 'Libros' también serán borrados automáticamente.

```
CREATE TABLE Autores (
    id int,
    nombre varchar(255),
    edad int,
    PRIMARY KEY (id)
);
```

```
CREATE TABLE Libros (
    id int,
    titulo varchar(255),
    id_autor int,
    PRIMARY KEY (id),
    FOREIGN KEY (id_autor) REFERENCES Autores(id) ON DELETE CASCADE
);
```

2. **Dejar en nulo**: Otra opción es configurar la llave foránea en la tabla 'Libros' para que se establezca en NULL cuando se elimina la tupla correspondiente en la tabla 'Autores'. Esto se puede hacer añadiendo la cláusula `ON DELETE SET NULL` a la definición de la llave foránea en la tabla 'Libros'. De esta manera, si borramos un autor de la tabla 'Autores', el campo 'id_autor' de todos sus libros correspondientes en la tabla 'Libros' se establecerá en NULL.

```
CREATE TABLE Autores (
    id int,
    nombre varchar(255),
    edad int,
    PRIMARY KEY (id)
);

CREATE TABLE Libros (
    id int,
    titulo varchar(255),
    id_autor int,
    PRIMARY KEY (id),
    FOREIGN KEY (id_autor) REFERENCES Autores(id) ON DELETE SET NULL
);
```

Es importante tener en cuenta que estas opciones deben usarse con cuidado, ya que pueden tener implicaciones significativas en la integridad de los datos en la base de datos.

PRODUCTO CRUZ: (FROM Tabla1,Tabla2)

El producto cruz (o producto cartesiano) en SQL se realiza implícitamente mediante el operador FROM cuando se especifican varias tablas. Esto genera una nueva tabla que combina cada fila de la primera tabla con cada fila de la segunda tabla.

Por ejemplo, si tenemos dos tablas, R y S:

Tabla R			Tabla S	
a	b		c	d
1	2		5	6
3	4		7	8

Podemos realizar un producto cruz de estas dos tablas con la siguiente consulta SQL:

```
SELECT * FROM R, S;
```

El resultado sería una nueva tabla que combina cada fila de R con cada fila de S:

a	b	c	d
1	2	5	6

1	2	7	8
3	4	5	6
3	4	7	8

Es importante tener en cuenta que el producto cruz puede generar una gran cantidad de datos, especialmente si las tablas involucradas tienen muchas filas. En la práctica, suele ser más útil realizar un producto cruz junto con una condición (a menudo utilizando la cláusula WHERE), para limitar los resultados a las combinaciones de filas que satisfacen cierta condición.

JOIN:(FROM Tabla1,Tabla2 + Where)

Un JOIN en SQL se puede realizar utilizando la cláusula WHERE para indicar la condición de coincidencia entre las tablas. Aunque actualmente es más común utilizar la cláusula JOIN para realizar estas operaciones, la cláusula WHERE puede ser utilizada para obtener el mismo resultado.

Por ejemplo, si tenemos dos tablas, 'Pedidos' y 'Clientes':

Pedidos			Clientes	
id_pedido	id_cliente	producto	id	nombre
1	1	Manzanas	1	Carlos
2	2	Bananas	2	Ana
3	3	Naranjas	3	Pedro

Podríamos querer obtener una nueva tabla que muestre el nombre del cliente junto con el producto que compró. Para ello, podríamos realizar un JOIN de las tablas 'Pedidos' y 'Clientes' utilizando la cláusula WHERE para indicar que queremos combinar las filas donde 'id_cliente' es el mismo en ambas tablas:

```
SELECT Pedidos.producto, Clientes.nombre FROM Pedidos, Clientes WHERE Pedidos.id_cliente = Clientes.id;
```

El resultado de esta consulta sería una nueva tabla que combina el producto de cada pedido con el nombre del cliente que realizó el pedido:

producto	nombre
Manzanas	Carlos
Bananas	Ana
Naranjas	Pedro

- Ejemplo de consulta interesante:

```
SELECT Peliculas.nombre,Actores.nombre
FROM Peliculas, Actuo_en, Actores
WHERE Peliculas.id = Actuo_en.id_pelicula
AND Actores.id = Actuo_en.id_actor;
```

