Sebastián Valencia Calderón.
Juan Camilo Bages.

Análisis.

Los requerimientos funcionales de la iteración anterior, contemplaban la inclusión de esquemas de concurrencia y de recuperación ante fallas, por lo que el análisis sobre los índices, se realizó sobre los requerimientos realizados sobre la presente entrega. Dada la metodología de desarrollo, la cual enfatiza el diseño de vistas por requerimientos, la implementación de los requerimientos pedidos para la presenta entrega, no intervienen en la implementación de los requerimientos pasados.

Los requerimientos para la presenta entregan se sintetizan a continuación, aquí se explica brevemente el requerimiento, su análisis para la implementación, aspectos de optimización y de diseño físico de la base de datos.

1. Consultar movimientos de valores (I).

   Mostrar los movimientos de valores realizados en un rango de tiempo que corresponda a un criterio de búsqueda asociado con el movimiento, es decir, un atributo del valor o del movimiento en sí, este criterio es uno no más y es dado por el usuario. La implementación de este requerimiento, requiere la inclusión o adopción del concepto de solicitud dentro de la aplicación, para esto, es necesario incluir una relación sobre las tablas de solicitud y valor, esto se hace a partir de la asociación muchos a muchos presente en el modelo de siempre. La consulta requiere la revisión de las tablas de solicitud, portafolio, y usuario, ya que el criterio de selección se restringe a uno, es fácil implementar desde el punto de vista de aplicación la consulta. Esto es, sin tener en cuenta la implementación física y aspectos de optimización de la base de datos.

2. Consultar movimientos de valores (II).

   Mostrar los movimientos de valores realizados en un rango de tiempo que corresponda a la negación de un criterio de búsqueda asociado con el movimiento, es decir, un atributo del valor o del movimiento en sí, este criterio es uno no más y es dado por el usuario. La implementación de este requerimiento, requiere la inclusión o adopción del concepto de solicitud dentro de la aplicación, para esto, es necesario incluir una relación sobre las tablas de solicitud y valor, esto se hace a partir de la asociación muchos a muchos presente en el modelo de siempre. La consulta requiere la revisión de las tablas de solicitud, portafolio, y usuario, ya que el criterio de selección se restringe a uno, es fácil implementar desde el punto de vista de aplicación la consulta. Esto es, sin tener en cuenta la implementación física y aspectos de optimización de la base de datos.

3. Consultar portafolios.

   Éste requerimiento, es para mostrar los portafolios que contienen valores de un tipo X que han tenido operaciones con valor mayor a Y. Los criterios de búsqueda son dados por el usuario. La implementación de este requerimiento, no amerita la inclusión de nuevas tablas, sino la relación entre las tablas existentes. Estas tablas son: portafolio, valor, y transacción. La optimización requiere la revisión de los atributos que intervienen así como los de discriminación de datos.

4. Consultar valores (II).

   Dado el identificador de un valor, mostrar la información de los portafolios en los que ha estado involucrado, incluyendo el periodo actual. Esta consulta requiere la revisión de las tablas valor, portafolio y solicitud.

A continuación se muestra el análisis para la selección de índices, y el análisis sobre los índices existentes.

El esquema imperativo (Pascal) para el requerimiento 1 es el siguiente:

```
PROCEDURE R1(name, tipo, tipo_renta, intermedio, oferente, t1, t2)
BEGIN
    solicitudes := executor('SELECT * FROM solicitudes')
    values_from_solicitude := executor('SELECT * FORM solicitudes_val')
    val := executor('SELECT * FROM vals')
    protfolios_val =: executor('SELECT * FROM portfolios_vals')
    ans := []

    vals := executor('SELECT value_id
                      FROM
                          (solicitudes
                          JOIN
                          (values_from_solicitude
                              JOIN val
                              ON val.pk_id = values_from_solicitude.val_id)
                          ON solicitudes.pk_id = values_from_solicitude.solicitude_id)
                      WHERE created_at BETWEEN t1 and t2')
    ans = List(executor('SELECT val
                         FROM
                          (SELECT * FROM vals JOIN val ON val.pk_id = vals.value_id)
                          JOIN portfolios_vals
                          ON val.pk_id = portfolios_vals.val_id'))
    return ans
END
```

Los atributos candidatos para el índice son los involucrados en los JOIN, y en los criterios de selección del WHERE.

El esquema imperativo (Pascal) para el requerimiento 2 es el siguiente:

```
PROCEDURE R2(name, tipo, tipo_renta, intermedio, oferente, t1, t2)
BEGIN
    solicitudes := executor('SELECT * FROM solicitudes')
    values_from_solicitude := executor('SELECT * FORM solicitudes_val')
    val := executor('SELECT * FROM vals')
    protfolios_val =: executor('SELECT * FROM portfolios_vals')
    ans := []

    vals := executor('SELECT value_id
                      FROM
                          (solicitudes
                          JOIN
                          (values_from_solicitude
                              JOIN val
                              ON val.pk_id = values_from_solicitude.val_id)
                          ON solicitudes.pk_id = values_from_solicitude.solicitude_id)
                      WHERE created_at BETWEEN t1 and t2')
    ans = List(executor('SELECT val
                         FROM
                          (SELECT * FROM vals JOIN val ON val.pk_id = vals.value_id)
                          JOIN portfolios_vals
                          ON val.pk_id = portfolios_vals.val_id'))
    return (val \ ans)
END
```

Los atributos candidatos para el índice son los involucrados en los JOIN, y en los criterios de selección del WHERE. Puede verse que el esquema es el mismo utilizado anteriormente pero el resultado se quita ans de los valores.

