

## Diseño Físico y Análisis

### 1. Índices:

#### a) Nuestros Índices:

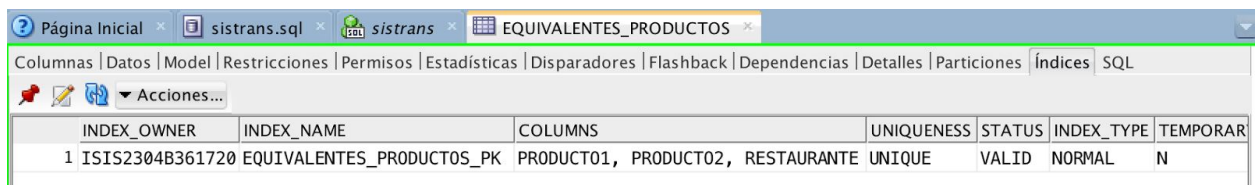
- **Plato\_PorPrecio:** Este índice se crea como un árbol B+ en la tabla PLATO sobre el atributo PRECIOVENTA, con el fin de facilitar las búsquedas por rangos cuando se quiere mirar los platos por precio. Además, sería un índice disperso y secundario, debido a que estas características tienen un menor costo de manutención y no sería la mejor manera para organizar los registros en memoria secundaria, ya que no es la búsqueda más usual. Específicamente, este índice es usado en: RFC1.
- **Plato\_PorCategoria:** Este índice se crea como una tabla de Hash en la tabla PLATO sobre el atributo CATEGORIA, debido a que específicamente se tienen 5 categorías, luego un árbol B+ sería ineficiente, dada la baja selectividad del atributo en la mayoría de los casos. Claramente, este índice es secundario y disperso. Específicamente, este índice es usado en: RFC1, RFC5.
- **Plato\_PorRestaurante:** Este índice se crea como un árbol B+ en la tabla PLATO sobre el atributo RESTAURANTE, aprovechando que este atributo es NN y lo comparten una gran cantidad de datos en la tabla, facilitando drásticamente la búsqueda por rangos de restaurantes en platos y los join entre la tabla Plato y Restaurante. Específicamente, este índice es usado en: RFC1, RFC2, RFC5, RF13, RFC8, RFC9, RFC10.
- **Restaurante\_PorZona:** Este índice se crea como un árbol B+ en la tabla RESTAURANTE sobre el atributo ZONA, aprovechando que este atributo es NN y lo comparten una gran cantidad de datos en la tabla, facilitando drásticamente la búsqueda por rangos de zonas en restaurantes y los join entre la tabla Restaurante y Zona. Específicamente, este índice es usado en: RFC2, RFC5.
- **Pedido\_PorCliente:** Este índice se crea como un árbol B+ en la tabla PEDIDO sobre el atributo EMAIL\_USER, aprovechando que este atributo es NN y lo comparten una gran cantidad de datos en la tabla, facilitando drásticamente la búsqueda por rangos de cliente en pedido y los join entre la tabla Pedido y Cliente. Específicamente, este índice es usado en: RFC3, RFC7.
- **MenuPlato\_PorPlato:** Este índice se crea como un árbol B+ en la tabla MENUPLATO sobre el atributo ID\_PLATO, aprovechando que este atributo es NN y lo comparten una gran cantidad de datos en la tabla, facilitando drásticamente la búsqueda por rangos de plato en MenuPlato y los join entre la tabla Plato y MenuPlato. Específicamente, este índice es usado en: RFC4, RFC5.
- **MenuPlato\_PorMenu:** Este índice se crea como un árbol B+ en la tabla MENUPLATO sobre el atributo ID\_MENU, aprovechando que este atributo es NN y lo comparten una gran cantidad de datos en la tabla, facilitando drásticamente la búsqueda por rangos de menú en MenuPlato y los join entre la tabla Menú y MenuPlato. Específicamente, este índice es usado en: RFC5, RFC8, RFC9, RFC10.
- **Menu\_PorRestaurante:** Este índice se crea como un árbol B+ en la tabla MENU sobre el atributo NOM\_RESTAURANTE, aprovechando que este atributo es NN y lo comparten una gran cantidad de datos en la tabla, facilitando drásticamente la búsqueda por rangos de restaurantes en menús y los join entre la tabla Menú y

Restaurante. Específicamente, este índice es usado en: RFC5, RF13, RFC8, RFC9, RFC10.

- **Pedio\_PorFecha:** Este índice se crea como un árbol B+ en la tabla PEDIDO sobre el atributo FECHA, con el fin de facilitar las búsquedas por rangos cuando se quiere mirar los pedidos por fecha. Además, sería un índice disperso y secundario, debido a que estas características tienen un menor costo de manutención y no sería la mejor manera para organizar los registros en memoria secundaria, ya que no es la búsqueda más usual. Específicamente, este índice es usado en: RFC6, RFC9, RFC10.
- **PedidoPlato\_PorPlato:** Este índice se crea como un árbol B+ en la tabla PEDIDOPLATO sobre el atributo ID\_PLATO, aprovechando que este atributo es NN y lo comparten una gran cantidad de datos en la tabla, facilitando drásticamente la búsqueda por rangos de plato en PedidoPlato y los join entre la tabla Plato y PedidoPlato. Específicamente, este índice es usado en: RF17, RFC7, RFC8, RFC9, RFC10.
- **PedidoMenu\_PorMenu:** Este índice se crea como un árbol B+ en la tabla PEDIDOMENU sobre el atributo ID\_MENU, aprovechando que este atributo es NN y lo comparten una gran cantidad de datos en la tabla, facilitando drásticamente la búsqueda por rangos de menú en PedidoMenu y los join entre la tabla Menú y PedidoMenu. Específicamente, este índice es usado en: RF17, RFC7, RFC8, RFC9, RFC10.

b) Índices creados por Oracle:

- **EQUIVALENTES\_PRODUCTOS:** Este índice fue creado para esta tabla en estas columnas debido a que conforman la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.



INDEX_OWNER	INDEX_NAME	COLUMNS	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORAR
1 ISIS2304B361720	EQUIVALENTES_PRODUCTOS_PK	PRODUCT01, PRODUCT02, RESTAURANTE	UNIQUE	VALID	NORMAL	N

- **CATEGORIA:** Este índice fue creado para esta tabla en esta columna debido a que conforma la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo

más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.

INDEX_OWNER	INDEX_NAME	COLUMNS	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORAR
1 ISIS2304B361720	CATEGORIA_PK	NOMCATEGORIA	UNIQUE	VALID	NORMAL	N

- **CLIENTEFR:** Este índice fue creado para esta tabla en esta columna debido a que conforma la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.

INDEX_OWNER	INDEX_NAME	COLUMNS	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORAR
1 ISIS2304B361720	CLIENTEFR_PK	EMAIL	UNIQUE	VALID	NORMAL	N

- **EQUIVALENTES:** Este índice fue creado para esta tabla en estas columnas debido a que conforman la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.

INDEX_OWNER	INDEX_NAME	COLUMNS	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORAR
1 ISIS2304B361720	SYS_C00216769	INGREDIENTE1, INGREDIENTE2	UNIQUE	VALID	NORMAL	N

- **INGREDIENTES:** Este índice fue creado para esta tabla en esta columna debido a que conforma la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice

claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.

The screenshot shows the Oracle SQL Developer interface with the 'Índices' tab selected for the 'INGREDIENTES' table. The table below represents the data shown in the interface.

	INDEX_OWNER	INDEX_NAME	COLUMNS	UNIQUEN...	STA...	INDEX_T...	TEMPOR.
1	ISIS2304B361720	INGREDIENTES_PK	NOMBRE	UNIQUE	VALID	NORMAL	N

- **INGREDIENTES\_PLATO:** Este índice fue creado para esta tabla en estas columnas debido a que conforman la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.

The screenshot shows the Oracle SQL Developer interface with the 'Índices' tab selected for the 'INGREDIENTES\_PLATO' table. The table below represents the data shown in the interface.

	INDEX_OWNER	INDEX_NAME	COLUMNS	UNIQUEN...	STA...	INDEX_T...	TEMPOR.
1	ISIS2304B361720	INGREDIENTES_PLATO_PK	ID_PLATO, NOM_INGR	UNIQUE	VALID	NORMAL	N

- **MENU:** Este índice fue creado para esta tabla en esta columna debido a que conforma la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.

The screenshot shows the Oracle SQL Developer interface with the 'Índices' tab selected for the 'MENU' table. The table below represents the data shown in the interface.

	INDEX_OWNER	INDEX_NAME	COLUMNS	UNIQUEN...	STA...	INDEX_T...	TEMPOR.
1	ISIS2304B361720	MENU_PK	ID_MENU	UNIQUE	VALID	NORMAL	N

- **MENU\_PLATO:** Este índice fue creado para esta tabla en estas columnas debido a que conforman la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.

Página Inicial

sistrans.sql

sistrans

MENU\_PLATO

Columnas

Datos

Model

Restricciones

Permisos

Estadísticas

Disparadores

Flashback

Dependencias

Detalles

Particiones

Índices

SQL

Acciones...

INDEX_OWNER	INDEX_NAME	COLUMNS	UNIQUEN...	STA...	INDEX_T...	TEMPOR.	
1	ISIS2304B361720	MENU_PLATO_PK	ID_MENU, ID_PLATO	UNIQUE	VALID	NORMAL	N

- ORGANIZADOR: Este índice fue creado para esta tabla en esta columna debido a que conforma la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.

</

- PEDIDO: Este índice fue creado para esta tabla en esta columna debido a que conforma la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.

Página Inicial

sistrans.sql

sistrans

PEDIDO

Columnas

Datos

Model

Restricciones

Permisos

Estadísticas

Disparadores

Flashback

Dependencias

Detalles

Particiones

Índices

SQL

Acciones...

INDEX_OWNER	INDEX_NAME	COLUMNS	UNIQUEN...	STA...	INDEX_T...	TEMPOR.
1	ISIS2304B361720 SYS_C00216661	NUM_PEDIDO	UNIQUE	VALID	NORMAL	N

- PEDIDO\_MENUS: Este índice fue creado para esta tabla en estas columnas debido a que conforman la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.

Página Inicial x sistrans.sql x sistrans x PEDIDO_MENUS x							
Columnas   Datos   Model   Restricciones   Permisos   Estadísticas   Disparadores   Flashback   Dependencias   Detalles   Particiones   Índices   SQL							
Acciones...							
INDEX_OWNER	INDEX_NAME	COLUMNS	UNIQUEN...	STA...	INDEX_T...	TEMPOR.	
1	ISIS2304B361720 SYS_C00216674	ID_MENU, NUM_PEDIDO	UNIQUE	VALID	NORMAL	N	

- **PEDIDO\_PLATO:** Este índice fue creado para esta tabla en estas columnas debido a que conforman la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.

Página Inicial x sistrans.sql x sistrans x PEDIDO_PLATO x							
Columnas   Datos   Model   Restricciones   Permisos   Estadísticas   Disparadores   Flashback   Dependencias   Detalles   Particiones   Índices   SQL							
Acciones...							
INDEX_OWNER	INDEX_NAME	COLUMNS	UNIQUEN...	STA...	INDEX_T...	TEMPOR.	
1	ISIS2304B361720 SYS_C00216669	ID_PLATO, NUM_PEDIDO	UNIQUE	VALID	NORMAL	N	

- **PLATO:** Este índice fue creado para esta tabla en esta columna debido a que conforma la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.

