SISTRANS Caso de estudio: Entrega 2

d.trivino, i.bermudezl, je.lopezu1

Noviembre del 2023

1. Integrantes

Juan Esteban Lopez Ulloa - 202021417 Issac David Bermudez Lara - 202014146 Daniel Felipe Triviño Santana - 202113218

2. Entrega del commit

Enlace al repositorio: https://github.com/SISTRANS-IDJ/HotelAndes/tree/repair **La rama repair contiene el último commit de la entrega**

SHA del último commit: 8b48c0144bd3fd697a470170f60765f1e478a980

Análisis

Para las modificaciones planteadas en el enunciado de esta iteración no se tuvo mayores inconvenientes. Gracias a un buen planteamiento de la base de datos en la en la primera iteración los únicos cambios que se debieron realizar fue actualizaciones de los tipos de datos para mejor manejo de las fechas (sobre todo para los requerimientos de los servicios y otros que dependen de las fechas). Por otra parte se debieron añadir unas entidades que si bien se habían tenido en cuenta en la primera iteración, no se habían implementado ya que no se vio la necesidad basándonos exclusivamente en los requerimientos; este fue el caso de cuenta de un cliente. Finalmente se realizar unos cambios en los atributos de las tablas relacionales los cuales se repetían entre relaciones porque no se había revisado minuciosamente este aspecto.

Diseño de la aplicación

2.1. Índices

RFC1 - MOSTRAR EL DINERO RECOLECTADO POR SERVICIOS EN CADA HABITACIÓN EN EL ÚLTIMO AÑO CORRIDO

Para este requerimiento se decidió crear un índice sobre la fecha de la relación consumo de cuenta (accountConsumption) y las reservas de habitaciones. Esta decocción se toma amiento que es un proyecto a largo plazo y se tendrán muchos años en la base de datos por lo que a través de tiempo la selectividad va a ir aumentando. Además como las fechas no son un dato cambiante y el requerimiento pide una consulta sobre un rago se puede concluir que la mejor opción para este caso es la estructura del árbol B+.

RFC2 - MOSTRAR LOS 20 SERVICIOS MÁS POPULARES

Para el segundo requerimiento se crea un índice con estructura Hashing ya que el criterio de agrupación (COUNT) es el id del servicio; como el manejador de bases de datos ya lo genera no es necesario implementarlo.

RFC3 - MOSTRAR EL ÍNDICE DE OCUPACIÓN DE CADA UNA DE LAS HABITACIONES DEL HOTEL

En este requerimiento lo mas importante es el id de la habitación del hotel, como el id de las habitaciones ya es una llave primaria no hay que generar un índice.

RFC4 - MOSTRAR LOS SERVICIOS QUE CUMPLEN CON CIERTA CARACTERÍSTICA

RFC5 - MOSTRAR EL CONSUMO EN HOTELANDES POR UN USUARIO DADO, EN UN RANGO DE FECHAS INDICADO

Se realizó un índice sobre las fechas de los consumos para poder, a través de la estructura B+, generar un rango de fechas en las que se pueda revisar. Además de esto como se pide sobre un cliente en especifico se necesita otro atributo, es decir un índice secundario, para filtrar con mayor rapidez en la relación de consumos el cliente que se necesita. Como solo se requiere un cliente la selectividad es alta y esta es otra razón por la que se debe generar el índice. Sin embargo como esto es una llave foránea, esta ya tiene un índice asociado por lo que no es necesario implementarlo.

RFC6 - ANALIZAR LA OPERACIÓN DE HOTELANDES

Como es una consulta de operaciones (`MAX`,`MIN`) pero sin ninguna restricción (`WHERE`, `GROUPBY`), es buena idea usar un índice.

RFC7 - ENCONTRAR LOS BUENOS CLIENTES

Como la agrupación y las sumas que se deben realizar para este requerimiento son sobre llaves primarias, o en su defecto llaves foráneas, el manejador de base de datos ya realiza estos índices de forma automática por lo que no hay que generar mas índices.

RFC8 - ENCONTRAR LOS SERVICIOS QUE NO TIENEN MUCHA DEMANDA

Similar al punto 2, como el manejador de bases de datos ya genera los índices para llaves primarias y foráneas, no es necesario implementarlos ya que en este requeriros funcional solo se utilizan esos valores.

RFC9 - CONSULTAR CONSUMO EN HOTELANDES

Como en esta sentencia se realizaran 2 WHERE's para filtrar registros no es necesario utilizar índices mas allá de los proporcionados automáticamente por el manejador de bases de datos.

RFC10 - CONSULTAR CONSUMO EN HOTELANDES V2

Al necesitar hacer una consulta en un rango de fechas especifico es buena idea hacer un índice sobre las fechas con la estructura de arbol B+.