Esta consulta SQL realiza dos operaciones JOIN. Primero, une las tablas 'Peliculas' y 'Actuo_en' basándose en una coincidencia entre los campos 'id' de 'Peliculas' y 'id_pelicula' de 'Actuo_en'. Luego, une el resultado de esta operación con la tabla 'Actores', basándose en una coincidencia entre los campos 'id' de 'Actores' y 'id_actor' de 'Actuo_en'. El resultado es una tabla que incluye los nombres de las películas y los nombres de los actores para cada película.

ALIAS: (AS)

```
SELECT P.nombre, A.nombre
FROM Peliculas as P, Actuo_en as Ae, Actores as A
WHERE P.id = Ae.id_pelicula
AND A.id = Ae.id_actor;
```

Nombramiento y Operaciones:

```
SELECT (nombre || ' dirigida por ' || director) as creditos, año
FROM Peliculas
```

id	nombre	director	año
1	Interestellar	C.Nolan	2014
2	...	...	

creditos	año
Interestellar dirigida por C.Nolan	2014

ORDENAMIENTO:

```
SELECT nombre, calificacion
FROM Peliculas
ORDER BY nombre, calificacion
```

→ Orden Ascendente por nombre (alfabéticamente), el desempate seria por calificación

→ (Order by resuelve un empate por el ultimo atributo entregado)

nombre	calificacion
Harry Potter	8.1
Inception	8.8
Interstellar	8.6
The Revenant	8.1
The Theory of Everything	7.7

```
SELECT nombre, calificacion
FROM Peliculas
ORDER BY nombre DESC, calificacion
```

nombre	calificacion
The Theory of Everything	7.7
The Revenant	8.1
Interstellar	8.6
Inception	8.8
Harry Potter	8.1

UNION:

```
SELECT nombre
FROM Actores
UNION
```

nombre
A. Iñárritu
C. Nolan

```
SELECT director
FROM Peliculas
```

D. Yates
J. Marsh
Jessica Chastain
Leonardo DiCaprio
Matthew McConaughey
Daniel Radcliffe

- EXCEPT: diferencia del álgebra
- UNION: unión del álgebra
- INTERSECT: intersección del álgebra
- UNION ALL: unión que admite duplicados

LIKE: (REGEX)

```
SELECT *
FROM Peliculas
WHERE name LIKE ‘%Potter%'
```

1. % : Cualquier otra secuencia de caracteres
2. _ : Cualquier otro caracter (1 solamente)
3. : No otro caracter → (correo Like ‘%@gmail.com’)

DISTINCT: (Elimina Repetidos)

```
SELECT DISTINCT nombre
FROM Peliculas
```

En esta tabla, se muestran los nombres únicos de las películas presentes en la tabla "Peliculas", sin ningún nombre repetido.

nombre
Interstellar
The Revenant
Harry Potter
The Theory of Everything
Inception

SQL 2

id	nombre	precio	fabricante
1	Corolla	25000	Toyota
2	Camry	30000	Toyota
3	Prius	28000	Toyota
4	Accord	28000	Honda
5	Civic	22000	Honda

```
SELECT AVG(precio) as promedio
FROM Productos
WHERE fabricante = ‘Toyota’
```

Entrega el precio promedio de los productos marca Toyota

promedio
27666.67

1. **AVG**: Promedio de una columna

2. **MIN**: Minimo de una columna
3. **MAX**: Maximo de una columna
4. **COUNT**: Cuenta el numero de filas

COUNT:

producto	fecha	precio	cantidad
tomates	07/02	100	6
tomates	06/07	150	
zapallo	08/02	800	1
zapallo	09/07	1000	
zapallo	01/01	600	1

```
SELECT COUNT(*)
FROM Compra
# 5
# contara todas las filas!
```

```
SELECT COUNT(cantidad)
FROM Compra
# 3
# contara solo las con info
```

Observación!:

SELECT DISTINCT COUNT(cantidad) = 3 ≠ SELECT COUNT(DISTINCT cantidad) = 2

1. `SELECT DISTINCT COUNT(cantidad)`:

   Esta consulta primero cuenta el número de filas que tienen valores en la columna "cantidad" y luego cuenta el número de resultados **únicos** obtenidos. Esto significa que si hay filas con el mismo valor de cantidad, se cuentan solo una vez.

