PRACTICA COMPLEMENTARIA 12
PROGRAMACIÓN CON SQL PARTE 1

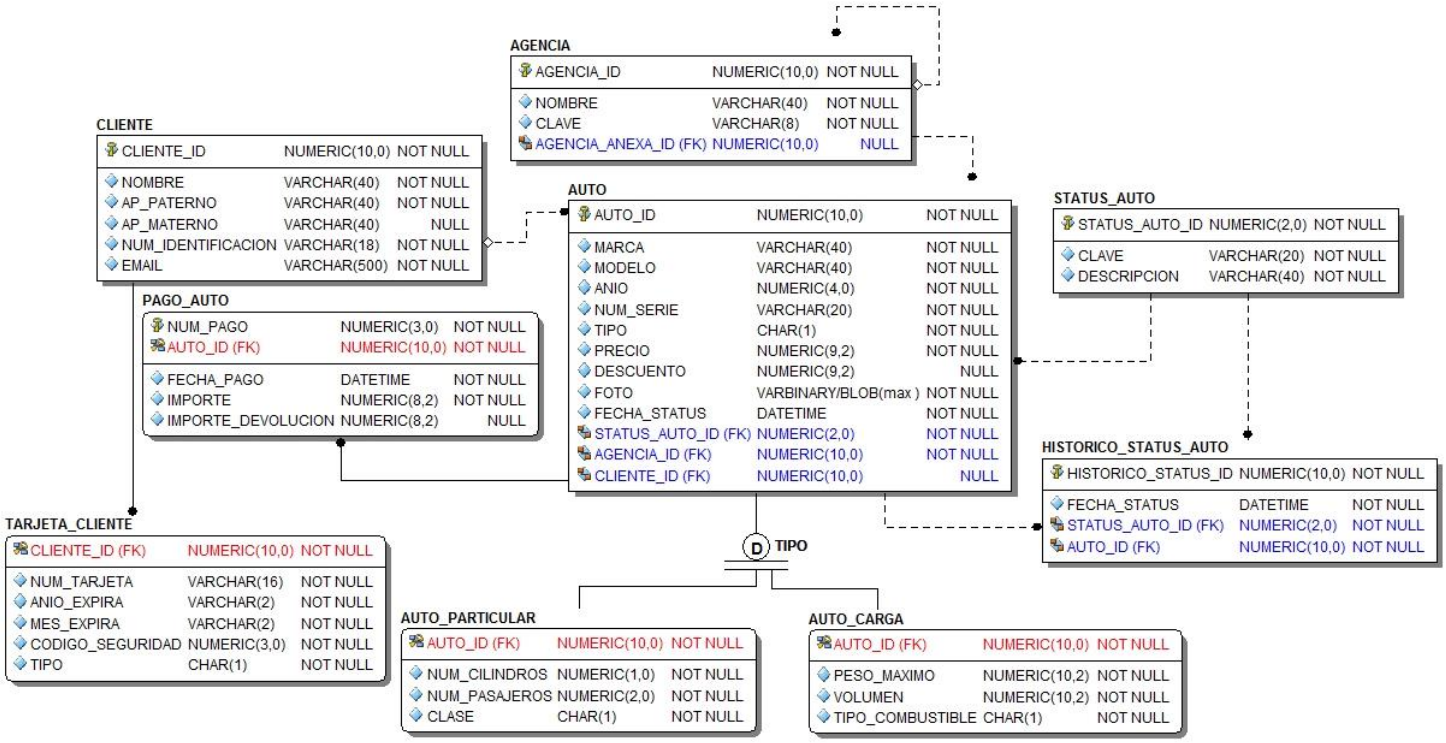
El reporte se entrega en equipos formado por máximo 2 integrantes.

1.1. OBJETIVO:

Poner en práctica los conceptos de programación PL/SQL para crear bloques anónimos, procedimientos, disparadores (triggers) así como funciones creadas por el usuario.

