Tarea 3

Fundamentos de Bases de Datos – Laboratorio 2023
29 de setiembre de 2023

Objetivos

- Experimentar en el desarrollo de **procedimientos almacenados**[1] para resolver problemas sencillos directamente en la base de datos.
- Experimentar en el uso de ***triggers***[2] como mecanismo de control del cumplimiento de restricciones de integridad, diferir tareas sobre la base de datos, etc.

Formato de entrega

- Cada script debe almacenarse en un archivo con extensión ***.sql***. En cada una de las partes se especificará el nombre que debe tener el archivo. Cada uno de estos archivos solo debe contener una consulta.
- Se debe entregar cada archivo ***.sql*** por separado. En esta entrega no se aceptarán archivos comprimidos.
- Se debe entregar un archivo de nombre ***grupo.txt*** conteniendo el número del grupo y los datos de cada uno de los integrantes.

Fecha de entrega

Cada grupo tendrá tiempo para entregar la tarea hasta el **martes 17/10/2023** a las 23:59 en el eva del curso.

Consideraciones generales

- No utilizar vistas.
- No utilizar secuencias salvo en los casos en que se pida explícitamente.
- En todas las funciones y/o procedimientos a definir utilizar el lenguaje PL/pgSQL[3]. No es necesario que defina el lenguaje, ya lo tiene disponible.
- Respetar todos los nombres dados, para secuencias, atributos, tablas, triggers, funciones, etc.
- Respetar el orden dado de los parámetros en las funciones y/o procedimientos.
- No definir triggers más allá de los que se piden explícitamente.
- No realizar DROP/TRUNCATE de ningún elemento de la base.
- No usar RAISE EXCEPTION para el manejo de los errores. Utilice RAISE NOTICE con este fin.
- No incluya sentencias que expliciten el usuario propietario de los elementos de la base en el código que entrega. Por ejemplo, evite sentencias del estilo “ALTER TABLE <nombretabla> OWNER TO <usuario>”.

1 https://www.postgresql.org/docs/current/sql-createfunction.html
https://www.postgresql.org/docs/current/sql-createprocedure.html

2 https://www.postgresql.org/docs/current/triggers.html

3 https://www.postgresql.org/docs/current/plpgsql.html

Introducción

En esta tarea, se trabajará sobre la base de datos que en la Tarea 2 se llamó “Sistema anterior”, cuyo MER y modelo relacional se encuentran en la sección 1 de la letra de dicha tarea. El esquema y datos son también aquellos de dicha tarea.

En lo que sigue, toda referencia a reservas y estadías corresponde a lo representado por las tablas ***reservas_anteriores*** y ***estadias_anteriores***.

Uso de procedimientos almacenados

1) Actividad del cliente

Es necesario conocer la actividad de los clientes con respecto a las reservas y a las estadías. Dado un cliente y un año, interesa determinar la cantidad de reservas y la cantidad de estadías de ese cliente en el año dado.

Se pide: Implementar una función que permita calcular la actividad del cliente. En la siguiente tabla se especifica la función:

Nombre	**actividad_cliente**
Parámetros	codigo **char(1)**, clientedoc **integer**, anio **integer**
Salida	**integer**
Acción	Si el código es 'R' o 'r' calcula la cantidad de reservas realizadas por el cliente con fecha de reserva en ese año. Si el código es 'E' o 'e' calcula la cantidad de estadías que tiene el cliente con fecha de check_in en ese año.
Observaciones	• Controlar que el documento del cliente existe en la tabla *clientes*. En caso contrario mostrar el mensaje 1. • Controlar que el código de operación es correcto. En caso contrario mostrar el mensaje 2. • Controlar el documento del cliente y, si existe, controlar el código operación. No es necesario hacer todos los controles; donde uno no se cumpla se informa con el mensaje correspondiente y se termina la funciòn.
Mensajes de error	1. No existe el cliente 2. Código de operación incorrecto

Se debe entregar: un archivo llamado **parte01.sql** que contenga la sentencia de creación de la función necesaria para resolver esta parte.

2) Ingresos espontáneos por tipo de habitación para un hotel

Se ha detectado que existen estadías que se concretan sin reserva previa. A este tipo de estadías se las llama “espontáneas” y generan ingresos que no estaban previstos. Es de interés para los hoteles conocer el monto correspondiente a estos ingresos inesperados, discriminados por tipo de habitación. El "monto de ingresos" se define como lo facturado y NO como el "márgen de ganancia". Esto es, no se debe descontar el costo de mantenimiento.

Se pide: Implementar una función que, dado un hotel, permita obtener para cada tipo de habitación el monto correspondiente al ingreso generado por estadías espontáneas.

En la siguiente tabla se especifica la función:

Nombre	**ingreso_extra**
Parámetros	codhotel **integer, out** tipohab **smallint, out** monto **numeric(8,2)**
Salida	**setof record**
Acción	Recibe un código de hotel y devuelve una tabla donde para cada tipo de habitación que tenga estadías espontáneas asociadas se muestra el monto de los ingresos que obtuvo el hotel.
Observaciones	• Una estadía se considera espontánea si no existe una reserva previa asociada con igual fecha de check_in. • La duración de la estadía se calcula como (fecha check_out – fecha check_in). • El costo y precio correspondientes a una estadía son los vigentes al momento del inicio (check_in) de la estadía. • Una estadía espontánea en una fecha donde la habitación no tiene precio se ignora para el cálculo del monto.

