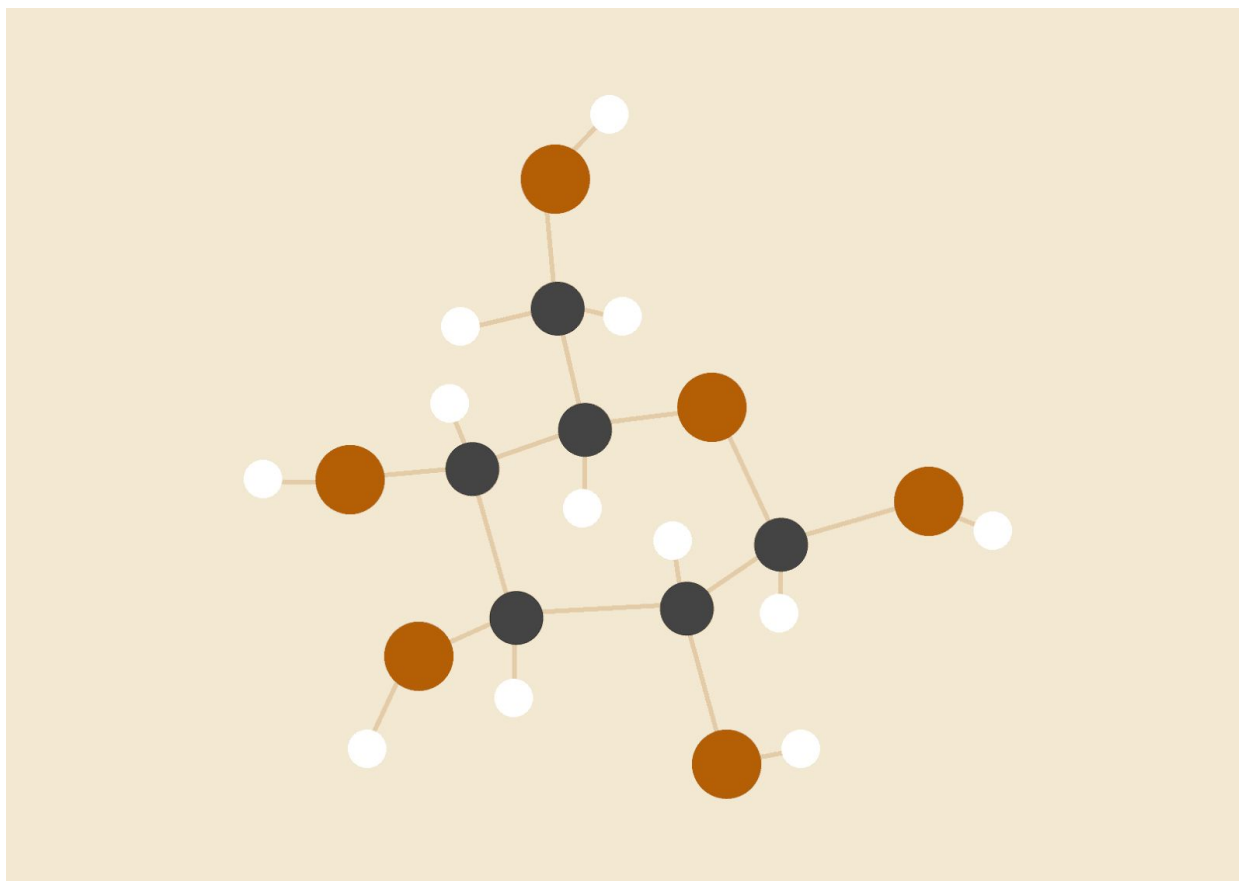


INFORME TPE BBDD 2019

Flamminio, Brenda (LU: 249441)

Zapata, Joaquín (LU: 249257)



GRUPO N°7

Fecha de entrega: 06/06/2019

INTRODUCCIÓN

En este informe mostraremos detalladamente el desarrollo del Trabajo Práctico Especial. Vamos a trabajar con una base de datos para el depósito de la Empresa “WMS Tandil”.

AJUSTE DEL ESQUEMA

Debemos crear un esquema y ajustarlo según las pautas de nomenclatura e incorporar el soporte para todos los aspectos indicados y crear un script de generación del mismo.

Empezamos crear un nuevo script SQL en Vertabelo utilizando el código SQL otorgado por la cátedra, para poder visualizar el esquema DERE así poder modificarlo y ajustarlo para cumplir con las pautas indicadas.