Página Inicial x sistrans.sql x sistrans x PLATO x							
Columnas   Datos   Model   Restricciones   Permisos   Estadísticas   Disparadores   Flashback   Dependencias   Detalles   Particiones   Índices   SQL							
Acciones...							
INDEX_OWNER	INDEX_NAME	COLUMNS	UNIQUEN...	STA...	INDEX_T...	TEMPOR.	
1	ISIS2304B361720 PLATO_PK1	ID_PLATO	UNIQUE	VALID	NORMAL	N	
2	ISIS2304B361720 UNIQUE_NAME	NOMBRE, RESTAURANTE	UNIQUE	VALID	NORMAL	N	

- **RESERVA:** Este índice fue creado para esta tabla en estas columnas debido a que conforman la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.



Página Inicial

sistrans.sql

sistrans

RESERVA

Columnas

Datos

Model

Restricciones

Permisos

Estadísticas

Disparadores

Flashback

Dependencias

Detalles

Particiones

Índices

SQL

Acciones...

	INDEX_OWNER	INDEX_NAME	COLUMNS	UNIQUEN...	STA...	INDEX_T...	TEMPOR.
1	ISIS2304B361720	RESERVA_PK	FECHA, ZONA	UNIQUE	VALID	NORMAL	N

- **USUARIOS:** Este índice fue creado para esta tabla en esta columna debido a que conforma la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.

Página Inicial

sistrans.sql

sistrans

USUARIOS

Columnas

Datos

Model

Restricciones

Permisos

Estadísticas

Disparadores

Flashback

Dependencias

Detalles

Particiones

Índices

SQL

Acciones...

	INDEX_OWNER	INDEX_NAME	COLUMNS	UNIQUEN...	STA...	INDEX_T...	TEMPOR.
1	ISIS2304B361720	UNIQUE_ID	IDENTIFICACION	UNIQUE	VALID	NORMAL	N
2	ISIS2304B361720	USUARIOS_PK	EMAIL	UNIQUE	VALID	NORMAL	N

- **RESTAURANTES:** Este índice fue creado para esta tabla en esta columna debido a que conforma la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.

- **ZONA:** Este índice fue creado para esta tabla en esta columna debido a que conforma la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.

Página Inicial

sistrans.sql

sistrans

ZONA

Columnas

Datos

Model

Restricciones

Permisos

Estadísticas

Disparadores

Flashback

Dependencias

Detalles

Particiones

Índices

SQL

Acciones...

	INDEX_OWNER	INDEX_NAME	COLUMNS	UNIQUEN...	STA...	INDEX_T...	TEMPOR.
1	ISIS2304B361720	ZONA_PK	NOMBRE	UNIQUE	VALID	NORMAL	N

- ZONAS\_CLIENTESFR: Este índice fue creado para esta tabla en estas columnas debido a que conforman la PK, y por tanto Oracle la crea en el instante que se define la PK. Esta elección de crear el índice en la Llave primaria por Oracle, se debe a que la PK es lo más usado en consultas y es, teóricamente, el orden natural de los registros en la memoria secundaria, por tal razón este índice es clustered, y denso. Este índice claramente optimiza las consultas, ya que es un índice primario indispensable para los join, que son altamente frecuentes en nuestras sentencias SQL.



## 2. Análisis:

### A. RFC9

- Sentencia SQL:

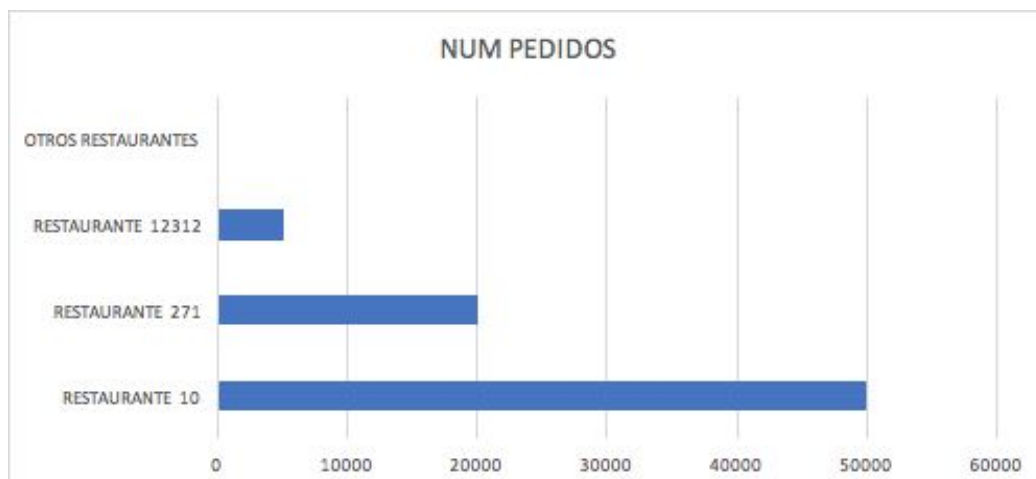
```
(SELECT EMAIL, NOMBRE, IDENTIFICACION FROM USUARIOS JOIN (  
(SELECT EMAIL_USER AS EMAIL_USER1  
FROM PEDIDO JOIN (PEDIDO_PLATO JOIN PLATO ON PEDIDO_PLATO.ID_PLATO = PLATO.ID_PLATO)  
ON PEDIDO.NUM_PEDIDO = PEDIDO_PLATO.NUM_PEDIDO WHERE PLATO.RESTAURANTE = '<PARAM RESTAURANTE>'  
AND PEDIDO.FECHA BETWEEN TO_DATE('<PARAM FECHA INIC>', 'DD/MM/YYYY') AND TO_DATE('<PARAM FECHA FIN>', 'DD/MM/YYYY'))  
UNION  
(SELECT EMAIL_USER AS EMAIL_USER1  
FROM PEDIDO JOIN (PEDIDO_MENUS JOIN MENU ON PEDIDO_MENUS.ID_MENU = MENU.ID_MENU)  
ON PEDIDO.NUM_PEDIDO = PEDIDO_MENUS.NUM_PEDIDO WHERE MENU.NOMRESTAURANTE = 'RESTAURANTE 1'  
AND PEDIDO.FECHA BETWEEN TO_DATE('<PARAM FECHA INIC>', 'DD/MM/YYYY') AND TO_DATE('<PARAM FECHA FIN>', 'DD/MM/YYYY'))  
ON EMAIL = EMAIL_USER1) ORDER BY EMAIL;
```

- Parámetros:

Los parámetros para la sentencia del RFC9 son el nombre de un restaurante de la Rotonda y un rango de tiempo, dado como fecha inicial y final. De esta manera, tenemos la siguiente distribución:

#### → Pedidos por Restaurante:

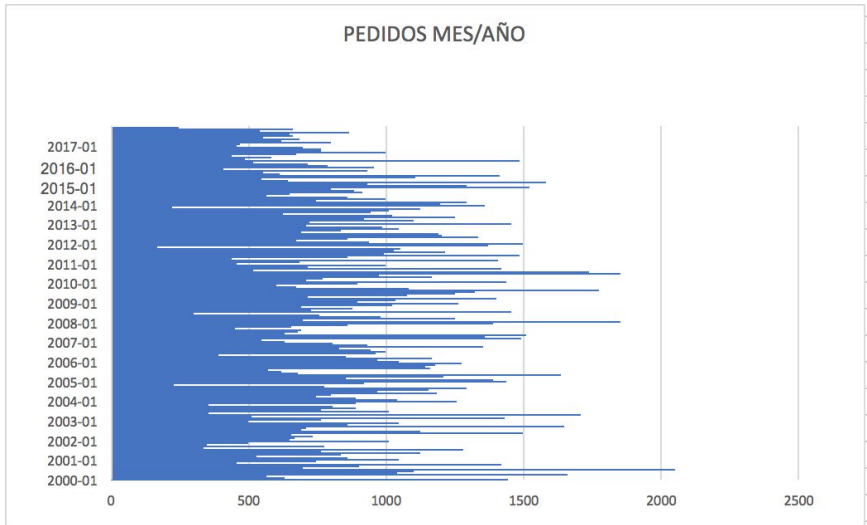
El número de pedidos por restaurante tiene una alta incidencia en el tamaño de la respuesta de la consulta, a mayor cantidad de pedidos, mas grande sera la respuesta para un mismo rango de fechas. Esto debido a que nuestros pedidos se encuentran distribuidos de manera equitativa por fecha, como se podrá ver en el siguiente análisis de parámetro. Debido a esto, restaurantes como “RESTAURANTE 10” tienen una densidad de pedidos dentro de la consulta que es muy alta, logrando así que la sentencia se vea empujada a realizar JOINS como HASH JOIN, debido a que las tablas no se pueden cargar a memoria.

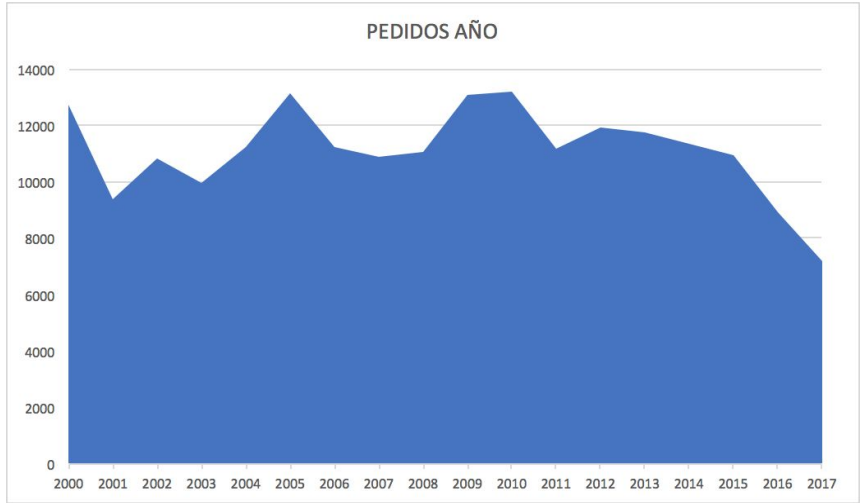


Restaurante	Selectividad
RESTAURANTE 10	67%
RESTAURANTE 271	27%
RESTAURANTE 12312	7%
OTROS RESTAURANTES	1%

→ Rango de Fechas:

La distribución de la base de de datos para las fechas se hizo de manera equitativa, como se podrá evidenciar en las tablas y gráficos anexos. Mediante la distribución equitativa de las fechas, este parámetro aumenta el tamaño de las fechas, y debido a su distribución uniforme es posible calcular el tamaño de del mismo con una baja incertidumbre.





