

Family name: Duran López Given name: Marc

Family name: López González Given name: Alex

Question 1

For each question in the 3rd Lab SQL quiz:

- 1) Explain what structures you created
 - a. Question 1:

En un principio, probamos con arboles B+ dado que los mapas de bits, que era otra de las opciones que barajamos al empezar, tienen un proceso de creación costoso. Sin embargo, en el momento de analizarlo, descubrimos que esta segunda opción resulta mucho más económica en comparación con la implementación de árboles B+, por la que optamos primeramente.

A continuación, pensamos que el modo de que fuera lo más eficiente posible era que todos los atributos tuvieran índices, la primera idea que se nos viene es crear un bitmap por cada atributo, pero después de probar y ver cómo actuaban estos índices creados caímos en que la posibilidad de agrupar los atributos 'cand' y 'val' en un solo bitmap generaba mejoras importantes, dado que este bitmap es el más utilizado.

Después de asignar los índices, también sabíamos que la Consulta 3 representaba la mitad de todas las ejecuciones totales y, por lo tanto, consideramos favorable la creación de una vista materializada específica para esta consulta. Esta aproximación nos permitiría acceder directamente a los datos almacenados en disco y así no habría que repetir el proceso de resolución de la consulta en múltiples ocasiones.

En resumen, las estructuras de datos que implementamos incluyeron tres mapas de bits diferentes: uno para el atributo 'población', otro para el atributo 'edad' y un tercero para los atributos 'cand' y 'val' agrupados. Además, incorporamos una vista materializada para la Consulta 3. Con esta configuración conseguimos un rendimiento muy eficiente y no se superaban los bloques establecidos para el ejercicio.

También aplicamos la compresión a los índices de los mapas de bits como vimos en clase, mediante el comando "ALTER TABLE poll answers MINIMIZE RECORDS PER BLOCK," y como resultado, obtuvimos mapas de bits eficazmente comprimidos.

b. Question 2:

Después de implementar el método de la matriz de afinidad, vimos que lo ideal era fragmentar nuestra tabla en tres partes:

- pobl y edat
- cand y val
- Respuestas y ref

Una vez implementado, vimos que a pesar de subir un poco el rendimiento, no podíamos seguir mejorando ya que indexar no producía ningún cambio en el resultado. A base de prueba y error, intentando mantenernos lo más cercanos a la estructura conseguida por la matriz de afinidad, la mejor estructura resultó ser la más alejada: todos los atributos a los que se acceden en las queries en un fragmento (es decir, pobl, edat, val y cand) y los resultados en el otro. Esto se debe a que Oracle, a pesar de estar equipado con tablas anidadas para implementar fragmentación vertical, es un sistema fundamentalmente orientado a filas.

Con esta estructura, hubo varias estrategias para indexar que mejoraban notablemente el rendimiento. Los bitmaps daban mejores resultados, ya que los atributos de las queries tenían un rango de valor pequeño y, por lo tanto, había considerables repeticiones de valor. Entre las estrategias probadas, las que mejor rendimiento dieron son:

- Indexar según los fragmentos obtenidos por la matriz de afinidad:
 - Bitmap(edat, pobl) y Bitmap(cand, val)
- Índice individual
 - Bitmap(edat), Bitmap(pobl), Bitmap(cand), Bitmap(val),

Pero la mejor resultó ser Bitmap(pobl), Bitmap(edat) y Bitmap(cand, val).

2) Explain Oracle's execution plan considering the structures you created

a. Question 1:

En la primera consulta, Oracle utiliza exclusivamente los índices de los mapas de bits 1 y 2, que corresponden a 'pobl' (bitmap1) y 'edat' (bitmap2).

Una vez que se realiza el escaneo, Oracle los transforma en rowids para su uso posterior en el acceso a la tabla, ya que Oracle está orientado a filas.

Después de convertir los dos mapas de bits en rowids, Oracle ejecuta una join de estos dos conjuntos. Esto resulta en la creación de una vista (index_join\$_001) y, posteriormente, se aplica la operación "group by" al campo 'pobl' de la consulta, como se puede ver en la figura 1. La decisión de utilizar los mapas de bits, convertirlos en rowids y luego realizar la join tiene sentido en este contexto, ya que la operación "group by" se realiza más adelante.

