Family name: Llanza Carmona Given name: Anna

Family name: Llobet Rodriguez Given name: Javier

Question 1

For each question in the 3rd Lab SQL quiz:

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

Ejercicio 1 del 3rd Lab SQL quiz:

Para resolver el ejercicio 1, hemos utilizado un bitmap para cada atributo de la tabla *poll_answers* y la vista materializada de la tercera query:

- **CREATE BITMAP INDEX** idxbit_pobl **ON** POLL_ANSWERS (pobl) **PCTFREE** 0;
- **CREATE BITMAP INDEX** idxbit_edat **ON** POLL_ANSWERS (edat) **PCTFREE** 0;
- **CREATE BITMAP INDEX** idxbit_cand **ON** POLL_ANSWERS (cand) **PCTFREE** 0;
- **CREATE BITMAP INDEX** idxbit_val **ON** POLL_ANSWERS (val) **PCTFREE** 0;
- **CREATE MATERIALIZED VIEW** view3 **ORGANIZATION HEAP PCTFREE** 0 **BUILD IMMEDIATE REFRESH COMPLETE ON DEMAND ENABLE QUERY REWRITE AS** (SELECT cand AS a, AVG(val) AS b FROM poll_answers GROUP BY cand);

La ejecución de Oracle de la primera query, utiliza los índices bitmap del atributo pobl (*idxbit_pobl*) y otro bitmap del atributo *edat* (*idxbit_edat*). Después de leer estos índices y realizar la conversión a rowids, hace un Hash Join con estos y una vista para más adelante hacer el group by y la proyección (select).

```
SELECT pobl, MIN(edat), MAX(edat), COUNT(*) FROM poll_answers GROUP BY pobl;
SELECT pobl, edat, cand, MAX(val), MIN(val), AVG(val) FROM poll_answers GROUP BY pobl, edat, cand;
SELECT cand, AVG(val) FROM poll_answers GROUP BY cand;


-- solucion con bitmaps
ALTER TABLE poll_answers MINIMIZE RECORDS_PER_BLOCK;
CREATE BITMAP INDEX idxbit_pobl ON POLL_ANSWERS (pobl) PCTFREE 0;
CREATE BITMAP INDEX idxbit_edat ON POLL_ANSWERS (edat) PCTFREE 0;
CREATE BITMAP INDEX idxbit_cand ON POLL_ANSWERS (cand) PCTFREE 0;
CREATE BITMAP INDEX idxbit_val ON POLL_ANSWERS (val) PCTFREE 0;
```

ults | Results - 2 | Results - 3 | Results - 4 | Statistics | Execution plan - 1

peración	Objeto	Optimizador	Coste	Cardinalidad	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)	IDXBIT_EDAT		0	0	0
BITMAP CONVERSION (TO ROWIDS)			6	20.000	140.000
BITMAP INDEX (FULL SCAN)	IDXBIT_POBL		0	0	0

La ejecución de Oracle de la segunda query, lee todos los índices bitmap (*idxbit_pobl, idxbit_edat, idxbit_cand, idxbit_val)* creados y realizar la conversión a rowids para después unirlos con Hash Join. También crea una vista para más adelante hacer el group by y la proyección (select).

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


-- solucion con bitmaps
ALTER TABLE poll_answers MINIMIZE RECORDS_PER_BLOCK;
CREATE BITMAP INDEX idxbit_pobl ON POLL_ANSWERS (pobl) PCTFREE 0;
CREATE BITMAP INDEX idxbit_edat ON POLL_ANSWERS (edat) PCTFREE 0;
CREATE BITMAP INDEX idxbit_cand ON POLL_ANSWERS (cand) PCTFREE 0;
CREATE BITMAP INDEX idxbit_val ON POLL_ANSWERS (val) PCTFREE 0;
```

Results | Results - 2 | Results - 3 | Results - 4 | Statistics | Execution plan - 1 | Execution plan - 2

Operación	Objeto	Optimizador	Coste	Cardinalidad	Bytes
SELECT STATEMENT		ALL_ROWS	33	20.000	260.000
HASH (GROUP BY)			33	20.000	260.000
VIEW	index$_join$_001		32	20.000	260.000
HASH JOIN			21	0	0
HASH JOIN			16	0	0
HASH JOIN			10	0	0
BITMAP CONVERSION (TO ROWIDS)			5	20.000	260.000
BITMAP INDEX (FULL SCAN)	IDXBIT_CAND		0	0	0
BITMAP CONVERSION (TO ROWIDS)			5	20.000	260.000
BITMAP INDEX (FULL SCAN)	IDXBIT_EDAT		0	0	0
BITMAP CONVERSION (TO ROWIDS)			6	20.000	260.000
BITMAP INDEX (FULL SCAN)	IDXBIT_POBL		0	0	0
BITMAP CONVERSION (TO ROWIDS)			5	20.000	260.000
BITMAP INDEX (FULL SCAN)	IDXBIT_VAL		0	0	0

La ejecución de Oracle de la tercera query, realiza una lectura de toda la vista materializada creada (view3), de donde puede hacer la proyección (select).

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


-- solucion con bitmaps
ALTER TABLE poll_answers MINIMIZE RECORDS_PER_BLOCK;
CREATE BITMAP INDEX idxbit_pobl ON POLL_ANSWERS (pobl) PCTFREE 0;
CREATE BITMAP INDEX idxbit_edat ON POLL_ANSWERS (edat) PCTFREE 0;
CREATE BITMAP INDEX idxbit_cand ON POLL_ANSWERS (cand) PCTFREE 0;
CREATE BITMAP INDEX idxbit_val ON POLL_ANSWERS (val) PCTFREE 0;
```