El esquema imperativo (Pascal) para el requerimiento 2 es el siguiente:

```
PROCEDURE R3(tipo, size)
BEGIN
    cardinal = executor('SELECT COUNT(val)
                         FROM
                            (SELECT *
                             FROM
                                (SELECT solicitudes
                                 FROM
                                    solicitudes
                                    JOIN
                                    solicitudes_val
                                    ON
                                    solicitudes.PK_ID = solicitudes_val.solicitude_id))
                            JOIN
                            (SELECT *
                             FROM vals)
                            WHERE vals.type = %s
                            HAVING COUNT(*) < %d', [tipo, size])


    ans = List(executor('SELECT portfolios
                         FROM
                            (SELECT *
                             FROM
                                portfolios
                                JOIN
                                portfolios_val)
                                ON
                                portfolios.pk_id = portfolios_val.portfolio_pk)
                            JOIN
                            cardinal
                            ON portfolios_val.val = cardinal.pk_id')
    return ans
END
```

Los atributos candidatos para el índice son los involucrados en los JOIN, y en los criterios de selección del WHERE.

El último requerimiento, por el diseño del modelo relacional, involucra la selección de valores en el portafolio dado el tipo del valor. Los parámetros de tiempo, se estiman por las solicitudes.

Los criterios seleccionados para la selección de los índices son:

- Se justifica la creación de índices en columnas que son frecuentemente filtradas, es decir comúnmente usadas en la condición del WHERE.
- No se justifica la creación de índices sobre columnas que incurran a la aplicación de funciones de agregación frecuentemente, pues esto requiere la revisión de toda la tabla (excluyendo MAX, MIN).
- Se debe crear índices sobre columnas usadas en las condiciones de los JOIN.
- Se debe crear índices sobre columnas usadas como atributos de ordenamiento (ORDER BY).
- Se debe crear índices sobre columnas que tienen pocos valores repetidos o valores únicos en la tabla.
- No se debe crear índices sobre tablas de cardinalidad baja, pues el escaneo total puede resultar más efectivo que el QUERY indexado.
- Mantener en lo posible los índices cortos, el escaneo de índices agrupadas largos puede ser más caro que la búsqueda lineal.

- Crear índices sobre columnas con alta selectividad, es decir, columnas que no tengan muchas valores duplicados.
- Intentar crear índices agrupados sobre columnas que no serán frecuentemente actualizadas, cada vez que se actualice el índice, el motor de base de datos debe mantener este índice, por lo que no es buena idea crear índices sobre muchas columnas a la vez.
- No crear índices extra sobre las mismas columnas, pues además de no afectar el rendimiento de la ejecución, es espacio perdido y memoria invertida en el mantenimiento de los índices.

Los índices seleccionados y su justificación, se exponen a continuación:

- Las llaves primarias de cada tabla: Son valores distintos y de baja actualización, además, son los fundamentales en la selección del JOIN.
- Las llaves foráneas de las tablas involucradas en el JOIN de selección: misma razón anterior.
- Las fechas, que son seleccionadas frecuentemente en los predicados de selección (WHERE), esto es siempre y cuando, la consulta no requiera la conversión de un tipo de dato, por ejemplo cadena de caracteres a fecha. Varios análisis sugieren la implementación de fechas a partir del UNIX timestamp por razones de eficiencia.
- Tipo de valor, de renta, son usados frecuentemente en las clausulas del WHERE.

Aparte de estos índices, Oracle indexa las siguientes columnas. Las tablas con columnas indexadas son las que tienen PK, esto favorece la implementación de los requerimientos por el diseño de la aplicación.

INDEX_NAME	TABLE_NAME
AUTH_GROUP_PERMISSIONS_0E9922A	AUTH_GROUP_PERMISSIONS
AUTH_GROUP_PERMISSIONS_837A862	AUTH_GROUP_PERMISSIONS
AUTH_PERMISSION_417F1B1C	AUTH_PERMISSION
AUTH_USER_GROUPS_0E939A4F	AUTH_USER_GROUPS
AUTH_USER_GROUPS_E8701AD4	AUTH_USER_GROUPS
AUTH_USER_USER_PERMISSIONS1CCA	AUTH_USER_USER_PERMISSIONS
AUTH_USER_USER_PERMISSIONS8AFE	AUTH_USER_USER_PERMISSIONS
D85D5DBB541FAB8F1DF26BA8B130BE	DJANGO_CONTENT_TYPE
DJANGO_ADMIN_LOG_417F1B1C	DJANGO_ADMIN_LOG
DJANGO_ADMIN_LOG_E8701AD4	DJANGO_ADMIN_LOG
DJANGO_SESSION_DE54FA62	DJANGO_SESSION
INDEX3	BEBEDORES
SYS_C00135604	USERS
SYS_C00135606	PASSIVES
SYS_C00135607	PASSIVES
SYS_C00135611	ACTIVES
SYS_C00135617	LEGALS
SYS_C00135618	LEGALS
SYS_C00135621	INVESTORS
SYS_C00135624	OFFERANTS

SYS_C00135632	RENTS
SYS_C00135640	VALS
SYS_C00144877	DJANGO_MIGRATIONS
SYS_C00144879	DJANGO_CONTENT_TYPE
SYS_C00144883	AUTH_PERMISSION
SYS_C00144884	AUTH_PERMISSION
SYS_C00144886	AUTH_GROUP
SYS_C00144887	AUTH_GROUP
SYS_C00144891	AUTH_GROUP_PERMISSIONS
SYS_C00144892	AUTH_GROUP_PERMISSIONS
SYS_C00144902	AUTH_USER
SYS_C00144903	AUTH_USER
SYS_C00144907	AUTH_USER_GROUPS
SYS_C00144908	AUTH_USER_GROUPS
SYS_C00144912	AUTH_USER_USER_PERMISSIONS
SYS_C00144913	AUTH_USER_USER_PERMISSIONS
SYS_C00144926	DJANGO_ADMIN_LOG
SYS_C00144931	DJANGO_SESSION
SYS_C00145022	SOLICITUDES
SYS_C00145081	SWAP_TRANSACTIONS
SYS_C00151606	PORTFOLIOS
SYS_C00151607	PORTFOLIOS_VALS
SYS_IL0000159313C00003$$	DJANGO_ADMIN_LOG
SYS_IL0000159313C00006$$	DJANGO_ADMIN_LOG
SYS_IL0000159322C00002$$	DJANGO_SESSION

