Ejemplos de la guía teórica

Juannie

06/06/2024

SQL Queries

c15: Averiguar el nombre y la edad de todos los marineros

```sql
SELECT nombrem, edad
FROM marineros;
```

c11: Averiguar todos los marineros con categoría superior a 7

```sql
SELECT *
FROM marineros AS M
WHERE M.categoría > 7;
```

> **Nota**: "M" es una "variable de rango". La cláusula `SELECT` es equivalente a la proyección del álgebra relacional, y la cláusula `FROM` es equivalente a la selección. Este desajuste entre la denominación de los operadores del álgebra relacional y la sintaxis de SQL es un accidente histórico desafortunado.

c1: Averiguar el nombre de los marineros que han reservado el barco 103

Opción 1: Utilizando la cláusula `NATURAL JOIN`

```sql
SELECT M.nombrem
FROM marineros AS M NATURAL JOIN reservas AS R
WHERE R.idb = 103;
```

Opción 2: Utilizando la cláusula `INNER JOIN`

```sql
SELECT M.nombrem
FROM marineros AS M INNER JOIN reservas AS R ON M.idm = R.idm
WHERE R.idb = 103;
```

Opción 3: Utilizando el producto cartesiano entre las tablas

```sql
SELECT M.nombrem
FROM marineros AS M, reservas AS R
WHERE M.idm = R.idm AND R.idb = 103;
```

c16: Averiguar el idm de los marineros que han reservado barcos rojos

```sql
SELECT DISTINCT R.idm
FROM reservas AS R NATURAL JOIN barcos AS B
WHERE B.color = 'rojo';
```

Nota: Agregué la cláusula `DISTINCT` porque los marineros con idm = 22 e idm = 31 reservaron dos barcos rojos. Si no agregaba este filtro, estos valores aparecerían dos veces en la tabla resultante. Si se hubiese pedido el idb junto al idm, no hubiese hecho falta agregar la cláusula `DISTINCT`.

c2: Averiguar el nombre de los marineros que han reservado barcos rojos

Opción 1: Utilizando la cláusula `NATURAL JOIN`

```sql
SELECT DISTINCT M.nombrem
FROM marineros AS M NATURAL JOIN reservas AS R NATURAL JOIN barcos AS B
WHERE B.color LIKE 'rojo';
```

Opción 2: Realizando un producto cartesiano entre las tablas

```sql
SELECT DISTINCT M.nombrem
FROM marineros M, reservas R, barcos B
WHERE M.idm = R.idm AND R.idb = B.idb AND B.color LIKE 'rojo';
```

Nota: La cláusula `AS` para asignar una variable de rango es opcional. El uso de comillas simples o dobles es indistinto. El operador `LIKE` permite encontrar strings similares a los que están en la tabla, sin requerir que el string sea idéntico. Por ejemplo, `B.color LIKE 'ro%'` devolvería las columnas donde el valor del dato "color" empiece con "ro". El uso de mayúsculas o minúsculas es indiferente, tanto con el operador `LIKE` como con el operador `=`.

c3: Averiguar el color de los barcos reservados por Domínguez

Opción 1:

```sql
SELECT DISTINCT B.color
FROM marineros AS M, reservas AS R, barcos AS B
WHERE M.idm = R.idm AND R.idb = B.idb AND M.nombrem LIKE 'Dom%';
```

Opción 2:

```sql
SELECT DISTINCT B.color
FROM marineros M NATURAL JOIN reservas R NATURAL JOIN barcos B
WHERE M.nombrem LIKE 'Domínguez';
```

Nota: Puede que haya más de un marinero llamado Domínguez, por lo que esta consulta devolverá el color de los barcos reservados por algún Domínguez.

c4: Averiguar el nombre de los marineros que han reservado, como mínimo, un barco

```sql
SELECT DISTINCT m.nombrem
FROM marineros AS m NATURAL JOIN reservas AS r;
```

c17: Calcular el incremento de la categoría de las personas que han navegado en dos barcos el mismo día

Opción 1:

```sql
SELECT DISTINCT M.nombrem, M.categoría + 1 AS categoría
FROM marineros AS M, reservas AS R1, reservas AS R2
WHERE M.idm = R1.idm AND M.idm = R2.idm AND R1.fecha = R2.fecha AND R1.idb <> R2.idb;
```

Opción 2:

```sql
SELECT DISTINCT m.nombrem, m.categoría + 1 AS categoría
FROM marineros AS m INNER JOIN reservas AS r1 ON m.idm = r1.idm INNER JOIN reservas
↪  AS r2 ON r1.idm = r2.idm
WHERE r1.fecha = r2.fecha AND r1.idb <> r2.idb;
```

Nota: Lo que viene a continuación estaba en el apunte y no sé muy bien a qué viene:

«Además, cada elemento de la condición puede ser tan general como expresión1 = expresión2».

```sql
SELECT m1.nombrem AS nombre1, m2.nombrem AS nombre2
FROM marineros AS m1, marineros AS m2
WHERE 2 * m1.categoría = m2.categoría - 1;
```

c18: Averiguar la edad de los marineros cuyo nombre comienza con B, acaba con O y tiene como mínimo seis caracteres

```sql
SELECT m.edad
FROM marineros AS m
WHERE m.nombrem LIKE 'B__%__O';
```

c5: Averiguar el nombre de los marineros que han reservado barcos rojos o verdes

Opción 1: Con `INNER JOIN`

```sql
SELECT DISTINCT m.nombrem
FROM marineros AS m INNER JOIN reservas AS r ON m.idm = r.idm INNER JOIN barcos AS b
↪  ON r.idb = b.idb
WHERE b.color LIKE 'rojo' OR b.color LIKE 'verde';
```

Opción 2: Con `NATURAL JOIN`

```sql
SELECT DISTINCT m.nombrem
FROM marineros AS m NATURAL JOIN reservas AS r NATURAL JOIN barcos AS b
WHERE b.color LIKE 'rojo' OR b.color LIKE 'verde';
```

Opción 3: Utilizando la cláusula `IN`

```sql
SELECT DISTINCT m.nombrem
FROM marineros AS m NATURAL JOIN reservas AS r NATURAL JOIN barcos AS b
WHERE b.color IN ('rojo', 'verde');
```

Opción 4: Hacer una unión entre tablas con `UNION`

```sql
SELECT DISTINCT m.nombrem
FROM marineros AS m INNER JOIN reservas AS r ON m.idm = r.idm INNER JOIN barcos AS b
↪  ON r.idb = b.idb
WHERE b.color LIKE 'rojo'

UNION

SELECT DISTINCT m.nombrem
FROM marineros AS m INNER JOIN reservas AS r ON m.idm = r.idm INNER JOIN barcos AS b
↪  ON r.idb = b.idb
WHERE b.color LIKE 'verde';
```

> **Nota**: Si ponemos la opción `ALL`, aparecerán todas las filas obtenidas a causa de la unión. No la pondremos si queremos eliminar las filas repetidas. Las tablas resultantes de las sentencias a unir deben ser compatibles con la unión.

c6: Averiguar el nombre de los marineros que han reservado barcos rojos y verdes

Opción 1: Haciendo una intersección con la cláusula `INTERSECT`

```sql
SELECT DISTINCT m.nombrem
FROM marineros AS m NATURAL JOIN reservas AS r NATURAL JOIN barcos AS b
WHERE b.color LIKE 'Rojo'

INTERSECT

SELECT DISTINCT m.nombrem
FROM marineros AS m NATURAL JOIN reservas AS r NATURAL JOIN barcos AS b
WHERE b.color LIKE 'Verde';
```

> **Nota**: Si ponemos la opción `ALL`, aparecerán todas las filas obtenidas a partir de la intersección. No la pondremos si queremos eliminar las filas repetidas. Las tablas resultantes de las sentencias a unir deben ser compatibles con la unión.

Opción 2: Utilizando `IN`

```sql
SELECT DISTINCT m.nombrem
FROM marineros AS m
WHERE m.idm IN (
    SELECT r.idm
    FROM reservas AS r INNER JOIN barcos AS b ON r.idb = b.idb
    WHERE b.color LIKE 'Rojo'
) AND m.idm IN (
    SELECT r.idm
    FROM reservas AS r INNER JOIN barcos AS b ON r.idb = b.idb
    WHERE b.color LIKE 'Verde'
);
```

Opción 3: Utilizando `IN` una sola vez:

```sql
SELECT DISTINCT m.nombrem
FROM marineros AS m NATURAL JOIN reservas AS r NATURAL JOIN barcos AS b
WHERE b.color LIKE "rojo" AND r.idm IN (
    SELECT r2.idm
    FROM marineros AS m2 NATURAL JOIN reservas AS r2 NATURAL JOIN barcos AS b2
    WHERE b2.color LIKE "verde"
    );
```

Opción 4: Utilizando `EXISTS`

```sql
SELECT DISTINCT m.nombrem
FROM marineros AS m
WHERE EXISTS (
    SELECT 1
    FROM reservas AS r INNER JOIN barcos AS b ON r.idb = b.idb
    WHERE r.idm = m.idm AND b.color LIKE 'Rojo'
) AND EXISTS (
    SELECT 1
    FROM reservas AS r INNER JOIN barcos AS b ON r.idb = b.idb
    WHERE r.idm = m.idm AND b.color LIKE 'Verde'
);
```

Explicación:

- `SELECT 1`: Esta es una convención comúnmente utilizada para indicar que solo se necesita comprobar la existencia de filas que cumplen la condición. 1 es un valor constante y no afecta el rendimiento ni el resultado de la subconsulta.
- `SELECT r.idm`: Selecciona una columna específica, `r.idm`, pero en el contexto de `EXISTS`, no se utiliza el valor seleccionado, solo se verifica la existencia de la fila.