Se debe entregar: un archivo llamado **parte02.sql** que contenga la sentencia de creación de la función necesaria para resolver esta parte.

3) *Resumen de ingresos extras*

La gerencia solicita un reporte por país que resuma la información del ingreso extra generado por las estadías espontáneas. Interesa que el resumen incluya todos los países y categorías de hoteles (según cantidad de estrellas) registrados en el sistema. En caso de que para un país no existan hoteles de una categoría dada o no existan estadías espontáneas bajo esa categoría, el valor en la columna “total_extra” debe ser 0.

La información debe de almacenarse en la siguiente tabla:

resumen (pais_codigo, cant_estrellas, total_extra), detallada a continuación:

Atributo	Descripción	Tipo
pais_codigo	Código del país	Igual que en la tabla *paises*.
cant_estrellas	Calificación en estrellas de los hoteles	Igual que estrellas en la tabla *hoteles*
total_extra	Suma de las extras para los hoteles de ese pais y de esa cantidad de estrellas	numeric (10,2)

Se pide: desarrollar una función o procedimiento almacenado con las siguientes características que implemente este reporte.

Nombre	**generar_reporte**
Parámetros	Sin parámetros.
Salida	**void**
Acción	Carga la tabla *resumen* con los datos especificados.
Observaciones	Asumir que la tabla resumen se encuentra creada y vacía al momento de iniciar el procedimiento.

Se debe entregar: un archivo llamado **parte03.sql** que contenga las sentencias de creación de la tabla de resumen y de la función o procedimiento *generar_reporte*. Se debe utilizar la función definida en la parte 2 pero **NO incluir la definición de la misma en este archivo**.

Uso de triggers

4) *Control en las fechas de costos de habitaciones*

Las estadías que se ingresan al sistema deben tener un costo asociado.

Se pide: Implementar un trigger que solo permita borrar o actualizar costos de habitaciones si dicha acción no deja estadías sin costo asociado. Para esto se debe controlar la fecha de check_in de las estadías en la habitación y la fecha_desde asociada a el precio y costo de mantenimiento de la habitación.

En la siguiente tabla se resumen las características del trigger a implementar:

Nombre del trigger	**control_costos**
Tabla	*costos_habitacion*
Acción	Sólo se permite el borrado o actualización de tuplas de la tabla *costos_habitacion* si el borrado o actualización de las mismas no provocan que una o más estadías se vean afectadas de forma de que no quede un costo asociado a la habitación. En caso de ser necesario, muestra el mensaje de error correspondiente.
Mensajes de error	1. La operación de borrado no es correcta 2. La actualización no es correcta

Se debe entregar: un archivo llamado **parte04.sql** que contenga las sentencias de creación del trigger y su función o procedimiento asociado, necesarios para resolver esta parte.

5) *Auditoría sobre el uso de la base por parte de los usuarios*

Interesa llevar registro sobre la cantidad de operaciones que realiza cada usuario de la base de datos sobre las tablas *estadias_anteriores*, *reservas_anteriores* y *clientes* en cada día. Para esto se maneja la siguiente tabla, donde ninguno de los atributos permite valores nulos:

registro_uso(usuario, tabla, fecha, cantidad) donde:

Atributo	**Descripción**	**Tipo**
usuario	Identificador del usuario que realiza una operación.	text
tabla	Nombre de la tabla sobre la que se realiza la operación.	name[4]
fecha	Fecha en la que se realiza la operación.	date
cantidad	Cantidad de operaciones (I,D,U) realizadas por el usuario sobre la tabla en ese día.	integer

Se pide: implementar triggers con las siguientes características que permitan realizar este registro.

Nombre del trigger	**registro_operaciones**
Tablas	*estadias_anteriores, reservas_anteriores y clientes*
Acción	Mantiene la cantidad de operaciones que realiza cada usuario en cada tabla en cada fecha.
Observaciones	Se deberá crear un trigger por cada una de las tres tablas involucradas, asociados a un mismo procedimiento.

Se debe entregar: un archivo llamado **parte05.sql** que contenga las sentencias para la creación de la tabla, de los triggers y su función o procedimiento asociado necesarios para resolver esta parte.