RFC11 - CONSULTAR FUNCIONAMIENTO

Esto se puede hacer con un índice sobre las fechas de ambas tablas; en este índice se manejará la estructura de árbol B+ debido a que es la mas eficiente para la búsqueda en rangos.

RFC12 - CONSULTAR LOS CLIENTES EXCELENTES

Al realizar consultas separadas y luego filtrar por id, no se necesita ningún

ÍNDICES ENCONTRADOS

	INDEX_NAME	TABLE_NAME	COLUMN_NAME
1	SYS_C001196168	CONSUMPTION_PLAN	ID
2	SYS_C001196171	HOTELANDES_USER_ROLE	ID
3	SYS_C001196179	HOTELANDES_USER	ID
4	SYS_C001196180	HOTELANDES_USER	EMAIL
5	SYS_C001196182	HOTELANDES_CLIENT	ID
6	SYS_C001196188	HOTEL_ROOM_TYPE	ID
7	SYS_C001196192	HOTEL_ROOM	ID
8	SYS_C001196193	HOTEL_ROOM	NAME
9	SYS_C001196199	HOTEL_ROOM_BOOKING	ID
10	SYS_C001196209	HOTELANDES_CLIENT_ACCOUNT	ID
11	SYS_C001196216	HOTELANDES_SERVICE	ID
12	SYS_C001196223	HOTELANDES_ACCOUNT_CONSUMPTION	ID
13	SYS_C001196230	PRODUCT	ID
14	SYS_C001196237	SERVICE_BOOKING	ID

Figura 1: Índices encontrados

ANÁLISIS DE ÍNDICES ENCONTRADOS

Se encontró que los índices generados por el manejador de bases de datos Oracle SQLDeveloper son las llaves primarias y además las columnas con campos únicos. Esto evidentemente ayuda a disminuir el tiempo de consultas ya que la mayoría de las consultas se realizan por la llave primaria, por lo que al tener un índice en ella ayuda a ubicar mas rápido los registros. Pueden existir excepciones como consultas con WHERE sobre otros campos diferentes al de los índices pero generalmente mejoran el rendimiento de las consultas.

2.2. Diseño de consultas para Requerimientos Funcionales

RFC1 - MOSTRAR EL DINERO RECOLECTADO POR SERVICIOS EN CADA HABITACIÓN EN EL ÚLTIMO AÑO CORRIDO

Para esto debemos hacer un JOIN entre las tablas de habitaciones, reservas, usuarios, consumos y servicios. Aquí debemos filtrar las filas por las fechas que incluyen el último año. A partir de esta relación, debemos hacer una suma agrupando por el identificador de la habitación. Una posible dificultad de este requerimiento es la alta cantidad de tablas que hay que unir, por lo que el uso de índices puede facilitar esta consulta.

RFC2 - MOSTRAR LOS 20 SERVICIOS MÁS POPULARES

Lo que se realizará es una consulta SQL en la que se retorne una lista de los id de los servicios y el conteo de las veces que estos han sido reservados. Esto se hace mediante una función COUNT sobre el id de los servicios; de esta forma se sabrá la cantidad de veces que se reservo un servicio.

```sql
-- consulta 1.1
SELECT sbs.service_id, COUNT(sbs.id) AS cantidad_de_reservas
FROM service_booking sbs
GROUP BY sbs.service_id
ORDER BY cantidad_de_reservas DESC
FETCH FIRST 5 ROWS ONLY;
```

RFC3 - MOSTRAR EL ÍNDICE DE OCUPACIÓN DE CADA UNA DE LAS HABITACIONES DEL HOTEL

Para realizar el calculo del índice de ocupación, es necesario una consulta usando la función agregada de `COUNT` por habitación (`GROUP BY`). Y dividiendo entre 365 cada resultado.

Tenemos que tener en cuenta que una habitación solamente esta disponible para reserva en un día, es decir que no puedo reservar una habitación para dos franjas horarias en el mismo día.

Y usualmente la hora de check-in es de la 13:00 hrs y de check-out es a las 12:00 hrs.

```sql
SELECT r.id                                                AS room_id,
       SUM(trunc(b.check_out) - trunc(b.check_in)) / 365 AS occupancy
FROM hotel_room_booking b
         JOIN hotel_room r ON b.hotel_room_id = r.id
GROUP BY r.id;
```

RFC4 - MOSTRAR LOS SERVICIOS QUE CUMPLEN CON CIERTA CARACTERÍSTICA

En la UI se puede hacer un dashboard con selectores para las diferentes carácterísticas que puede tener un servicio: Estas características son :

- Rango de precios
- Tipo o categoría
- Fecha de Consumo
- Empleado que lo registro

Para solucionarlo se va a crear una consulta dentro de uno de los servicios del backend que permita agregar las condiciones requeridas al WHERE de la sentencia SQL según se necesite.