Las demás tablas indexadas corresponden a tablas de administración por parte del servidor de aplicaciones usador, estas son para la administración de sesión y demás. A continuación se muestran las tablas, con sus índices, así como las sentencias para obtener tal información. Claramente, los índices mostrados corresponden a las llaves primarias de cada tabla, pues sólo se muestran tablas con llave primaria, esto es por que es un dato de baja actualización, y además de alta dispersión sobre el dominio. No ayuda si la tabla posee pocos valores. Con respecto a los requerimientos funcionales, ayuda pues las llaves primarias son altamente usadas en la selectividad de los JOINS y de los predicados del WHERE.

```
SELECT i.table_name, i.uniqueness, c.column_name
FROM   user_indexes i, user_ind_columns c
WHERE  i.index_name = c.index_name
  AND  i.table_name = UPPER('users')
ORDER  BY c.index_name, c.column_position;

SELECT i.table_name, i.uniqueness, c.column_name
FROM   user_indexes i, user_ind_columns c
WHERE  i.index_name = c.index_name
  AND  i.table_name = UPPER('passives')
ORDER  BY c.index_name, c.column_position;

SELECT i.table_name, i.uniqueness, c.column_name
FROM   user_indexes i, user_ind_columns c
WHERE  i.index_name = c.index_name
  AND  i.table_name = UPPER('actives')
ORDER  BY c.index_name, c.column_position;

SELECT i.table_name, i.uniqueness, c.column_name
FROM   user_indexes i, user_ind_columns c
WHERE  i.index_name = c.index_name
  AND  i.table_name = UPPER('legals')
ORDER  BY c.index_name, c.column_position;

SELECT i.table_name, i.uniqueness, c.column_name
FROM   user_indexes i, user_ind_columns c
WHERE  i.index_name = c.index_name
  AND  i.table_name = UPPER('investors')
ORDER  BY c.index_name, c.column_position;
```

```
SELECT i.table_name, i.uniqueness, c.column_name
FROM   user_indexes i, user_ind_columns c
WHERE  i.index_name = c.index_name
  AND  i.table_name = UPPER('offerants')
ORDER  BY c.index_name, c.column_position;

SELECT i.table_name, i.uniqueness, c.column_name
FROM   user_indexes i, user_ind_columns c
WHERE  i.index_name = c.index_name
  AND  i.table_name = UPPER('rents')
ORDER  BY c.index_name, c.column_position;

SELECT i.table_name, i.uniqueness, c.column_name
FROM   user_indexes i, user_ind_columns c
WHERE  i.index_name = c.index_name
  AND  i.table_name = UPPER('vals')
ORDER  BY c.index_name, c.column_position;

SELECT i.table_name, i.uniqueness, c.column_name
FROM   user_indexes i, user_ind_columns c
WHERE  i.index_name = c.index_name
  AND  i.table_name = UPPER('solicitudes')
ORDER  BY c.index_name, c.column_position;

SELECT i.table_name, i.uniqueness, c.column_name
FROM   user_indexes i, user_ind_columns c
WHERE  i.index_name = c.index_name
  AND  i.table_name = UPPER('SWAP_TRANSACTIONS')
ORDER  BY c.index_name, c.column_position;
```

```
SELECT i.table_name, i.uniqueness, c.column_name
FROM   user_indexes i, user_ind_columns c
WHERE  i.index_name = c.index_name
  AND  i.table_name = UPPER('PORTFOLIOS')
ORDER  BY c.index_name, c.column_position;

SELECT i.table_name, i.uniqueness, c.column_name
FROM   user_indexes i, user_ind_columns c
WHERE  i.index_name = c.index_name
  AND  i.table_name = UPPER('PORTFOLIOS_VALS')
ORDER  BY c.index_name, c.column_position;
```

TABLE_NAME	UNIQUENESS	COLUMN_NAME
USERS	UNIQUE	LOGIN

TABLE_NAME	UNIQUENESS	COLUMN_NAME
PASSIVES	UNIQUE	PASSIVE_REGISTER
PASSIVES	UNIQUE	USER_LOGIN

TABLE_NAME	UNIQUENESS	COLUMN_NAME
ACTIVES	UNIQUE	USER_LOGIN

TABLE_NAME	UNIQUENESS	COLUMN_NAME
LEGALS	UNIQUE	LEGAL_ID
LEGALS	UNIQUE	USER_LOGIN

TABLE_NAME	UNIQUENESS	COLUMN_NAME
INVESTORS	UNIQUE	USER_LOGIN

TABLE_NAME	UNIQUENESS	COLUMN_NAME
OFFERANTS	UNIQUE	USER_LOGIN

TABLE_NAME	UNIQUENESS	COLUMN_NAME
RENTS	UNIQUE	PK_ID

TABLE_NAME	UNIQUENESS	COLUMN_NAME
VALS	UNIQUE	PK_ID

TABLE_NAME	UNIQUENESS	COLUMN_NAME
SOLICITUDES	UNIQUE	PK_ID

TABLE_NAME	UNIQUENESS	COLUMN_NAME
SWAP_TRANSACTI	UNIQUE	PK_ID

Las sentencias de ejecución para cada requerimiento son:

Consultar movimientos valores (I):

```
SELECT SOLICITUDE, VAL, VAL_TYPE, CREATED_AT, ACTIVE_LOGIN, PASSIVE_REGISTER, USER_LOGIN, RENT_TYPE
FROM
    (
        SELECT SOLICITUDE, VAL, VAL_TYPE, RENT_ID, CREATED_AT, ACTIVE_LOGIN, PASSIVES.PASSIVE_REGISTER, USER_LOGIN
        FROM
            (
                SELECT SOLICITUDE, VAL, VAL_TYPE, RENT_ID, CREATED_AT, ACTIVESPASSIVES.ACTIVE_LOGIN, PASSIVE_REGISTER
                FROM
                    (
                        SELECT SOLICITUDE, VAL, VAL_TYPE, RENT_ID, CREATED_AT, ACTIVE_LOGIN
                        FROM
                            (
                                SELECT SOLICITUDE, VAL, VAL_TYPE, RENT_ID, CREATED_AT, ACTIVE_LOGIN
                                FROM
                                    (
                                        SELECT SOLICITUDE, VAL, VAL_TYPE, RENT_ID
                                        FROM
                                        (
                                            SOLICITUDES_VAL INNER JOIN VALS
                                            ON VALS.PK_ID = SOLICITUDES_VAL.VAL
                                        )
                                    )
                                SOLICITUDES_INFO INNER JOIN SOLICITUDES
                                ON SOLICITUDES_INFO.SOLICITUDE = SOLICITUDES.PK_ID
                            )
                        INFO1 INNER JOIN ACTIVES
                        ON INFO1.ACTIVE_LOGIN = ACTIVES.USER_LOGIN
                    )
                INFO2 INNER JOIN ACTIVESPASSIVES
                ON ACTIVESPASSIVES.ACTIVE_LOGIN = INFO2.ACTIVE_LOGIN
            )
        INFO3 INNER JOIN PASSIVES
        ON PASSIVES.PASSIVE_REGISTER = INFO3.PASSIVE_REGISTER
    )
INFO4 INNER JOIN RENTS
ON INFO4.RENT_ID = RENTS.PK_ID
WHERE
    val = 7617612432630759234 AND
    VAL_TYPE = '0' AND
    RENT_TYPE = '1' AND
    ACTIVE_LOGIN = 'cloWoo' AND
    (PASSIVE_REGISTER = 19193 OR USER_LOGIN = 'hanosi') AND
    CREATED_AT >= TO_TIMESTAMP('2014-11-05','yyyy-mm-dd') AND
    CREATED_AT < TO_TIMESTAMP('2014-12-07','yyyy-mm-dd');
```

Consultar movimientos valores (II):

```
SELECT SOLICITUDE, VAL, VAL_TYPE, CREATED_AT, ACTIVE_LOGIN, PASSIVE_REGISTER, USER_LOGIN, RENT_TYPE
FROM
    (
        SELECT SOLICITUDE, VAL, VAL_TYPE, RENT_ID, CREATED_AT, ACTIVE_LOGIN, PASSIVES.PASSIVE_REGISTER, USER_LOGIN
        FROM
            (
                SELECT SOLICITUDE, VAL, VAL_TYPE, RENT_ID, CREATED_AT, ACTIVESPASSIVES.ACTIVE_LOGIN, PASSIVE_REGISTER
                FROM
                    (
                        SELECT SOLICITUDE, VAL, VAL_TYPE, RENT_ID, CREATED_AT, ACTIVE_LOGIN
                        FROM
                            (
                                SELECT SOLICITUDE, VAL, VAL_TYPE, RENT_ID, CREATED_AT, ACTIVE_LOGIN
                                FROM
                                    (
                                        SELECT SOLICITUDE, VAL, VAL_TYPE, RENT_ID
                                        FROM
                                        (
                                            SOLICITUDES_VAL INNER JOIN VALS
                                            ON VALS.PK_ID = SOLICITUDES_VAL.VAL
                                        )
                                    )
                                SOLICITUDES_INFO INNER JOIN SOLICITUDES
                                ON SOLICITUDES_INFO.SOLICITUDE = SOLICITUDES.PK_ID
                            )
                        INFO1 INNER JOIN ACTIVES
                        ON INFO1.ACTIVE_LOGIN = ACTIVES.USER_LOGIN
                    )
                INFO2 INNER JOIN ACTIVESPASSIVES
                ON ACTIVESPASSIVES.ACTIVE_LOGIN = INFO2.ACTIVE_LOGIN
            )
        INFO3 INNER JOIN PASSIVES
        ON PASSIVES.PASSIVE_REGISTER = INFO3.PASSIVE_REGISTER
    )
INFO4 INNER JOIN RENTS
ON INFO4.RENT_ID = RENTS.PK_ID
WHERE
    val <> 7617612432630759234 AND
    VAL_TYPE <> '0' AND
    RENT_TYPE <> '1' AND
    ACTIVE_LOGIN <> 'cloWoo' AND
    (PASSIVE_REGISTER <> 19193 OR USER_LOGIN <> 'hanosi');
```

```
SELECT SOLICITUDE, VAL, VAL_TYPE, CREATED_AT, ACTIVE_LOGIN, PASSIVE_REGISTER, USER_LOGIN, RENT_TYPE
FROM
    (
        SELECT SOLICITUDE, VAL, VAL_TYPE, RENT_ID, CREATED_AT, ACTIVE_LOGIN, PASSIVES.PASSIVE_REGISTER, USER_LOGIN
        FROM
            (
                SELECT SOLICITUDE, VAL, VAL_TYPE, RENT_ID, CREATED_AT, ACTIVESPASSIVES.ACTIVE_LOGIN, PASSIVE_REGISTER
                FROM
                    (
                        SELECT SOLICITUDE, VAL, VAL_TYPE, RENT_ID, CREATED_AT, ACTIVE_LOGIN
                        FROM
                            (
                                SELECT SOLICITUDE, VAL, VAL_TYPE, RENT_ID, CREATED_AT, ACTIVE_LOGIN
                                FROM
                                    (
                                        SELECT SOLICITUDE, VAL, VAL_TYPE, RENT_ID
                                        FROM
                                        (
                                            SOLICITUDES_VAL INNER JOIN VALS
                                            ON VALS.PK_ID = SOLICITUDES_VAL.VAL
                                        )
                                    )
                                SOLICITUDES_INFO INNER JOIN SOLICITUDES
                                ON SOLICITUDES_INFO.SOLICITUDE = SOLICITUDES.PK_ID
                            )
                        INFO1 INNER JOIN ACTIVES
                        ON INFO1.ACTIVE_LOGIN = ACTIVES.USER_LOGIN
                    )
                INFO2 INNER JOIN ACTIVESPASSIVES
                ON ACTIVESPASSIVES.ACTIVE_LOGIN = INFO2.ACTIVE_LOGIN
            )
        INFO3 INNER JOIN PASSIVES
        ON PASSIVES.PASSIVE_REGISTER = INFO3.PASSIVE_REGISTER
    )
INFO4 INNER JOIN RENTS
ON INFO4.RENT_ID = RENTS.PK_ID
WHERE
    NOT(
    val = 7617612432630759234 AND
    VAL_TYPE = '0' AND
    RENT_TYPE = '1' AND
    ACTIVE_LOGIN = 'cloWoo' AND
    (PASSIVE_REGISTER = 19193 OR USER_LOGIN = 'hanosi'));
```