4 https://www.postgresql.org/docs/current/datatype-character.html (ver tabla 8.5)

6) *Auditoría de la tabla de estadías (estadias_anteriores)*

Se desea mantener una auditoría de los cambios efectuados sobre la tabla *estadías_anteriores*. Para esto se trabaja con la tabla *audit_estadia***,** con el siguiente esquema, donde ninguno de los atributos permite valores nulos:

audit_estadia (idop, accion, fecha, usuario, cliente_documento, hotel_codigo, nro_habitacion, check_in)
donde:

Atributo	Descripción	Tipo
idop	Valores de una secuencia	Secuencia[5] **logidseq**
accion	Valores posibles: 'I','U','D' que representan las operaciones de insert, update y delete respectivamente.	char(1)
fecha	Fecha en la que se realiza la operación.	date
usuario	Usuario de la base de datos que realiza la operación.	text
cliente_documento	Documento del cliente que realiza la estadía.	Igual que en la tabla *clientes*.
hotel_codigo	Identificación del hotel sobre el cual se realiza la estadía.	Igual que en la tabla *hoteles*.
nro_habitacion	Número de la habitación donde se realiza la estadía.	Igual que en la tabla *habitaciones*.
check_in	Fecha de comienzo de la estadía.	Igual que en la tabla *estadias_anteriores*.

Datos sobre la secuencia a utilizar para el atributo idop:

Nombre	**logidseq**
Valor Incial	1
Incremento	1

Se pide implementar un trigger con las siguientes características, que permita realizar este registro.

Nombre del trigger	**auditoria_estadias**
Tabla	*estadias_anteriores*
Acción	Insertar una tupla en la tabla *audit_estadia* con los datos de la operación que se realiza sobre la tabla *estadias_amteriores*
Observaciones	• Para las operaciones D y U se deben almacenar los valores anteriores a la operación. En el caso de la I se deben almacenar los nuevos valores. • Se deben registrar únicamente las operaciones que tienen efecto sobre la tabla.

Se debe entregar: un archivo llamado **parte06.sql** que contenga la creación de la secuencia, creación de la tabla, la creación del trigger y su procedimiento asociado, necesarios para resolver esta parte.

5 https://www.postgresql.org/docs/current/sql-createsequence.html

7) *Afinidad de usuarios*

Los usuarios del hotel (clientes que se hospedan) generan puntos denominados “finguitos” con cada una de sus estadías. Para esto se trabajará sobre la siguiente tabla:

finguitos_usuarios (cliente_documento, hotel_codigo, check_in, check_out, fecha_inicio, fecha_fin, finguitos, fecha_operacion, estado) donde:

Atributo	Descripción	Tipo
cliente_documento	Usuario asociado a la estadía	Igual que en la tabla *clientes*.
hotel_codigo	Hotel en el cual se efectúa la estadía	Igual que en la tabla *hoteles*.
check_in	Fecha de inicio de la estadía	Igual que en la tabla *estadías_anteriores*.
check_out	Fecha de fin de la estadía	Igual que en la tabla *estadías_anteriores*.
fecha_inicio	Fecha de inicio de vigencia de los “finguitos”	date
fecha_fin	Fecha de fin de vigencia de los “finguitos”	date
finguitos	Cantidad de finguitos generados	int
fecha_operacion	Timestamp actual al momento de efectuar la operación.	timestamp[6]
estado	Estado de los finguitos (1:Activos, 2:Vencidos, 3:Cancelados)	smallint

Cada vez que se haga INSERT en la tabla *estadias_anteriores* se deberá insertar una tupla en *finguitos_usuarios*. Se deben considerar las siguientes pautas:

- La cantidad de finguitos se calcula como el monto que el cliente paga por la estadía dividido entre diez. Considerar el último precio vigente de la habitación, independientemente de la fecha de la estadía. Se debe truncar el resultado a un entero[7].
- Si el usuario tiene estadías previas en el mismo hotel, se deben generar 5 finguitos adicionales.
- *fecha_inicio* = check_in + 1 mes
- *fecha_fin* = check_out + 2 años
- *estado* = 1 (Activo), si la fecha_fin es igual o mayor a la fecha actual

Cada vez que se efectúe UPDATE sobre una estadía, en *estadas_anteriores*, se deberá actualizar la cantidad de finguitos para la estadía correspondiente en *finguitos_usuario*s. Si no existe tupla en *finguitos_usuarios* no se debe tomar ninguna acción.

Tanto para el INSERT como para el UPDATE se debe mantener el histórico de finguitos. Si durante cualquiera de estas operaciones se detectan, para el usuario afectado, finguitos vencidos, se debe cambiar el estado de los mismos a “Vencidos”. Se entiende que los finguitos vencieron cuando *fecha_fin* < fecha actual.

Cada vez que se efectúe DELETE sobre una estadía, en *estadas_anteriores*, se deberá cambiar el *estado* de los finguitos afectados a “Cancelados”.

Para todas las operaciones se debe colocar en *fecha_operacion* el timestamp actual. En particular, las operaciones de UPDATE y DELETE deben sobrescribir el valor de f*echa_operacion* original con el valor de timestamp actual.

6 https://www.postgresql.org/docs/current/datatype-datetime.html

7 https://www.postgresql.org/docs/current/functions-math.html

Se pide: Implementar un trigger que cumpla con el requisito especificado. En la siguiente tabla se resumen las características del trigger a implementar:

Nombre del trigger	**finguitos**
Tabla	*estadias_anteriores*
Acción	Mantener el registro de los finguitos de los usuarios de acuerdo a las reglas establecidas.

Se debe entregar: un archivo llamado **parte07.sql** que contenga las sentencias de creación de la tabla, del trigger y su función o procedimiento asociado, necesarios para resolver esta parte.