Año	Selectividad
2000	6%
2001	5%
2002	5%
2003	5%
2004	6%
2005	7%
2006	6%
2007	5%
2008	6%
2009	7%
2010	7%
2011	6%
2012	6%
2013	6%
2014	6%

2015	5%
2016	4%
2017	4%

- Plan de ejecución:

Operation	Params	Rows	Total Cost	Raw desc
Select				
Merge join (MERGE JOIN)		1	7.0	cpu_cost = 24383358...
Index scan (TABLE ACCESS BY INDEX ROWID)	table: USUARIOS;	1	7.0	cpu_cost = 24383358...
Full index scan (INDEX FULL SCAN)	index: USUARIOS_PK;	1	0.0	cpu_cost = 200, io_co...
Sort (SORT JOIN)		2	7.0	cpu_cost = 24383338...
Access (VIEW)		2	6.0	cpu_cost = 20412564...
Sort unique (SORT UNIQUE)		2	6.0	cpu_cost = 20412564...
Union all (UNION-ALL)				cpu_cost = null, io_co...
Nested loops (NESTED LOOPS)		1	1.0	cpu_cost = 6587684, i...
Nested loops (NESTED LOOPS)		5013	1.0	cpu_cost = 6587684, i...
Nested loops (NESTED LOOPS)		5013	1.0	cpu_cost = 8121, io_co...
Access (TABLE ACCESS BY INDEX ROWID BATCHED)	table: PLATO;	4	1.0	cpu_cost = 7321, io_c...
Index scan (INDEX SKIP SCAN)	index: UNIQUE_NAME;	1	1.0	cpu_cost = 7321, io_c...
Index scan (INDEX RANGE SCAN)	index: SYS_C00216669;	1645938342	0.0	cpu_cost = 200, io_co...
Unique index scan (INDEX UNIQUE SCAN)	index: SYS_C00216661;	1	0.0	cpu_cost = 1050, io_c...
Index scan (TABLE ACCESS BY INDEX ROWID)	table: PEDIDO;	1	0.0	cpu_cost = 1313, io_co...
Nested loops (NESTED LOOPS SEMI)		1	3.0	cpu_cost = 118122683...
Nested loops (NESTED LOOPS)		1	3.0	cpu_cost = 118121313...
Full index scan (INDEX FULL SCAN)	index: SYS_C00216674;	89997	0.0	cpu_cost = 250, io_co...
Index scan (TABLE ACCESS BY INDEX ROWID)	table: PEDIDO;	1	0.0	cpu_cost = 1313, io_co...
Unique index scan (INDEX UNIQUE SCAN)	index: SYS_C00216661;	1	0.0	cpu_cost = 1050, io_c...
Index scan (TABLE ACCESS BY INDEX ROWID)	table: MENU;	1	0.0	cpu_cost = 1370, io_c...
Unique index scan (INDEX UNIQUE SCAN)	index: MENU_PK;	1	0.0	cpu_cost = 1050, io_c...

- Ejecución sentencia:

```

sql> (SELECT EMAIL, NOMBRE, IDENTIFICACION FROM USUARIOS JOIN (
(SELECT EMAIL_USER AS EMAIL_USER1
FROM PEDIDO JOIN (PEDIDO_PLATO JOIN PLATO ON PEDIDO_PLATO.ID_PLATO = PLATO.ID_PLATO)
ON PEDIDO.PLATO_NUM_PEDIDO = PEDIDO_PLATO.NUM_PEDIDO WHERE PLATO.RESTaurANTE = 'RESTAURANTE 12312'
AND PEDIDO.FECHA BETWEEN TO_DATE('10-10-2010', 'DD/MM/YYYY') AND TO_DATE('10-10-2011', 'DD/MM/YYYY'))
UNION
(SELECT EMAIL_USER AS EMAIL_USER1
FROM PEDIDO JOIN (PEDIDO_MENU JOIN MENU ON PEDIDO_MENU.ID_MENU = MENU.ID_MENU)
ON PEDIDO.NUM_PEDIDO = PEDIDO_MENU.NUM_PEDIDO WHERE MENU.NOMRESTAURANTE = 'RESTAURANTE 12312'
AND PEDIDO.FECHA BETWEEN TO_DATE('10-10-2010', 'DD/MM/YYYY') AND TO_DATE('10-10-2011', 'DD/MM/YYYY'))
ON EMAIL = EMAIL_USER1) ORDER BY EMAIL
[2017-11-21 19:59:28] 210 rows retrieved starting from 1 in 1s 57ms (execution: 994ms, fetching: 63ms)

```

EMAIL	NOMBRE	IDENTIFICACION
EMAIL 6528	Juan	6528
EMAIL 6529	Juan	6529
EMAIL 6530	Juan	6530
EMAIL 6531	Juan	6531
EMAIL 6532	Juan	6532
EMAIL 6533	Juan	6533
EMAIL 6534	Juan	6534
EMAIL 6535	Juan	6535
EMAIL 6536	Juan	6536

- Selectividad y plan de ejecución nuestro:

Para la sentencia del requerimiento RFC9, se tiene en cuenta la manera en la cual están distribuidas las tablas. Es por eso que se ejecutará primero el JOIN entre la tabla PEDIDO\_PLATO y PLATO, haciendo un JOIN INDEX, debido a que esté JOIN en su cláusula ON detalla que es por la PK de PLATO que está referenciada como FK en PEDIDO\_PLATO, para los cuales se tiene índice, uno generado por Oracle, y el otro por nosotros. Posterior a ese JOIN, se realiza el JOIN con PEDIDO, otra vez por el método PK FK con INDEX JOIN, ya que ambos índices están creados en la base de datos. Este mismo proceso se realiza con las tablas PEDIDO, PEDIDO\_MENU y MENU, seguidos de una cláusula SELECT con un WHERE tal que la información final consiste en los NUM\_PEDIDO tal que algún producto es del restaurante indicado por parámetro, y este pedido tiene fecha dentro del rango indicado por parametro. Después, se le

hace UNION a estas tablas, para consolidarlas y facilitar el siguiente y último paso que consiste en realizar el JOIN de esta tabla con USUARIOS, tal que se le haga un “append” a la información que se pide originalmente en la sentencia, ya que se quiere la información completa de los usuarios que consuman en un restaurante en un rango de fechas específico. Si se tiene una cláusula ORDER BY al final de la sentencia, esté ORDER BY se debe realizar posteriormente, mediante un MERGE sobre los datos en memoria secundaria, como lo vimos en clase.

➔ Selectividad Alta:

Parámetros:

‘RESTAURANTE 12’ (1%)

10-10-2010 -> 10-10-2011 (6.5%)

The screenshot shows the SQL Developer interface with a query executed in the Database Console. The query is a complex JOIN involving USUARIOS, PEDIDO, PEDIDO\_PLATO, PLATO, and PEDIDO\_MENU tables, filtered by 'RESTAURANTE 12' and a date range from 10-10-2010 to 10-10-2011. The result set, titled 'Result 178', displays two rows of data with columns EMAIL, NOMBRE, and IDENTIFICACION.

	EMAIL	NOMBRE	IDENTIFICACION
1	EMAIL 40017	Maria	40017
2	EMAIL 40018	Maria	40018

➔ Selectividad Media:

Parámetros:

‘RESTAURANTE 271’ (27%)

10-10-2010 -> 10-10-2011 (6.5%)

The screenshot shows the SQL Developer interface with a query executed in the Database Console. The query is similar to the previous one but filtered by 'RESTAURANTE 271'. The result set, titled 'Result 180', displays 14 rows of data with columns EMAIL, NOMBRE, and IDENTIFICACION.

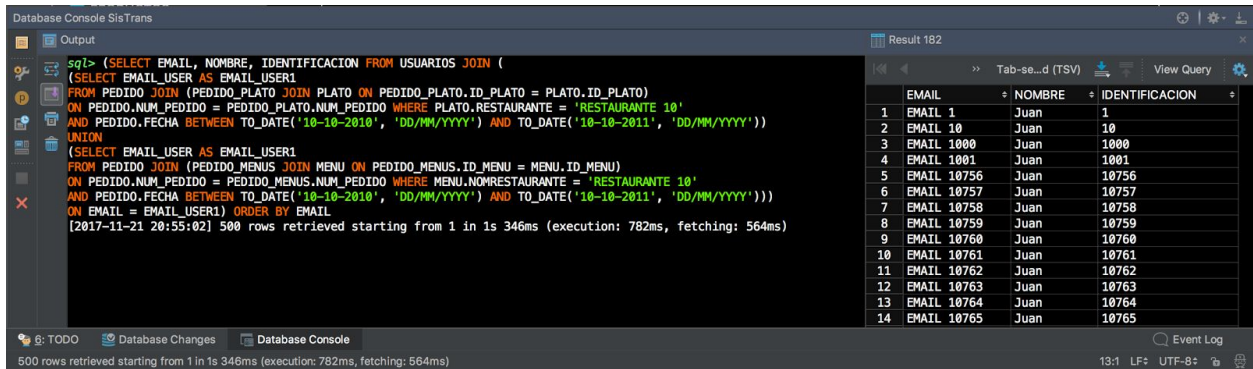
	EMAIL	NOMBRE	IDENTIFICACION
1	EMAIL 1	Juan	1
2	EMAIL 10	Juan	10
3	EMAIL 10756	Juan	10756
4	EMAIL 10757	Juan	10757
5	EMAIL 10758	Juan	10758
6	EMAIL 10759	Juan	10759
7	EMAIL 10760	Juan	10760
8	EMAIL 10761	Juan	10761
9	EMAIL 10762	Juan	10762
10	EMAIL 10763	Juan	10763
11	EMAIL 10764	Juan	10764
12	EMAIL 10765	Juan	10765
13	EMAIL 10766	Juan	10766
14	EMAIL 10767	Juan	10767

➔ Selectividad Baja:

Parámetros:

‘RESTAURANTE 10’ (67%)

10-10-2010 -> 10-10-2011 (6.5%)



	EMAIL	NOMBRE	IDENTIFICACION
1	EMAIL 1	Juan	1
2	EMAIL 10	Juan	10
3	EMAIL 1000	Juan	1000
4	EMAIL 1001	Juan	1001
5	EMAIL 10756	Juan	10756
6	EMAIL 10757	Juan	10757
7	EMAIL 10758	Juan	10758
8	EMAIL 10759	Juan	10759
9	EMAIL 10760	Juan	10760
10	EMAIL 10761	Juan	10761
11	EMAIL 10762	Juan	10762
12	EMAIL 10763	Juan	10763
13	EMAIL 10764	Juan	10764
14	EMAIL 10765	Juan	10765

## B. RFC10

- Sentencia SQL:

```
((SELECT EMAIL, NOMBRE, IDENTIFICACION FROM USUARIOS LEFT OUTER JOIN (
  (SELECT EMAIL_USER AS EMAIL_USER1
   FROM PEDIDO JOIN (PEDIDO_PLATO JOIN PLATO ON PEDIDO_PLATO.ID_PLATO = PLATO.ID_PLATO)
   ON PEDIDO.NUM_PEDIDO = PEDIDO_PLATO.NUM_PEDIDO WHERE PLATO.RESTAURANTE = '<PARAM RESTAURANTE>'
   AND PEDIDO.FECHA BETWEEN TO_DATE('<PARAM FECHA INIC>', 'DD/MM/YYYY') AND TO_DATE('<PARAM FECHA FIN>', 'DD/MM/YYYY'))
  ON EMAIL = EMAIL_USER1 WHERE EMAIL_USER1 IS NULL) UNION (SELECT EMAIL, NOMBRE, IDENTIFICACION FROM USUARIOS JOIN (
  (SELECT EMAIL_USER AS EMAIL_USER1
   FROM PEDIDO JOIN (PEDIDO_MENU JOIN MENU ON PEDIDO_MENU.ID_MENU = MENU.ID_MENU)
   ON PEDIDO.NUM_PEDIDO = PEDIDO_MENU.NUM_PEDIDO WHERE MENU.NOMRESTAURANTE = 'RESTAURANTE 10'
   AND PEDIDO.FECHA BETWEEN TO_DATE('<PARAM FECHA INIC>', 'DD/MM/YYYY') AND TO_DATE('<PARAM FECHA FIN>', 'DD/MM/YYYY'))
  ON EMAIL = EMAIL_USER1 WHERE EMAIL_USER1 IS NULL))
ORDER BY EMAIL;
```