Cambiamos la nomenclatura de todas las tablas dadas y de las claves foráneas para que queden de la siguiente manera:



Luego pasamos al agregado de los atributos que creemos necesarios para el cumplimiento de ciertos servicios, por ejemplo el **id_posicion** de la tabla **gr7_posicion**, para darle a las posiciones un número único en el depósito, con el objetivo de utilizarlo como clave secundaria. Esto implica generar una restricción de unicidad sobre este atributo.



También agregamos el atributo **peso_max_kg** en la tabla **gr7_fila** para aplicar una restricción de peso en las filas.

Lo siguiente es arreglar todas las claves foráneas, y agregar algunas otras para ciertas funciones. Decidimos que la tabla **gr7_mov_interno** debe referenciarse a sí misma para tener un tracking de los movimientos anteriores; además tanto los movimientos de salida

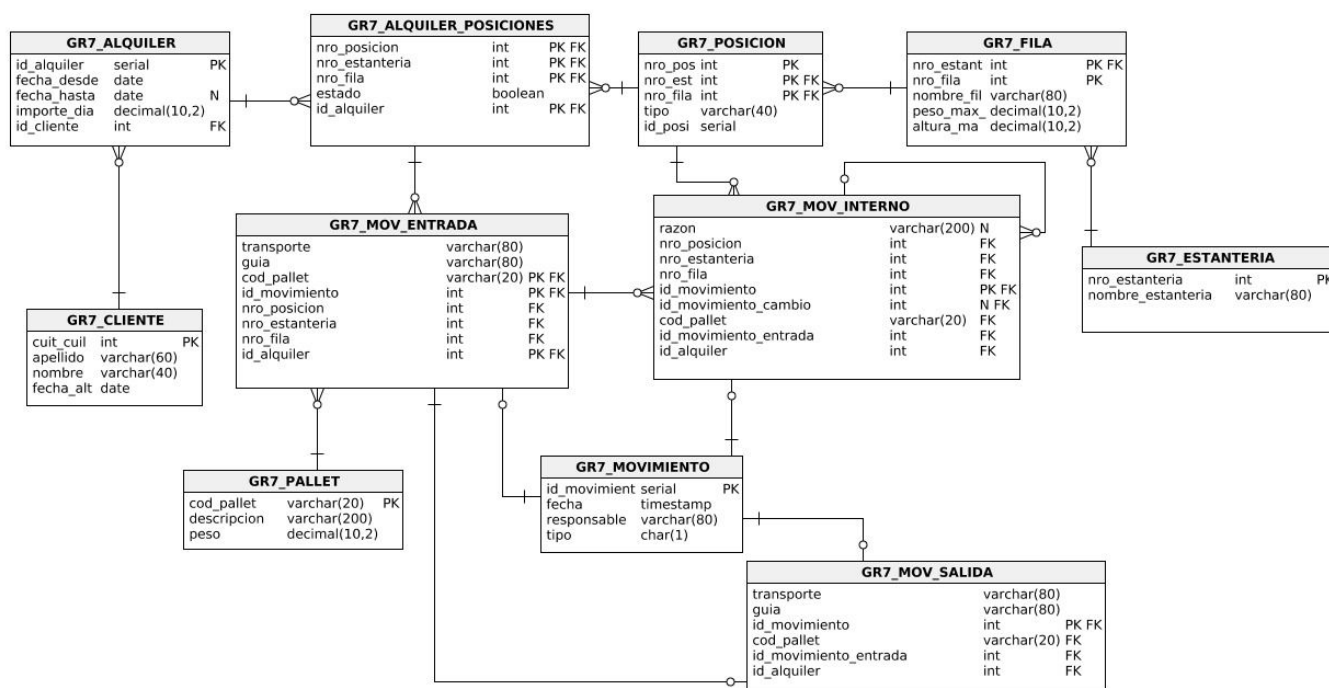
como los internos deben referenciar a un movimiento de entrada.

Tuvimos en cuenta a la hora de crear las claves foráneas, que sería conveniente configurarlas para la **eliminación y actualización en cascada**. Esto nos ayudará más adelante a la hora de generar el script de borrado de las tablas.

Update constraint action:
cascade

Delete constraint action:
cascade

El modelo DERE final queda de esta manera:



Decidimos que usar el tipo de variable **serial** para que las ID sean auto incrementales, y ahorrarnos tiempo y algunos errores a la hora de ingresar registros a las tablas.

Ahora vamos a agregar registros a las tablas, con el objetivo de probar que las relaciones están bien implementadas y funcionen de manera correcta. Para nombrar las estanterías y las filas decidimos usar nombres de animales y números, respectivamente.

Con todo esto ya podemos generar el script **G7_Creacion.sql**, y pasar al siguiente punto.

