PRACTICA COMPLEMENTARIA 12
PROGRAMACIÓN CON SQL PARTE 1

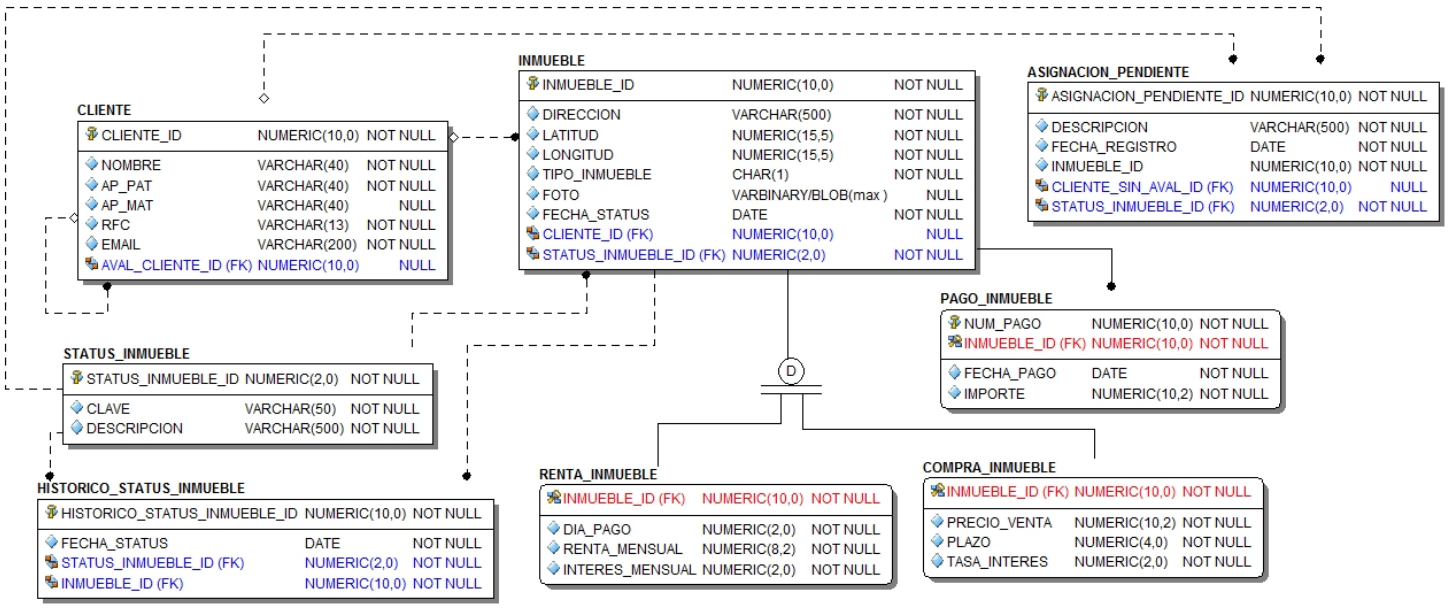
El reporte se entrega en equipos máximo 2 personas.

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 llevar el control de la venta y renta de inmuebles.



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>_p1203_inmuebles`, 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, create sequence.`

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_P1203_INMUEBLES';
  if v_count > 0 then
    dbms_output.put_line('Eliminando usuario existente');
    execute immediate 'drop user jrc_p1203_inmuebles cascade';
  end if;
end;
/

create user jrc_p1203_inmuebles identified by jorge quota unlimited on users;
grant create session, create table, create procedure, create sequence,
create trigger to jrc_p1203_inmuebles;
```

1.2.2. Creación de objetos

- De la carpeta compartida correspondiente a esta práctica, descargar un script llamado `s-02-inmuebles-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.

```
curl "https://api.mockaroo.com/api/6ff13690?count=500&key=b71bb7e0" > s-03-carga-cliente.sql
curl "https://api.mockaroo.com/api/b97d2aa0?count=5&key=b71bb7e0"   > s-03-carga-status-inmueble.sql
curl "https://api.mockaroo.com/api/c929d5a0?count=250&key=b71bb7e0" > s-03-carga-inmueble.sql
curl "https://api.mockaroo.com/api/ecad28c0?count=125&key=b71bb7e0" > s-03-carga-renta-inmueble.sql
curl "https://api.mockaroo.com/api/de641a50?count=125&key=b71bb7e0" > s-03-carga-compra-inmueble.sql
```

- Obtener los siguientes scripts de la carpeta compartida de esta práctica que realizan carga de datos adicionales.