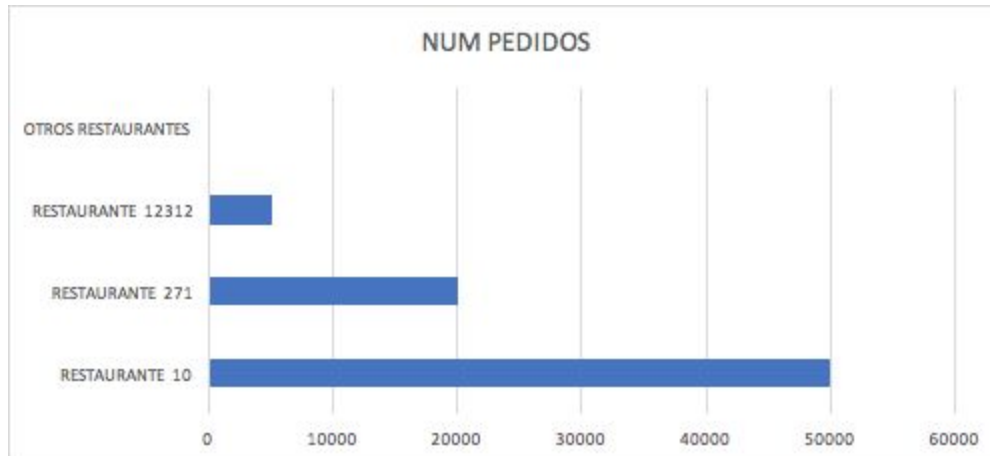
- Parámetros:

Los parámetros para la sentencia del RFC10 son el nombre de un restaurante de la Rotonda y un rango de tiempo , dado como fecha inicial y final. De esta manera, tenemos la siguiente distribución:

→ Pedidos por Restaurante:

El número de pedidos por restaurante tiene una alta incidencia en el tamaño de la respuesta de la consulta, a mayor cantidad de pedidos, más pequeña será la respuesta para un mismo rango de fechas. Esto debido a que nuestros pedidos se encuentran distribuidos de manera equitativa por fecha, como se podrá ver en el siguiente análisis de parámetro. Debido a esto, restaurantes como “RESTAURANTE 9912” tienen una densidad de pedidos baja dentro de la consulta que es muy alta, logrando así que la sentencia se vea empujada a realizar JOINS como HASH JOIN, debido a que las tablas no se pueden cargar a memoria.

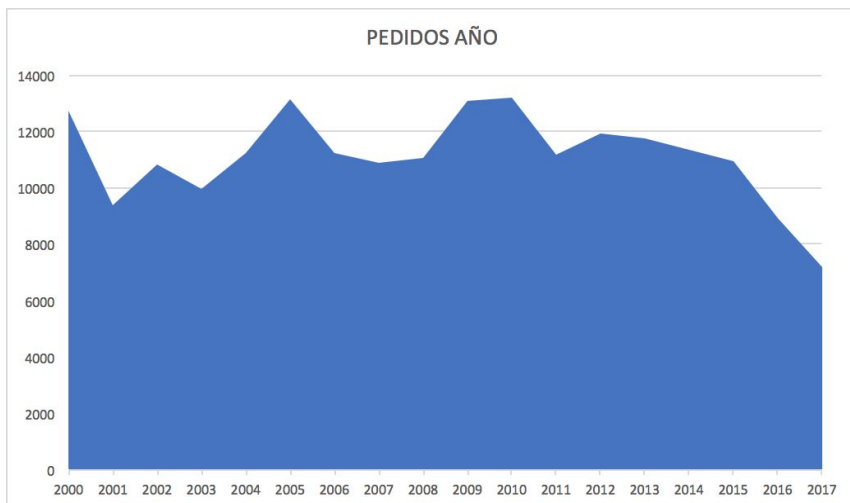
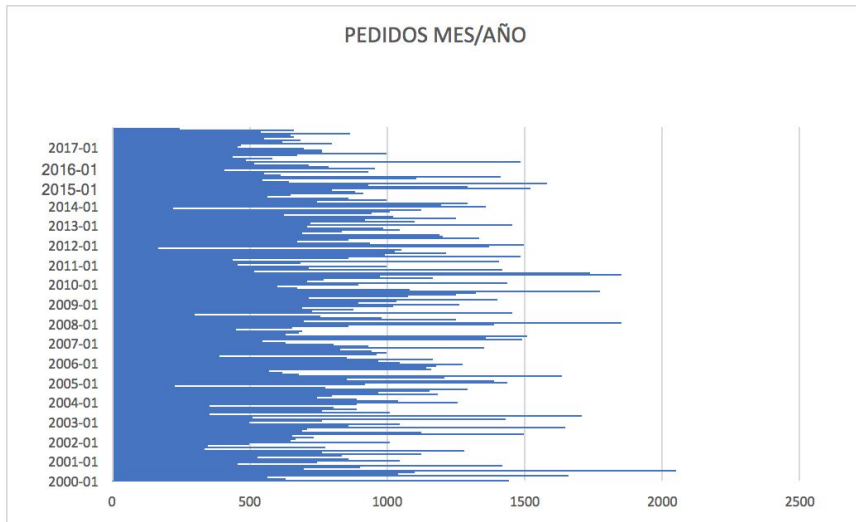




Restaurante	Selectividad
RESTAURANTE 10	67%
RESTAURANTE 271	27%
RESTAURANTE 12312	7%
OTROS RESTAURANTES	1%

→ Rango de Fechas:

La distribución de la base de de datos para las fechas se hizo de manera equitativa, como se podrá evidenciar en las tablas y gráficos anexos. Mediante la distribución equitativa de las fechas, este parámetro aumenta el tamaño de las fechas, y debido a su distribución uniforme es posible calcular el tamaño de del mismo con una baja incertidumbre.



Año	Selectividad
2000	6%
2001	5%
2002	5%
2003	5%
2004	6%
2005	7%
2006	6%

2007	5%
2008	6%
2009	7%
2010	7%
2011	6%
2012	6%
2013	6%
2014	6%
2015	5%
2016	4%
2017	4%

- Plan de ejecución:

Operation	Params	Rows	Total Cost	Raw desc
Select		2	5.0	cpu_cost = 165418539,...
▼ Sort unique (SORT UNIQUE)		2	4.0	cpu_cost = 125710808,...
▼ Union all (UNION-ALL)				cpu_cost = null, io_co...
▼ Merge join (MERGE JOIN ANTI)		1	2.0	cpu_cost = 46295525,...
▼ Index scan (TABLE ACCESS BY INDEX ROWID)	table: USUARIOS;	1	0.0	cpu_cost = 200, io_co...
▼ Full index scan (INDEX FULL SCAN)	index: USUARIOS_PK;	1	0.0	cpu_cost = 200, io_co...
▼ Sort unique (SORT UNIQUE)		1	2.0	cpu_cost = 46295325,...
▼ Access (VIEW)		1	1.0	cpu_cost = 6587684, l...
▼ Nested loops (NESTED LOOPS)		1	1.0	cpu_cost = 6587684, l...
▼ Nested loops (NESTED LOOPS)		5013	1.0	cpu_cost = 6587684, l...
▼ Nested loops (NESTED LOOPS)		5013	1.0	cpu_cost = 8121, io_co...
▼ Access (TABLE ACCESS BY INDEX ROWID BATCHED)	table: PLATO;	4	1.0	cpu_cost = 7321, io_c...
▼ Index scan (INDEX SKIP SCAN)	index: UNIQUE_NAME;	1	1.0	cpu_cost = 7321, io_c...
▼ Index scan (INDEX RANGE SCAN)	index: SYS_C00216669;	1645938342	0.0	cpu_cost = 200, io_co...
▼ Unique index scan (INDEX UNIQUE SCAN)	index: SYS_C00216661;	1	0.0	cpu_cost = 1050, io_c...
▼ Index scan (TABLE ACCESS BY INDEX ROWID)	table: PEDIDO;	1	0.0	cpu_cost = 1313, io_co...
▼ Filter (FILTER)				cpu_cost = null, io_co...
▼ Nested loops (NESTED LOOPS SEMI)		1	3.0	cpu_cost = 118123943,...
▼ Nested loops (NESTED LOOPS)		1	3.0	cpu_cost = 118122573,...
▼ Nested loops (NESTED LOOPS)		1	3.0	cpu_cost = 118121313,...
▼ Full index scan (INDEX FULL SCAN)	index: SYS_C00216674;	89997	0.0	cpu_cost = 250, io_co...
▼ Index scan (TABLE ACCESS BY INDEX ROWID)	table: PEDIDO;	1	0.0	cpu_cost = 1313, io_co...
▼ Unique index scan (INDEX UNIQUE SCAN)	index: SYS_C00216661;	1	0.0	cpu_cost = 1050, io_c...
▼ Index scan (TABLE ACCESS BY INDEX ROWID)	table: USUARIOS;	1	0.0	cpu_cost = 1260, io_c...
▼ Unique index scan (INDEX UNIQUE SCAN)	index: USUARIOS_PK;	1	0.0	cpu_cost = 1050, io_c...
▼ Index scan (TABLE ACCESS BY INDEX ROWID)	table: MENU;	1	0.0	cpu_cost = 1370, io_c...
▼ Unique index scan (INDEX UNIQUE SCAN)	index: MENU_PK;	1	0.0	cpu_cost = 1050, io_c...

- Ejecución de la sentencia:

Database Console SisTrans

Output

```

sql> ((SELECT EMAIL, NOMBRE, IDENTIFICACION FROM USUARIOS LEFT OUTER JOIN (
  (SELECT EMAIL_USER AS EMAIL_USER1
  FROM PEDIDO JOIN (PEDIDO_PLATO JOIN PLATO ON PEDIDO_PLATO.ID_PLATO = PLATO.ID_PLATO)
  ON PEDIDO.NUM_PEDIDO = PEDIDO_PLATO.NUM_PEDIDO WHERE PLATO.RESTAURANTE = 'RESTAURANTE 12312'
  AND PEDIDO.FECHA BETWEEN TO_DATE('10-10-2010', 'DD/MM/YYYY') AND TO_DATE('10-10-2011', 'DD/MM/YYYY'))
  ON EMAIL = EMAIL_USER1 WHERE EMAIL_USER1 IS NULL) UNION (SELECT EMAIL, NOMBRE, IDENTIFICACION FROM USUARIOS
  (SELECT EMAIL_USER AS EMAIL_USER1
  FROM PEDIDO JOIN (PEDIDO_MENU JOIN MENU ON PEDIDO_MENU.ID_MENU = MENU.ID_MENU)
  ON PEDIDO.NUM_PEDIDO = PEDIDO_MENU.NUM_PEDIDO WHERE MENU.NOMRESTAURANTE = 'RESTAURANTE 12312'
  AND PEDIDO.FECHA BETWEEN TO_DATE('10-10-2010', 'DD/MM/YYYY') AND TO_DATE('10-10-2011', 'DD/MM/YYYY'))
  ON EMAIL = EMAIL_USER1 WHERE EMAIL_USER1 IS NULL))
ORDER BY EMAIL
[2017-11-21 20:02:07] 500 rows retrieved starting from 1 in 1s 332ms (execution: 1s 177ms, fetching: 155ms)