Finalmente, se realiza la selección de los resultados. Todo esto se puede ver en el plan de ejecución con el trazador seleccionando la primera consulta, como se muestra en la figura 1.

Operation	Object	Optimizer	Cost	Cardinality	Bytes
SELECT STATEMENT		ALL_ROWS	13	120	840
HASH (GROUP BY)			13	120	840
VIEW	index\$_join\$_001		11	20,000	140,000
HASH JOIN			11	0	0
BITMAP CONVERSION (TO ROWIDS)			5	20,000	140,000
BITMAP INDEX (FULL SCAN)	BITMAP2		0	0	0

SELECT pobl, MIN(edat), MAX(edat), COUNT(*) FROM poll_answers GROUP BY pobl

Figura 1: Plan de ejecución de la Consulta 1 (no se llega a ver el final)

En la segunda consulta, de manera similar a lo que se hizo en la consulta anterior, se inicia con el escaneo de los mapas de bits para los atributos 'edat', correspondientes al bitmap2, y ('cand', 'val'), correspondientes al bitmap3.

Después de convertir los dos mapas de bits en rowids, Oracle ejecuta una join de estos dos conjuntos. Esto resulta en la creación de una vista (index_join\$_001), hasta el momento se sigue el procedimiento realizado en la primera consulta.

Posteriormente, se crea una vista con el resultado y se aplica la operación "group by" a los campos 'pobl', 'edat' y 'cand'. Al igual que en la consulta 1, la decisión de utilizar los mapas de bits, convertirlos en rowids y realizar la join está justificada, ya que la operación "group by" se realiza más adelante.

Finalmente, se realiza la selección de los resultados. Esto se puede observar en el plan de ejecución con el rastreador seleccionando la segunda consulta, como se muestra en la figura 2.

Operation	Object	Optimizer	Cost	Cardinality	Bytes
SELECT STATEMENT		ALL_ROWS	24	20,000	260,000
HASH (GROUP BY)			24	20,000	260,000
VIEW	index\$_join\$_001		23	20,000	260,000
HASH JOIN			17	0	0
HASH JOIN			11	0	0
BITMAP CONVERSION (TO ROWIDS)			5	20,000	260,000
BITMAP INDEX (FULL SCAN)	BITMAP2		0	0	0
BITMAP CONVERSION (TO ROWIDS)			6	20,000	260,000
BITMAP INDEX (FULL SCAN)	BITMAP3		0	0	0
BITMAP CONVERSION (TO ROWIDS)			6	20,000	260,000
BITMAP INDEX (FULL SCAN)	BITMAP1		0	0	0

SELECT pobl, edat, cand, MAX(val), MIN(val), AVG(val) FROM poll_answers GROUP BY pobl, edat, cand

Figura 2: Plan de ejecución de la Consulta 2

En la consulta 3, se puede observar que tendrá un costo bajo, ya que, como se ha creado una vista materializada específica para esta consulta llamada "view_nameQ3", la única operación que se debe realizar es examinar la vista materializada y seleccionar los resultados.

Esto se puede ver en el plan de ejecución con el rastreador seleccionando la tercera consulta, como se muestra en la figura 3.

Operation	Object	Optimizer	Cost	Cardinality	Bytes
SELECT STATEMENT		ALL_ROWS	5	10	250
MAT_VIEW REWRITE ACCESS (FULL)	VIEW_NAMEQ3	ANALYZED	5	10	250

SELECT cand, AVG(val) FROM poll_answers GROUP BY cand

Figura 3: Plan de ejecución de la Consulta 3

b. Question 2:

Q1

En el plan de ejecución de la primera consulta con tablas anidadas, Oracle accede a los bitmaps correspondientes a pobl y edat. Seguidamente, Oracle convierte los mapas de bits en identificadores de filas. Aquí vemos que a pesar de estar usando estrategias de fragmentación vertical, Oracle continúa con un enfoque orientado a filas.