Consultar portafolio:

```
SELECT *
FROM
    (
        SELECT *
        FROM
            (
                SELECT *
                FROM
                    (
                        (
                            SELECT val, COUNT(val) AS Frequency
                            FROM SOLICITUDES_VAL
                            GROUP BY val
                            ORDER BY COUNT(val) DESC
                        )
                        FREQ INNER JOIN VALS
                        ON FREQ.val = VALS.PK_ID
                    )
                WHERE VAL_TYPE = '0' AND FREQUENCY > 1
            )
            VALORES INNER JOIN PORTFOLIOS_VALS
            ON VALORES.VAL = PORTFOLIOS_VALS.PK_VAL
    )
    PORTFOLIO_INFO INNER JOIN PORTFOLIOS
    ON PK_PORTFOLIO = PK_ID;
```

Consultar valores 2:

```
SELECT *
FROM
    (
        SELECT PK_PORTFOLIO
        FROM
        PORTFOLIOS_VALS INNER JOIN VALS
        ON PORTFOLIOS_VALS.PK_VAL = VALS.PK_ID
        WHERE PK_VAL = 9810102381089460792
    )
INFO INNER JOIN PORTFOLIOS
ON INFO.PK_PORTFOLIO = PORTFOLIOS.PK_ID;
```

Los índices creados fueron:

```
CREATE INDEX index10 ON SOLICITUDES_VAL(VAL);
CREATE INDEX index11 ON ACTIVES(USER_LOGIN);
CREATE INDEX index12 ON SOLICITUDES_VAL(SOLICITUDES);
CREATE INDEX index13 ON SOLICITUDES_VAL(VAL);
CREATE INDEX index14 ON ACTIVESPASSIVES(ACTIVE_LOGIN);
CREATE INDEX index15 ON ACTIVESPASSIVES(PASSIVE_REGISTER);
CREATE INDEX index16 ON OFFERANTS(USER_LOGIN);
CREATE INDEX index17 ON PASSIVES(PASSIVE_REGISTER);
CREATE INDEX index18 ON PASSIVES(USER_LOGIN);
CREATE INDEX index19 ON PORTFOLIOS(USER_LOGIN);
CREATE INDEX index01 ON PORTFOLIOS_VALS(PK_PORTFOLIO);
CREATE INDEX index02 ON PORTFOLIOS_VALS(PK_VAL);
CREATE INDEX index03 ON RENTS(RENT_TYPE);
CREATE INDEX index04 ON SOLICITUDES(CREATED_AT);
CREATE INDEX index05 ON SOLICITUDES(ACTIVE_LOGIN);
CREATE INDEX index06 ON SOLICITUDES_VAL(SOLICITUDE);
CREATE INDEX index07 ON SOLICITUDES_VAL(VAL);
CREATE INDEX index08 ON VALS(VAL_TYPE);
```

Para mostrar la creación, se incluye la tabla con los índices sobre la tabla SOLICITUDES.

Worksheet | Query Builder

```
SELECT i.table_name, i.uniqueness, c.column_name
FROM   user_indexes i, user_ind_columns c
WHERE  i.index_name = c.index_name
  AND  i.table_name = UPPER('SOLICITUDES')
ORDER  BY c.index_name, c.column_position;
```

Explain Plan | Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.533 seconds

	TABLE_NAME	UNIQUENESS	COLUMN_NAME
1	SOLICITUDES	NONUNIQUE	CREATED_AT
2	SOLICITUDES	NONUNIQUE	ACTIVE_LOGIN
3	SOLICITUDES	UNIQUE	PK_ID