1.2. PREPARACIÓN DE LA BASE DE DATOS.

Considerar el siguiente modelo relacional que representa la implementación de una base de datos para una empresa automotriz dedicada a la venta de autos.



1.2.1. Creación de usuario

- Crear un script llamado `s-01-creacion-usuario.sql` El script deberá contener el código necesario para crear un usuario llamado `<iniciales>_p1201_autos`, donde: `<iniciales>` corresponde con las primeras 4 letras de los integrantes del equipo: letra inicial de los apellidos del primer integrante, letra inicial de los apellidos del segundo integrante. Si es individual, emplear las iniciales del nombre completo. No olvidar agregar el encabezado al archivo.
- Asignarle los privilegios necesarios para poder crear los objetos del diagrama relacional (tablas y secuencias). Adicionalmente asignarle el privilegio `create procedure, create trigger.`

Ejemplo:

```
--@Autor:           Jorge Rodriguez
--@Fecha creación:  dd/mm/yyyy
--@Descripción:     Creación de usuario Practica 12
Prompt proporcione el password del usuario sys
connect sys as sysdba

--permite la salida de mensajes a consula empleabo dbms_output.put_line
set serveroutput on

--este bloque anómimo valida la existencia del usuario, si existe lo elimina.
declare
 v_count number(1,0);
begin
 select count(*) into v_count
 from dba users
 where username = 'JRC_P1201_AUTOS';
 if v_count > 0 then
  dbms_output.put_line('Eliminando usuario existente');
  execute immediate 'drop user jrc p1201 autos cascade';
 end if;
end;
/

create user jrc p1201 autos identified by jorge quota unlimited on users;
grant create session, create table, create procedure, create trigger,
 create sequence to jrc_p1201_autos;
```

1.2.2. Creación de objetos

- De la carpeta compartida correspondiente a esta práctica, descargar un script llamado `s-02-autos-ddl.sql` el cual será el encargado de crear los objetos del modelo relacional anterior.

1.2.3. Carga inicial

Para cada tabla se proporciona un script de carga inicial que puede ser descargado del sitio Web Mockaroo (generador de datos SQL). El Script se puede descargar directamente del navegador web empleando la dirección http, o a través de una terminal empleando el comando `curl` que será encargado de obtener los datos y guardarlos en el script SQL correspondiente. Se sugiere abrir una terminal, colocarse en el directorio donde se encuentran todos los scripts SQL y ejecutar las siguientes instrucciones.

- Tabla Agencia

```
curl "https://api.mockaroo.com/api/d41216c0?count=50&key=b71bb7e0" > s-03-agencia.sql
```

- Tabla Cliente

```
curl "https://api.mockaroo.com/api/14412eb0?count=500&key=b71bb7e0" > "s-03-cliente.sql"
```

- Tabla tarjeta_cliente

```
curl "https://api.mockaroo.com/api/65b0bc50?count=40&key=b71bb7e0" > "s-03-tarjeta-cliente.sql"
```

- Tabla auto

```
curl "https://api.mockaroo.com/api/05a3dd20?count=100&key=b71bb7e0" > "s-03-auto.sql"
```

- Tabla Status_auto

```
curl "https://api.mockaroo.com/api/213b2550?count=6&key=b71bb7e0" > "s-03-status-auto.sql"
```

- Tabla auto_carga

```
curl "https://api.mockaroo.com/api/67b8efb0?count=50&key=b71bb7e0" > "s-03-auto-carga.sql"
```

- Tabla auto_particular

```
curl "https://api.mockaroo.com/api/9f2a1360?count=50&key=b71bb7e0" > "s-03-auto-particular.sql"
```

- Tabla Historico_status_auto

```
curl "https://api.mockaroo.com/api/5f670810?count=1000&key=b71bb7e0" > "s-03-historico-status-auto.sql"
```

- Tabla pago_auto

```
curl "https://api.mockaroo.com/api/7b763d50?count=100&key=b71bb7e0" > "s-03-pago-auto.sql"
```

1.2.4. Ejecución de scripts.

- Crear un script `s-00-main.sql` que invoque a cada uno de los scripts anteriores empleando los usuarios correspondientes. Ejecutar el script, verificar que no existan errores.

