CENTRO DE ENSEÑANZA TÉCNICA Y SUPERIOR

Escuela de Ingeniería en Ciencias Computacionales

Ingeniería en Ciencias Computacionales

Materia: Sistemas de bases de datos

Presenta:

Christian Moisés Cuevas Larin 29794

Juan Pablo Hernández Vázquez 23641

Pablo Díaz Ochoa 30343

Oscar Altamirano Cerezo 30233

Profesor: Ricardo Martínez Soto

Proyecto Final: Hotel PostgreSQL

Tijuana, B.C., 3 de diciembre de 2020

Sistema de Hotel en PostgreSQL

1st Christian Moises Cuevas Larin
engr. Computer Science
CETYS Universidad
Sistemas de base de datos
Profesor: Dr. Ricardo Martinez Soto
Tijuana, Mexico
christian.cuevas@cetys.edu.mx

2nd Juan Pablo Hernandez Vázquez
engr. Computer Science
CETYS Universidad
Sistemas de base de datos
Profesor: Dr. Ricardo Martinez Soto
Tijuana, Mexico
juanp.hernandez@cetys.edu.mx

3rd Pablo Díaz Ochoa
engr. Computer Science
CETYS Universidad
Sistemas de base de datos
Profesor: Dr. Ricardo Martinez Soto
Tijuana, Mexico
pablo.diaz@cetys.edu.mx

4rd Oscar Altamirano Cerezo
engr. Computer Science
CETYS Universidad
Sistemas de base de datos
Profesor: Dr. Ricardo Martinez Soto
Tijuana, Mexico
oscar.altamirano@cetys.edu.mx

I. Resumen

En este reporte, mostraremos el desarrollo de la solución al problema del hotel de 5 estrellas, debido a que deben tener una mejor administración en el manejo de información. Para esta solución, implementaremos el administrador de bases de datos PostgreSQL, y posteriormente aplicaremos triggers y stored procedures. Esto con el fin, de mejorar el rendimiento y velocidad de la base de datos.

II. Introducción

Si bien el problema se plantea a continuación, es importante definir que el objetivo principal es satisfacer las necesidades del cliente. Describiendo más las capacidades del equipo para desarrollar el sistema del hotel, podemos decir que se tuvo conciencia del progreso del proyecto y se registró todas las etapas de planeación como diagramas UML y descripción de cada una de las entidades para completa transparencia con las necesidades del cliente. Una vez que la fase de planeación se terminó, la implementación se llevó acabo abierta a retroalimentación por posibles cambios a la estructura y arquitectura de la base de datos a desarrollarse. Los resultados son desplegados en el reporte demostrando la funcionalidad de la base de datos con consultas ejemplo que el cliente pudiera generar en situaciones hipotéticas.

III. Planteamiento del problema

Se desea desarrollar una solución de base de datos que permita llevar a cabo el control de información de un hotel 5 estrellas, los cuales se están contemplando reservaciones de habitación (por internet, teléfono, visita personal), paquetes de servicios internos y paquete de servicios externos y manejo de quejas y satisfacción del cliente.

IV. Descripción del tipo de solución al problema

Para la solución de este problema se usará el gestor de bases de datos llamado PostgreSQL, ya que es open source y está basado en bases de datos de tipo relacional. Elegimos implementar una base de datos relacional por que las entidades tienen cierta dependencia entre ellas. Para poner en contexto, mencionaremos un ejemplo: no puede existir una reservación en el hotel, si no hay una cliente/huésped que la solicite. Por otra parte, postgreSQL no requiere usar bloqueos de lectura al realizar una transacción lo que nos brinda una mayor escalabilidad. También PostgreSQL tiene Hot-Standby. Este permite que los clientes hagan búsquedas (sólo de lectura) en los servidores mientras están en modo de recuperación o espera. Así podemos hacer tareas de mantenimiento o recuperación sin bloquear completamente el sistema. [1]

V. Requerimientos de Cliente

Requisito Mínimo de Hardware: 2.8 GHZ processor, 8 gb de RAM, 1 tb de HDD or SSD
Software: Windows 2016 Server
RDBMS: PostgreSQL

VI. Desarrollo

A. Lenguaje unificado de modelado

Para comenzar, construiremos un diagrama de Lenguaje unificado de modelado o también conocido como UML, esto nos servirá para identificar las relaciones y atributos de cada entidad. Se tuvieron que realizar varias iteraciones por la retroalimentación con el cliente. La primera versión del diagrama según las especificaciones terminó como se puede ver en la figura 1 en anexos.