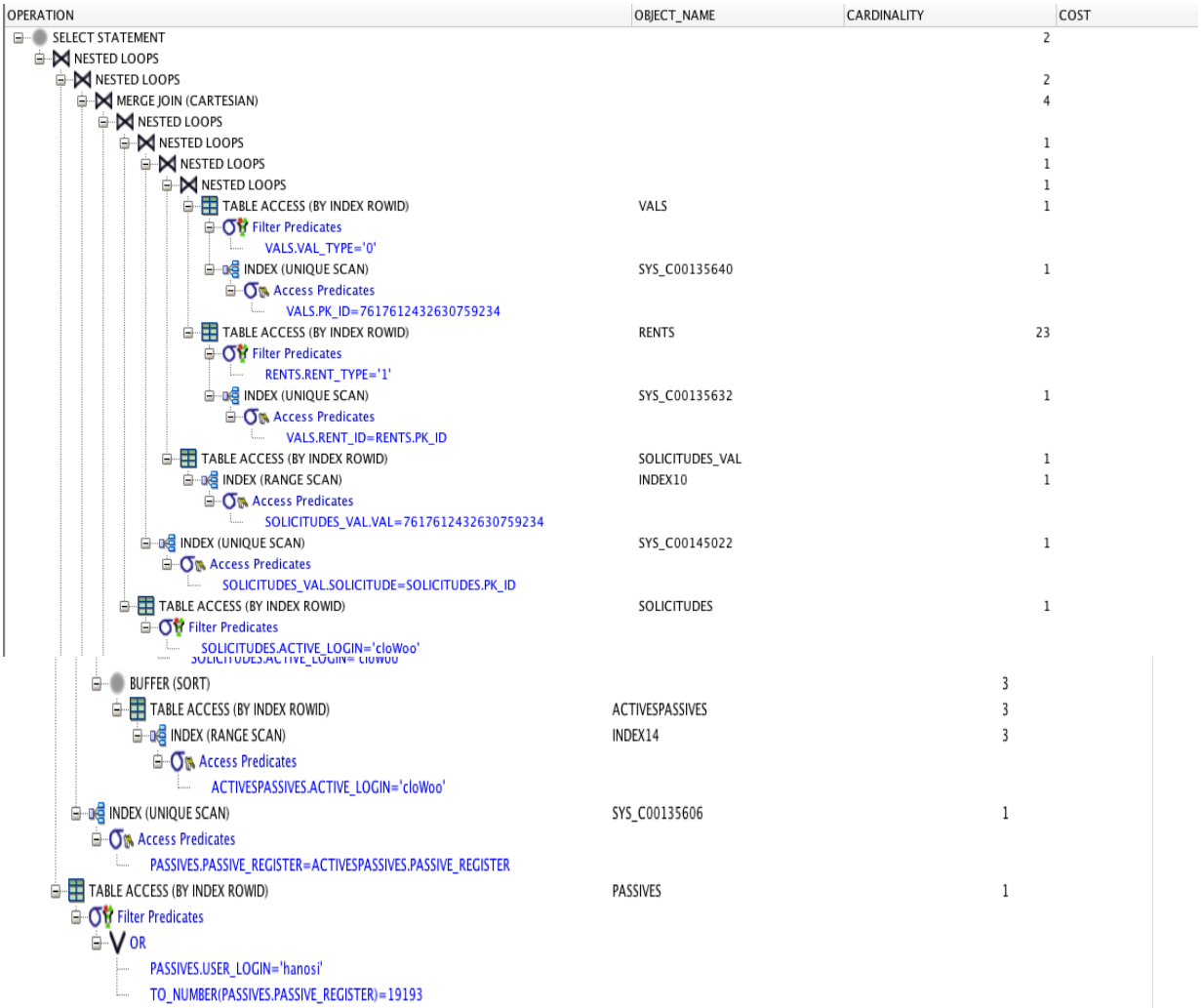
Los valores de prueba seleccionados se encuentran en las fotos de los comandos. Para llenar la base de datos, se usó un programa hecho en Python que selecciona valores aleatorios sobre grandes dominios, por lo que no hay datos diferenciadores sobre la ejecución.

El gráfico de distribución se muestra en el archivo adjunto DISTRIBUTION

Los planes de consulta por requerimiento son los siguientes:

R1:

R2:


OPERATION
OBJECT_NAME
CARDINALITY
COST
SELECT STATEMENT
NESTED LOOPS
NESTED LOOPS
MERGE JOIN (CARTESIAN)
NESTED LOOPS
NESTED LOOPS
NESTED LOOPS
NESTED LOOPS
TABLE ACCESS (BY INDEX ROWID)
VALS
Filter Predicates
VALS.VAL_TYPE='0'
INDEX (UNIQUE SCAN)
SYS_C00135640
Access Predicates
VALS.PK_ID=7617612432630759234
TABLE ACCESS (BY INDEX ROWID)
RENTS
Filter Predicates
RENTS.RENT_TYPE='1'
INDEX (UNIQUE SCAN)
SYS_C00135632
Access Predicates
VALS.RENT_ID=RENTS.PK_ID
TABLE ACCESS (BY INDEX ROWID)
SOLICITUDES_VAL
INDEX (RANGE SCAN)
INDEX10
Access Predicates
SOLICITUDES_VAL.VAL=7617612432630759234
INDEX (UNIQUE SCAN)
SYS_C00145022
Access Predicates
SOLICITUDES_VAL.SOLICITUDE=SOLICITUDES.PK_ID
TABLE ACCESS (BY INDEX ROWID)
SOLICITUDES
Filter Predicates
SOLICITUDES.ACTIVE_LOGIN='cloWoo'
BUFFER (SORT)
3
TABLE ACCESS (BY INDEX ROWID)
ACTIVESPASSIVES
3
INDEX (RANGE SCAN)
INDEX14
3
Access Predicates
ACTIVESPASSIVES.ACTIVE_LOGIN='cloWoo'
INDEX (UNIQUE SCAN)
SYS_C00135606
1
Access Predicates
PASSIVES.PASSIVE_REGISTER=ACTIVESPASSIVES.PASSIVE_REGISTER
TABLE ACCESS (BY INDEX ROWID)
PASSIVES
1
Filter Predicates
OR
PASSIVES.USER_LOGIN='hanosi'
TO_NUMBER(PASSIVES.PASSIVE_REGISTER)=19193