Ejemplo:

```
--@Autor:           Jorge Rodriguez
--@Fecha creación:  dd/mm/yyyy
--@Descripción:     Archivo principal

--si ocurre un error, se hace rollback de los datos y
--se sale de SQL *Plus
whenever  sqlerror exit rollback

Prompt creando usuario jrc p1201 autos
@s-01-creacion-usuario.sql

Prompt conectando como usuario jrc_p1201_autos
connect jrc p1201 autos

Prompt creando objetos
@s-02-autos-ddl.sql

Prompt realizando la carga de datos
@s-03-agencia.sql
@s-03-cliente.sql
@s-03-tarjeta-cliente.sql
@s-03-status-auto.sql
@s-03-auto.sql
@s-03-auto-carga.sql
@s-03-auto-particular.sql
@s-03-historico-status-auto.sql
@s-03-pago-auto.sql

Prompt confirmando cambios
commit;

--Si se encuentra un error, no se sale de SQL *Plus
--no se hace commit ni rollback, es decir, se
--regresa al estado original.
whenever sqlerror continue none

Prompt Listo!
```

1.3. EJERCICIOS DE PROGRAMACIÓN

1.3.1. Ejercicio 1: Registrando pagos faltantes de autos.

La empresa ha detectado que el pago número 1 de algunos autos cuyo id está en el rango [1,100] no fue registrado. Para corregir el problema se requiere generar un procedimiento almacenado llamado `p_corrige_pagos`. El programa deberá realizar las siguientes acciones:

- Deberá iterar en el rango de identificadores indicado anteriormente: 1 a 100.
- Para cada identificador, el programa deberá validar que el pago numero 1 no fue registrado. Se recomienda la siguiente instrucción:

```
select count(*) into v_pago_faltante
from pago auto
where auto_id = v_indice
and num_pago = 1;
```

En la instrucción anterior, se realiza el conteo de los registros donde el número de pago es 1 y se asigna a la variable `v_pago_faltante`. Si el valor de dicha variable es cero, significa que el pago 1 no fue registrado. La variable `v_indice` corresponde con en número de iteración actual y se emplea para hacer referencia al identificador del auto.

- En caso de no existir el pago, el programa deberá insertar un nuevo registro con importe = 2500, num. Pago = 1, y fecha de pago = fecha del sistema.
- Al final de la ejecución el programa deberá imprimir el número de registros insertados y el total de los importes registrados:

```
----------Validacion concluida ------------
Numero de pagos faltantes:              50
Importe total:                      125000
```

- No olvidar las instrucciones `set serveroutput on` para habilitar los mensajes en consola y la instrucción `show errors` al final del procedimiento para validar posibles errores de compilación:

```
--@Autor:           Jorge Rodriguez
--@Fecha creación:  dd/mm/yyyy
--@Descripción:     Script encargado de validar e insertar  pagos

--Habilita la salida de mensajes dbms_output.put_line
set serveroutput on
create or replace procedure p_corrige_pagos is
 --completar
begin
 --completar
end;
/
--En caso de existir errores de compilación los muestra.
show errors
```

- El programa deberá estar contenido en un script llamado `s-04-ejercicio-pago-autos.sql`
- Ejecutar el procedimiento almacenado desde SQL *Plus empleando la instrucción `exec`
- Si todo es exitoso, hacer `commit` para que los cambios realizados por el procedimiento sean permanentes.
- **C1. Incluir en el reporte únicamente** el código del script debidamente formateado y el resultado de ejecutar el comando `exec`.