Después de la primera etapa de desarrollo, se pensó que sería mejor estructurar la entidad previamente concebida como ”retroalimentación” como ”queja” solamente, ofreciendo

fechas de registro de queja y fechas de resolución de esta, al igual que la oportunidad de registrar a que departamento es esta pertinente. Fuera de eso, se realizaron algunos cambios pertinentes de relaciones entre entidades. La segunda versión del diagrama según las especificaciones y nuevas consideraciones terminó como se puede ver en la figura 2 en anexos.

Finalmente, considerando ... se pensó que sería mas pertinente realizar una tercera versión del diagrama para iniciar con la fase de implementación. La tercera versión del diagrama según las consideraciones finales terminó como se puede ver en la figura 3 en anexos.

B. Rutinas de backup

Los backup de esta base de datos van a tener una rutina de backup total cada 6 meses, debido a que tenemos que prevenir posibles fallos de disco, desastres, etc. Pero también es importante implementar un backup que se generé cada mes, en este caso diferencial. Por último, se debe crear un backup incremental cada 3 días , ya que es el más rápido y solo copia los regitros que cambiaron a partir de un backup diferencial.

C. Roles y usuarios creados

Para el manejo organizacional en el uso de la base de datos, se crearon distintos roles con diferentes privilegios para distribuir las tareas y responsabilidades entre los usuarios de la base de datos.

- db_administrator: Es el usuario que tiene todos los privilegios, reservado para el administrador de la base de datos.
- user_insert: Es el usuario que tiene solo privilegios de inserción a la base de datos.
- user_privileges: Es el usuario que tiene privilegios de otorgar privilegios a otros usuarios.
- user_selecter: Es el usuario que tiene privilegios de consulta solamente y puede realizar solo comandos de palabra clave SELECT.

D. Descripción de entidades

A continuación, explicaremos cada entidad para justificar el por qué se encuentran en nuestro diagrama:

En la primera versión:

- Reservación: Es la entidad principal,y es el pivote de la funcionalidad de esta base de datos.Es esencial reservar una habitación, por que si no existe una reservación, puede que dos o más clientes tengan conflictos al momento de querer usar una habitación.
- Huésped_principal: Esta entidad es necesaria para tener la información disponible de la persona que haga la reservación. No se le puede rentar una habitación a alguien que no esté registrado.
- Huésped_acompañante: Es necesaria esta entidad por el hecho de que una reservación no solo puede ser para una persona (huésped_principal) si no que la habitación reservada puede tener una capacidad para más de 1 huésped.
- Retroalimentación: Esta entidad se encarga de registrar todo tipo de retroalimentación que los huéspedes decidan dejar sobre la calidad del servicio o de su estancia en general. Puede ser positiva o negativa por lo que cubre la función de quejas también.
- Habitación: Esta entidad es necesaria porque es donde se guarda las habitaciones que hay disponibles para rentar. También tiene información valiosa como su disponibilidad y capacidad.
- Categoria_habitación: Esta entidad es necesaria ya que funge como el catálogo del tipo de habitación que uno puede rentar. Contiene una descripción general del tipo de habitación y el precio de esta.
- Factura: El motivo de esta entidad es englobar todos los cargos que haya podido tener el huésped. Contiene un identificador del huésped y su reservación correspondiente.
- Cargo: La entidad cargo tiene como propósito guardar la información que podría tener cualquier gasto del huésped. Contiene información como el tipo de servicio al que pertenece el cargo, categoría de este, posible paquete al que pertenezca, y detalles como la fecha y el total monetario.
- Paquete: Esta entidad es importante ya que se encarga de en listar los tipos de paquetes que un huésped podría elegir para su estadía en el hotel. Contiene la información de inicio y fin de la disponibilidad de la promoción, al igual que el precio.
- Empleado: Esta entidad guarda los datos necesarios de los empleados para todas las actividades administrativas realizadas en el hotel.
- Departamento: Esta entidad lista las áreas de trabajo en las cuales dividimos nuestros empleados para una mejor organización.
- Bono: Esta entidad registra los bonos dados a los empleados. Es necesaria para llevar un historial de las razones, la cantidad monetaria, y fecha de estos bonos.