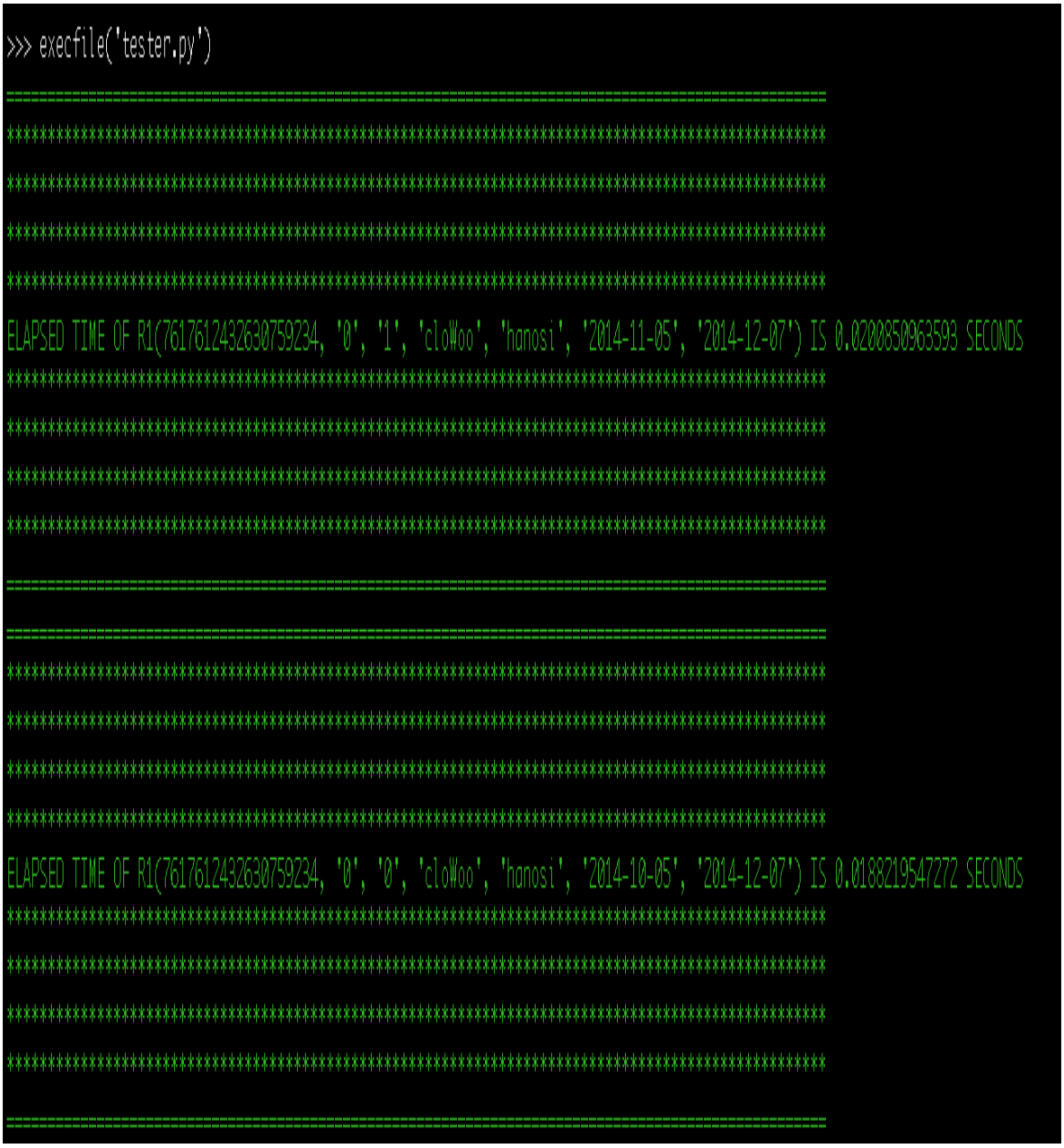
R3:

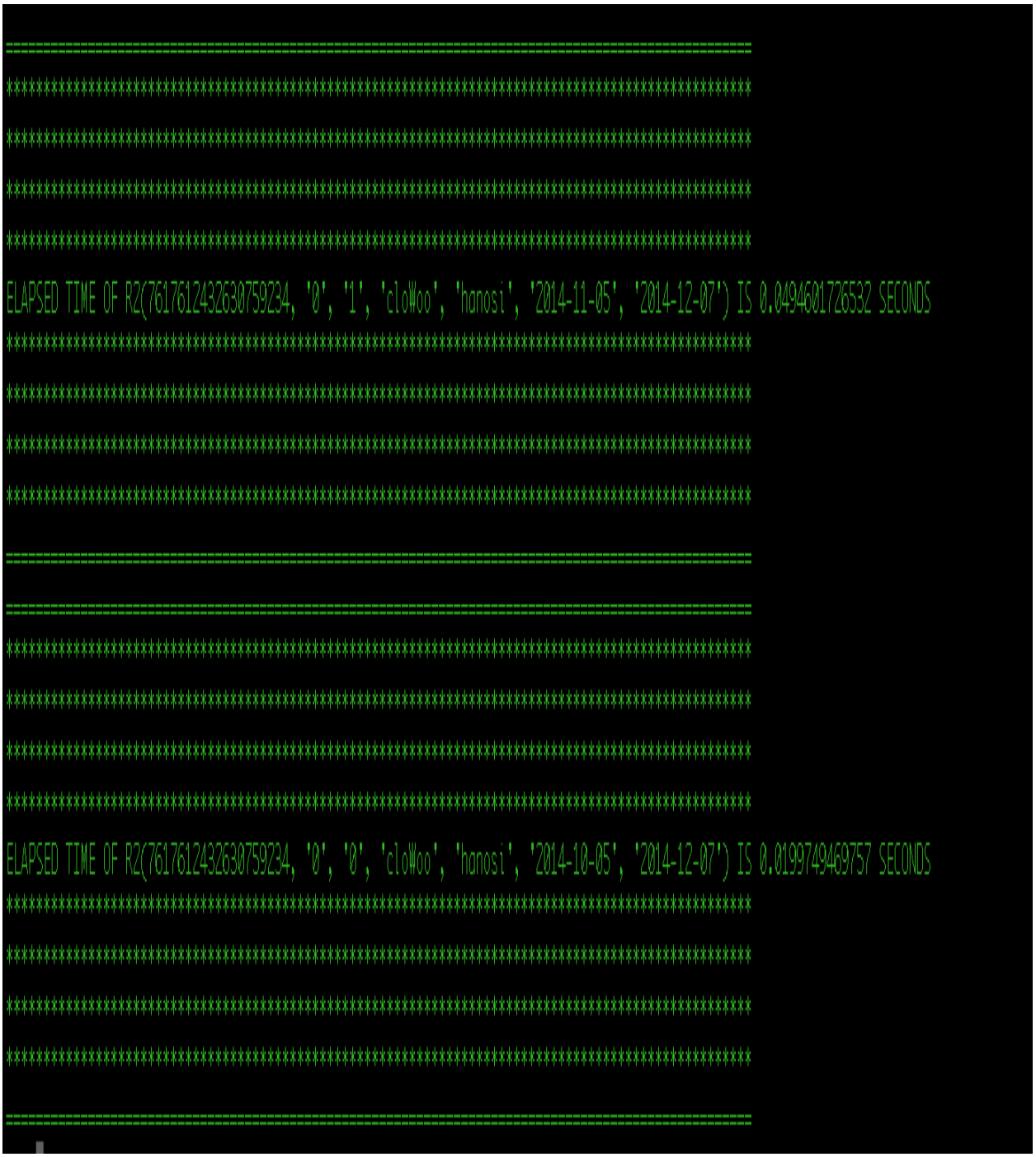
OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1523	12201
NESTED LOOPS			
NESTED LOOPS		1523	12201
NESTED LOOPS		1523	12201
NESTED LOOPS		1523	3060
VIEW		3053	6
SORT (ORDER BY)		3053	6
FILTER			
Filter Predicates			
COUNT(VAL)			
HASH (GROUP BY		3053	6
TABLE ACCES	SOLICITUDES_VAL	3053	4
TABLE ACCESS (BY INDEX RO	VALS	1	1
Filter Predicates			
VALS.VAL_TYPE='0'			
INDEX (UNIQUE SCAN)	SYS_C00135640	1	0
Access Predicates			
FREQ.VAL=VALS.			
TABLE ACCESS (BY INDEX ROWID	PORTFOLIOS_VALS	1	6
INDEX (RANGE SCAN)	INDEX02	17	0
Access Predicates			
FREQ.VAL=PORTFOL			
INDEX (UNIQUE SCAN)	SYS_C00151606	1	0
Access Predicates			
PORTFOLIOS_VALS.PK_PORT			
TABLE ACCESS (BY INDEX ROWID)	PORTFOLIOS	1	0