ELABORACIÓN DE RESTRICCIONES

El siguiente paso es dar soporte a las restricciones o reglas del negocio enunciadas:

- A. La fecha de todos los alquileres deben ser consistentes, es decir la fecha a partir de la cual se inicia un alquiler debe ser menor o igual a la fecha de finalización del mismo. En este caso, determinamos que al ser una restricción de tupla que involucra los atributos **fecha_desde** y **fecha_hasta** de la tabla **gr7_alquiler**, utilizaremos una restricción **CHECK**, que controle que las fechas ingresadas sean coherentes. En caso de que intentemos violar esta restricción, el DBMS arroja este error:

Error de SQL:

```
ERROR: new row for relation "gr7_alquiler" violates check constraint "ck_gr7_gr7_alquiler_fecha_valida"
DETAIL: Failing row contains (2, 2019-06-04, 2018-05-03, 25.00, 21753).
```

En la sentencia:

```
INSERT INTO "unc_249257"."gr7_alquiler" ("id_alquiler","fecha_desde","fecha_hasta","importe_dia","id_cliente")
VALUES (nextval('gr7_alquiler_id_alquiler_seq'::regclass), '2019-06-04', '2018-05-03', '25', '21753')
```

- B. El peso de los pallets de una fila no debe superar al máximo de la fila. Para controlar esto utilizaremos una función que mantenga actualizado el peso total de las filas, y antes de mover un pallet a una fila, controlaremos que no supere el límite asignado a la misma (**peso_max_kg**). Para esto habría que crear una **ASSERTION** como restricción general, pero los DBMS no las soportan. La manera declarativa sólo implicaría unir las tablas y chequear que el peso del pallet que ingresa, más el de los que ya están allí, supere el peso total soportado por la fila:

```
CREATE ASSERTION peso_valido
CHECK ( NOT EXISTS (
    SELECT 1
    FROM gr7_pallet p,
    JOIN gr7_mov_entrada me ON ( me.cod_pallet = p.cod_pallet )
    JOIN gr7_fila f ON (
        me.nro_estanteria = f.nro_estanteria AND me.nro_fila = f.nro_fila )
    WHERE ( SELECT sum(p1.peso) FROM gr7_pallet p1
    JOIN gr7_fila f1 ON (
        me.nro_estanteria = f1.nro_estanteria AND me.nro_fila = f1.nro_fila ) )
    + p.peso > f.peso_max_kg ) );
```

Al no contar con la posibilidad de usar assertions, lo resolveremos mediante la implementación de **TRIGGERS** y **FUNCIONES**.

Ejemplo de funcionamiento:

Error de SQL:

```
ERROR: LA FILA NO PUEDE SOPORTAR MAS PESO
CONTEXT: PL/pgSQL function trfn_gr7_ctrl_peso() line 23 at RAISE
```

En la sentencia:

```
INSERT INTO gr7_mov_entrada VALUES
('transporte 1','guia 1',1174,2,1,1,1,1);
```

- C. El tipo de posición puede tomar los siguientes valores “general”, “vidrio”, “insecticidas”, “inflamable”. Para esto simplemente agregamos una restricción de atributo usando un **CHECK** sobre **tipo** en la tabla **gr7_posicion** que especifica los valores que puede tomar el mismo. En caso de que intentemos agregar un tipo inválido (plástico), el DBMS nos arroja el siguiente error:

Error de SQL:

```
ERROR: new row for relation "gr7_posicion" violates check constraint "ck_gr7_gr7_posicion_tipo_valido"
DETAIL: Failing row contains (5, 4, 1, plastico, 15).
```

En la sentencia:

```
INSERT INTO "unc_249257"."gr7_posicion" ("nro_posicion","nro_estanteria","nro_fila","tipo","id_posicion")
VALUES ('5','4','1','plastico',nextval('gr7_posicion_id_posicion_seq'::regclass))
```

SERVICIOS

Debemos implementar lo siguiente a través de triggers, procedimientos y vistas:

1. Para una fecha determinada dar la lista de las posiciones libres; esto es número de estantería, número de fila y número de posición.

Aclaración: Para este punto consideramos como libres aquellas posiciones del depósito que no están alquiladas por ningún cliente actualmente.

Debemos crear una **función** que seleccionará aquellas posiciones que no hayan sido alquiladas y las que estaban alquiladas pero se han vencido, en una fecha dada. Para ello la función retornará un **SET OF** de la tabla **gr7_posicion**, y dejaremos fuera los resultados no deseados utilizando **NOT IN** sobre una subconsulta compleja con un **WHERE** sobre la tabla de alquileres que tenga en

cuenta la fecha recibida por parámetro.