RFC5 - MOSTRAR EL CONSUMO EN HOTELANDES POR UN USUARIO DADO, EN UN RANGO DE FECHAS INDICADO

Se debe revisar sobre el rango de fechas dado, los consumo que ha realizado el cliente que ha ingresado por parámetro. Con esto ya se puede hacer la suma de los valores de los consumos para al final retornar el valor.

```sql
SELECT SUM(COST) AS suma_consumos
FROM HOTELANDES_ACCOUNT_CONSUMPTION
WHERE cliente_id = 'ID_DEL_CLIENTE'
    AND fecha >= 'FECHA_DE_INICIO'
```

```
    AND fecha <= 'FECHA_DE_FIN';
```

RFC6 - ANALIZAR LA OPERACIÓN DE HOTELANDES

Para este requerimiento lo que debemos hacer es primero calcular 3 agregados, uno es sobre la ocupación (Mayor/Menor) y ganancias.

Esto lo debemos agrupar por un día desde que se tiene historia, y nos debería votar el agregado junto a la fecha

En resumen seria un `COUNT` sobre el agregado seguido de un `MAX/MIN`.

```
WITH date_range AS (
    SELECT
        MIN(trunc(b.check_in))  AS min_start_date,
        MAX(trunc(b.check_out)) AS max_end_date
    FROM
        hotel_room_booking b
), operation_dates AS (
    SELECT
        TO_DATE(to_char(min_start_date + level - 1, 'YYYY-MM-DD'),
                'YYYY-MM-DD') AS date_within_range
    FROM
        date_range
    CONNECT BY
        level <= ( max_end_date - min_start_date ) + 1
), room_occupancy AS (
    SELECT
        d.date_within_range "date",
        COUNT(*) AS occupancy
    FROM
        operation_dates    d
        LEFT JOIN hotel_room_booking b ON d.date_within_range BETWEEN
    trunc(b.check_in) AND trunc(b.check_out)
    WHERE
        b.client_id IS NOT NULL
    GROUP BY
        d.date_within_range
), room_occupancy_min AS (
    SELECT
        MIN(o.occupancy) AS min_occupancy
    FROM
        room_occupancy o
)
SELECT
    o."date",
    occupancy AS MIN_OCCUPANCY
FROM
    room_occupancy  o WHERE o.occupancy = (SELECT min_occupancy from
    room_occupancy_min);

WITH date_range AS (
    SELECT
```

```
        MIN(trunc(b.check_in))  AS min_start_date,
        MAX(trunc(b.check_out)) AS max_end_date
    FROM
        hotel_room_booking b
), operation_dates AS (
    SELECT
        TO_DATE(to_char(min_start_date + level - 1, 'YYYY-MM-DD'),
                'YYYY-MM-DD') AS date_within_range
    FROM
        date_range
    CONNECT BY
        level <= ( max_end_date - min_start_date ) + 1
), room_occupancy AS (
    SELECT
        d.date_within_range "date",
        COUNT(*) AS occupancy
    FROM
        operation_dates    d
        LEFT JOIN hotel_room_booking b ON d.date_within_range BETWEEN
    trunc(b.check_in) AND trunc(b.check_out)
    WHERE
        b.client_id IS NOT NULL
    GROUP BY
        d.date_within_range
), room_occupancy_max AS (
    SELECT
        MAX(o.occupancy) AS max_occupancy
    FROM
        room_occupancy o
)
SELECT
    o."date",
    occupancy AS max_occupancy
FROM
    room_occupancy  o WHERE o.occupancy = (SELECT max_occupancy from
    room_occupancy_max);
```

RFC7 - ENCONTRAR LOS BUENOS CLIENTES

Un cliente es bueno si ha acumulado más de dos semanas de estadía en el hotel en todas sus estadías o si ha consumido más de 15000000 en el último año de funcionamiento de Hotelandes. Para verificar la segunda condición se debe hacer una consulta que agrupe y sume todos los servicios consumidos por cada cliente, y filtrar con el piso de 15 millones. Para la otra consulta se debe realizar lo mismo extrayendo el tiempo entre el checkin y el checkout de cada reserva de cada cliente .

RFC8 - ENCONTRAR LOS SERVICIOS QUE NO TIENEN MUCHA DEMANDA

Parecido al requerimiento 2, lo que se realizara una consulta en la que se obtiene los id de los servicios y se mostrarán los servicios que menos se reserven. La lógica es la misma, se realiza un COUNT sobre el id de los servicios, lo único es que no se retorna los servicios mas reservados

sino lo que menos ser reservan.

```sql
-- consulta 1.1
SELECT sbs.service_id, COUNT(sbs.id) AS cantidad_de_reservas
FROM service_booking sbs
GROUP BY sbs.service_id
ORDER BY cantidad_de_reservas
FETCH FIRST 5 ROWS ONLY;
```

RFC9 - CONSULTAR CONSUMO EN HOTELANDES

La entrada principal del requerimiento es nombre del servicio junto al rango de fechas.