```

Result 167

	EMAIL	NOMBRE	IDENTIFICACION
1	EMAIL 1	Juan	1
2	EMAIL 10	Juan	10
3	EMAIL 100	Juan	100
4	EMAIL 1000	Juan	1000
5	EMAIL 10000	Juan	10000
6	EMAIL 100000	Marcos	100000
7	EMAIL 10001	Juan	10001
8	EMAIL 10002	Juan	10002
9	EMAIL 10003	Juan	10003

36 cells 1:1 8:81 LF: UTF-8

- Selectividad y plan de ejecución nuestro:

Para la sentencia del requerimiento RFC10, se tiene en cuenta la manera en la cual están distribuidas las tablas. Es por eso que se ejecutará primero el JOIN entre la tabla PEDIDO\_PLATO y PLATO, haciendo un JOIN INDEX, debido a que esté JOIN en su cláusula ON detalla que es por la PK de PLATO que está referenciada como FK en PEDIDO\_PLATO, para los cuales se tiene índice, uno generado por Oracle, y el otro por nosotros. Posterior a ese JOIN, se realiza el JOIN con PEDIDO, otra vez por el método PK FK con INDEX JOIN, ya que ambos índices están creados en la base de datos. Este mismo proceso se realiza con las tablas PEDIDO, PEDIDO\_MENU y MENU, seguidos de una cláusula SELECT con un WHERE tal que la información final consiste en los NUM\_PEDIDO tal que algún producto es del restaurante indicado por parámetro, y este pedido tiene fecha dentro del rango indicado por parámetro. Después, se le hace UNION a estas tablas, para consolidarlas y facilitar el siguiente paso que consiste en realizar el LEFT OUTER JOIN de esta tabla con USUARIOS, tal que se le haga un “append” a la información que se pide originalmente en la sentencia, ya que se quiere la información completa de los usuarios que consuman en un restaurante en un rango de fechas específico. Si se tiene una cláusula ORDER BY al final de la sentencia, esté ORDER BY se debe realizar posteriormente, mediante un MERGE sobre los datos en memoria secundaria, como lo vimos en clase. Ahora, cómo se realizó un LEFT OUTER JOIN, se corre un SELECT sobre el resultset, retornando sólo las tuplas que tienen null las columnas del outer join, dando como resultado aquellos que no consumieron productos de el restaurante indicado en el rango de fechas por parámetro.

➔ Selectividad Alta:

Parámetros:

‘RESTAURANTE 12’ (1%)

10-10-2010 -> 10-10-2011 (6.5%)

Database Console SisTrans

Output

```

sql> ((SELECT EMAIL, NOMBRE, IDENTIFICACION FROM USUARIOS LEFT OUTER JOIN (
  (SELECT EMAIL_USER AS EMAIL_USER1
  FROM PEDIDO JOIN (PEDIDO_PLATO JOIN PLATO ON PEDIDO_PLATO.ID_PLATO = PLATO.ID_PLATO)
  ON PEDIDO.NUM_PEDIDO = PEDIDO_PLATO.NUM_PEDIDO WHERE PLATO.RESTAURANTE = 'RESTAURANTE 12'
  AND PEDIDO.FECHA BETWEEN TO_DATE('10-10-2010', 'DD/MM/YYYY') AND TO_DATE('10-10-2011', 'DD/MM/YYYY'))
  ON EMAIL = EMAIL_USER1 WHERE EMAIL_USER1 IS NULL) UNION (SELECT EMAIL, NOMBRE, IDENTIFICACION FROM USUARIOS JOIN (
  (SELECT EMAIL_USER AS EMAIL_USER1
  FROM PEDIDO JOIN (PEDIDO_MENU JOIN MENU ON PEDIDO_MENU.ID_MENU = MENU.ID_MENU)
  ON PEDIDO.NUM_PEDIDO = PEDIDO_MENU.NUM_PEDIDO WHERE MENU.NOMRESTAURANTE = 'RESTAURANTE 12'
  AND PEDIDO.FECHA BETWEEN TO_DATE('10-10-2010', 'DD/MM/YYYY') AND TO_DATE('10-10-2011', 'DD/MM/YYYY'))
  ON EMAIL = EMAIL_USER1 WHERE EMAIL_USER1 IS NULL))
  ORDER BY EMAIL
[2017-11-21 20:38:00] 500 rows retrieved starting from 1 in 6s 254ms (execution: 6s 63ms, fetching: 191ms)

```

Result 170

EMAIL	NOMBRE
EMAIL 1	Juan
EMAIL 10	Juan
EMAIL 100	Juan
EMAIL 1000	Juan
EMAIL 10000	Juan
EMAIL 100000	Marcos
EMAIL 10001	Juan
EMAIL 10002	Juan
EMAIL 10003	Juan
EMAIL 10004	Juan
EMAIL 10005	Juan
EMAIL 10006	Juan
EMAIL 10007	Juan
EMAIL 10008	Juan

8: TODO Database Changes Database Console

500 rows retrieved starting from 1 in 1s 73ms (execution: 841ms, fetching: 232ms)

29:1 LF: UTF-8

→ Selectividad Media:

Parámetros:

'RESTAURANTE 271' (27%)

10-10-2010 -> 10-10-2011 (6.5%)

Database Console SisTrans

Output

```

sql> ((SELECT EMAIL, NOMBRE, IDENTIFICACION FROM USUARIOS LEFT OUTER JOIN (
  (SELECT EMAIL_USER AS EMAIL_USER1
  FROM PEDIDO JOIN (PEDIDO_PLATO JOIN PLATO ON PEDIDO_PLATO.ID_PLATO = PLATO.ID_PLATO)
  ON PEDIDO.NUM_PEDIDO = PEDIDO_PLATO.NUM_PEDIDO WHERE PLATO.RESTAURANTE = 'RESTAURANTE 271'
  AND PEDIDO.FECHA BETWEEN TO_DATE('10-10-2010', 'DD/MM/YYYY') AND TO_DATE('10-10-2011', 'DD/MM/YYYY'))
  ON EMAIL = EMAIL_USER1 WHERE EMAIL_USER1 IS NULL) UNION (SELECT EMAIL, NOMBRE, IDENTIFICACION FROM USUARIOS JOIN (
  (SELECT EMAIL_USER AS EMAIL_USER1
  FROM PEDIDO JOIN (PEDIDO_MENU JOIN MENU ON PEDIDO_MENU.ID_MENU = MENU.ID_MENU)
  ON PEDIDO.NUM_PEDIDO = PEDIDO_MENU.NUM_PEDIDO WHERE MENU.NOMRESTAURANTE = 'RESTAURANTE 271'
  AND PEDIDO.FECHA BETWEEN TO_DATE('10-10-2010', 'DD/MM/YYYY') AND TO_DATE('10-10-2011', 'DD/MM/YYYY'))
  ON EMAIL = EMAIL_USER1 WHERE EMAIL_USER1 IS NULL))
  ORDER BY EMAIL
[2017-11-21 20:42:49] 500 rows retrieved starting from 1 in 729ms (execution: 552ms, fetching: 177ms)

```

Result 174

EMAIL	NOMBRE	IDENTIFICACION
EMAIL 100	Juan	100
EMAIL 1000	Juan	1000
EMAIL 10000	Juan	10000
EMAIL 100000	Marcos	100000
EMAIL 10001	Juan	10001
EMAIL 10002	Juan	10002
EMAIL 10003	Juan	10003
EMAIL 10004	Juan	10004
EMAIL 10005	Juan	10005
EMAIL 10006	Juan	10006
EMAIL 10007	Juan	10007
EMAIL 10008	Juan	10008
EMAIL 10009	Juan	10009
EMAIL 1001	Juan	1001

8: TODO Database Changes Database Console

14:1 LF: UTF-8

→ Selectividad Baja:

Parámetros:

'RESTAURANTE 10' (67%)

10-10-2010 -> 10-10-2011 (6.5%)

Database Console SisTrans

Output

```

sql> ((SELECT EMAIL, NOMBRE, IDENTIFICACION FROM USUARIOS LEFT OUTER JOIN (
  (SELECT EMAIL_USER AS EMAIL_USER1
  FROM PEDIDO JOIN (PEDIDO_PLATO JOIN PLATO ON PEDIDO_PLATO.ID_PLATO = PLATO.ID_PLATO)
  ON PEDIDO.NUM_PEDIDO = PEDIDO_PLATO.NUM_PEDIDO WHERE PLATO.RESTAURANTE = 'RESTAURANTE 10'
  AND PEDIDO.FECHA BETWEEN TO_DATE('10-10-2010', 'DD/MM/YYYY') AND TO_DATE('10-10-2011', 'DD/MM/YYYY'))
  ON EMAIL = EMAIL_USER1 WHERE EMAIL_USER1 IS NULL) UNION (SELECT EMAIL, NOMBRE, IDENTIFICACION FROM USUARIOS JOIN (
  (SELECT EMAIL_USER AS EMAIL_USER1
  FROM PEDIDO JOIN (PEDIDO_MENU JOIN MENU ON PEDIDO_MENU.ID_MENU = MENU.ID_MENU)
  ON PEDIDO.NUM_PEDIDO = PEDIDO_MENU.NUM_PEDIDO WHERE MENU.NOMRESTAURANTE = 'RESTAURANTE 10'
  AND PEDIDO.FECHA BETWEEN TO_DATE('10-10-2010', 'DD/MM/YYYY') AND TO_DATE('10-10-2011', 'DD/MM/YYYY'))
  ON EMAIL = EMAIL_USER1 WHERE EMAIL_USER1 IS NULL))
  ORDER BY EMAIL
[2017-11-21 20:44:30] 500 rows retrieved starting from 1 in 688ms (execution: 529ms, fetching: 159ms)

```

Result 176

EMAIL	NOMBRE	IDENTIFICACION
EMAIL 100	Juan	100
EMAIL 10000	Juan	10000
EMAIL 100000	Marcos	100000
EMAIL 10001	Juan	10001
EMAIL 10002	Juan	10002
EMAIL 10003	Juan	10003
EMAIL 10004	Juan	10004
EMAIL 10005	Juan	10005
EMAIL 10006	Juan	10006
EMAIL 10007	Juan	10007
EMAIL 10008	Juan	10008
EMAIL 10009	Juan	10009
EMAIL 10010	Juan	10010
EMAIL 10011	Juan	10011