En este ejemplo, usando la fecha actual la consulta funciona correctamente y devuelve esta lista de posiciones:

| nro_posicion | nro_estanteria | nro_fila | tipo | id_posicion |
|--------------|----------------|----------|--------------|-------------|
| 1 | 1 | 1 | general | 1 |
| 1 | 1 | 2 | general | 3 |
| 1 | 2 | 1 | vidrio | 4 |
| 5 | 3 | 1 | vidrio | 6 |
| 1 | 3 | 1 | inflamable | 7 |
| 2 | 4 | 1 | insecticidas | 8 |
| 2 | 2 | 1 | general | 9 |
| 3 | 2 | 1 | general | 10 |
| 4 | 2 | 1 | vidrio | 11 |
| 5 | 2 | 1 | general | 12 |
| 6 | 2 | 1 | vidrio | 13 |
| 7 | 2 | 1 | general | 14 |
| 5 | 4 | 1 | vidrio | 16 |
| 3 | 4 | 1 | vidrio | 17 |

14 fila(s)

Servidor?: PostgreSQL (127.0.0.1:5432:allow) ▼

Ruta de la búsqueda en los esquemas?: unc_249257

`select * from fn_gr7_posicioneslibres(current_date)`

o subir un script SQL: Ningún archivo seleccionado

Si integramos este procedimiento podremos hacer consultas fácilmente desde el sitio web que crearemos más adelante.

2. Dar la lista de los clientes que en una cierta cantidad de días (configurable) se les debe avisar que se vence su alquiler. Para ello creamos una **función** que devuelve filas de la tabla de clientes, donde se verifica que en una determinada cantidad de días (que pasamos por parámetro) se vence el alquiler del mismo.

Resultado de la consulta

| cuit_cuil | apellido | nombre | fecha_alta |
|-----------|-------------|--------|------------|
| 22944 | Artiguenabe | Camila | 2019-05-10 |

1 fila(s)

Ruta de la búsqueda en los esquemas?: unc_249257

`select * from fn_gr7_notificacionclientes(10)`

Para ello utilizamos el tipo de retorno **SET OF** en la tabla **gr7_cliente**. En el ejemplo elegimos la cantidad de días 10, y luego verificamos que devuelve los datos correctamente, con otra consulta:

Resultado de la consulta

| apellido | nombre | current_date | fecha_hasta |
|-------------|--------|--------------|-------------|
| Artiguenabe | Camila | 2019-06-05 | 2019-06-15 |

1 fila(s)

Ruta de la búsqueda en los esquemas?: unc_249257

`select c.apellido,c.nombre, current date, a.fecha_hasta from gr7_alquiler a join gr7_cliente c on (c.cuit_cuil = a.id_cliente) where c.cuit_cuil = 22944`

DEFINICIÓN DE VISTAS

Para finalizar el script **G7_Cambios.sql** debemos agregar algunas vistas para facilitarnos la búsqueda de ciertos datos relevantes:

1. Realizar una vista que para cada una de las posiciones indique su estado libre u ocupada, y para éste último caso se indique la cantidad de días que restan de alquiler. Indicar los datos completos de la posición.

Para crear esta vista hemos tenido que comprobar que se muestren todas las posiciones, y controlamos con un sentencias **CASE** que devuelva distintos resultados según lo que obtengamos del atributo **estado**. También manejamos los resultados que sean negativos, porque no sería coherente que el tiempo restante de un alquiler sea una cantidad de días negativa.

Hacemos una prueba y examinamos la vista para comprobar que los resultados sean los esperados:

```
SELECT * FROM unc_249257.gr7_estado_posiciones;
```

Enviar

| nombre_estanteria | nombre_fila | nro_posicion | días restantes |
|-------------------|-------------|--------------|----------------|
| perro | fila 1 | 1 | 184 |
| perro | fila 1 | 2 | 186 |
| gato | fila 1 | 1 | 237 |
| perro | fila 2 | 1 | 98 |
| perro | fila 1 | 2 | 0 |
| pez | fila 1 | 5 | 0 |
| perro | fila 2 | 3 | 0 |

Consideramos que esta vista no es actualizable porque no conserva todas las columnas de la clave primaria.

2. Realizar una vista que liste los 10 clientes que más dinero han invertido en el último año (tomar el momento en el que se ejecuta la consulta hacia atrás).