- Servicio: Esta entidad se encarga de guardar los servicios proporcionados por nuestro hotel. Esto nos ayuda a poder facturar mas fácilmente y dividir los empleados por departamentos.

En la segunda versión: se dejo la entidad retroalimentación por la entidad queja.

- Queja: Esta entidad se encarga de guardar posibles quejas de algún huésped con respecto a un servicio especifico. Cuenta con información pertinente a la queja, como fecha de registro, estado de la queja, fecha de finalización, motivo, huésped y servicio responsable.

En la tercera versión: agregamos la entidad promoción para complementar la información de paquete, al igual que un cambio en paquete.

- Promoción: Esta entidad se encarga de registrar los elementos para un paquete, es decir almacena la información de el servicio correspondiente a la promoción incluida dentro del paquete.

- Paquete: Se agregó el atributo ”nombre” para identificar más fácilmente a los paquetes.

E. Stored procedures realizados

Para facilitar el trabajo de los usuarios de la base de datos, a veces es necesario realizar funciones que automaticen algunas de las tareas que podría realizar alguien junto a una consulta o 'update' a la base de datos. PostgreSQL no cuenta con stored procedures como tal [2], pero pueden registrarse funciones que cumplen el mismo objetivo.

A continuación las funciones que se realizaron fueron:

- get_employee_by_name: función que toma como parámetro el nombre de un empleado y retorna la información de él o ella. Esta función puede visualizarse en el *Listing 1* en los anexos.

- get_charges_from_factura: función que toma como parámetro el id de una factura y regresa todos los cargos pertenecientes a la factura indicada. Esta función puede visualizarse en el *Listing 2* en los anexos.

- get_reservations_between: función que toma una fecha de inicio y una fecha final y regresa todas las reservaciones que hayan sido registradas durante el tiempo entre esas dos fechas. Esta función puede visualizarse en el *Listing 3* en los anexos.

- get_rooms_available: función que toma como parámetro la categoría de la habitación y regresa todas las habitaciones disponibles pertenecientes a dicha categoría. Esta función puede visualizarse en el *Listing 4* en los anexos.

- get_active_quejas_per_department: función que toma como parámetro el departamento de servicio y regresa todas las quejas activas que sean sobre el departamento consultado. Esta función puede visualizarse en el *Listing 5* en los anexos.

F. Disparadores realizados

Para facilitar el trabajo de los usuarios, pueden automatizarse ciertas acciones antes o después de una inserción, una alteración a un registro, un borrado, etc. Por lo tanto, se realizaron los siguientes disparadores:

- verificar_fecha_cancelacion: Evita que se cancele una reservación que ya se había completado. Este disparador puede visualizarse en el *Listing 6* en los anexos.

- evitar_modificacion_paquete: evita la modificación de un paquete si ya se eligió uno. Este disparador puede visualizarse en el *Listing 7* en los anexos.

- agregar_bono: agrega un bono al empleado que haya realizado una venta de registro de reservación. Si es mayor a 3 días es de 380 pesos, si es mayor a 5 días es de 500 pesos y por último de 10 días en adelante el bono es de 1500 pesos. Este disparador puede visualizarse en el *Listing 8* en los anexos.