En ambos casos, la subconsulta solo necesita determinar si alguna fila que cumple con los criterios especificados existe. Por lo tanto, puedes usar cualquiera de los dos métodos en una subconsulta `EXISTS` sin diferencia en el resultado o el rendimiento.

c19: Averiguar el idm de todos los marineros que han reservado barcos rojos pero no verdes

Opción 1: Hacer una diferencia de conjuntos con `EXCEPT`

```sql
SELECT DISTINCT r.idm
FROM reservas AS r INNER JOIN barcos AS b ON r.idb = b.idb
WHERE b.color LIKE 'rojo'

EXCEPT

SELECT DISTINCT r.idm
FROM reservas AS r INNER JOIN barcos AS b ON r.idb = b.idb
WHERE b.color LIKE 'verde';
```

Opción 2: Utilizar `NOT IN`

```sql
SELECT DISTINCT r.idm
FROM reservas AS r INNER JOIN barcos AS b ON r.idb = b.idb
WHERE b.color LIKE 'rojo' AND r.idm NOT IN (
    SELECT r2.idm
    FROM reservas AS r2 INNER JOIN barcos AS b2 ON r2.idb = b2.idb
    WHERE b2.color LIKE 'verde'
);
```

Opción 3: Utilizar `NOT EXISTS`

```sql
SELECT DISTINCT r.idm
FROM reservas AS r INNER JOIN barcos AS b ON r.idb = b.idb
WHERE b.color LIKE 'rojo' AND NOT EXISTS (
    SELECT 1
    FROM reservas AS r2 INNER JOIN barcos AS b2 ON r2.idb = b2.idb
    WHERE r2.idm = r.idm AND b2.color LIKE 'verde'
);
```

Consulta extra inventada por mí: Averiguar el idm de los marineros que tengan una categoría de 10 y hayan reservado el barco 104.

```sql
SELECT DISTINCT r.idm
FROM reservas AS r NATURAL JOIN marineros AS m
WHERE m.categoría = 10 AND r.idm IN (
    SELECT r2.idm
    FROM reservas AS r2
    WHERE r2.idb = 104
);
```

Como vimos anteriormente, aquí también se hubiese podido usar la clásula "INTERSECT"

c20: Averiguar el idm de los marineros que tengan una categoría de 10 o hayan reservado el barco 104.

Opción 1: Utilizando una unión

```sql
SELECT r.idm
FROM marineros AS r
WHERE r.categoría = 10

UNION

SELECT r.idm
FROM reservas AS r
WHERE r.idb = 104;
```

Opción 2: Utilizando la cláusula `IN`

```sql
SELECT m.idm
FROM marineros AS m
WHERE m.categoría = 10 OR m.idm IN (
```

```
    SELECT r.idm
    FROM reservas AS r
    WHERE r.idb = 104
);
```

Un último punto a destacar sobre la `UNION`, `INTERSECT` y `EXCEPT` es: a diferencia del criterio predeterminado de que de la forma básica de las consultas no se eliminan los duplicados a menos que se especifique `DISTINCT`, el criterio predeterminado para las consultas `UNION` es que los duplicados sí se eliminan, a menos que escribamos `UNION AL`, que fuerza a dejar los duplicados. Igual para el caso de `INTERSECT ALL` o de `EXCEPT ALL`.

c9: Averiguar el nombre de los marineros que han reservado todos los barcos.

Para resolver esta consigna necesitamos hacer una división, operación no soportada en SQL de forma "directa". Las opciones que tenemos son las siguientes.

Opción 1: Utilizando una subconsulta con `NOT EXISTS`

```
SELECT nombrem
FROM marineros AS m
WHERE NOT EXISTS (
    SELECT *
    FROM barcos AS b
    WHERE NOT EXISTS (
        SELECT *
        FROM reservas AS r
        WHERE r.idm = m.idm AND r.idb = b.idb
    )
);
```

La consulta SQL busca encontrar los nombres de los marineros que han reservado todos los barcos. La consulta utiliza una combinación de subconsultas anidadas y el operador `NOT EXISTS` para lograr esto. Vamos a desglosar paso a paso cómo funciona:

1. **Consulta Principal**:

```
SELECT nombrem
FROM marineros AS m
WHERE NOT EXISTS (
    -- subconsulta
);
```

Esta parte selecciona el nombre de los marineros (`nombrem`) de la tabla `marineros` (alias `m`). La condición en el `WHERE` especifica que debe ser cierto que no existe algo. Vamos a ver qué es ese "algo".

2. **Primera Subconsulta**:

```
SELECT *
FROM barcos AS b
WHERE NOT EXISTS (
    -- subconsulta anidada
);
```

Aquí se seleccionan todas las filas de la tabla `barcos` (alias `b`). La condición en el `WHERE` especifica que debe ser cierto que no existe algo más, lo que nos lleva a la segunda subconsulta anidada.

3. **Segunda Subconsulta Anidada**:

```
SELECT *
FROM reservas AS r
WHERE r.idm = m.idm AND r.idb = b.idb
```

   Esta subconsulta anidada busca en la tabla `reservas` (alias `r`) filas donde el `idm` de la reserva coincide con el `idm` del marinero de la consulta principal y donde el `idb` de la reserva coincide con el `idb` del barco de la primera subconsulta. Es decir, está comprobando si existe una reserva que relacione al marinero `m` con el barco `b`.

