PROYECTO FINAL DE CURSO “MySQL”

Joaquin Pessina

PROYECTO FINAL DE CURSO “MySQL”

Este proyecto tiene como objetivo principal brindar una solución sencilla, eficiente y accesible para el análisis de datos poblacionales. Con la implementación de una base de datos MySQL, se busca mejorar la legibilidad y accesibilidad de los datos, así como facilitar la tarea de leer y analizar diferentes aspectos de cada área poblacional.

Uno de los principales desafíos del proyecto fue lograr la correcta implementación de las tablas interconectadas, para maximizar la eficiencia del script y garantizar el correcto funcionamiento del sistema. Además, fue necesario desarrollar un script adecuado para cumplir con todas las funciones requeridas.

El objetivo de este proyecto es ofrecer una herramienta confiable y eficiente para la gestión de datos poblacionales. A través de la base de datos MySQL, se busca proporcionar una plataforma fácil de usar y navegar, que permita a los usuarios analizar y compilar información relevante para sus necesidades. El sistema también tiene como objetivo simplificar la tarea de recopilar y actualizar datos poblacionales en tiempo real.

El modelo de negocio de este proyecto es muy versátil y adaptable a las necesidades del cliente. La base de datos MySQL puede ser utilizada por diversas empresas y organizaciones, como empresas de investigación de mercado, gobiernos locales, organizaciones sin fines de lucro y otras entidades interesadas en analizar datos poblacionales. Al ofrecer una herramienta sencilla y eficiente para el análisis de datos, este proyecto tiene el potencial de ser una solución valiosa para cualquier organización que necesite información detallada sobre las tendencias y patrones poblacionales en diferentes áreas geográficas.

Herramientas y tecnologías:

Lenguaje de consulta estructurado (SQL): un lenguaje de programación utilizado para interactuar con la base de datos y realizar consultas y modificaciones de datos.

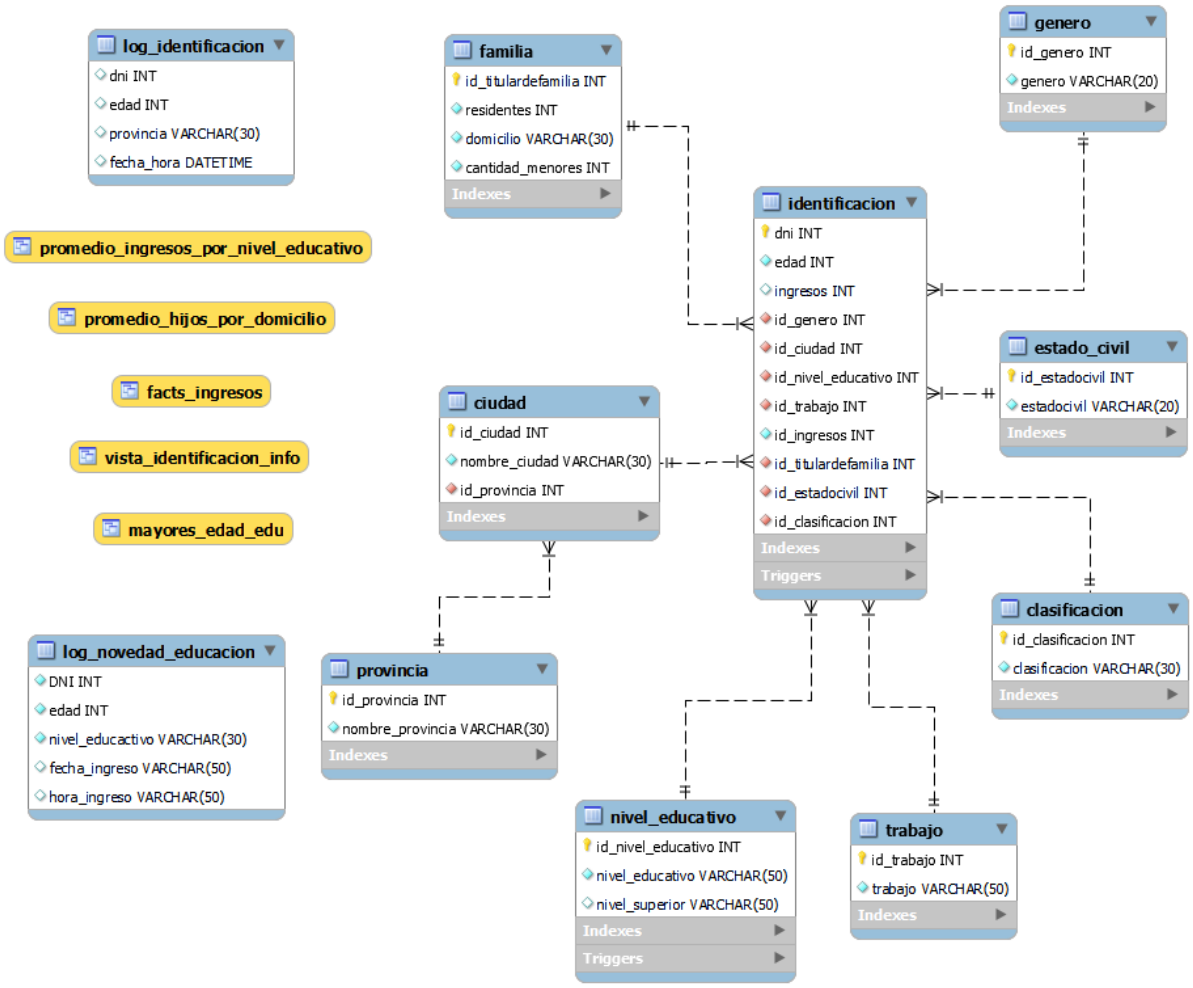
MySQL: un sistema de gestión de bases de datos relacional (RDBMS) utilizado para almacenar y gestionar datos en el proyecto.