1.3.2. Ejercicio 2: Registro de autos nuevos.

Crear un script `s-05-ejercicio-crea-auto.sql` El script deberá contener un procedimiento almacenado llamado `p_crea_auto` encargado de registrar un auto nuevo:

- Deberá recibir los siguientes parámetros en el orden indicado: `p_auto_id`, `p_marca`, `p_modelo`, `p_anio`, `p_num_serie`, `p_tipo`, `p_precio`, `p_agencia_id`, `p_num_cilindros`, `p_num_pasajeros`, `p_clase`, `p_peso_maximo`, `p_volumen`, `p_tipo_combustible`
- El status y su fecha no se deberán solicitar, se le deberá asignar el valor 2: En Agencia.
- El identificador del auto deberá ser un parámetro tanto de salida como de entrada ya que el programa deberá asignarle el siguiente valor de la secuencia `auto_seq` la cual fue creada en el script de creación de objetos.
- El valor para la foto será un campo binario vacío, es decir, asignarle el valor que regresa la función `empty_blob()`. El auto no tendría cliente asignado.
- El procedimiento deberá crear el nuevo auto, así como los datos particulares con base a su tipo (de carga o particular).
- El procedimiento deberá validar que el tipo de auto sea correcto 'P' o 'C'. De ser incorrecto deberá generar una excepción con código -20010 y un mensaje que indique el error.
- Adicionalmente, el procedimiento deberá ingresar una entrada en el histórico de status. Hacer uso de la secuencia `historico_status_auto_seq`
- En un programa PL/SQL es posible emplear la siguiente instrucción para lanzar una excepción.

```
raise_application_error(-20001,'<mensaje/detalle del error>');
```

- Ejecutar el script, verificar que no existan problemas de compilación.
- **C2. Incluir en el reporte** únicamente el contenido del script.

Se recomienda crear un script con un bloque PL/SQL anónimo encargado de invocar al procedimiento almacenado anteriormente para verificar resultados. Opcionalmente se puede revisar con el validador mismo que se describe más adelante.

Ejemplo:

```
--@Autor:           Jorge Rodriguez
--@Fecha creación:  dd/mm/yyyy
--@Descripción:     Bloque anonimo que crea un auto
set serveroutput on
Prompt insertando un nuevo auto
declare
 v_auto_id auto.auto_id%type;
begin
 p_crea_auto(v_auto_id,'GMC','Suburban',2018,'SDJWEH2839','P',500004.35,1,8,10,'B',null,null,null);
 dbms_output.put_line('Auto creado con éxito, id: '||v_auto_id);
end;
/
--haciendo commit
commit;
```

- Emplear datos distintos a los mostrados en el ejemplo.
- Ejecutar el script.

1.3.1. Ejercicio 3: Actualización de status de autos

Se ha decidido implementar un trigger para validar que los cambios de status de un auto sean congruentes. El trigger debe validar el cumplimiento de las siguientes reglas de negocio justo antes de insertar un nuevo auto o antes de modificar su status.

- Si el status se establece a EN TRANSITO, EN AGENCIA o DEFECTUOSO, el trigger deberá validar que no existan pagos registrados asociados al auto.
- En caso de detectar pagos, el trigger deberá impedir la inserción o modificación. Para ello, deberá lanzar una excepción con código -20001 y un mensaje que indique que un auto con estos status no puede contar con pagos registrados.
- Si el status se establece a APARTADO, deberá existir un único pago registrado. En caso de no existir el trigger deberá generar una excepción con código -20002 y su correspondiente mensaje.
- El monto del pago del auto con status APARTADO detectado en el punto anterior servirá para aplicar un descuento al precio total del auto. El trigger deberá registrar el 50% del pago registrado como valor del campo `descuento` en la tabla `auto`. Tip: para asignar el valor de este atributo emplear la siguiente expresión `:new.descuento = 0.5*v_importe_pago;`
- Si el status del auto se establece a vendido, el trigger deberá validar que el total de los pagos registrados cubra el precio del auto. En caso contrario, el trigger deberá generar una excepción con código -20003 y su mensaje correspondiente.
- Finalmente, si el status del auto es establecido a EN REPARACION, el status anterior debe ser VENDIDO. Es decir, para que un auto sea reparado, este debió haber sido vendido anteriormente. De no cumplir con esta regla, el trigger deberá lanzar una excepción con código -20004.
- Crear un script `s-06-ejercicio-valida-status-auto.sql` que contenga la definición del trigger con nombre `tr_valida_status_auto`
- **C3. Incluir en el reporte** únicamente el contenido del script. Se recomienda realizar pruebas con un programa PL/SQL anónimo para validar el correcto funcionamiento del trigger o en su defecto probar el trigger con el validador mismo que se describe más adelante.