```
s-03-carga-pagos.sql
s-03-carga-historico.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_p1203_inmuebles
@s-01-creacion-usuario.sql

Prompt conectando como usuario jrc_p1203_inmuebles
connect jrc_p1203_inmuebles

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

set define off
```

```
!curl "https://api.mockaroo.com/api/6ff13690?count=500&key=b71bb7e0" > s-03-cliente.sql
!curl "https://api.mockaroo.com/api/b97d2aa0?count=5&key=b71bb7e0" > s-03-status-inmueble.sql
!curl "https://api.mockaroo.com/api/c929d5a0?count=250&key=b71bb7e0" > s-03-inmueble.sql
!curl "https://api.mockaroo.com/api/ecad28c0?count=125&key=b71bb7e0" >  s-03-renta-inmueble.sql
!curl "https://api.mockaroo.com/api/de641a50?count=125&key=b71bb7e0" >  s-03-compra-inmueble.sql

-- Se modifica el formato por default de la fecha para realizar la carga de datos.
alter session set nls_date_format="yyyy-mm-dd hh24:mi:ss";

Prompt realizando la carga de datos
@s-03-cliente.sql
@s-03-status-inmueble.sql
@s-03-inmueble.sql
@s-03-renta-inmueble.sql
@s-03-compra-inmueble.sql
@s-03-carga-pagos.sql
@s-03-carga-historico.sql

set define on

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: Registro de inmuebles nuevos.

Crear un script `s-04-ejercicio-crea-inmueble.sql` El script deberá contener un procedimiento almacenado llamado `sp_crea_inmueble` encargado de registrar un inmueble nuevo:

- Deberá recibir los siguientes parámetros con base a la siguiente estructura:

```
p_inmueble_id in out number,
p_direccion varchar2,
p_latitud number,
p longitud number,
p_tipo char,
p_cliente_id number default null,
p_dia_pago number default null,
p_renta_mensual number default null,
p_interes_mensual number default null,
p_precio_venta number default null,
p_plazo number default null,
p_tasa_interes number default null
```

- El status y su fecha no se deberán solicitar, se le deberá asignar el valor 1: `DISPONIBLE`.
- El identificador del inmueble deberá ser un parámetro tanto de salida como de entrada ya que el programa deberá asignarle el siguiente valor de la secuencia `seq_inmueble` 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 inmueble no tendría cliente asignado.
- Observar que algunos parámetros pueden ser nulos. Por ejemplo, si se crea un inmueble para compra, todos los valores del otro subtipo deberán ser nulos. El uso de `default null` permite omitir la asignación de valores nulos al momento de invocar al procedimiento. En el siguiente ejemplo, se invoca al procedimiento únicamente con los parámetros necesarios para crear un inmueble para compra:

```
declare
 v_inmueble_id number;
begin

 sp_crea_inmueble (
    p_inmueble_id  => v_inmueble_id,
    p_direccion    => 'direccion inmueble',
    p_latitud      => 38.7097954,
    p_longitud     =>-9.4164575,
    p_tipo         => 'C',
    p_precio_venta => 2700000,
    p_plazo        => 36,
    p_tasa_interes => 0.5
  );

 dbms_output.put_line('inmueble creado: '||v_inmueble_id);

end;
/
--haciendo commit
commit;
```

- Observar que el procedimiento solo se invoca con los parámetros que se necesitan. Para los demás, su valor será el valor por default asignado.
- Observar la notación `<nombre_parametro> => <valor_parametro>`. Este estilo permite invocar al procedimiento especificando los valores de sus parámetros en cualquier orden. El nombre del parámetro debe coincidir con el nombre empleado en la instrucción `create or replace procedure`.
- El procedimiento deberá crear el nuevo inmueble, así como los datos particulares con base a su tipo (renta o compra).
- El procedimiento deberá validar que el tipo de inmueble sea correcto 'R' 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 `seq_hist_status_inmueble`

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. No olvidar anexar la instrucción `show errors`.
- Se recomienda crear un script con un bloque PL/SQL anónimo similar al anterior, encargado de invocar al procedimiento almacenado anteriormente para verificar resultados. Opcionalmente se puede revisar con el validador mismo que se describe más adelante.