Oracle realiza una operación de "join" entre estos conjuntos de datos, lo que crea una vista temporal que se usará para la operación "group by" justo antes del select.

Como vemos, el plan de ejecución es muy parecido al de la sección de la primera pregunta

Operación	Objeto	Optimizador	Coste	Cardinalidad	Bytes
SELECT STATEMENT		ALL_ROWS	94	200	4,800
HASH (GROUP BY)			94	200	4,800
VIEW	index\$_join\$_002		93	20,000	480,000
HASH JOIN			103	0	0
HASH JOIN			13	0	0
BITMAP CONVERSION (TO ROWIDS)			7	20,000	480,000
BITMAP INDEX (FULL SCAN)	ID7		0	0	0
BITMAP CONVERSION (TO ROWIDS)			6	20,000	480,000
BITMAP INDEX (FULL SCAN)	ID8		0	0	0
INDEX (FAST FULL SCAN)	SYS_FK0002149...	ANALYZED	90	20,000	480,000

SELECT t.pobl AS a, MIN(t.edat) AS b, MAX(t.edat) AS c, COUNT(*) AS d
FROM poll_answers pa, TABLE(pa.pobl_edat) t
GROUP BY t.pobl

Figura 4: Plan de ejecución de la Consulta 1 con tablas anidadas

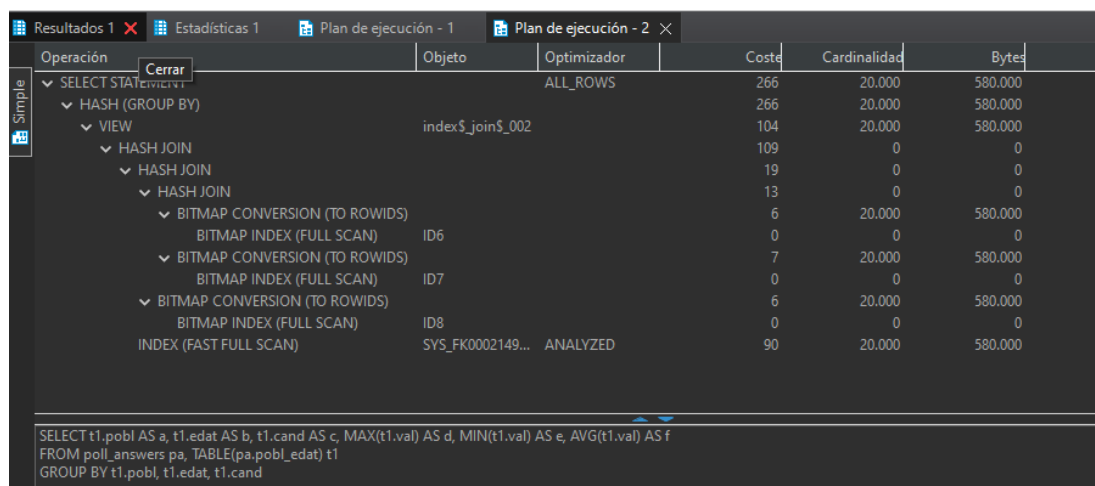
Q2

En el plan de ejecución de la segunda consulta, Oracle comienza su proceso a través de todos los mapas de bits creados (cand y val, edat, pobl). Estos índices pasan por un proceso de conversión, transformándose en identificadores de fila.

Después, Oracle procede a realizar múltiples operaciones de "join". Estas uniones permiten combinar la información de diferentes índices o tablas, aportando cohesión a los datos que se quieren obtener.

Por encima de estos "joins", Oracle genera una vista temporal, identificada como "index_join\$_002". Posteriormente se hace una agrupación, realizada con "HASH(GROUP BY)" justo antes del select.

Como en la primera consulta, el plan de ejecución con tablas anidadas es muy parecido al plan de ejecución sin tablas anidadas, haciendo otra vez evidente que la manera de procesar estas consultas de Oracle, a pesar de estar usando fragmentación vertical, es orientada a filas.