- package_updater: agrega promociones nuevas al paquete que se le relacione dependiendo del servicio nuevo que se registre. Este disparador puede visualizarse en el *Listing 9* en los anexos.

VII. Resultado

Como se ve en la figura 3 de los diagramas UML, la versión 3 fue la versión final de la que se realizaron todas las tablas. Se integraron todos los datos prueba para demostrar el funcionamiento de la base de datos con 15 o más inserciones, al igual que las funciones y disparadores. No se detectó ningún error. En los anexos se presentan algunas capturas de pantalla demostrando algunas de las tablas sobre las que se trabajo (Figuras 4 y 5).

VIII. Conclusión

En conclusión, se logró realizar un sistema de base de datos en base a las especificaciones del cliente con la posibilidad de realizar todas las funcionalidades básicas del hotel y facilitar el trabajo de los empleados en cuanto al registro de empleados, habitaciones, promociones, etc.

IX. Anexos

Fig. 1. Versión 1 del diagrama UML

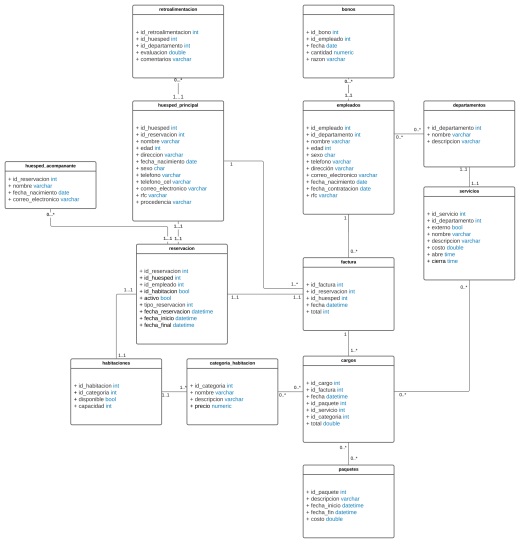


Fig. 2. Versión 2 del diagrama UML

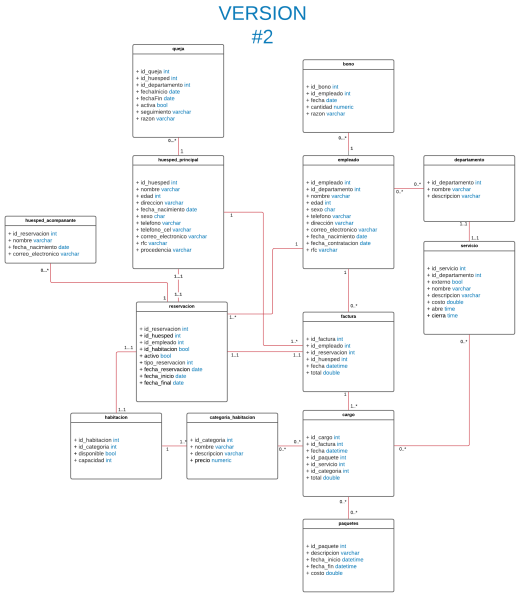
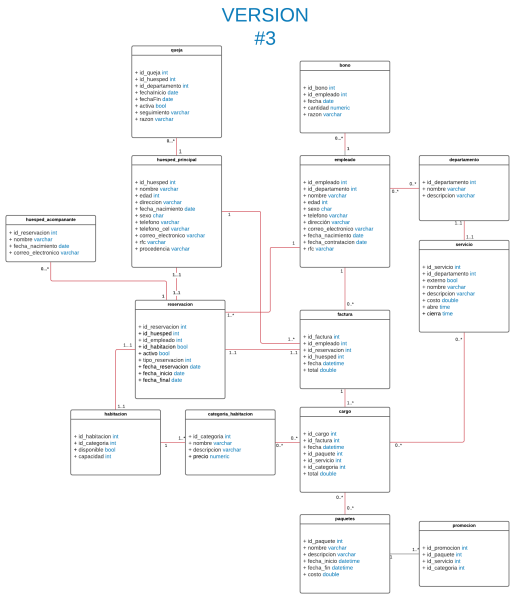