Ahora, vamos a juntar todo y explicar cómo se relacionan estas subconsultas para lograr el objetivo:

Explicación Completa del Proceso

1. **Iteración por cada Marinero**: La consulta principal itera sobre cada marinero en la tabla `marineros`.
2. **Verificación de Barcos**: Por cada marinero, la primera subconsulta itera sobre cada barco en la tabla `barcos`.
3. **Comprobación de Reservas**: Por cada combinación de marinero y barco, la subconsulta anidada comprueba si existe una entrada en la tabla `reservas` que indique que ese marinero ha reservado ese barco.
4. **Condición `NOT EXISTS`**:
   - Si la subconsulta anidada (que busca reservas) no encuentra ninguna fila que relacione al marinero `m` con el barco `b` (`NOT EXISTS` devuelve verdadero), significa que el marinero `m` no ha reservado el barco `b`.
   - Si la subconsulta anidada encuentra al menos una fila que relacione al marinero con el barco, `NOT EXISTS` devuelve falso.
5. **Condición Principal `NOT EXISTS`**:
   - Si hay al menos un barco para el cual no existe una reserva hecha por el marinero `m`, la primera subconsulta devolverá al menos una fila, lo que hace que `NOT EXISTS` en la consulta principal devuelva falso.
   - Si para todos los barcos existe una reserva hecha por el marinero `m`, la primera subconsulta no devolverá ninguna fila, haciendo que `NOT EXISTS` en la consulta principal devuelva verdadero.
6. **Resultado Final**: La consulta principal seleccionará los nombres de aquellos marineros para los cuales no existe ningún barco que no hayan reservado, es decir, aquellos marineros que han reservado todos los barcos disponibles.

Ejemplo Supongamos que tenemos las siguientes tablas:

- `marineros`:

idm	nombrem	categoría	edad
1	John	5	35
2	Paul	3	28
3	George	4	40

- `barcos`:

idb	nombreb	color
1	Poseidon	azul
2	Triton	rojo

- `reservas`:

idm	idb	fecha
1	1	2024-01-01
1	2	2024-01-02
2	1	2024-01-03
3	1	2024-01-04
3	2	2024-01-05

Para este ejemplo: - John (idm 1) ha reservado ambos barcos, por lo que será incluido en el resultado. - Paul (idm 2) ha reservado solo un barco, por lo que no será incluido en el resultado. - George (idm 3) ha reservado ambos barcos, por lo que será incluido en el resultado.

El resultado de la consulta será:

nombrem
John
George

Opción 2: Utilizando `GROUP BY`, `HAVING` y `COUNT`

```sql
SELECT m.nombrem
FROM marineros AS m INNER JOIN reservas AS r ON m.idm = r.idm
GROUP BY m.idm, m.nombrem
HAVING COUNT(DISTINCT r.idb) = (
    SELECT COUNT(*)
    FROM barcos
);
```

Paso a Paso:

1. **JOIN entre `marineros` y `reservas`**:
   - `JOIN reservas AS r ON m.idm = r.idm`: Esta parte de la consulta une la tabla `marineros` con la tabla `reservas` usando la columna `idm` (el identificador del marinero) que está presente en ambas tablas. El resultado de esta unión incluye todas las filas de `marineros` junto con las filas correspondientes de `reservas` donde los marineros han hecho reservas.
2. **Agrupación por marinero (`GROUP BY m.idm, m.nombrem`)**:
   - `GROUP BY m.idm, m.nombrem`: Después del `JOIN`, agrupamos los resultados por las columnas `idm` (identificador del marinero) y `nombrem` (nombre del marinero). Agrupar por estas columnas significa que cualquier cálculo de agregación (como `COUNT`) se realizará por cada marinero individualmente.
3. **Condición de agregación (`HAVING COUNT(DISTINCT r.idb) = (SELECT COUNT(*) FROM barcos)`)**:
   - `HAVING COUNT(DISTINCT r.idb) = (SELECT COUNT(*) FROM barcos)`: Esta es una condición de filtrado que se aplica a los grupos formados por el `GROUP BY`. Funciona de la siguiente manera:

- `COUNT(DISTINCT r.idb)`: Cuenta el número de barcos distintos (`idb`) que cada marinero ha reservado. El uso de `DISTINCT` asegura que cada barco se cuente solo una vez por marinero, incluso si el marinero ha reservado el mismo barco más de una vez.
- `(SELECT COUNT(*) FROM barcos)`: Esta subconsulta cuenta el número total de barcos disponibles en la tabla `barcos`.
- `HAVING COUNT(DISTINCT r.idb) = (SELECT COUNT(*) FROM barcos)`: Esta condición asegura que solo los marineros que han reservado un número de barcos igual al total de barcos disponibles sean seleccionados. En otras palabras, solo selecciona a los marineros que han reservado todos los barcos.