Operación	Objeto	Optimizador	Coste	Cardinalidad	Bytes
SELECT STATEMENT		ALL_ROWS	266	20.000	580.000
HASH (GROUP BY)			266	20.000	580.000
VIEW	index_join\$_002		104	20.000	580.000
HASH JOIN			109	0	0
HASH JOIN			19	0	0
HASH JOIN			13	0	0
BITMAP CONVERSION (TO ROWIDS)			6	20.000	580.000
BITMAP INDEX (FULL SCAN)	ID6		0	0	0
BITMAP CONVERSION (TO ROWIDS)			7	20.000	580.000
BITMAP INDEX (FULL SCAN)	ID7		0	0	0
BITMAP CONVERSION (TO ROWIDS)			6	20.000	580.000
BITMAP INDEX (FULL SCAN)	ID8		0	0	0
INDEX (FAST FULL SCAN)	SYS_FK0002149...	ANALYZED	90	20.000	580.000

SELECT t1.pobl AS a, t1.edat AS b, t1.cand AS c, MAX(t1.val) AS d, MIN(t1.val) AS e, AVG(t1.val) AS f
FROM poll_answers pa, TABLE(pa.pobl_edat) t1
GROUP BY t1.pobl, t1.edat, t1.cand

Figura 5: Plan de ejecución de la Consulta 2 con tablas anidadas

Q3

En la tercera consulta, el plan de ejecución es parecido a las anteriores pero más sencilla, ya que solo tiene que acceder a un mapa de bits (el de *cand* y *val*)

Operación	Objeto	Optimizador	Coste	Cardinalidad	Bytes
SELECT STATEMENT		ALL_ROWS	80	10	230
HASH (GROUP BY)			80	10	230
VIEW	index\$_join\$_002		78	20.000	460.000
HASH JOIN			96	0	0
BITMAP CONVERSION (TO ROWIDS)			6	20.000	460.000
BITMAP INDEX (FULL SCAN)	ID6		0	0	0
INDEX (FAST FULL SCAN)	SYS_FK0002149...	ANALYZED	90	20.000	460.000

SELECT t2.cand AS a, AVG(t2.val) AS b
FROM poll_answers pa, TABLE(pa.pobl_edat) t2
GROUP BY t2.cand

Figura 6: Plan de ejecución de la Consulta 3 con tablas anidadas

- 3) For the execution plan given, discuss what steps would be executed differently by a column-oriented databases (refer to the physical data structures as well as query processing techniques used)

(a) Question 1:

Si consideramos una base de datos orientada a columnas en lugar de una orientada a filas como en la situación anterior, notaríamos cambios significativos en relación con la "Pregunta 1":

En la Consulta 1, primero, en el contexto de una base de datos orientada a columnas, como no se requiere reconstruir la tabla para hacer el select, no sería necesario realizar la join de la población y la edad. Sería posible examinar las referencias (identificador) y obtener una solución ya que se podría acceder directamente a la tabla de edad, sin la necesidad de realizar ninguna optimización ni estrategia adicional, lo cual, simplifica considerablemente todo el proceso.

En la consulta 2, aplicaríamos la misma lógica que en la primera consulta. Con las referencias no sería necesario aplicar ninguna optimización, como podría ser la materialización tardía, ya que tiene toda la información que se necesita.

La consulta 3 ya es inherentemente eficiente por sí misma, ya que, existe una vista materializada.

En el caso de una base de datos orientada a columnas, la situación podría ser diferente. Ya que no habría materialización de la tabla y se tendría que recorrer igualmente para responder a las consultas. Esto es debido a que habría que realizar los procesos comentados en las otras dos consultas anteriores. Todo esto provoca que la tercera consulta sea más costosa si nos encontramos en una base de datos orientada a columnas.