Fig. 3. Versión 3 del diagrama UML



Listing 1. Función get_employee_by_name

```
create function
get_employee_by_name(name varchar)
  returns setof empleado
as
$$
select * from empleado where nombre = name;
$$
language sql;
```

Listing 2. Función get_charges_from_factura

```
create function
get_charges_from_factura(idfactura int)
  returns setof cargo
as
$$
select * from cargo where
id_factura = idfactura;
$$
language sql;
```

Listing 3. Función get_reservations_between

```
create function
get_reservations_between(fechaInicio date,
fechaFinal date)
  returns setof reservacion
as
$$
select * from reservacion where
fecha_reservacion between fechaInicio and
fechaFinal;
$$
language sql;
```

Listing 4. Función get_rooms_available

```
create function
get_rooms_availables(categoria int)
  returns setof habitacion
as
$$
select * from habitacion where
id_categoria = categoria and
disponible = '1';
$$
language sql;
```

Listing 5. Función get_active_quejas_per_department

```
create function
get_active_quejas_per_department
(departamento int)
  returns setof queja
as
$$
select * from queja where
id_departamento = departamento and
activa = '1';
$$
language sql;
```

Listing 6. Disparador verificar_fecha_cancelacion

```
CREATE OR REPLACE FUNCTION
verificar_fecha_cancelacion()
RETURNS TRIGGER AS $verificar_fecha_cancelacion$
  DECLARE
  BEGIN
   if OLD.activo = '1' and NEW.activo = '0'
   and OLD.fecha_final <= current_date THEN
        RAISE unique_violation USING MESSAGE =
        'COMPLETED RESERVATIONS CANNOT BE
        CANCELLED';
        END if;
   RETURN NEW;
  END;
$verificar_fecha_cancelacion$
LANGUAGE plpgsql;


CREATE TRIGGER verificar_fecha_cancelacion
    BEFORE UPDATE ON reservacion
    FOR EACH ROW EXECUTE PROCEDURE
    verificar_fecha_cancelacion();
```

Listing 7. Disparador evitar_modificacion_paquete

```
CREATE OR REPLACE FUNCTION
evitar_modificacion_paquete()
RETURNS TRIGGER AS
$evitar_modificacion_paquete$
  DECLARE
  BEGIN
    if OLD.id_paquete <> NEW.id_paquete
    THEN
        RAISE unique_violation
        USING MESSAGE =
        'CANNOT CHANGE PACKET ONCE
        REGISTERED';
        END if;
   RETURN NEW;
  END;
$evitar_modificacion_paquete$ LANGUAGE
plpgsql;


CREATE TRIGGER evitar_modificacion_paquete
BEFORE UPDATE ON cargo FOR EACH ROW
    EXECUTE PROCEDURE
    evitar_modificacion_paquete();
```

Listing 8. Disparador agregar_bono

```
CREATE OR REPLACE FUNCTION agregar_bono()
RETURNS TRIGGER AS $agregar_bono$
  DECLARE
  BEGIN
        if NEW.fecha_final -
        NEW.fecha_inicio > 10 THEN
                insert into bono values(
                default,NEW.id_empleado,
                current_date,1500,
                'registro de reservacion
                mas de 10 dias'
                );
        elseif NEW.fecha_final -
        NEW.fecha_inicio > 5 THEN
                insert into bono values(
                default,NEW.id_empleado,
                current_date,500,
                'registro de reservacion
                mas de 5 dias'
                );
        elseif NEW.fecha_final -
        NEW.fecha_inicio > 3 THEN
                insert into bono values(
                default,NEW.id_empleado,
                current_date,380,
                'registro de reservacion
                mas de 3 dias'
                );
        END if;
   RETURN NEW;
  END;
$agregar_bono$ LANGUAGE plpgsql;


CREATE TRIGGER agregar_bono AFTER INSERT
    ON reservacion FOR EACH ROW
    EXECUTE PROCEDURE agregar_bono();
```