Detalle del Funcionamiento:

- **JOIN**: Se combinan las filas de `marineros` y `reservas` donde `marineros.idm = reservas.idm`. Esto nos da una lista de todas las reservas hechas por cada marinero.
- **GROUP BY**: Agrupa los resultados de la combinación por `idm` y `nombrem`, de modo que cualquier cálculo de agregación, como `COUNT`, se aplique por separado a cada marinero.
- **COUNT(DISTINCT r.idb)**: Dentro de cada grupo (cada marinero), cuenta los diferentes `idb` (barcos) que el marinero ha reservado. Si un marinero ha reservado el mismo barco más de una vez, este barco solo se cuenta una vez debido al `DISTINCT`.
- **Subconsulta `(SELECT COUNT(*) FROM barcos)`**: Esta subconsulta cuenta cuántos barcos hay en total en la tabla `barcos`.
- **HAVING**: Después de agrupar, `HAVING` filtra los grupos para incluir solo aquellos donde el número de barcos distintos reservados por el marinero es igual al número total de barcos. Esto asegura que solo los marineros que han reservado todos los barcos estén en el resultado final.

Ejemplo con Datos: Supongamos las siguientes tablas:

`marineros`:

idm	nombrem	categoría	edad
1	John	1	30.5
2	Alice	2	25.0
3	Bob	1	28.0

`barcos`:

idb	nombreb	color
1	Boat1	Red
2	Boat2	Blue
3	Boat3	Green

`reservas`:

idm	idb	fecha
1	1	2023-01-01
1	2	2023-02-01
1	3	2023-03-01
2	1	2023-01-01

idm	idb	fecha
2	2	2023-02-01

Aplicación de la Consulta:

1. **JOIN**:

```sql
SELECT m.idm, m.nombrem, r.idb
FROM marineros AS m INNER JOIN reservas AS r ON m.idm = r.idm;
```

Resulta en:

idm	nombrem	idb
1	John	1
1	John	2
1	John	3
2	Alice	1
2	Alice	2

2. **GROUP BY y HAVING**:

```sql
SELECT m.idm, m.nombrem
FROM marineros AS m INNER JOIN reservas AS r ON m.idm = r.idm
GROUP BY m.idm, m.nombrem
HAVING COUNT(DISTINCT r.idb) = (
    SELECT COUNT(*)
    FROM barcos
);
```

Resulta en:

- `John` ha reservado 3 barcos, y hay 3 barcos en total: coincide.
- `Alice` ha reservado 2 barcos, y hay 3 barcos en total: no coincide.

Resultado Final:

nombrem
John

El resultado muestra que solo `John` ha reservado todos los barcos disponibles.

Opción 3: Utilizando `NOT EXISTS` y `EXCEPT`

```sql
SELECT m.nombrem
FROM marineros AS m
WHERE NOT EXISTS (
    SELECT b.idb
    FROM barcos AS b

    EXCEPT
```

```
        SELECT r.idb
        FROM reservas AS r
        WHERE r.idm = m.idm
    );
```

Paso a Paso:

1. **Consulta externa (`SELECT m.nombrem FROM marineros AS m`)**:
   - Esta consulta selecciona los nombres (`nombrem`) de todos los marineros (`marineros`).
2. **Subconsulta en `NOT EXISTS`**:
   - La subconsulta verifica si para cada marinero (`m`), la diferencia entre todos los barcos (`barcos`) y los barcos reservados por ese marinero (`reservas`) está vacía. Si la diferencia está vacía, significa que el marinero ha reservado todos los barcos.
3. **Subconsulta interna (`SELECT b.idb FROM barcos AS b EXCEPT SELECT r.idb FROM reservas AS r WHERE r.idm = m.idm`)**:
   - La subconsulta interna tiene dos partes:
     - **Primera parte (`SELECT b.idb FROM barcos AS b`)**: Selecciona los identificadores de todos los barcos (`idb`) de la tabla `barcos`.
     - **Segunda parte (`SELECT r.idb FROM reservas AS r WHERE r.idm = m.idm`)**: Selecciona los identificadores de barcos (`idb`) que han sido reservados por el marinero actual (`m.idm`) de la tabla `reservas`.

Explicación del `EXCEPT`:

- La operación `EXCEPT` devuelve todas las filas del primer conjunto (todos los barcos) que no están en el segundo conjunto (barcos reservados por el marinero actual). Si esta diferencia es vacía, significa que el marinero ha reservado todos los barcos.

Detalle del Funcionamiento:

1. **Subconsulta interna (`SELECT b.idb FROM barcos AS b EXCEPT SELECT r.idb FROM reservas AS r WHERE r.idm = m.idm`)**:
   - Esta subconsulta devuelve los `idb` de los barcos que no han sido reservados por el marinero actual (`m.idm`).
   - Si el resultado de esta subconsulta es vacío para un marinero, eso significa que el marinero ha reservado todos los barcos.
2. **Condición `NOT EXISTS`**:
   - `NOT EXISTS` evalúa si la subconsulta interna devuelve algún resultado. Si no devuelve ningún resultado (es decir, todos los barcos han sido reservados por el marinero), la condición se evalúa como verdadera.
   - Si la subconsulta interna devuelve algún barco (es decir, hay al menos un barco que no ha sido reservado por el marinero), la condición se evalúa como falsa.