1.3.2. Ejercicio 4: Asignación de autos en una agencia.

Se ha decidido implementar la siguiente regla de negocio a través del uso de un trigger. Cada vez que se crea o se modifica la agencia a la que se asigna un auto, se debe validar que la agencia tenga espacio para resguardar al auto en cuestión. En caso de no contar con cupo, se intentará asignar el auto a la agencia anexa (en caso de existir y en caso de tener lugares disponibles). Si el segundo intento falla, el trigger deberá impedir que se registre o actualice la agencia del auto. Para realizar esta acción, el trigger deberá lanzar una excepción con código de error -20005 y con un mensaje que describa el error. En caso de encontrar cupo en la agencia alterna, el trigger deberá actualizar el valor del campo `agencia_id`.
Para efectos de la práctica, asumir que en cada agencia se pueden resguardar hasta 5 autos.

Debido a que se requieren hacer consultas (conteo de registros) sobre la misma tabla que genera el evento, este trigger puede generar el problema de "Tabla mutante". Para evitar este inconveniente, se deberá crear un trigger DML compuesto. La estrategia para construir este trigger es la siguiente:

- La consulta para saber el número de autos que están asociados a una agencia NO podrá realizarse en el bloque `before each row`. Esto con la finalidad de evitar el error de "Tabla mutante". Dicha consulta se realizará en el bloque `after statement`. En este mismo bloque se deberá programar la lógica para determinar si hay cupo, si se cambia a la anexa o si se lanza excepción.
- Existe un inconveniente: las variables :new y :old no existen en el bloque `after statement`, solo existen en el bloque `before each row`.
- Para resolver este detalle, en el bloque `before for each row` se deberán recuperar los valores de `:new.auto_id`, `:new.agencia_id` y el valor de `agencia_anexa_id` en caso de existir una agencia anexa.
- Un punto más a considerar es que una sentencia `update` podría modificar la agencia de varios autos en una sola instrucción. Esto implica que se tendría una lista de autos a validar ya que el bloque `before each row` se va a ejecutar tantas veces como registros hayan sido afectados y el bloque `after statement` solo se ejecuta una vez.
- Para resolver este último punto, en el bloque `before each row` se deberán guardar en "una lista" los valores de `:new.auto_id`, `:new.agencia_id` `agencia_anexa_id` por cada registro afectado. Se deberán emplear colecciones.
- Finalmente, en el bloque `after statement` de deberá iterar dicha lista y aplicar la lógica de programación antes mencionada.