(b) Question 2:

En una base de datos orientada a columnas, dado que su manera de almacenar datos ya proporciona un nivel de escaneo eficiente para este tipo de consultas, no habría sido necesario el uso de bitmaps. Además, no habría la necesidad de convertir el mapa en rowids ni de crear una vista temporal durante el join, lo que habría ahorrado un coste considerable

Question 2

For the second exercise, use the affinity matrix method to decide how to fragment the database vertically. Consider now your solution for Exercise 2 in the 3rd Lab SQL quiz. Is Oracle yielding the best result when using the same vertical fragmentation strategy as suggested by the affinity matrix method? **Justify your answer.**

El método de la matriz de afinidad implica evaluar la frecuencia de los patrones de acceso a los atributos en diferentes consultas para identificar el esquema de fragmentación más adecuado. En este laboratorio, lo hemos utilizado para fragmentar verticalmente nuestra relación mediante tablas anidadas. Para llevar a cabo este método, primero hemos generado la matriz de uso de los atributos para las *queries* del ejercicio, donde se muestra si el atributo ha sido accedido (con un 1) o no (con un 0).

Partiendo de esta matriz, hemos generado la matriz de afinidad de los atributos donde, por cada pareja de atributos, computamos la afinidad añadiendo las frecuencias con las que aparecen juntos. Esto nos lleva a ver qué atributos son más adecuados para crear un cluster

	ref	pobl	edat	cand	val	r_1	r_2	r_3	r_4	r_5
Q1	0	1	1	0	0	0	0	0	0	0
Q2	0	1	1	1	1	0	0	0	0	0
Q3	0	0	0	1	1	0	0	0	0	0

Figura 7: Matriz de uso de los atributos.

r_i (1 ≤ i ≤ 5) representa el atributo respuesta_i

	ref	pobl	edat	cand	val	r_1	r_2	r_3	r_4	r_5
ref	0	0	0	0	0	0	0	0	0	0
pobl	0	50	50	30	30	0	0	0	0	0
edat	0	50	50	30	30	0	0	0	0	0
cand	0	30	30	80	80	0	0	0	0	0
val	0	30	30	80	80	0	0	0	0	0
r_1	0	0	0	0	0	0	0	0	0	0
r_2	0	0	0	0	0	0	0	0	0	0
r_3	0	0	0	0	0	0	0	0	0	0
r_4	0	0	0	0	0	0	0	0	0	0
r_5	0	0	0	0	0	0	0	0	0	0

Figura 8: Matriz de afinidad de los atributos.

r_i ($1 \leq i \leq 5$) representa el atributo *respuesta_i*

Finalmente, reorganizamos la matriz para formar clusters donde haya una alta afinidad entre los atributos

	cand	val	pobl	edat	ref	r_1	r_2	r_3	r_4	r_5
cand	80	80	30	30	0	0	0	0	0	0
val	80	80	30	30	0	0	0	0	0	0
pobl	30	30	50	50	0	0	0	0	0	0
edat	30	30	50	50	0	0	0	0	0	0
ref	0	0	0	0	0	0	0	0	0	0
r_1	0	0	0	0	0	0	0	0	0	0
r_2	0	0	0	0	0	0	0	0	0	0
r_3	0	0	0	0	0	0	0	0	0	0
r_4	0	0	0	0	0	0	0	0	0	0
r_5	0	0	0	0	0	0	0	0	0	0

Figura 9: Matriz de afinidad de los atributos reordenada.

r_i ($1 \leq i \leq 5$) representa el atributo *respuesta_i*

Como muestra la matriz reordenada, la mejor opción es formar un cluster de alta afinidad entre *cand* y *val* y otro entre *pobl* y *edat*. Por lo tanto, a la hora de fragmentar la base de datos, quedaría un primer fragmento con los atributos *cand* y *val*, un segundo fragmento con *pobl* y *edat*, y finalmente otro con *ref*, *respuesta_1*, *respuesta_2*, *respuesta_3*, *respuesta_4* y *respuesta_5*.

Con esta estrategia de fragmentación el rendimiento supera al original. Sin embargo, Oracle es fundamentalmente un sistema orientado a filas optimizado para recuperar filas completas de datos. Aunque el algoritmo de matriz de afinidad se diseñó para bases de datos orientadas a columnas y Oracle ofrece tablas anidadas para tales arquitecturas, sigue procesando las consultas como un sistema basado en filas (como hemos podido ver en las secciones anteriores). Por lo tanto, nuestra fragmentación vertical no es la que mejores resultados da.