Ejemplo con Datos: Supongamos las siguientes tablas:

`marineros`:

idm	nombrem	categoría	edad
1	John	1	30.5
2	Alice	2	25.0
3	Bob	1	28.0

`barcos`:

idb	nombreb	color
1	Boat1	Red
2	Boat2	Blue
3	Boat3	Green

`reservas`:

idm	idb	fecha
1	1	2023-01-01
1	2	2023-02-01
1	3	2023-03-01
2	1	2023-01-01
2	2	2023-02-01

Aplicación de la Consulta: Para cada marinero (`m`), evaluamos la subconsulta:

Para John (idm = 1):

- **Todos los barcos**: {1, 2, 3}
- **Barcos reservados por John**: {1, 2, 3}
- **Diferencia**: {} (vacío)

Para Alice (idm = 2):

- **Todos los barcos**: {1, 2, 3}
- **Barcos reservados por Alice**: {1, 2}
- **Diferencia**: {3}

Para Bob (idm = 3):

- **Todos los barcos**: {1, 2, 3}
- **Barcos reservados por Bob**: {} (vacío)
- **Diferencia**: {1, 2, 3}

Evaluación de `NOT EXISTS`:

- **Para John**: La subconsulta devuelve vacío, `NOT EXISTS` es verdadero.
- **Para Alice**: La subconsulta devuelve {3}, `NOT EXISTS` es falso.
- **Para Bob**: La subconsulta devuelve {1, 2, 3}, `NOT EXISTS` es falso.

Resultado Final:

nombrem
John

El resultado muestra que solo `John` ha reservado todos los barcos disponibles.

Resumen del Funcionamiento:

- La consulta selecciona los nombres de los marineros que no tienen ningún barco sin reservar.
- Utiliza `NOT EXISTS` para verificar que no haya barcos que el marinero no haya reservado.
- La subconsulta con `EXCEPT` compara todos los barcos con los barcos reservados por el marinero actual y verifica que la diferencia sea vacía.

  Esta es mi opción favorita de las tres.

c25: Averiguar el promedio de edad de los marineros.

```
SELECT AVG (m.edad)
FROM marineros AS m;
```

Encerrar entre paréntesis el parámetro es importante, ya que de otra forma la consulta no funciona.

Si quisiéramos averiguar el marinero más joven o el más viejo podríamos utilizar los operadores de agregación `MIN` o `MAX`, respectivamente.

Consulta extra: Averiguar qué edad tiene el marinero más joven.

```
SELECT MIN (m.edad)
FROM marineros AS m;
```

Esta consulta devuelve, con la tabla que tenemos cargada, el valor 16, sin ningún otro campo más que lo acompañe.

¿Y qué pasaría si me pidieran que, además de su edad, tengo que hallar el nombre del marinero, digamos esta vez, más viejo?

Sirva otra vuelta de consulta extra, pulpero: Averiguar el nombre y la edad del marinero más vetusto.

Opción 1:

```
SELECT m.nombrem, m.edad
FROM marineros AS m
WHERE m.edad IN (
    SELECT MAX (m2.edad)
    FROM marineros AS m2
);
```

Opción 2:

```sql
SELECT m.nombrem, m.edad
FROM marineros AS m
WHERE m.edad = (
    SELECT MAX(m2.edad)
    FROM marineros AS m2
);
```

Opción 3: Si hacés esta, te falla.

```sql
SELECT m.nombrem, m.edad
FROM marineros AS m
WHERE (SELECT MAX(m2.edad) FROM marineros AS m2) = m.edad;
```

c28: Contar el número de marineros.

```sql
SELECT COUNT(*)
FROM marineros AS m;
```

El resultado de esta consulta es 10.

El argumento * resulta útil cuando se desea contar todas las filas. Lo podemos considerar como una "abreviatura" de todas las columnas. También podemos pasarle como argumento el número 1 y el resultado será el mismo.

c29: Contar el número de nombres de marineros diferentes.

```sql
SELECT COUNT(DISTINCT m.nombrem)
FROM marineros AS m;
```

El resultado de esta consulta es 9 (porque el nombre "Horacio" está repetido dos veces).

c30: Averiguar el nombre de los marineros de más edad que el marinero más viejo de la categoría 10.

Opción 1:

```sql
SELECT m.nombrem
FROM marineros AS m
WHERE m.edad > (
    SELECT MAX(m2.edad)
    FROM marineros AS m2
    WHERE m2.categoría = 10
);
```

Los operadores de agregación ofrecen una alternativa al uso de las estructuras `ANY/SOME` y `ALL`, pues otra forma de escribir esta consulta es la siguiente:

```sql
SELECT m.nombrem
FROM marineros AS m
WHERE m.edad > ALL (
    SELECT m2.edad
    FROM marineros AS m2
```

```
    WHERE m2.categoría = 10
);
```

Dato de color: La consulta con `ALL` es más susceptible a sufrir errores. El empleo de `ANY`/`SOME` se corresponde con el de `MIN` en lugar de `MAX`.

c33: Para cada barco rojo, averiguar el número de reservas realizadas.

```
SELECT b.idb, COUNT (*) AS número_de_reservas
FROM barcos AS b INNER JOIN reservas AS r ON b.idb = r.idb
WHERE b.color LIKE "rojo"
GROUP BY b.idb;
```

c34: Averiguar la edad media de los marineros de cada categoría que tenga, como mínimo, dos marineros.

```
SELECT m.categoría, AVG(m.edad) AS edad_media
FROM marineros AS m
GROUP BY m.categoría
HAVING COUNT (*) >= 2;
```

c34: Averiguar la edad media de los marineros con derecho a voto (mayores de 18 años) para cada categoría que tenga, como mínimo, dos marineros.