1.3.2. Ejercicio 2: Actualizando pagos de inmuebles.

La empresa ha detectado que existen inmuebles que fueron comprados (tipo = C) con clave de status `PAGADO` con un número incorrecto de pagos registrados. Deben existir N pagos dependiendo el valor del campo `plazo` el cual está expresado en semanas. Considerar el siguiente ejemplo:

Suponer que el inmueble 10 tiene un costo de $8,065,854.79 y el valor del campo plazo es de 5 semanas. Suponer que la fecha de status del inmueble es 01/06/2005 09:02:35. La lista de pagos que debería existir en la base de datos es:

Inmueble_id	Num_pago	Fecha_pago	Importe
10	1	01/**02**/2005 09:02:35	1613170.95
10	2	01/**03**/2005 09:02:35	1613170.95
10	3	01/**04**/2005 09:02:35	1613170.95
10	4	01/**05**/2005 09:02:35	1613170.95
10	5	01/**06**/2005 09:02:35	1613170.**99**

- Observar que se generan 5 registros debido a que se tienen 5 meses para pagar el inmueble (plazo = 5)
- Notar que la fecha de cada pago inicia 4 meses antes con respecto a la fecha de status de tal forma que la fecha del último pago corresponda con la fecha del status del inmueble. Tip: emplear la función `add_months(fecha_status,-1*(plazo-1))`. Esta expresión permite regresar N-1 meses atrás una fecha, donde N representa el plazo que se le otorgó al cliente para pagar su inmueble.
- Observar que el último pago tiene 4 centavos más. Los pagos parciales se registran truncados a 2 decimales. Este truncamiento puede provocar que la suma de los pagos no sea exacta con respecto al precio de compra. Para resolver esta pequeña diferencia, en el último pago se le deberá agregar esta diferencia redondeada a 2 dígitos. Si se suman las 5 cantidades, se obtendrá el costo total exacto.
- Crear un procedimiento `sp_corrige_pago_inmuebles` que realice esta corrección de pagos.
    - Recibirá los siguientes parámetros en orden: identificador del inmueble a revisar (parámetro de entrada), bandera que indica si los pagos del inmueble fueron actualizados: (1 = sus pagos fueron actualizados, 0 sus pagos no fueron actualizados).

- El programa deberá validar que el número de pagos existentes sea igual al valor del campo `plazo`. Si esta condición no se cumple, los pagos del inmueble deberán actualizarse. En este caso, los pagos existentes deberán eliminarse para ser corregidos.

Ejemplo:

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

set serveroutput on
create or replace procedure sp_corrige_pago_inmuebles(p_inmueble_id number,
  p actualiado out number) is

 --completar
begin
 --completar
end;
/
show errors
```

- El programa deberá estar contenido en un script llamado `s-04-ejercicio-pago-inmuebles.sql`
- Para probar el procedimiento, se recomienda crear un bloque anónimo.
- Si todo es exitoso, hacer `commit` para que los cambios realizados por el procedimiento sean permanentes.

```
set serveroutput on
declare
  v_inmueble_id number := 207;
  v_actualizado number(1,0);
begin

  sp_corrige_pago_inmuebles(v_inmueble_id,v_actualizado);

  dbms_output.put_line(
    'Procedimiento invocado, resultado (inmueble_id,actualizado): '
      ||v_inmueble_id
      ||','
      ||v_actualizado
  );
end;
/
```

- **C1. Incluir en el reporte únicamente** el código del script debidamente formateado.

1.3.1. Ejercicio 3: Revisión de los avales de un inmueble.

Se ha decidido implementar un trigger para realizar la correcta asignación de un cliente a un inmueble. Cuando se le asigne un cliente a un inmueble, ya sea al momento de su registro, o al aplicar un cambio, el trigger deberá revisar las siguientes reglas:

- Si el cliente es nulo, el trigger deberá verificar que el status del inmueble sea DISPONIBLE (`status_inmueble_id = 1`) , es decir, el inmueble se encuentra disponible. Cualquier otro status se considera incorrecto, por lo que el trigger deberá lanzar un error con código -20010 indicando la razón del error.
- Si el cliente es nulo y el status del inmueble es DISPONIBLE, significa que el inmueble está disponible para ser asignado a un cliente. En este caso no se requiere hacer validación de aval, el programa termina permitiendo la actualización o modificación.
- Si el cliente es no nulo, y el status del inmueble es DISPONIBLE, se deberá lanzar error -20011 indicando que un inmueble disponible no puede tener un cliente asignado.

Finalmente, si el cliente es no nulo y tiene un status diferente a DISPONIBLE, se requiere validar su aval con base a las siguientes reglas:

- El nuevo cliente asignado debe contar con un aval. Dicho aval debe contar con al menos un inmueble con status PAGADO. Esto garantizará el respaldo del aval hacia el cliente.
- La única excepción a la regla anterior es que el status del inmueble a validar sea PAGADO. En ese caso el programa termina ya que no se requiere aval.
- En caso de no cumplir con estas condiciones, el trigger deberá registrar los datos del inmueble en `asignacion_pendiente`. En el campo `descripcion` se deberá indicar que el cliente no cuenta con un aval válido. Posterior a la inserción, el trigger deberá lanzar un error con código -20012 indicando el error, y adicionalmente, indicar que se ha generado un nuevo registro en `asignacion_pendiente`. Hacer uso de una secuencia llamada `seq_asignacion_pendiente` para generar los valores de la PK.
- De no ocurrir error, el trigger permitirá la inserción o actualización.
- El trigger deberá estar en un `script s-06-ejercicio-revisa-aval.sql`
- Se recomienda crear un bloque PL/SQL anónimo para probar el trigger.

- **C2. Incluir en el reporte únicamente** el código del script debidamente formateado.

1.4. VALIDACIÓN DE RESULTADOS.

En esta actividad se realizará la validación de las respuestas del ejercicio anterior. Para ello, realizar las siguientes acciones:

- En la carpeta compartida de la práctica obtener todos los scripts `sql/plb`.
- Editar el script `s-07-validador-main.sql` con los valores correspondientes
- En una nueva terminal cambiarse al directorio donde se encuentran los scripts, y ejecutar el script editado (no se requiere emplear al usuario Oracle).

