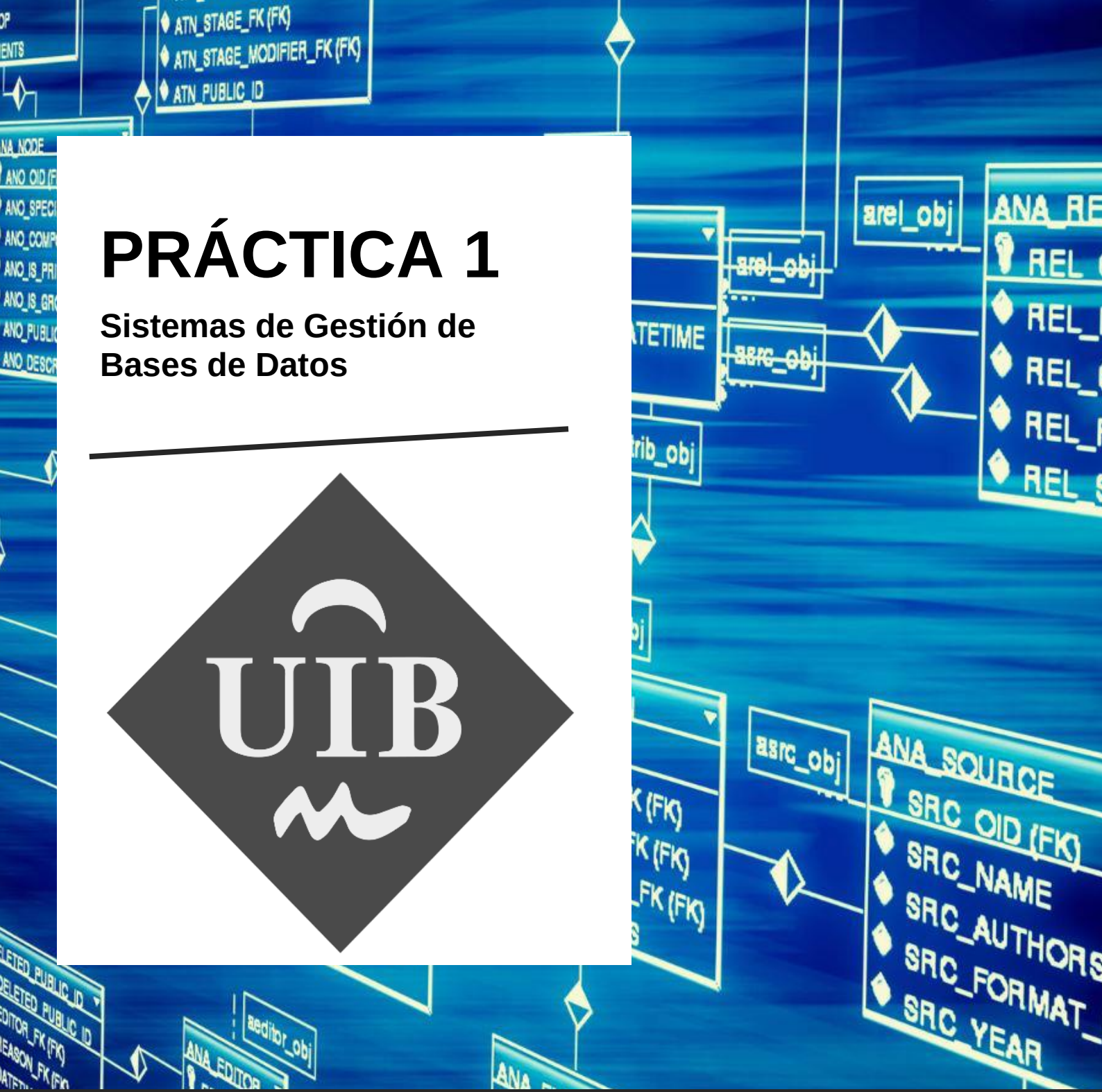

PRÁCTICA 1

Sistemas de Gestión de Bases de Datos

22 OCTUBRE

PRÁCTICA 1 – EJERCICIO 17
Mario Ventura Burgos 43223476-J
Grado en Ingeniería Informática (GIN 3)
CURSO 2023-2024

Para optimizar las consultas y buscar mejorar el modelo de forma significativa se podría estudiar la posibilidad de aumentar la velocidad de las consultas añadiendo índices, creando vistas de diferentes tipos, etc

Analicemos cada una de las consultas para ver si seria posible mejorar si rendimiento o no:

1. PRIMERA CONSULTA

La consulta 1 del examen se desarrolló en el documento mvb135_2.sql y es la siguiente:

```
SELECT ALI.referencia, ALI.nom
FROM Aliment ALI
    JOIN Linia_Albara LA
    ON LA.referencia = ALI.referencia
        JOIN Albara ALB
        ON ALB.codi = LA.codi_alb
            JOIN  roveidor PRO
            ON PRO.nif = ALB.nif_pro
            AND PRO.nom = 'UIBFruita'
ORDER BY ALI.referencia ASC;
```

En este caso, la consulta busca un proveedor de nombre UIBFruita. Teniendo en cuenta que, pese a que el nombre del proveedor no es PK de su tabla, todos los nombres de los proveedores presentan unicidad y por tanto son todos diferente entre sí; podríamos aprovechar esta propiedad para tratar esta columna como si verdaderamente fuera una clave y añadir un índice que permitiese que esta consulta se ejecutase más rápido. Esto permitiría obtener el mismo resultado en menor tiempo, y eso supone una mejora evidente.