Opción 1:

```
SELECT m.categoría, AVG(m.edad) AS edad_media
FROM marineros AS m
WHERE m.edad >= 18
GROUP BY m.categoría
HAVING COUNT (*) >= 2;
```

La cláusula `WHERE` se aplica antes de llevar a cabo la agrupación; por lo tanto solo quedan los marineros con edad >= 18 cuando se agrupa.

Opción 2:

```
SELECT temp.categoría, temp.edad_media
FROM (
    SELECT m.categoría, AVG(m.edad) AS edad_media, COUNT(*) AS num_categ
    FROM marineros AS m
    WHERE m.edad >= 18
    GROUP BY m.categoría
    ) AS temp
WHERE temp.num_categ >= 2;
```

Esta alternativa trae a colación varios puntos interesantes:

1. La cláusula `FROM` puede contener subconsultas anidadas.
2. La cláusula `HAVING` no es necesaria en lo absoluto. Cualquier consulta con con cláusula `HAVING` se puede reescribir sin ella. La misma existe porque muchas veces hace que la consulta

sea más sencilla.

3. Cuando aparecen subconsultas en la cláusula `FROM` es necesario emplear la palabra clave `AS` para darle un nombre que nos permita utilizar la tabla.

c37: Averiguar las categorías para las que la edad media de los marineros es mínima.

Opción 1: Utilizando subconsultas anidadas

```
SELECT sub.categoría, sub.edad_media
FROM (
    SELECT m.categoría, AVG(m.edad) AS edad_media
    FROM marineros AS m
    GROUP BY m.categoría
) AS sub
WHERE sub.edad_media = (
    SELECT MIN(sub2.edad_media)
    FROM (
        SELECT AVG(m2.edad) AS edad_media
        FROM marineros AS m2
        GROUP BY m2.categoría
    ) AS sub2
);
```

Objetivo de la Consulta: Encontrar las categorías (`categoría`) en las que la edad media (`edad_media`) de los marineros es la mínima entre todas las categorías.

Explicación Paso a Paso:

1. **Subconsulta Interna `sub2`**:

```
    SELECT AVG(m2.edad) AS edad_media
    FROM marineros AS m2
    GROUP BY m2.categoría
```

   - **Descripción**:
     - Esta subconsulta calcula la edad media (`AVG(m2.edad)`) para cada categoría (`m2.categoría`).
     - Agrupa los resultados por `m2.categoría` para obtener una edad media por cada categoría.
   - **Resultado**:
     - Una tabla con la edad media para cada categoría. Por ejemplo:

categoría	edad_media
1	29
2	24.67
3	35

2. **Subconsulta `sub`**:

```
    SELECT m.categoría, AVG(m.edad) AS edad_media
    FROM marineros AS m
    GROUP BY m.categoría
```

- **Descripción**:
  - Similar a `sub2`, esta subconsulta también calcula la edad media (`AVG(m.edad)`) para cada categoría (`m.categoría`).
  - Agrupa los resultados por `m.categoría` para obtener una edad media por cada categoría.
- **Resultado**:
  - Una tabla con la edad media para cada categoría, que se usará en la consulta principal. Por ejemplo:

categoría	edad_media
1	29
2	24.67
3	35

3. **Encontrar la Edad Media Mínima**:

```sql
SELECT MIN(sub2.edad_media)
FROM (
    SELECT AVG(m2.edad) AS edad_media
    FROM marineros AS m2
    GROUP BY m2.categoría
) AS sub2
```

- **Descripción**:
  - Esta subconsulta anidada encuentra la edad media mínima (`MIN(sub2.edad_media)`) de todas las categorías calculadas en la subconsulta `sub2`.
- **Resultado**:
  - Un solo valor que representa la edad media mínima. Por ejemplo: 24.67.

4. **Consulta Principal**:

```sql
SELECT sub.categoría, sub.edad_media
FROM (
    SELECT m.categoría, AVG(m.edad) AS edad_media
    FROM marineros AS m
    GROUP BY m.categoría
) AS sub
WHERE sub.edad_media = (
    SELECT MIN(sub2.edad_media)
    FROM (
        SELECT AVG(m2.edad) AS edad_media
        FROM marineros AS m2
        GROUP BY m2.categoría
    ) AS sub2
);
```

- **Descripción**:
  - La consulta principal selecciona las categorías (`sub.categoría`) y sus edades medias (`sub.edad_media`) de la subconsulta `sub`.
  - Filtra los resultados para incluir solo aquellas categorías cuya edad media es igual a la edad media mínima obtenida de la subconsulta que encuentra la edad media mínima.