Results | Results - 2 | Results - 3 | Results - 4 | Statistics | Execution plan - 1 | Execution plan - 2

Operación	Objeto	Optimizador	Coste	Cardinalidad	Bytes
SELECT STATEMENT		ALL_ROWS	5	10	250
MAT_VIEW REWRITE ACCESS (FULL)	VIEW3	ANALYZED	5	10	250

La ejecución de las queries con una base de datos column-oriented se diferenciará porque no será necesario reconstruir las tuplas mediante el uso de joins. Además se beneficiará de la compresión de las tuplas y del uso de las técnicas específicas de procesamiento de queries mediante la vectorización.

Ejercicio 2 del 3rd Lab SQL quiz:

Para resolver el ejercicio 2, hemos utilizado 3 bitmaps, uno para el atributo *pobl*, otro para el atributo *edat* y otro para los atributos *cand* y *val* sobre la tabla *poll_answers*, fragmentada verticalmente, con una partición sobre *pobl, edat, cand* y *val* y otra partición sobre resposta_1, resposta_2, resposta_3, resposta_4 y resposta_5:

```
CREATE OR REPLACE TYPE pobl_edat_cand_val_t AS OBJECT (
  pobl INTEGER,
  edat INTEGER,
  cand INTEGER,
  val INTEGER);

CREATE OR REPLACE TYPE pobl_edat_cand_val_tab IS TABLE OF pobl_edat_cand_val_t;

CREATE OR REPLACE TYPE respostes_t AS OBJECT (
  resposta_1 VARCHAR2(10),
  resposta_2 VARCHAR2(10),
  resposta_3 VARCHAR2(10),
  resposta_4 VARCHAR2(10),
  resposta_5 VARCHAR2(10));

CREATE OR REPLACE TYPE respostes_tab IS TABLE OF respostes_t;

CREATE TABLE poll_answers (
  ref  INTEGER,
  pobl_edat_cand_val pobl_edat_cand_val_tab,
  respostes respostes_tab
)
NESTED TABLE pobl_edat_cand_val STORE AS pobl_edat_cand_val_store
NESTED TABLE respostes STORE AS respostes_store
PCTFREE 0 ENABLE ROW MOVEMENT;
```

- **CREATE BITMAP INDEX idxbit_pobl ON pobl_edat_cand_val_store (pobl) PCTFREE 0;**
- **CREATE BITMAP INDEX idxbit_edat ON pobl_edat_cand_val_store (edat) PCTFREE 0;**
- **CREATE BITMAP INDEX idxbit_cand_val ON pobl_edat_cand_val_store (cand,val) PCTFREE 0;**

La ejecución de Oracle de la primera query, utiliza los índices bitmap sobre los atributos *pobl* y edat para realizar una lectura de estos y después de transformarlos a rowids, realizar el hash join y así poder realizar el group by y la selección.

```
CREATE VIEW view1 AS
SELECT pbe.pobl AS a, MIN(pbe.edat) AS b, MAX(pbe.edat) AS c, COUNT(*) AS d
FROM poll_answers p, TABLE(p.pobl_edat_cand_val) pbe
GROUP BY pbe.pobl;

SELECT * FROM view1;
```

Results | Results - 2 | Results - 3 | Results - 4 | Results - 5 | Statistics | Execution plan - 1