```
sqlplus /nolog
start s-07-validador-main.sql
```

- En caso de existir errores, revisar y leer cuidadosamente los mensajes de error, corregir y reintentar. **C3. Incluir en el reporte** la captura de pantalla con la salida del validador.

1.5. CONTENIDO DEL REPORTE.

Para realizar la evaluación de la práctica se deberá anexar en la última página del reporte la rúbrica correspondiente:

- Rúbrica para el grupo de laboratorio.
- Rúbrica para el grupo de teoría plan 2010
- Rúbrica para grupo de teoría plan 2016 inscritos en otro grupo de laboratorio

Imprimir alguna de las siguientes páginas de este documento e incluirla en el reporte. Prácticas que no incluyan esta tabla se considerarán como prácticas no entregadas. La rúbrica permite conocer a detalle los criterios empleados para asignar la calificación final.

PRÁCTICA 12

Rúbrica para grupo del laboratorio

Contenido	Penalizaciones	Puntaje máximo	Observaciones
Carátula *	-5P	**5P**	
objetivos e Introducción *	-5P	**5P**	
Actividades Práctica complementaria			
C1. Código ejercicio 2	-10P Incompleto, ausente o mal formato	**10P**	• Si se detecta copia de sentencias la práctica se anula.
C2. Código ejercicio 3	-10P Incompleto, ausente o mal formato	**10P**	• Si se detecta copia de sentencias la práctica se anula
C3. Resultado del validador.	-10P por cada error detectado	**60P**	• La práctica no se evalúa si no se incluye la salida del validador.
Conclusiones, comentarios, recomendaciones *	-5P	**5P**	
Bibliografía *	-5P	**5P**	

* Ver Rubrica general de prácticas para mayores detalles en cuanto a los requisitos que debe cumplir el elemento de evaluación y los puntajes asignados.

PRACTICA 12

Rúbrica para grupo de teoría plan 2010

Contenido	Penalizaciones	Puntaje máximo	Observaciones
Carátula *	-5P	**5P**	
objetivos e Introducción *	-5P	**5P**	
Actividades Práctica complementaria			
C1. Código ejercicio 2	-10P Incompleto, ausente o mal formato	**10P**	• Si se detecta copia de sentencias la práctica se anula.
C2. Código ejercicio 3	-10P Incompleto, ausente o mal formato	**10P**	• Si se detecta copia de sentencias la práctica se anula
C3. Resultado del validador.	-10P por cada error detectado	**60P**	• La práctica no se evalúa si no se incluye la salida del validador.
Conclusiones, comentarios, recomendaciones *	-5P	**5P**	
Bibliografía *	-5P	**5P**	

* Ver Rubrica general de prácticas para mayores detalles en cuanto a los requisitos que debe cumplir el elemento de evaluación y los puntajes asignados.

PRÁCTICA 12

Rúbrica para grupo de teoría plan 2016 inscritos en otro grupo de laboratorio

Contenido	Penalizaciones	Puntaje máximo	Observaciones
Carátula *	-5P	**5P**	
Actividades Práctica complementaria			
C1. Código ejercicio 2	-10P Incompleto, ausente o mal formato	**10P**	• Si se detecta copia de sentencias la práctica se anula.
C2. Código ejercicio 3	-10P Incompleto, ausente o mal formato	**10P**	• Si se detecta copia de sentencias la práctica se anula
C3. Resultado del validador.	-10P por cada error detectado	**70P**	• La práctica no se evalúa si no se incluye la salida del validador.
Conclusiones, comentarios, recomendaciones *	-5P	**5P**	

* Ver Rubrica general de prácticas para mayores detalles en cuanto a los requisitos que debe cumplir el elemento de evaluación y los puntajes asignados.