Se podría crear un índice de la siguiente manera:

```
CREATE UNIQUE INDEX IND_PRO_UIBFRUITA ON PROVEIDOR (NOM) WHERE NOM = 'UIBFruita';
```

Una vez hecho esto, podemos hacer el EXPLAIN SELECT de igual manera que hicimos para obtener el plan de ejecución inicial. De esta forma, viendo los valores del coste podremos comparar y deducir si se ha obtenido alguna mejor.

El output obtenido es el siguiente:

Sort (cost=30.35..30.35 rows=2 width=162)
** Sort Key: ali.referencia**
** -> Nested Loop (cost=8.45..30.34 rows=2 width=162)**
**   -> Nested Loop (cost=8.30..29.90 rows=2 width=44)**
**   -> Hash Join (cost=8.15..29.28 rows=2 width=12)**
**   Hash Cond: ((alb.nif_pro)::text = (pro.nif)::text)**

-> Seq Scan on albara alb (cost=0.00..18.80 rows=880 width=52)

-> Hash (cost=8.14..8.14 rows=1 width=40)

-> Index Scan using IND_PRO_UIBFRUITA on proveidor pro (cost=0.12..8.14 rows=1 width=40)

-> Index Only Scan using pk_linalb on linia_albara la (cost=0.15..0.27 rows=4 width=56)

Index Cond: (codi_alb = alb.codi)

-> Index Scan using pk_aliment on aliment ali (cost=0.15..0.21 rows=1 width=162)

Index Cond: ((referencia)::text = (la.referencia)::text)

Tal y como puede verse, ahora, en lugar de realizar una búsqueda sencuencial sobre la tabla Proveidor en busca de coincidencias con el nombre de este, se usa el índice (**-> Index Scan using IND_PRO_UIBFRUITA on proveidor pro (cost=0.12..8.14 rows=1 width=40)**).

Esto hace que el coste total de las operaciones ha disminuido. Cuando se hizo el EXPLAIN SELECT la primera vez, el coste era de 38.71..38.72 unidades de coste, y ahora esta magnitud se ha visto reducida a 30.35 unidades. Esto supone más de un 20% de mejora y demuestra que el uso del índice IND_PRO_UIBRUITA ha servido para mejorar la consulta.

2. SEGUNDA CONSULTA

La segunda consulta del examen es la siguiente:

```
SELECT ALB.codi, PRO.nom
FROM Albara ALB
    JOIN Proveidor PRO
    ON PRO.nif = ALB.nif_pro    /*Hay que entrar en la tabla ya que se quiere algo de
ella*/
        JOIN Linia_ALbara LA
        ON LA.codi_alb = ALB.codi
        AND LA.preu IS NULL     /*Sin especificar precio = columna precio con valor
NULL*/
WHERE ALB.facturat = 'S';       /*Facturados = columna facturat con valor 'S'*/
```

Tal y como se hizo anteriormente para mejorar la primera consulta, en este caso también se pueden usar índices para mejorar la consulta. Esto se debe a que vemos que, en este caso, se establecen condiciones de unión entorno a que el precio en Linia_Albara no esté especificado (preu IS NULL) y el albarán no este facturado (facturat = 'S'). Se pueden crear otros dos índices en estas dos columnas con el objetivo de mejorar la consulta:

```
CREATE INDEX IND_ALB_FACT ON ALBARA (FACTURAT) WHERE FACTURAT IN ('S');
CREATE INDEX IND_PREU_NULL ON LINIA_ALBARA (PREU) WHERE PREU IS NULL;
```

EL output obtenido es el siguiente:

Nested Loop (cost=8.41..30.61 rows=1 width=130)
-> Nested Loop (cost=8.26..25.42 rows=1 width=52)
Join Filter: (la.codi_alb = alb.codi)
-> Bitmap Heap Scan on albara alb (cost=4.13..12.59 rows=4 width=52)
Recheck Cond: ((facturat)::text = 'S'::text)
-> Bitmap Index Scan on IND_ALB_FACT (cost=0.00..4.13 rows=4 width=0)
-> Materialize (cost=4.13..12.60 rows=4 width=12)
-> Bitmap Heap Scan on linia_albara la (cost=4.13..12.58 rows=4 width=12)
Recheck Cond: (preu IS NULL)
-> Bitmap Index Scan on IND_PREU_NULL (cost=0.00..4.13 rows=4 width=0)
-> Index Scan using pk_proveidor on proveidor pro (cost=0.15..5.17 rows=1 width=158)
Index Cond: ((nif)::text = (alb.nif_pro)::text)

Tal y como puede observarse, una vez más, el coste máximo de la consulta ha disminuido, pasando de 46.24 originalmente a 30.61 conseguidos mediante el uso de índices. Esto supone que la consulta ha mejorado un 33,81% respecto a la consulta original que no usaba índices.
Esta mejora se puede atribuir a, entre otras cosas, el uso de técnicas como *Bitmap Heap Scan* con índices Bitmap (*Bitmap index*).