Operación	Objeto	Optimizador	Coste	Cardinalidad	Bytes
SELECT STATEMENT		ALL_ROWS	92	200	4.800
HASH (GROUP BY)			92	200	4.800
VIEW	index$_join$_003		91	20.000	480.000
HASH JOIN			102	0	0
HASH JOIN			12	0	0
BITMAP CONVERSION (TO ROWIDS)			6	20.000	480.000
BITMAP INDEX (FULL SCAN)	IDXBIT_EDAT		0	0	0
BITMAP CONVERSION (TO ROWIDS)			6	20.000	480.000
BITMAP INDEX (FULL SCAN)	IDXBIT_POBL		0	0	0
INDEX (FAST FULL SCAN)	SYS_FK0001379...	ANALYZED	90	20.000	480.000

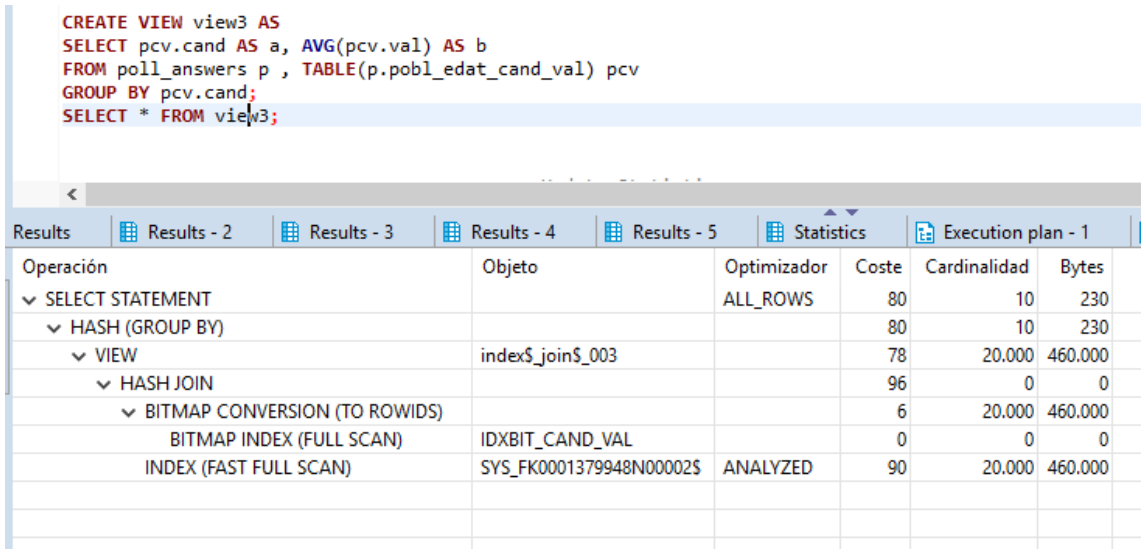
La ejecución de Oracle de la segunda query, realiza una lectura de los tres índices bitmap sobre los atributos *pobl, edat, cand y val.* Después de hacer la conversión a rowids, realiza los respectivos hash join y así puede agrupar y seleccionar las tuplas que busca.

```
CREATE VIEW view2 AS
SELECT pbe.pobl AS a, pbe.edat AS b, pbe.cand AS c,  MAX(pbe.val) AS d, MIN(pbe.val) AS e,
    AVG(pbe.val) AS f
FROM poll_answers p ,TABLE(p.pobl_edat_cand_val) pbe
GROUP BY pbe.pobl, pbe.edat, pbe.cand;
SELECT * FROM view2;
```

Results | Results - 2 | Results - 3 | Results - 4 | Results - 5 | Statistics | Execution plan - 1 | Executio

Operación	Objeto	Optimizador	Coste	Cardinalidad	Bytes
SELECT STATEMENT		ALL_ROWS	265	20.000	580.000
HASH (GROUP BY)			265	20.000	580.000
VIEW	index$_join$_003		103	20.000	580.000
HASH JOIN			108	0	0
HASH JOIN			18	0	0
HASH JOIN			12	0	0
BITMAP CONVERSION (TO ROWIDS)			6	20.000	580.000
BITMAP INDEX (FULL SCAN)	IDXBIT_CAND_VAL		0	0	0
BITMAP CONVERSION (TO ROWIDS)			6	20.000	580.000
BITMAP INDEX (FULL SCAN)	IDXBIT_EDAT		0	0	0
BITMAP CONVERSION (TO ROWIDS)			6	20.000	580.000
BITMAP INDEX (FULL SCAN)	IDXBIT_POBL		0	0	0
INDEX (FAST FULL SCAN)	SYS_FK0001379948N00002$	ANALYZED	90	20.000	580.000