2. `SELECT COUNT(DISTINCT cantidad)`:

   Esta consulta primero encuentra todos los valores únicos en la columna "cantidad" y luego cuenta el número total de valores encontrados. Esto significa que si hay filas con el mismo valor de cantidad, cada uno de esos valores se cuenta solo una vez

GROUP BY:

producto	fecha	precio	cantidad
tomates	07/02	100	6
tomates	06/07	150	4
zapallo	08/02	800	1
zapallo	09/07	1000	2
zapallo	01/01	600	1

```
SELECT producto, SUM(precio*cantidad) as ventaTotal
FROM Compra
WHERE fecha > '10/01'
GROUP BY producto
```

1) FROM Y WHERE:

producto	fecha	precio	cantidad
tomates	07/02	100	6
tomates	06/07	150	4

zapallo	08/02	800	1
zapallo	09/07	1000	2

2) GROUP BY:

producto	fecha	precio	cantidad
tomates	07/02	100	6
	06/07	150	4
zapallo	08/02	800	1
	09/07	1000	2

3) Ejecutar Proyección:

producto	ventaTotal
tomates	1200
zapallo	2800

HAVING:

(Para la misma consulta anterior, pero solo para vendidos mas de 100 veces)

```
SELECT producto, SUM(precio*cantidad) as ven
FROM Compra
WHERE fecha > '10/01'
GROUP BY producto
HAVING SUM(cantidad) > 100
```

La cláusula WHERE se utiliza para filtrar filas antes de que se realice la agregación y el agrupamiento, mientras que la cláusula HAVING se utiliza para filtrar resultados después de que se han realizado las operaciones de agregación y agrupamiento.

En este caso específico:

- La cláusula WHERE `fecha > '10/01'` filtra las filas de la tabla "Compra" para incluir solo aquellas con fechas posteriores al 10 de enero.
- La cláusula GROUP BY `producto` agrupa las filas resultantes según el campo "producto".
- La función de agregación SUM se usa para calcular la suma del producto del precio y la cantidad para cada grupo de productos.

Ahora, la cláusula HAVING `SUM(cantidad) > 100` se aplica después de la operación de agregación y agrupamiento. Esto filtra los resultados del grupo para incluir solo aquellos grupos donde la suma de la cantidad de productos vendidos es mayor que 100.

Si intentáramos incluir `SUM(cantidad) > 100` en la cláusula WHERE, SQL intentaría aplicarlo antes de que se realice la agregación y el agrupamiento, lo que provocaría un error porque no se puede usar una función de agregación en la cláusula WHERE. Por lo tanto, para aplicar una condición basada en una función de agregación, como SUM, COUNT, AVG, etc., se debe utilizar la cláusula HAVING después del GROUP BY.

CONSULTAS ANIDADAS:

Bandas(nombre, vocalista, ...)
Estudiantes_UC(nombre, ...)

Toco_en(nombre_banda, nombre_festival)

```
SELECT Bandas.nombre
FROM Bandas, Estudiantes_UC
WHERE Bandas.vocalista = Estudiantes_UC.nombre
AND Bandas.nombre IN (
    SELECT Toco_en.nombre_banda
    FROM Toco_en
    WHERE Toco_en.nombre_festival = ‘Lollapalooza’
)
```

Comprobamos que Bandas.nombre esté dentro del listado de bandas que han tocado en Lollapalooza.

1. IN : forma abreviada de escribir una condición comúnmente utilizada para verificar si un valor está presente en un conjunto de resultados.
    - Por ejemplo, si deseas verificar si el valor de una subconsulta está presente en un conjunto de resultados, puedes usar `s IN R`. Si el valor devuelto por la subconsulta está presente en el conjunto de resultados, la condición será verdadera; de lo contrario, será falsa.