Para esto lo que se debe hacer es un filtro con respecto al rango con un `WHERE`. Y otro filtro con respecto al servicio, pero este puede ser aplicado con un `JOIN` o un `WHERE` dependiendo de como se desarrolle la consulta.

Como parámetros opcionales se nos indica que deberíamos hacerlo con los datos del cliente y el numero de veces que se ha utilizado el servicio.

Para los datos del cliente primero podríamos hacer una pre-consulta con IDs de los clientes que cumplen el criterio, para luego ser unida (`JOIN`) con la consulta global. Y el numero de veces es filtrar por el agregado de utilizacion de servicio.

RFC10 - CONSULTAR CONSUMO EN HOTELANDES V2

Para encontrar los clientes que nunca consumieron un servicio específico en un rango de fechas hay que:

- Filtrar todos los consumos por el rango de fechas
- Obtener los clientes que están presentes en este subconjunto
- Hallar la diferencia entre el conjunto de todos los clientes y el conjunto obtenido previamente
- Agrupar y ordenar por los criterios mencionados abajo, según lo indique el usuario de la aplicación en la UI.

Los criterios de clasificación para ordenamiento y agrupación son:

- Datos del cliente
- Fecha
- Servicio

RFC11 - CONSULTAR FUNCIONAMIENTO

En este caso se realizara un filtro por fecha en la tabla de reserva de habitaciones y reserva de servicios, de esta forma se obtendrán las habitaciones mas/menos utilizadas y los servicios mas/menos utilizados.

RFC12 - CONSULTAR LOS CLIENTES EXCELENTES

Podríamos realizar 3 consultas por cada criterio, y unirlas con `UNION`, esta consulta nos daría los IDs de los clientes que queremos. Luego a partir de los IDs ya filtrados obtenemos la información que se nos pide.

3. Cargue Masivo de datos

3.1. Diseño de Datos

Para el diseño de los datos se optó por las siguientes decisiones:

- Hacer uso de la misma aplicación para implementar métodos de generación y carga de datos masiva dentro de los servicios de spring. Al generar los datos dentro de la aplicación garantizamos consistencia en los datos insertados en las tablas ya que están pasando a través de las restricciones de la apliación, en vez de entrar directamente a SQL developer.
- Hacer uso de la libreria Data Faker de Java para crear datos aleatorios y únicos que representen elementos de la vida real (nombres, identificaciones, correos, dominios)
- Las entidades con datos predeterminados por el enunciado del curso como los tipos de usuario, los servicios, y los tipos de habitación se limitaron a estos datos recomendados y no se les hizo una carga masiva
- Se cargo a la base de datos un volumen de más de 700 000 datos para las tablas en las que tiene sentido (usuarios, clientes, reservas, compras).

3.2. Documentación de la carga de datos

Para la carga de datos realizamos los siguientes pasos:

1. Se creó un servicio para cada entidad del dominio llamada *entidadGenerator* la cual hace uso de la librería Faker y de los servicios CRUD previamente creados para generar volúmenes de datos
2. Un servicio maestro llamado *DataGeneratorService* agrupa todos estos generadores y otras dependencias necesarias como repositorios JDBC para invocar la generación de los datos y su posterior inserción a la base de datos Oracle por medio de queries.
3. Estas invocaciones se realizaron en un orden tal que se generaran e insertaran primero las entidades que no tienen dependencias (llaves foráneas) y progresivamente se generaban las entidades con más dependencias de otras entidades. Esto permite la integridad de los datos y evita errores al momento de insertar las filas en la DB.
4. Se creó una clase de tipo controlador que expusiera unos endpoints de API REST para poder invocar la creación y eliminación de las tablas y la inserción de los datos con un *request* HTTP. Esto permitiría hacer pruebas sin depender del FrontEnd de la aplicación y además hacerlas desde la herramienta Postman.

4. Construcción de la aplicación, ejecución de pruebas y análisis de resultados

Para esta entrega el equipo de trabajo logró completar con éxito la carga masiva de datos con una buena calidad en estos y manteniendo la integridad de las tablas. Sin embargo no se lograrón completar las consultas para todos los requerimientos funcionales ni la implementación del Front End de la apliación web. Se hicieron multiples correcciones sobre los requerimientos de la entrega 1 del proyecto, tanto en el esquema de la base de datos como en algunos métodos de las entidades en el Back End que permitieron la inserción automatizada de datos masivos.