La ejecución de Oracle de la tercera query, lee el índice bitmap sobre los atributos *cand* y *val*. Utiliza la operación bitmap conversión para poder tener identificadores y más adelante hacer la agrupación y la selección de las tuplas.



```
CREATE VIEW view3 AS
SELECT pcv.cand AS a, AVG(pcv.val) AS b
FROM poll_answers p , TABLE(p.pobl_edat_cand_val) pcv
GROUP BY pcv.cand;
SELECT * FROM view3;
```

Results | Results - 2 | Results - 3 | Results - 4 | Results - 5 | Statistics | Execution plan - 1

Operación	Objeto	Optimizador	Coste	Cardinalidad	Bytes
SELECT STATEMENT		ALL_ROWS	80	10	230
HASH (GROUP BY)			80	10	230
VIEW	index$_join$_003		78	20.000	460.000
HASH JOIN			96	0	0
BITMAP CONVERSION (TO ROWIDS)			6	20.000	460.000
BITMAP INDEX (FULL SCAN)	IDXBIT_CAND_VAL		0	0	0
INDEX (FAST FULL SCAN)	SYS_FK0001379948N00002$	ANALYZED	90	20.000	460.000

Al igual que en la pregunta anterior, la ejecución de las queries con una base de datos column-oriented se diferenciará porque no será necesario reconstruir las tuplas mediante el uso de joins. Además de beneficiarse de la compresión de las tuplas y del uso de las técnicas específicas de procesamiento de queries mediante la vectorización.

Además, la fragmentación vertical que se utilizaría en una base de datos column-oriented también sería un aspecto a diferenciar, ya que las particiones que se realizaría serían completamente diferentes, tal y como se ve en el resultado de la affinity matrix de la siguiente pregunta.

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.**

Q1: (20%) SELECT pobl, MIN(edat), MAX(edat), COUNT(*) FROM poll_answers GROUP BY pobl;

Q2: (30%) SELECT pobl, edat, cand, MAX(val), MIN(val), AVG(val) FROM poll_answers GROUP BY pobl, edat, cand;

Q3: (50%) SELECT cand, AVG(val) FROM poll_answers GROUP BY cand;

Para realizar las respectivas matrices, no hemos tenido en cuenta los atributos *ref, resposta_1, resposta_2, resposta_3, resposta_4 y resposta_5* ya que no se usa en las queries. Cada uno de ellos debería de ser una partición, pero para nuestra solución en Oracle, hemos visto más conveniente dejarlos agrupados en la partición *respostes* dentro de la tabla *poll_answers*.

Attribute Usage Matrix:

	pobl	edat	cand	val
Q1	1	1	0	0
Q2	1	1	1	1
Q3	0	0	1	1

Attribute Affinity Matrix:

	pobl	edat	cand	val
pobl	50	50	30	30
edat	50	50	30	30
cand	30	30	80	80
val	30	30	80	80

Se generan las siguientes particiones según el resultado obtenido:

Part1: cand, val

Part2: pobl, edat

Ratio de lectura efectivo:

Q1: SELECT pobl, MIN(edat), MAX(edat), COUNT(*) FROM poll_answers GROUP BY pobl;

#Atr(Q1,P1) / #Atr(P1): 0 / 2 = 0

#Atr(Q1,P2) / #Atr(P2): 2 / 2 = 1

Q2: SELECT pobl, edat, cand, MAX(val), MIN(val), AVG(val) FROM poll_answers GROUP BY pobl, edat, cand;

#Atr(Q2,P1) / #Atr(P1): 2 / 2 = 1

#Atr(Q2,P2) / #Atr(P2): 2 / 2 = 1

Q3: SELECT cand, AVG(val) FROM poll_answers GROUP BY cand;

#Atr(Q3,P1) / #Atr(P1): 2 / 2 = 1

#Atr(Q3,P2) / #Atr(P2): 0 / 2 = 0

Con la partición P1 satisfacemos las queries Q2 y Q3 con 100% de efectividad y con P2 y P1 satisfacemos con 100% efectividad la query Q2. Con la partición P2 satisfacemos las queries Q1 y Q2 con 100% de efectividad.

En Oracle, en contra de una column-oriented, juntar las particiones se realiza mediante joins utilizando índices internos con sus nested tables (particiones).

En Oracle no obtenemos el mejor rendimiento si utilizamos la fragmentación vertical sugerida por la matriz de afinidad. Se obtiene un mejor rendimiento generando una partición agrupada por todos los atributos y definiendo índices sobre éstos.

Al tener una tabla principal donde anida las nested, al final realiza otro join. Con la consiguiente pérdida de eficiencia. Oracle no se beneficia de no utilizar joins, debido a que no tiene un tamaño fijo de datos ni ordenación en las particiones.