En general, el trigger se verá así:

```
--@Autor:           Jorge Rodriguez
--@Fecha creación:  dd/mm/yyyy
--@Descripción:     trigger encargado de validar la asignacion de agencias
set serveroutput on
create or replace trigger tr valida asignacion agencia
 for insert or update of agencia_id on auto
 compound trigger

 --sección común
 --declara un objeto type para guardar los valores que se van a validar en una colección
 --cada elemento de la colección contendrá 3 atributos_ auto_id, agencia_id y agencia_anexa_id.
 --estos 3 atributos serán empleados para validar la asignación de agencia.
 type agencia_auto_type is record (
  auto_id auto.auto_id%type,
  agencia_id agencia.agencia_id%type,
  agencia_anexa_id agencia.agencia_anexa_id%type
 );

 --Crea un objeto tipo collection para almacenar los objetos
 type agencia_auto_list_type is table of agencia_auto_type;

 --Crea una colección y la inicializa.
 v_lista agencia_auto_list_type := agencia_auto_list_type();

 ------bloque before --------------
 --aqui solamente se llena la colección por cada registro afectado---
 before each row is

  v_index number;
  v agencia anexa id agencia.agencia id%type;

 begin
  --asigna espacio a la colección
  v_lista.extend;
  --obtiene el índice siguiente para guardar los datos
  v_index := v_lista.last;
  --obtiene a la agencia anexa.
  select agencia_anexa_id into v_agencia_anexa_id
  from agencia
  where agencia_id = :new.agencia_id;
  dbms output.put line('T: Nuevo elemento en coleccion para auto id: '
   || :new.auto_id);
  v lista(v index).auto id := :new.auto id;
  v_lista(v_index).agencia_id := :new.agencia_id;
  v_lista(v_index).agencia_anexa_id := v_agencia_anexa_id;

 end before each row;

  ----- bloque after statement
  -- en este bloque se deberá recorrer la colección creada anteriormente
  -- y aplicar la lógica para validar la asignación de  las agencias.
 after statement is
  v num autos number(1,0);
  v_agencia_id  agencia.agencia_id%type;
  v agencia anexa id  agencia.agencia anexa id%type;
  v_auto_id auto.auto_id%type;

 begin
  for i in v_lista.first .. v_lista.last loop
     --se obtienen los valores de las 3 variables contenidas en la colección.
     v_agencia_id := v_lista(i).agencia_id;
     v_agencia_anexa_id := v_lista(i).agencia_anexa_id;
     v_auto_id := v_lista(i).auto_id;

     --completar aquí la lógica para validar la asignación de agencia.

  end loop;
 end after statement;
end;
/
show errors
```

- Se recomienda hacer uso del validador para confirmar el correcto funcionamiento del trigger.
- Generar un script llamado `s-07-ejercicio-valida-asignacion-agencia.sql` que contenga la definición del trigger compuesto llamado `tr_valida_asignacion_agencia`
- **C4. Incluir en el reporte únicamente** el código que se requiere para completar este trigger.

1.3.3. Ejercicio 5: Función para exportar datos.

Para efectos del sistema web que tiene la agencia, se requiere generar una cadena en formato CSV (campos separados por coma) con la siguiente información:

Auto_id	Num. serie	tipo	precio	Num. cilindros	Peso maximo	Num. Pago	Importe pago	Email cliente
1	35354	P	33456	4	N/A	1	7648.45	c@m.com
1	35354	P	33456	4	N/A	2	7648.45	c@m.com

- Para implementa lo anterior, crear una función llamada `exporta_datos_auto_csv_fx` que acepte como parámetro el identificador del auto. La función deberá regresar una cadena en formato CSV con la información mostrada en la tabla.
- En caso de que un valor sea nulo, se deberá incluir la cadena N/A
- Crear un script llamado `s-08-ejercicio-exporta-datos-csv.sql` que contenga la definición de la función
- **C5. Incluir en el reporte únicamente** el código dela función.

1.4. VALIDACIÓN DE RESULTADOS.

- Obtener todos los archivos con extensión `plb` de la carpeta compartida correspondiente a la práctica.
- Ejecutar el script `s-09-valida-ejercicios.plb`, similar a cualquier script SQL. Proporcionar los datos solicitados.
- Los archivos `plb` pueden ser ejecutados de forma separada para validar cada uno de los ejercicios de la práctica.
- **C6. Incluir en el reporte únicamente** la salida del script a partir del punto indicado.

1.5. CONTENIDO DEL REPORTE

- Introducción
- Objetivo
- Desarrollo de la práctica.
  - C1 – C5 Scripts con el código del trigger/función, script de prueba y salida de ejecución.
  - C6. Salida del script de validación.
- Conclusiones, comentarios, experiencias y/o sugerencias.
- Bibliografía.