8: TODO Database Changes Database Console

14:1 LF: UTF-8

### C. RFC11

- Sentencia SQL:

```
SELECT ID_PLATO, PLATO.NOMBRE, PLATO.CATEGORIA, COUNT(*) as numVendidos, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') as day
FROM pedido_plato NATURAL JOIN PEDIDO NATURAL JOIN PLATO WHERE TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') = 'MIÉRCOLES'
GROUP BY ID_PLATO, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY'), PLATO.NOMBRE, PLATO.CATEGORIA
ORDER BY numVendidos ASC
fetch first 1 rows only;

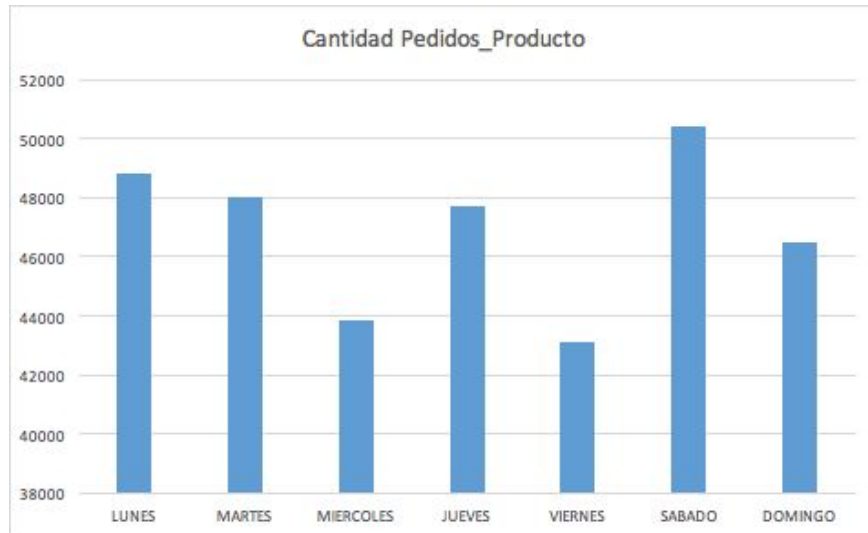
SELECT ID_PLATO, PLATO.NOMBRE, PLATO.CATEGORIA, COUNT(*) as numVendidos, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') as day
FROM pedido_plato NATURAL JOIN PEDIDO NATURAL JOIN PLATO WHERE TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') = 'MIÉRCOLES'
GROUP BY ID_PLATO, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY'), PLATO.NOMBRE, PLATO.CATEGORIA
ORDER BY numVendidos DESC
fetch first 1 rows only;

SELECT RESTAURANTES.NOMBRE, COUNT(PEDIDO_PLATO.NUM_PEDIDO) AS NUM, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') AS DAY
FROM RESTAURANTES INNER JOIN
PLATO ON RESTAURANTES.NOMBRE = PLATO.RESTAURANTE JOIN
PEDIDO_PLATO ON PLATO.ID_PLATO = PEDIDO_PLATO.ID_PLATO JOIN PEDIDO ON PEDIDO_PLATO.NUM_PEDIDO =
PEDIDO.NUM_PEDIDO
WHERE TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY')
= 'MIÉRCOLES'
GROUP BY RESTAURANTES.NOMBRE, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY')
ORDER BY NUM DESC
fetch first 1 rows only;

SELECT RESTAURANTES.NOMBRE, COUNT(PEDIDO_PLATO.NUM_PEDIDO) AS NUM, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') AS DAY
FROM RESTAURANTES INNER JOIN
PLATO ON RESTAURANTES.NOMBRE = PLATO.RESTAURANTE JOIN
PEDIDO_PLATO ON PLATO.ID_PLATO = PEDIDO_PLATO.ID_PLATO JOIN PEDIDO ON PEDIDO_PLATO.NUM_PEDIDO =
PEDIDO.NUM_PEDIDO
WHERE TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY')
= 'MIÉRCOLES'
GROUP BY RESTAURANTES.NOMBRE, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY')
ORDER BY NUM ASC
fetch first 1 rows only;
```

Aunque en este requerimiento no existan parámetros, debido a que se pide el mas y el menos consumido producto de la rotonda para cada día, junto con el mas y menos frecuentado restaurante también para cada día. De este modo, se deben realizar las consultas por cada uno de los días de la semana, sin embargo, con el fin de forzar al máximo la sentencia sql, se decidió distribuir las fechas de los pedidos de forma equitativa para tener que siempre forzar la sentencia. Así, la distribución dependiendo de los días es la siguiente:





Dia	Cantidad Pedidos_Producto
LUNES	48838
MARTES	47994
MIERCOLES	43828
JUEVES	47717
VIERNES	43080
SABADO	50395
DOMINGO	46482

D. RFC12

```

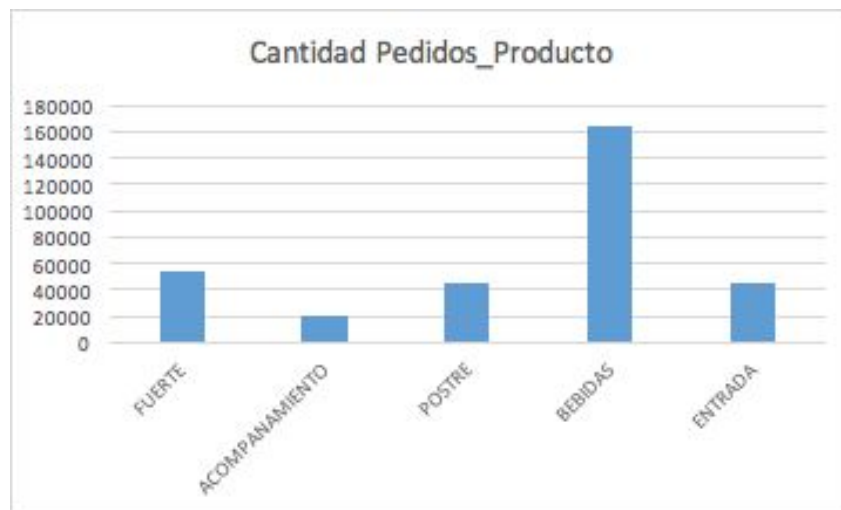
SELECT CLIENTEFR.EMAIL, CLIENTEFR.NOMBRE, CLIENTEFR.IDENTIFICACION FROM CLIENTEFR WHERE EMAIL NOT IN (SELECT EMAIL_USER FROM
PEDIDO NATURAL JOIN PEDIDO_PLATO INNER JOIN PLATO ON PEDIDO_PLATO.ID_PLATO = PLATO.ID_PLATO where PRECIO < 36885 AND CATEGORIA != 'FUERTE');

SELECT EMAIL, NOMBRE, IDENTIFICACION FROM (SELECT SUM(NUMWEEKS) AS TOTALWEEKS
FROM (SELECT COUNT(DISTINCT(WEEK)) AS NUMWEEKS, YEAR FROM (SELECT NUM_PEDIDO, EXTRACT(YEAR FROM FECHA) AS YEAR,
to_number(to_char(FECHA, 'WW')) AS WEEK FROM (PEDIDO)) GROUP BY YEAR)) NATURAL JOIN (SELECT EMAIL, NOMBRE, IDENTIFICACION,
SUM(NUMWEEKS) AS TOTALWEEKS FROM (SELECT COUNT(*) AS NUMWEEKS, EMAIL, NOMBRE, IDENTIFICACION, YEAR FROM
(SELECT EXTRACT(YEAR FROM FECHA) AS YEAR, EMAIL, NOMBRE, IDENTIFICACION, to_number(to_char(FECHA, 'WW')) AS WEEK
FROM (PEDIDO INNER JOIN CLIENTEFR ON PEDIDO.EMAIL_USER = CLIENTEFR.EMAIL)) GROUP BY EMAIL, NOMBRE, IDENTIFICACION, YEAR) GROUP BY EMAIL, NOMBRE, IDENTIFICACION, YEAR);

SELECT NOMBRE, EMAIL, IDENTIFICACION FROM (PEDIDO NATURAL JOIN PEDIDO_PLATO NATURAL JOIN CLIENTEFR) WHERE EMAIL NOT IN (SELECT EMAIL_USER
FROM (PEDIDO NATURAL JOIN PEDIDO_MENUS));

```

En el requerimiento 12, así como en el 11, no hay parámetros, ya que se pide encontrar todos los clientes buenos, los cuales entran en esta categoría por 3 diferentes factores. Sin embargo, en este caso, debido a que siempre se escogen los clientes que cumplan las mismas condiciones, aunque exista una distribución, esta no va cambiar de ninguna manera los resultados debido a que estos serán siempre los mismos, a menos que aumente la cantidad de clientes que cumplan cierta característica, pero esto se debería a factores “externos”, como lo son el hecho de que un cliente efectúe más pedido de platos fuertes, o que efectúe pedidos más caros, o que efectúe pedidos únicamente de platos y no de menús. Aún así, existe una amplia distribución en las Categorías de los platos, de modo que lo que sí es afectado, es la densidad de datos al efectuar la búsqueda, de este modo, tenemos la siguiente distribución:



Categoria	Cantidad Pedidos_Producto
FUERTE	53335
ACOMPAÑAMIENTO	19999
POSTRE	45000
BEBIDAS	165000

ENTRADA	45000
---------	-------

- Planes y tiempo sentencias:

Vale la pena aclarar que, debido a factores externos, las sentencias corrían en tiempos muy lentos, siendo tal la gravedad de la situación que se recurrió a los monitores de la materia y nos supieron darnos razón, pues las mismas sentencias corrían mucho más rápido en tablas de otros y hasta con más datos, de modo que no se pudo concluir cuál es el error y por tanto fue inevitable que los tiempos fueran mayores que lo que deberían. Sin embargo, se hizo todo lo posible para que los tiempos fueran lo menor posible. Esto es tanto para el RFC11 como para el RFC12

- RFC11

Tiempo Sentencia 1:

<pre> SELECT ID_PLATO, PLATO.NOMBRE, PLATO.CATEGORIA, COUNT(*) as num/Vendidos, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') as day FROM pedido_plato NATURAL JOIN PEDIDO NATURAL JOIN PLATO WHERE TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') = 'MIÉRCOLES' GROUP BY ID_PLATO, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY'), PLATO.NOMBRE, PLATO.CATEGORIA ORDER BY num/Vendidos ASC Fetch First 1 rows only: </pre>				
<p>Schema de base A   Exploración del Plan A   Resultado de la Consulta X</p> <p>Tratamiento: File Respuestas: 1 en 1,127 segundos</p>				
ID PLATO	NOMBRE	CATEGORIA	NUM/VENDIDOS	DAY
15175	Bebida	15175 BEBIDAS		MIÉRCOLES

Plan de consulta sentencia 1:

<pre> SELECT ID_PLATO, PLATO.NOMBRE, PLATO.CATEGORIA, COUNT(*) as numPedidos, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') as day FROM pedido_plato NATURAL JOIN PEDIDO NATURAL JOIN PLATO WHERE TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') = 'MIÉRCOLES' GROUP BY ID_PLATO, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY'), PLATO.NOMBRE, PLATO.CATEGORIA ORDER BY numPedidos ASC Fetch First 1 rows only </pre>					
<p>Salida de Script: <span>Explicación del Plan</span> <span>Resultado de la Consulta</span></p> <p>SQL 2.22 segundos</p>					
OPERATION	OBJECT_NAME	OPEROR	COST%MEMO	ROWS	
SELECT STATEMENT				1	31
HASH				1	42
VIEW	SQL_TABLE			1	15
Plan Predicate					
Plan Predicate	from_subquery_alias_subquery_alias_subquery_alias				
VIEW			SORT PUSHED NAME	1	15
Plan Predicate					
Plan Predicate	ROW_NUMBER() OVER (ORDER BY COUNT(*) ASC)				
HASH			GROUP BY	1	15
SELECT STATEMENT				1	17
NESTED LOOPS				1	17
NESTED LOOPS				1	17
INDEX	SQL_TABLE		FULL SCAN	1	17
Plan Predicate					
Plan Predicate	pedido_plato.num_pedido <> 0				
Plan Predicate					
TABLE ACCESS	pedido		BY INDEX ROWID	1	0
Plan Predicate					
Plan Predicate	TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') = 'MIÉRCOLES'				
INDEX	SQL_TABLE		UNIQUE SCAN	1	0
Plan Predicate					
Plan Predicate	pedido_plato.num_pedido = pedido_plato.num_pedido				
INDEX	plato_pla		UNIQUE SCAN	1	0
Plan Predicate					
Plan Predicate	pedido_plato.id_plato = plato.id_plato				
TABLE ACCESS	plato		BY INDEX ROWID	1	0

Tiempo sentencia 2:

<pre> SELECT ID_PLATO, PLATO.NOMBRE, PLATO.CATEGORIA, COUNT(*) as numPedidos, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') as day FROM pedido_plato NATURAL JOIN PEDIDO NATURAL JOIN PLATO WHERE TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') = 'MIÉRCOLES' GROUP BY ID_PLATO, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY'), PLATO.NOMBRE, PLATO.CATEGORIA ORDER BY numPedidos DESC Fetch First 1 rows only </pre>					
<p>Salida de Script: <span>Explicación del Plan</span> <span>Resultado de la Consulta</span></p> <p>SQL 1.22 segundos</p>					
ID_PLATO	NOMBRE	CATEGORIA	NUMPEDIDOS	DAY	
1	10 Tarta de 10 MEJORES	6560 MIÉRCOLES			

Plan de consulta sentencia 2:

<pre> SELECT PLATO, COUNT(PEDIDO_PLATO.SUM_PEDIDO) AS SUM, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') AS DAY FROM RESTAURANTES INNER JOIN PLATO ON RESTAURANTES.NUMER = PLATO.RESTAURANTE JOIN PEDIDO_PLATO ON PLATO.ID_PLATO = PEDIDO_PLATO.ID_PLATO JOIN PEDIDO ON PEDIDO_PLATO.NUM_PEDIDO = PEDIDO.NUM_PEDIDO WHERE TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') = 'MIÉRCOLES' GROUP BY RESTAURANTES.SUMMER, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') ORDER BY SUM DESC fetch first 1 rows only; </pre>					
<p>Salida de Script   Explicación del Plan   Resultado de la Consulta</p> <p>Todas las Filas Recuperadas: 1 en 3 segundos</p>					
NOMBRE	NUM	DAY			
1 RESTAURANTE 10	4601	MIÉRCOLES			

Tiempo sentencia 3:

<pre> SELECT RESTAURANTES.NOMBRE, COUNT(PEDIDO_PLATO.SUM_PEDIDO) AS SUM, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') AS DAY FROM RESTAURANTES INNER JOIN PLATO ON RESTAURANTES.NUMER = PLATO.RESTAURANTE JOIN PEDIDO_PLATO ON PLATO.ID_PLATO = PEDIDO_PLATO.ID_PLATO JOIN PEDIDO ON PEDIDO_PLATO.NUM_PEDIDO = PEDIDO.NUM_PEDIDO WHERE TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') = 'MIÉRCOLES' GROUP BY RESTAURANTES.SUMMER, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') ORDER BY SUM DESC fetch first 1 rows only; </pre>					
<p>Salida de Script   Explicación del Plan   Resultado de la Consulta</p> <p>Todas las Filas Recuperadas: 1 en 3 segundos</p>					
NOMBRE	NUM	DAY			
1 RESTAURANTE 10	4601	MIÉRCOLES			

Plan de consulta sentencia 3:





<pre> SELECT RESTAURANTES.NOMBRE, COUNT(PEDIDO_PLATO.NUM_PEDIDO) AS NUM, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') AS DAY FROM RESTAURANTES INNER JOIN PLATO ON RESTAURANTES.NOMBRE = PLATO.RESTAURANTE JOIN PEDIDO_PLATO ON PLATO.ID_PLATO = PEDIDO_PLATO.ID_PLATO JOIN PEDIDO ON PEDIDO_PLATO.NUM_PEDIDO = PEDIDO.NUM_PEDIDO WHERE TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') = 'MIÉRCOLES' GROUP BY RESTAURANTES.NOMBRE, TO_CHAR(TO_DATE(PEDIDO.FECHA, 'DD-MM-YY'), 'DAY') ORDER BY NUM ASC fetch first 1 rows only; </pre>					
<div> <div>Salida de Script</div> <div>Resultado de la Consulta</div> <div>Explicación del Plan</div> </div> <div>SQL   1,585 segundos</div>					
OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	
SELECT STATEMENT				1	20
SORT		ORDER BY		1	20
VIEW	SYS_NULL			1	19
Filter Predicates					
from\$_subquery\$_008.rowlimit_\$\$_rownumber <= 1					
WINDOW					
Filter Predicates					
ROW_NUMBER() OVER ( ORDER BY COUNT(*) ) <= 1					
HASH		GROUP BY		1	19
NESTED LOOPS				1	17
NESTED LOOPS				1	17
NESTED LOOPS				1	17
INDEX	SYS_C00216662	FULL SCAN		328334	0
Access Predicates					
PEDIDO_PLATO.NUM_PEDIDO > 0					
Filter Predicates					
PEDIDO_PLATO.NUM_PEDIDO > 0					
TABLE ACCESS	PEDIDO	BY INDEX ROWID		1	0
Filter Predicates					
TO_CHAR(TO_DATE(INTERNAL_FUNCTION(PEDIDO.FECHA), 'DD-MM-YY'), 'DAY') = 'MIÉRCOLES'					
INDEX	SYS_C00216661	UNIQUE SCAN		1	0
Access Predicates					
PEDIDO_PLATO.NUM_PEDIDO = PEDIDO.NUM_PEDIDO					
INDEX	PLATO_PK1	UNIQUE SCAN		1	0
Access Predicates					
PLATO.ID_PLATO = PEDIDO_PLATO.ID_PLATO					
TABLE ACCESS	PLATO	BY INDEX ROWID		1	0
<div>Other XML</div> <div>(info)</div> <div>info type = "db_version"</div>					

- Análisis de eficiencia

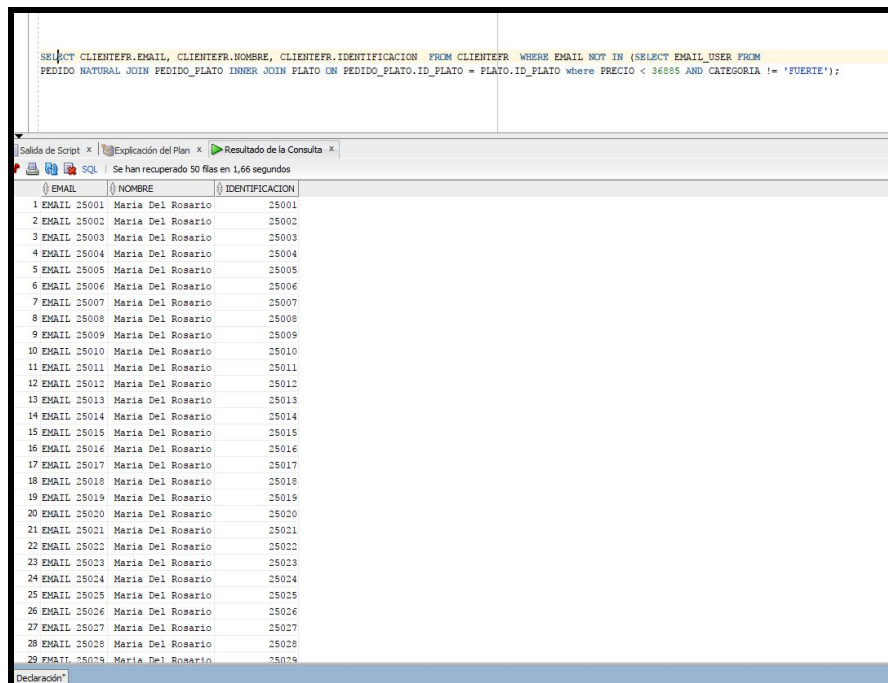
Para obtener diferentes selectividades en este requerimiento, sería necesario usar escenarios en donde se cambia el día ya que aunque no posean una amplia distribución, siguen teniendo valores diferentes y una selectividad diferente (aunque no sea muy drástica), de este modo, un escenario sería querer ver los productos más y menos consumidos junto con los restaurantes mas y menos frecuentados en un día específico, como por ejemplo, el Miércoles, de este modo, diría que la manera más efectiva de realizar la consulta sería usando hash table, ya que aunque se esté buscando todos los elementos que pertenecen a un rango (un día) este elemento no está en ninguna columna, lo que esta es la fecha y por tanto, no sería tan útil. Por tanto, lo mejor sería agrupar los elementos por la cantidad de veces consumido, y para efectuar esta agrupación sería útil usar índices hash, por otro lado, los JOIN se efectúan en las PK, de modo que este proceso sería utilizando los índices de las correspondidas PK.

En el plan efectuado por Oracle, fue perceptible que efectivamente usó una tabla hash para facilitar el agrupamiento, sin embargo, también se vio que como uso varios nested loops por la cantidad de datos, ya que no todo cabía en memoria y por tanto tuvo que subir solo unos e ir ordenando poco a poco, manera de efectuar los joins de forma eficiente cuando no caben todos los datos en la memoria principal. Por otro lado, es notable que hizo uso diversas veces

de los índices, efectuando varios index rowid por medio de ellos para facilitar las búsquedas de los elementos. De este modo, si es en parte como se esperaba, sin embargo, Oracle hace muchas más acciones de las que uno imagina, haciendo los procesos lo más eficiente posibles.

- RFC12

Tiempo sentencia 1:



The screenshot shows an Oracle SQL Developer window with a query in the top pane and its results in the bottom pane. The query is:

```
SELECT CLIENTEPR.EMAIL, CLIENTEPR.NOMBRE, CLIENTEPR.IDENTIFICACION FROM CLIENTEPR WHERE EMAIL NOT IN (SELECT EMAIL_USER FROM PEDIDO NATURAL JOIN PEDIDO_PLATO INNER JOIN PLATO ON PEDIDO_PLATO.ID_PLATO = PLATO.ID_PLATO where PRECIO < 36885 AND CATEGORIA != 'FUERTE');
```

The results pane shows 29 rows of data. The status bar at the bottom indicates "Se han recuperado 50 filas en 1,66 segundos".

EMAIL	NOMBRE	IDENTIFICACION
EMAIL 25001	Maria Del Rosario	25001
EMAIL 25002	Maria Del Rosario	25002
EMAIL 25003	Maria Del Rosario	25003
EMAIL 25004	Maria Del Rosario	25004
EMAIL 25005	Maria Del Rosario	25005
EMAIL 25006	Maria Del Rosario	25006
EMAIL 25007	Maria Del Rosario	25007
EMAIL 25008	Maria Del Rosario	25008
EMAIL 25009	Maria Del Rosario	25009
EMAIL 25010	Maria Del Rosario	25010
EMAIL 25011	Maria Del Rosario	25011
EMAIL 25012	Maria Del Rosario	25012
EMAIL 25013	Maria Del Rosario	25013
EMAIL 25014	Maria Del Rosario	25014
EMAIL 25015	Maria Del Rosario	25015
EMAIL 25016	Maria Del Rosario	25016
EMAIL 25017	Maria Del Rosario	25017
EMAIL 25018	Maria Del Rosario	25018
EMAIL 25019	Maria Del Rosario	25019
EMAIL 25020	Maria Del Rosario	25020
EMAIL 25021	Maria Del Rosario	25021
EMAIL 25022	Maria Del Rosario	25022
EMAIL 25023	Maria Del Rosario	25023
EMAIL 25024	Maria Del Rosario	25024
EMAIL 25025	Maria Del Rosario	25025
EMAIL 25026	Maria Del Rosario	25026
EMAIL 25027	Maria Del Rosario	25027
EMAIL 25028	Maria Del Rosario	25028
EMAIL 25029	Maria Del Rosario	25029

Plan de consulta sentencia 1:

Salida de Script x

Resultado de la Consulta x

Explicación del Plan x

SQL 3,663 segundos

SELECT STATEMENT

MERGE JOIN

TABLE ACCESS CLIENTEFR

INDEX CLIENTEFR\_PK

SORT

Access Predicates EMAIL=EMAIL\_USER

Filter Predicates EMAIL=EMAIL\_USER

VIEW SYS\_VW\_NSO\_1

NESTED LOOPS

NESTED LOOPS

INDEX SYS\_C00216662

Access Predicates PEDIDO\_PLATO.NUM\_PEDIDO>0

Filter Predicates PEDIDO\_PLATO.NUM\_PEDIDO>0

TABLE ACCESS PEDIDO

Filter Predicates PEDIDO.PRECIO<36885

INDEX SYS\_C00216661

Access Predicates PEDIDO.NUM\_PEDIDO=PEDIDO\_PLATO.NUM\_PEDIDO

TABLE ACCESS PLATO

Filter Predicates PLATO.CATEGORIA<>'FUERTE'

INDEX PLATO\_PK1

Access Predicates PEDIDO\_PLATO.ID\_PLATO=PLATO.ID\_PLATO

Other XML

{info}

info type="db\_version"

12.1.0.2

Tiempo sentencia 2:

Salida de Script x

Explicación del Plan x

Resultado de la Consulta x

SQL Todas las Filas Recuperadas: 0 en 1,985 segundos

EMAIL NOMBRE IDENTIFI...