2. > ALL: Esta condición se utiliza para verificar si un valor escalar devuelto por una subconsulta es mayor que **TODOS** los valores devueltos por otra consulta.
    - Por ejemplo, si deseas verificar si un valor es mayor que todos los valores devueltos por una subconsulta, puedes usar `s > ALL R`. Si el valor es mayor que todos los valores devueltos por la subconsulta, la condición será verdadera; de lo contrario, será falsa.

3. > ANY: Esta condición se utiliza para verificar si un valor escalar devuelto por una subconsulta es mayor que al menos uno de los valores devueltos por otra consulta.
    - Por ejemplo, si deseas verificar si un valor es mayor que al menos uno de los valores devueltos por una subconsulta, puedes usar `s > ANY R`. Si el valor es mayor que al menos uno de los valores devueltos por la subconsulta, la condición será verdadera; de lo contrario, será falsa.

4. EXISTS: Esta condición se utiliza para verificar si la subconsulta devuelve algún resultado o no.
    - Si la subconsulta devuelve algún resultado, la condición `EXISTS R` será verdadera; de lo contrario, será falsa.
    - Se utiliza comúnmente en combinación con una cláusula `WHERE` para filtrar filas basadas en la existencia de resultados en una subconsulta.

ALL:

1. Cervezas más baratas que la Austral Calafate

```
SELECT Cervezas.nombre
FROM Cervezas
WHERE Cervezas.precio < ALL (
    SELECT Cervezas2.precio
    FROM Cervezas AS Cervezas2
```

```
    WHERE Cervezas2.nombre = ‘Austral Calafate’
)
```

ANY:

2. Cervezas que no son la mas cara

```
SELECT Cervezas.nombre
FROM Cervezas
WHERE Cervezas.precio < ANY (
    SELECT Cervezas2.precio
    FROM Cervezas AS Cervezas2
)
```

Consultas Monotonas:

→ SELECT, FROM, WHERE: Son consultas monótonas: Es una consulta que al agregar mas datos nunca reduce el conjunto de resultados que produce

→ All No es monotona!

SUBCONSULTAS RELACIONADAS:

Las sub-consultas dependen de la externa!

```
SELECT p.nombre
FROM Películas AS p
WHERE p.año <> ANY (
    SELECT año
    FROM Películas
    WHERE nombre = p.nombre
)
```

Join Anidado:

Ejemplo 1:

“El nombre de cada actor junto con el total de películas en las que ha actuado.”

```
SELECT Actores.nombre, agg.cuenta
FROM Actores, (
    SELECT id_actor as id, COUNT(*) as cuenta
    FROM Actuo_en
    GROUP BY id_actor
) as agg
WHERE Actores.id = agg.id
```

Actores		Actuo_en	
id	nombre	id_actor	pelicula_id
1	Leonardo DiCaprio	1	1
		1	2

2	Matthew McConaughey	1	3
3	Daniel Radcliffe	2	1
4	Jessica Chastain	3	2
		4	3

SUB-CONSULTA:

```
SELECT id_actor as id, COUNT(*) as cuenta
FROM Actuo_en
GROUP BY id_actor;
```

→ Cuenta el número de películas en las que cada actor ha actuado. Devuelve dos columnas: "id" (el id del actor) y "cuenta" (el número de películas en las que ha actuado cada actor).