R4:

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		2	2
NESTED LOOPS			
NESTED LOOPS		2	2
TABLE ACCESS (BY INDEX ROWID)	PORTFOLIOS_VALS	2	2
INDEX (RANGE SCAN)	INDEX02	2	1
Access Predicates			
PORTFOLIOS_VALS.PK_V			
INDEX (UNIQUE SCAN)	SYS_C00151606	1	0
Access Predicates			
PORTFOLIOS_VALS.PK_PORT			
TABLE ACCESS (BY INDEX ROWID)	PORTFOLIOS	1	0

Los tiempos de ejecución son:




ELAPSED TIME OF RZ(7617612432630759234, '0', '1', 'clowoo', 'honosi', '2014-11-05', '2014-12-07') IS 0.0494601726532 SECONDS
ELAPSED TIME OF RZ(7617612432630759234, '0', '0', 'clowoo', 'honosi', '2014-10-05', '2014-12-07') IS 0.0199749469757 SECONDS

```
**********************************************************************
**********************************************************************
ELAPSED TIME OF R3('0', '1') IS 0.0213899612427 SECONDS
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************

======================================================================
======================================================================
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
ELAPSED TIME OF R3('1', '4') IS 0.0242450237274 SECONDS
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************

======================================================================
======================================================================
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
ELAPSED TIME OF R3('1', '4') IS 0.0205900669098 SECONDS
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************

======================================================================
======================================================================
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
ELAPSED TIME OF R3('0', '889') IS 0.019159078598 SECONDS
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************

======================================================================
```

R1:

Alta selectividad por naturaleza del predicado.
Realizar JOIN usando loops anidados, y seleccionar usando IF en comparación, el de Oracle usa el INDEX.

R2:

Igual que R1

R3:

Loop anidado y selección sobre datos de índice, alta selectividad por cardinalidad de las tablas que intervienen.

R4:

Loop anidado y selección sobre datos de índice, alta selectividad por cardinalidad de las tablas que intervienen.

```
foreach tuple r ∈ R do
    foreach tuple s ∈ S do
        if r_i==s_j then add ⟨r, s⟩ to result
```

```
foreach tuple r ∈ R do
    foreach tuple s ∈ S do
        if r_i==s_j then add ⟨r, s⟩ to result
```

Implementación.

Se expone ahora de manera general un análisis comparativo de la discriminación de datos por parte de Oracle, y la discriminación de datos por parte del programador. El esquema de datos propuesto es el siguiente:

Entrada

Titulo	**Contenido**	**Fecha**
El cáncer de mama	El cáncer de mama es malo, muy muy malo	24-10-2014
Los atracos de transmilenio	Los atracos de transmilenio son malos, muy muy malos.	22-10-2014
Las drogas ilegales	Las drogas ilegales son malas, muy muy malas.	20-10-2014
Viajando por el mundo	Viajar por el mundo es bueno, muy muy bueno	03-11-2014

Comentario

Contenido	**Entrada**	**Fecha**
De acuerdo	El cáncer de mama	27-10-2014
No de acuerdo	Los atracos de transmilenio	24-10-2014
Muy en contra	El cáncer de mama	01-11-2014
Mal mal mal	Viajando por el mundo	06-11-2014

Operaciones

- **Selección (SELECT * FROM Entrada WHERE Fecha > to_date('15-10-2014', 'DD-MM-YYYY')**

Desarrollador:

Proceso:

Para un desarrollador dando uso de un lenguaje de programación, la siguiente operación se realizaría con un recorrido sobre la tabla comparando los valores respectivos de fecha para saber si cumplen lo requerido y por ende son agregados al resultado.

Complejidad:

O(n) Donde n es la cantidad de datos en la tabla, esto es porque debe recorrer todos los elementos de la tabla al menos una vez.

Base de datos:

Proceso:

Para una base de datos, es posible realizar la creación de un índice en las fechas por lo que la cantidad de datos a recorrer disminuiría y de esta forma comparar cada fecha para saber si cumple el parámetro requerido.

Complejidad:

O(m) Donde m es la cantidad de distintas fechas en la tabla, la cual será considerablemente menor a la cantidad de datos en gran escala.

- **Join (SELECT * FROM (Entrada INNER JOIN Comentario ON Titulo = Entrada) WHERE Entrada.Fecha > to_date(26-10-2014))**

Desarrollador:

Proceso:

Para un desarrollador, es posible realizar esta operación mediante bucles anidados, en donde para cada dato válido encontrado en la tabla de Entrada, se realiza un recorrido sobre la tabla Comentario para obtener los datos correspondientes a esta tupla.

Complejidad:

O(nm) Donde n es la cantidad de datos en la primera tabla y m es la cantidad de datos en la segunda tabla.

Base de datos:

Proceso:

Busqueda lineal

Complejidad:

O(mnlog(mn))