Plan de consulta sentencia 2:

SELECT EMAIL, NOMBRE, IDENTIFICACION FROM (SELECT SUM(NUMWEEKS) AS TOTALWEEKS  
FROM (SELECT COUNT(DISTINCT(WEEK)) AS NUMWEEKS, YEAR FROM (SELECT NUM\_PEDIDO, EXTRACT(YEAR FROM FECHA) AS YEAR,  
to\_number(to\_char(FECHA,'WW')) AS WEEK FROM (PEDIDO)) GROUP BY YEAR)) NATURAL JOIN (SELECT EMAIL, NOMBRE, IDENTIFICACION,  
SUM(NUMWEEKS) AS TOTALWEEKS FROM (SELECT COUNT(\*) AS NUMWEEKS, EMAIL, NOMBRE, IDENTIFICACION, YEAR FROM  
(SELECT EXTRACT(YEAR FROM FECHA) AS YEAR, EMAIL, NOMBRE, IDENTIFICACION, to\_number(to\_char(FECHA,'WW')) AS WEEK  
FROM (PEDIDO INNER JOIN CLIENTEFR ON PEDIDO.EMAIL\_USER = CLIENTEFR.EMAIL)) GROUP BY EMAIL, NOMBRE, IDENTIFICACION, YEAR) GROUP BY EMAIL, NOMBRE, IDENTIFICACION, YEAR);

Salida de Script x Resultado de la Consulta x Explicación del Plan x

SQL 0,141 segundos

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	661
NESTED LOOPS			1	661
VIEW			1	149
SORT		AGGREGATE	1	149
VIEW			1	149
HASH		GROUP BY	1	149
VIEW	SYS.VM_NWWW_1		1	148
HASH		GROUP BY	1	148
INDEX	INDICEFECHA	FAST FULL SCAN	1	147
VIEW			1	512
Filter Predicates		from\$_subquery\$001.TOTALWEEKS=from\$_subquery\$005.TOTALWEEKS		
SORT		GROUP BY	1	512
NESTED LOOPS			1	511
NESTED LOOPS			1	511
TABLE ACCESS	PEDIDO	FULL	1	511
INDEX	CLIENTEFR_PK	UNIQUE SCAN	1	0
Access Predicates		PEDIDO.EMAIL_USER=CLIENTEFR.EMAIL		
TABLE ACCESS	CLIENTEFR	BY INDEX ROWID	1	0

Other XML (info) info type="db\_version"

Tiempo sentencia 3:

SELECT NOMBRE, EMAIL, IDENTIFICACION FROM (PEDIDO NATURAL JOIN PEDIDO\_PLATO NATURAL JOIN CLIENTEFR) WHERE EMAIL NOT IN (SELECT EMAIL\_USER  
FROM (PEDIDO NATURAL JOIN PEDIDO\_MENUS));

Salida de Script x Explicación del Plan x Resultado de la Consulta x

SQL Se han recuperado 50 filas en 15,077 segundos

	NOMBRE	EMAIL	IDENTIFICACION
1	Marcos	EMAIL 73577	73577
2	Marcos	EMAIL 73578	73578
3	Marcos	EMAIL 73579	73579
4	Marcos	EMAIL 73580	73580
5	Marcos	EMAIL 73581	73581
6	Marcos	EMAIL 73582	73582
7	Marcos	EMAIL 73583	73583
8	Marcos	EMAIL 73584	73584
9	Marcos	EMAIL 73585	73585
10	Marcos	EMAIL 73586	73586
11	Marcos	EMAIL 73587	73587
12	Marcos	EMAIL 73588	73588
13	Marcos	EMAIL 73589	73589
14	Marcos	EMAIL 73590	73590
15	Marcos	EMAIL 73591	73591
16	Marcos	EMAIL 73592	73592
17	Marcos	EMAIL 73593	73593
18	Marcos	EMAIL 73594	73594
19	Marcos	EMAIL 73595	73595
20	Marcos	EMAIL 73596	73596
21	Marcos	EMAIL 73597	73597
22	Marcos	EMAIL 73598	73598
23	Marcos	EMAIL 73599	73599
24	Marcos	EMAIL 73600	73600
25	Marcos	EMAIL 73601	73601
26	Marcos	EMAIL 73602	73602
27	Marcos	EMAIL 73603	73603
28	Marcos	EMAIL 73604	73604
29	Marcos	EMAIL 73605	73605
30	Marcos	EMAIL 73606	73606

a Declaración\*

### Plan de consulta sentencia 3:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	275
FILTER				
NOT EXISTS (SELECT 0 FROM PEDIDO_MENUS PEDIDO_MENUS, PEDIDO PEDIDO WHERE PEDIDO.NUM_PEDIDO = PEDIDO_MENUS.NUM_PEDIDO AND PEDIDO.EMAIL_USER = B1 AND PEDIDO_MENUS.NUM_PET)				
MERGE JOIN				
INDEX	SYS_C00216669	CARTESIAN	1	272
BUFFER		FULL SCAN	1	0
TABLE ACCESS	CLIENTEFR	SORT	1	272
NESTED LOOPS				
INDEX	SYS_C00216674	FULL	1	272
Access Predicates		SEMI	89997	3
PEDIDO_MENUS.NUM_PEDIDO > 0		FULL SCAN	89997	0
Filter Predicates				
PEDIDO_MENUS.NUM_PEDIDO > 0				
TABLE ACCESS	PEDIDO	BY INDEX ROWID	1	0
Filter Predicates				
PEDIDO.EMAIL_USER = B1				
INDEX	SYS_C00216661	UNIQUE SCAN	1	0
Access Predicates				
PEDIDO.NUM_PEDIDO = PEDIDO_MENUS.NUM_PEDIDO				

- **Análisis de eficiencia:**

Para este requerimiento, la única forma de obtener diferentes selectividades es aumentando o disminuyendo la cantidad de productos pertenecientes a la categoría Fuerte, sin embargo esto no se puede hacer por medio de ningún parámetro. Aún así, es posible considerar un escenario donde la categoría Fuerte tiene aproximadamente un quinto de los productos totales, de modo que como se requiere los que únicamente consumen platos fuertes, se debe escoger los que no hagan parte de aquellos que consuman algo diferente a fuerte, pudiendo así obtener los deseados. Para efectuar todo esto, la manera óptima sería efectuar, posiblemente, nested loops ya que la información no cabe en memoria principal, y a partir de este procedimiento efectuar los joins necesarios usando en la mayoría de los casos las PK, de modo que se usen los índices y consecuentemente, los join, así sean nested loops, son mucho más rápidos.

Como se puede observar en las imágenes anexadas en el punto anterior, lo que hace la consola es, efectivamente, realizar varios nested loops debido al tamaño de los datos, sin embargo, efectúa también un merge join al comienzo de todas las operaciones, de modo que escoge que tipo de join efectuar dependiendo de los datos y del tipo de join por usar. Además, es perceptible como en casi todos los casos recorre los elementos a partir de los índices o en el peor de los casos, unique scan. Este procedimiento acelera mucho el proceso debido a que por medio de los índices se puede comprar los elementos y escoger correctamente sin tanto esfuerzo. Así, como se dijo anteriormente, efectivamente usó varios nested loops y se apoyó en casi todos los casos en los índices debido a la facilidad que ellos dan, sin embargo también usó varios unique scan y efectuó un merge join, mostrando nuevamente, que siempre encuentra maneras de efectuar los procesos de manera muy óptima y eficiente.

## **Comparación de operaciones IF, WHILE vs JOIN, SELECT**

Para realizar las consultas y que estas se hagan a gran velocidad, el administrador de bases de datos cuenta con un optimizador de consultas, que tiene como objetivo, que la consulta se realice lo más rápido posible aun cuando la sentencia sea poco eficiente. Esto lo logra por medio de herramientas como índices y árboles, que guarda en la memoria principal del servidor.

Cuando llega una consulta, el administrador de bases de datos la analiza y genera un plan de ejecución en el que se optimiza el tiempo requerido para realizarla. Para esto tiene en cuenta los joins que se requiere hacer, la cantidad de datos que se deben leer y los índices y árboles existentes sobre los datos. Al consultar datos, por ejemplo, el smbd mirara si existen indices sobre el campo buscado. Si los hay realizará una aproximación del porcentaje de datos que debe leer para encontrarlo y dependiendo de eso decidirá si usar el indice o leer todos los datos. En el caso del join, el smdb calculara si es más eficiente hacer la búsqueda 'WHERE' primero y después unir las tablas o si une las tablas primero y después filtra las tuplas. Otra vez esto lo decide teniendo en cuenta los índices y árboles existentes sobre cada tabla y el tiempo de procesamiento que se requiere tanto para la búsqueda en cada caso, como para la unión de tablas en cada uno de los planes de ejecución posible.

Las bases de datos optimizan el tiempo de realización de consultas de diferentes maneras. Para empezar implementan algoritmos muy eficientes que optimizan la capacidad de procesamiento de memoria principal al máximo, logrando realizar consultas rápidamente. Al contrario de los algoritmos tradicionales de java, las BD están diseñadas para tener una parte de los datos en disco mientras que hay otras en la memoria e igualmente realizar consultas eficientes. Además tienen herramientas para encontrar la información, diferente a recorrerla toda, como lo son los árboles y los índices, que ayudan a realizar búsquedas en menos tiempo aún con datos desorganizados. En java para realizar un algoritmo de búsqueda eficiente se debe ordenar los datos primero.

Estas diferencias son más claras con un ejemplo. Si por ejemplo se quisiera hacer un join, en una base de datos el administrador buscaría la manera más eficiente de hacerlo. Utilizando índices, árboles y algoritmos de búsqueda, la concatenación de tablas quedará lista en cuestión de segundos. Por otro lado si en java queremos concatenar 2 grupos de datos debemos recorrer todos los datos del grupo 1 y múltiples veces los datos del grupo 2. Haciendo que este proceso, para cantidades de datos muy grandes, sea extremadamente ineficiente. Por otra parte si los datos no caben en la memoria principal, con lo algoritmos tradicionales no se lograría pues se requiere que todos estén en memoria para poder ser procesados adecuadamente.