Listing 9. Disparador package_updater

```
CREATE OR REPLACE FUNCTION package_updater()
RETURNS TRIGGER AS $package_updater$
  DECLARE
  BEGIN
   if new.id_departamento = 3 THEN
        INSERT INTO promocion VALUES (
        default,
        1,
        NEW.id_servicio,NULL);
        END if;
   RETURN NEW;
  END;
$package_updater$ LANGUAGE plpgsql;


CREATE TRIGGER package_updater after insert
    ON servicio FOR EACH ROW
    EXECUTE PROCEDURE package_updater();
```

Fig. 4. Tabla de huéspedes

id_reservacion [PK] integer	id_huesped integer	id_empleado integer	id_habitacion integer	activo boolean	tipo_reservacion integer	fecha_reservacion date	fecha_inicio date	fecha_final date
1	6	6	5	true	1	2017-09-14	2017-10-14	2017-10-15
2	8	8	7	true	2	2018-01-05	2018-01-09	2018-01-12
3	14	14	14	false	3	2018-02-26	2018-02-27	2018-02-28
4	15	15	15	false	1	2018-05-16	2018-05-18	2018-05-19
5	9	9	8	true	2	2018-05-25	2018-05-28	2012-05-29
6	11	11	10	true	3	2018-06-17	2018-06-23	2018-06-26
7	13	13	12	false	1	2018-07-03	2018-07-03	2018-07-06
8	10	10	9	true	2	2018-10-25	2019-06-14	2019-06-15
9	15	15	15	false	3	2018-11-04	2018-11-12	2018-11-13
10	3	3	2	false	1	2019-01-06	2019-01-13	2019-01-15
11	12	12	11	false	2	2019-02-22	2019-02-26	2019-02-27
12	4	4	3	false	3	2019-03-01	2019-03-01	2019-03-02
13	2	1	1	true	1	2019-03-12	2019-03-13	2019-03-16
14	7	7	6	false	2	2019-04-24	2019-05-27	2019-05-29
15	5	5	4	false	3	2019-11-10	2019-11-24	2019-11-25

Fig. 5. Tabla de reservaciones

id_reservacion integer	nombre character varying	fecha_nacimiento date	correo_electronico character varying
1	Macie Stehr	1993-11-29	irving.torphy@yahoo.com
2	Ivah Kertzmann	1995-02-04	eichmann.evangeline@r...
3	Waino Jacobson	2019-09-29	kassulke.sarai@gmail.c...
4	Chanel Mraz	1989-02-03	conner@gmail.com
5	Ethan Casper	1989-03-10	adickens@robel.com
6	Prof. Chelsie Franecki	2013-04-19	wiza.hassie@lebsack.co...
7	Dan Roberts	2012-02-21	janessa32@hotmail.com
8	Toy Heidenreich I	1991-07-21	angeline16@jerde.com
9	Dedrick Runte	1972-10-09	rick.runolfsdottir@yahoo...
10	Keshaun Koelpin MD	1981-02-26	darron.koepp@gmail.com
11	Cathryn Wiegand	2018-05-13	daugherty.breanne@yah...
12	Bennett Hand II	1971-07-12	newell.schuster@oreillyl...
13	Virgie Graham	2011-11-18	bryon43@gmail.com
14	Conor Hirthe	1979-01-21	jaqueline.wuckert@gmai...
15	Miss Leta Heathcote	2019-01-03	sydnee.mann@gmail.com

REFERENCES

[1] C. A. Dorantes. Qué es postgresql y cuáles son sus ventajas. platzi.com. [Online]. Available: https://platzi.com/blog/que-es-postgresql/

[2] (2020, Nov) Create procedure. [Online]. Available: https://www.postgresql.org/docs/11/sql-createprocedure.html