RESULTADO:

nombre	cuenta
Leonardo DiCaprio	3
Matthew McConaughey	1
Daniel Radcliffe	1
Jessica Chastain	1

Ejemplo 2:

“El nombre de cada actor junto con el año de la primera película que actuó”

```
SELECT Actores.nombre, agg.año
FROM Actores, (
    SELECT Actuo_en.id_actor as id, MIN(Peliculas.año) as año
    FROM Actuo_en, Peliculas
    WHERE Actuo_en.id_pelicula = Peliculas.id
    GROUP BY id_actor
) as agg
WHERE Actores.id = agg.id
```

Actores		Actuo_en	
id	nombre	id_actor	pelicula_id
1	Leonardo DiCaprio	1	1
		1	2
2	Matthew McConaughey	1	3
		2	1
3	Daniel Radcliffe	3	2
		4	3
4	Jessica Chastain		

Peliculas		
id	nombre	año
1	Titanic	1997
2	Interstellar	2014
3	Harry Potter and the Sorcerer's Stone	2001

SUB-CONSULTA:

```
SELECT Actuo_en.id_actor as id, MIN(Peliculas.año)
FROM Actuo_en, Peliculas
WHERE Actuo_en.id_pelicula = Peliculas.id
GROUP BY id_actor
```

Esta subconsulta une las tablas "Actuo_en" y "Peliculas" utilizando la condición `Actuo_en.id_pelicula = Peliculas.id` para obtener el año mínimo (el año más antiguo) de las películas en las que cada actor ha actuado. Devuelve dos columnas: "id" (el id del actor) y "año" (el año mínimo de las películas en las que ha actuado cada actor).

id	año
1	1997
2	2014
3	2001
4	2001

RESULTADO FINAL:

nombre	año
Leonardo DiCaprio	1997
Matthew McConaughey	2014
Daniel Radcliffe	2001
Jessica Chastain	2001

NULL:

En sql se modela la falta de informacion mediante nulos (**NULL**)

En general los nulos pueden significar:

- Valor existe, pero no tengo la información
- Valor no existe (si nombre = null la información no existe)
- Ni siquiera sé si el valor existe o no

- SELECT * FROM R ≠ SELECT * FROM R WHERE R.b=3 OR R.b<>3 → (Que no sea ni 3 ni distinto a 3 NO ES NULO!!!)

```
SELECT * FROM R
# <=>

SELECT * FROM R WHERE R.b = 3
UNION
SELECT * FROM R WHERE R.b <> 3
UNION
SELECT * FROM R WHERE R.b IS NULL
```

- IS NULL: True si es nulo
- IS NOT NULL: True si no es nulo

R	S

A	B
1	2
null	3

```
SELECT R.A + S.B
FROM R, S
```

R.A + R.S
3
4
Null
Null

Si tenemos R.A = S.B

1. R.A = 1 y S.B = 2 → FALSE
2. R.A = Null S.B= 2 → UNKWNOWN

Ejemplos:

R	S
A	A
1	1
2	2
	3
	4

```
SELECT S.A
FROM S
WHERE S.A NOT IN (
    SELECT R.A
    FROM R
)
```

RESULTADO: {3,4}

R	S
A	A
1	1
2	2
null	3
	4

```
SELECT S.A
FROM S
WHERE S.A NOT IN (
    SELECT R.A
    FROM R
)
```

RESULTADO: TABLA VACIA!!!

Cuando intentas hacer la comparación `S.A NOT IN (SELECT R.A FROM R)`, el valor `NULL` se maneja de manera especial. La comparación `NULL NOT IN (...)` se comporta como si la condición fuera desconocida. Por lo tanto, no se puede determinar si el valor `NULL` está presente o no en el conjunto resultante de la subconsulta. Como resultado, se devuelve una tabla vacía debido a que el resultado de la subconsulta no se puede determinar en presencia de valores `NULL`.

EXPLICACION:

```
3 NOT IN {1,2,null}
= NOT (3 IN {1,2,null})
= NOT (3=1 OR 3=2 OR 3 = null)
= NOT (FALSE OR FALSE OR UNKNOWN)
= NOT (UNKNOWN)
= UNKWNOWN

# MISMO RAZONAMIENTO PARA 4 NOT IN {1,2,null}
```

COUNT:

R | A: {1,null}

SELECT COUNT(*) FROM R = 2

SELECT COUNT(R.A) FROM R = 1

- ojo!: SELECT SUM(R.A) FROM R = 1

RESTRICCION DE NULOS!

```
CREATE TABLE <Nombre> (
    id int NOT NULL
)
```