- **Resultado**:
  - Una tabla con las categorías que tienen la edad media mínima. Por ejemplo:

categoría	edad_media
2	24.67

Resumen del Funcionamiento:

1. **Subconsulta `sub`**: Calcula la edad media para cada categoría.
2. **Subconsulta `sub2` dentro de `WHERE`**: Calcula nuevamente la edad media para cada categoría y encuentra la mínima de esas edades medias.
3. **Condición `WHERE`**: Filtra las categorías en la consulta principal para incluir solo aquellas cuya edad media coincide con la edad media mínima calculada.

Ejemplo con Datos Supongamos la siguiente tabla `marineros`:

idm	nombrem	categoría	edad
1	John	1	30
2	Alice	2	25
3	Bob	1	28
4	Carol	2	22
5	Dave	3	35
6	Eve	2	27

1. **Resultado de `sub2`**:
   - Calcula las edades medias:

categoría	edad_media
1	29
2	24.67
3	35

   - Edad media mínima: 24.67
2. **Resultado de `sub`**:
   - Calcula las edades medias:

categoría	edad_media
1	29
2	24.67
3	35

3. **Consulta Principal**:
   - Filtra por `edad_media` = 24.67:

categoría	edad_media
2	24.67

Este análisis muestra cómo la consulta encuentra las categorías para las cuales la edad media de los marineros es mínima, utilizando subconsultas para calcular y comparar las edades medias.

Opción 2: Utilizar `CTEs`: `Common Table Expressions`

Existe una forma más eficiente y sencilla de escribir la consulta para averiguar las categorías para las que la edad media de los marineros es mínima. En lugar de utilizar subconsultas anidadas, podemos emplear una combinación de `WITH` (Common Table Expressions, CTE) para estructurar la consulta de una manera más clara y eficiente

```
WITH categoría_edades AS (
    SELECT m.categoría, AVG(m.edad) AS edad_media
    FROM marineros AS m
    GROUP BY m.categoría
),
edad_media_mínima AS (
    SELECT MIN(ce.edad_media) AS edad_media_min
    FROM categoría_edades AS ce
)
SELECT ce.categoría, ce.edad_media
FROM categoría_edades AS ce INNER JOIN edad_media_mínima AS em ON ce.edad_media =
↪  em.edad_media_min;
```

Explicación Paso a Paso

1. **CTE `categoría_edades`**:
   - Calcula la edad media (`AVG(m.edad)`) para cada categoría (`m.categoría`).
   - Agrupa los resultados por `m.categoría` para obtener una edad media por cada categoría.

   ```
   WITH categoría_edades AS (
       SELECT m.categoría, AVG(m.edad) AS edad_media
       FROM marineros AS m
       GROUP BY m.categoría
   )
   ```

2. **CTE `edad_media_mínima`**:
   - Encuentra la edad media mínima (`MIN(ce.edad_media)`) de todas las categorías calculadas en `categoría_edades`.

   ```
   edad_media_mínima AS (
       SELECT MIN(ce.edad_media) AS edad_media_min
       FROM categoría_edades AS ce
   )
   ```

3. **Consulta Principal**:
   - Selecciona las categorías (`ce.categoría`) y sus edades medias (`ce.edad_media`) de `categoría_edades` donde la edad media es igual a la edad media mínima encontrada en `edad_media_mínima`.

```
SELECT ce.categoría, ce.edad_media
FROM categoría_edades AS ce INNER JOIN edad_media_mínima AS em ON ce.edad_media
↪  = em.edad_media_min;
```

Ejemplo con Datos Supongamos la siguiente tabla `marineros`:

idm	nombrem	categoría	edad
1	John	1	30
2	Alice	2	25
3	Bob	1	28
4	Carol	2	22
5	Dave	3	35
6	Eve	2	27

Cálculo de Edad Media por Categoría

1. **`categoría_edades`**:
   - Categoría 1: Edad Media = (30 + 28) / 2 = 29
   - Categoría 2: Edad Media = (25 + 22 + 27) / 3 = 24.67
   - Categoría 3: Edad Media = 35
2. **`edad_media_mínima`**:
   - Edad Media Mínima: 24.67

Resultado Final

categoría	edad_media
2	24.67

Ventajas de la Versión Optimizada

1. **Claridad**:
   - La consulta es más fácil de leer y entender debido a la estructura clara proporcionada por los CTEs.
2. **Mantenimiento**:
   - Es más fácil de mantener y modificar en el futuro, ya que cada parte de la consulta está claramente separada y etiquetada.
3. **Rendimiento**:
   - Dependiendo del motor de base de datos, el uso de CTEs puede mejorar el rendimiento, ya que algunos motores pueden optimizar mejor las consultas estructuradas de esta manera.

Resumen La versión optimizada de la consulta utiliza CTEs para calcular primero las edades medias por categoría y luego la edad media mínima. Finalmente, selecciona las categorías cuya edad media coincide con la edad media mínima. Esta estructura hace que la consulta sea más clara y posiblemente más eficiente.