3. TERCERA CONSULTA

La tercera consulta es la siguiente:

```
SELECT ALI.referencia, LA.preu
FROM Aliment ALI
    JOIN linia_albara LA
    ON LA.referencia = ALI.referencia
        JOIN Albara ALB
        ON ALB.codi = LA.codi_alb
WHERE ALB.data = (
        SELECT MAX(A.data)
        FROM ALBARA A
            JOIN linia_albara LINALB
            ON LINALB.codi_alb = A.codi
        WHERE LINALB.referencia = LA.referencia
    )
AND LA.preu IS NOT NULL;
```

Teniendo en cuenta la complejidad de esta consulta, sabiendo que usa una subconsulta (SELECT anidado) y teniendo en cuenta la poca eficiencia que tienen los selects anidados, lo mejor probablemente sería eliminar esta subconsulta y sustituirla por una vista tal y como se hace en el ejercicio 4 con el uso de vistas con sus respectivas consultas cada una.

Probemos si esto mejora la eficiencia de la consulta:

```
CREATE MATERIALIZED VIEW MAXDATA AS
    SELECT LA.referencia, MAX(A.data) AS MAX
    FROM ALBARA A
        JOIN Linia_ALbara LA
        ON LA.codi_alb = A.codi
    GROUP BY LA.referencia;
```

Y la consulta modificada sería la siguiente:

```
SELECT ALI.referencia, LA.preu
FROM Aliment ALI
    JOIN linia_albara LA
    ON LA.referencia = ALI.referencia
        JOIN Albara ALB
        ON ALB.codi = LA.codi_alb
            JOIN MAXDATA MD
            ON MD.referencia = LA.referencia
WHERE ALB.data = MD.MAX
AND LA.preu IS NOT NULL;
```

Además, se puede ver que la consulta de dentro de la vista materializada creada puede contener índices para ser todavía más eficiente:

```
CREATE INDEX IND_MAXDATA ON MAXDATA (referencia,MAX);
```

Si comprobamos el resultado que se obtiene al hacer el EXPLAIN SELECT veremos que el coste máximo de la consulta ha pasado de ser 1904.30 (un coste totalmente desproporcionado e inaceptable. Es el mayor de los costes de todas las consultas y se puede ver que es notablemente mayor al coste máximo del resto de estas) a ser 121.01. Esto supone que se ha mejorado un 93,65% el rendimiento de la consulta.

Esto supone una mejora enorme y pone en alza la poca eficiencia del uso de consultas anidadas dentro de otras en contraposición con el uso de vistas, índices o ambas como en este caso.

4. CUARTA Y QUINTA CONSULTA

Las consultas 4 y 5 son las siguientes:

Consulta 4

```
CREATE VIEW QUILOSVENDA AS
    SELECT VEN.referencia, SUM(VEN.quilograms) AS KG_VENUTS
    FROM Venda VEN
    GROUP BY VEN.referencia;

CREATE VIEW QUILOSALBARA AS
    SELECT LA.referencia, SUM(LA.quilograms) AS KG
    FROM linia_albara LA
    GROUP BY LA.referencia;

SELECT ALI.referencia, ALI.nom, (COALESCE(QUILOSA.KG, 0) - COALESCE(QUILOSV.KG_VENUTS,
0)) AS STOCK
FROM Aliment ALI
    LEFT JOIN QUILOSALBARA QUILOSA
    ON QUILOSA.referencia = ALI.referencia
        LEFT JOIN QUILOSVENDA QUILOSV
        ON QUILOSV.referencia = ALI.referencia;
```

Consulta 5

```
SELECT VEN.codi, COUNT(*) AS NUM_COD
FROM Venda VEN
GROUP BY VEN.codi
HAVING COUNT(*) > 1;
```

En estos dos últimos casos, no se han añadido mejoras a las consultas.

- En el caso de la consulta 4, esta ya usa vistas y es “bastante eficiente”. De hecho, es la más eficiente de las consultas mostradas ya que el uso de vistas hace que el coste sea ínfimo en comparación al resto de queries desarrolladas.

  Si esta consulta, al igual que la anterior, usase selects anidados (subconsultas) en lugar de vistas, se añadirían las vistas como mejoras y se podría apreciar una mejora notable en el rendimiento de la consulta ; pero en este caso la consulta original ya fue planteada con vistas.

- Por otro lado, dada la sencillez de la quinta consulta, tampoco se usará ni vistas ni índices para mejorarla. De hecho, usar vistas sería absurdo. Como mucho podríamos plantearnos el uso de índices, pero teniendo en cuenta la simplicidad de la consulta, el uso de índices no mejoraría el rendimiento