Para ello necesitamos realizar una consulta que cuente la cantidad de días de alquiler del último año para cada cliente, y haga la multiplicación con el importe

por día correspondiente al alquiler del cliente.

```
sum(a.importe_dia::double precision *  
CASE  
  WHEN a.fecha_desde <= (CURRENT_DATE - '1 year'::interval) AND a.fecha_hasta >= CURRENT_DATE THEN date_part('day'::text, now()) - '1 year'::interval)  
  ELSE date_part('day'::text, now()) - a.fecha_desde::timestamp with time zone)  
END) AS invertido
```

Sabiendo que todos los clientes que hemos cargado pagan el mismo **importe por día**, no aportará lo mismo alguien que está alquilando hace un año que alguien que alquiló durante los últimos seis meses. Por ello es que hay que buscar el **número exacto de días** que aportó este año. Conseguimos esto calculando desde la fecha actual hasta la fecha desde la que se inició el alquiler. También tuvimos que comprobar que devuelva los resultados correctamente para aquellos clientes que tienen más de una posición alquilada.

```
SELECT * FROM unc_249257.gr7_top_clientes_anual;
```

Enviar

| id_cliente | apellido | nombre | total invertido |
|------------|-------------|---------|-----------------|
| 20458 | Gomez | Miguel | 8725 |
| 22158 | Sanchez | Mario | 7200 |
| 20124 | Pastor | Daiana | 6175 |
| 21731 | Arias | Julieta | 4300 |
| 22944 | Artiguenabe | Camila | 675 |

Examinamos la vista y obtuvimos lo esperado. Sabemos que Miguel Gómez y Julieta Arias tienen a su nombre dos alquileres cada uno, y ambos resultados se han sumado naturalmente al total invertido. Se dice que esta vista no es actualizable porque contiene por lo menos una función de agregación.

SCRIPT DE BORRADO

El último script que debemos realizar es el de borrado. Esto implica eliminar todas las tablas, relaciones, funciones, disparadores y todo lo relacionado con el proyecto y dejar la base completamente limpia. Para hacerlo correctamente debemos tener en cuenta las jerarquías de cada tabla, y eliminar todo de manera ordenada.

Decidimos eliminar todo mediante sentencias **DROP** en el siguiente orden: vistas, funciones, disparadores, chequeos, claves foráneas, tablas.

Luego de agregar todas las sentencias de borrado, queda listo el script **G7_Borrado.sql**.

SITIO A REALIZAR

Creamos un sitio en PHP con el fin de poder acceder a la base de datos mediante una interfaz y dar soporte a los servicios indicados de una manera simple para el usuario.

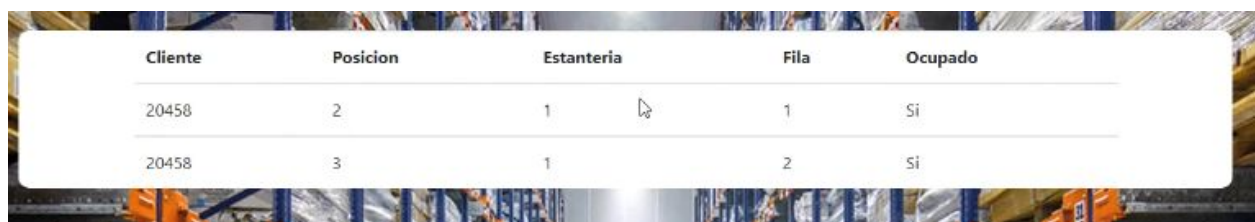
Le ofrecemos al usuario un campo para que ingrese una **fecha**, que se pasa como parámetro al procedimiento que habíamos creado antes, y ejecuta la sentencia directamente a nuestra base de datos. Esto nos devuelve una lista de todas las posiciones libres a la fecha, y quedan plasmados en la tabla.



The screenshot shows a web form titled "Formulario de consultas" (Consultation Form). It contains two sections for user input:

- The first section is labeled "Consulte por posiciones libres:" (Consult by free positions:). It features a text input field with the placeholder "dd/mm/aaaa" and a blue button labeled "Consultar" (Consult).
- The second section is labeled "Consulte el estado de sus alquileres:" (Consult the status of your rentals:). It features a text input field with the placeholder "XXXXX" and a blue button labeled "Consultar" (Consult).

También le permitimos buscar por **id_cliente** y devolver una lista de las posiciones que este tiene ocupado actualmente en el depósito.



The screenshot displays a table with the following data:

| Cliente | Posicion | Estanteria | Fila | Ocupado |
|---------|----------|------------|------|---------|
| 20458 | 2 | 1 | 1 | Si |
| 20458 | 3 | 1 | 2 | Si |

En este ejemplo vemos que devuelve todos los datos de la posiciones que el cliente tiene ocupadas actualmente.

Con esto damos por finalizado el desarrollo del TPE de Bases de Datos 2019.