GitHub: una plataforma de alojamiento de código utilizada para alojar el repositorio del proyecto y colaborar con otros miembros del equipo.

MySQL Workbench: una herramienta de modelado de datos utilizada para diseñar y visualizar el esquema de la base de datos y las relaciones entre las tablas.

Microsoft Excel: herramientas de hojas de cálculo utilizadas para analizar y visualizar los datos extraídos de la base de datos.

Diagrama de Entidad - Relación



A continuación hago una descripción de cada columna de cada tabla particular

Tabla: clasificación

Columna	Tipo de dato	Descripción
id_clasificacion	int	Identificador único de la clasificación
clasificacion	varchar(30)	Nombre de la clasificación

Tabla: trabajo

Columna	Tipo de dato	Descripción
id_trabajo	int	Identificador único del trabajo
trabajo	varchar(50)	Nombre del trabajo

Tabla: provincia

Columna	Tipo de dato	Descripción
id_provincia	int	Identificador único de la provincia
nombre_provincia	varchar(30)	Nombre de la provincia

Tabla: ciudad

Columna	Tipo de dato	Descripción
id_ciudad	int	Identificador único de la ciudad
nombre_ciudad	varchar(30)	Nombre de la ciudad
id_provincia	int	Identificador único de la provincia a la que pertenece la ciudad, enlazado con la tabla provincia

Tabla: genero

Columna	Tipo de dato	Descripción
id_genero	int	Identificador único del género
genero	varchar(20)	Nombre del género

Tabla: nivel_educativo

Columna	Tipo de dato	Descripción
id_nivel_educativo	int	Identificador único del nivel educativo
nivel_educativo	varchar(50)	Nombre del nivel educativo
nivel_superior	varchar(50)	Nombre del nivel educativo superior (si aplica)

Tabla: Familia

Columna	Descripción
id_titulardefamilia	Identificador único de cada familia, utilizado como clave primaria.
residentes	Número de personas que viven en la familia.
domicilio	Dirección del domicilio de la familia.
cantidad_menores	Número de menores de edad que viven en la familia.

Tabla: Estado_civil

Columna	Descripción
id_estadocivil	Identificador único de cada estado civil, utilizado como clave primaria.
estadocivil	Descripción del estado civil de la persona (soltero/a, casado/a, viudo/a, divorciado/a, separado/a)

Tabla: Identificación

Columna	Descripción
dni	Identificación única de la persona, utilizada como clave primaria.
edad	Edad de la persona.
ingresos	Ingresos de la persona.
id_genero	Identificador único del género de la persona, utilizado como clave foránea que hace referencia a la tabla Genero.
id_ciudad	Identificador único de la ciudad de la persona, utilizado como clave foránea que hace referencia a la tabla Ciudad.
id_nivel_educativo	Identificador único del nivel educativo de la persona, utilizado como clave foránea que hace referencia a la tabla Nivel_educativo.
id_trabajo	Identificador único del trabajo de la persona, utilizado como clave foránea que hace referencia a la tabla Trabajo.
id_ingresos	Identificador único de la clasificación de ingresos de la persona, utilizado como clave foránea que hace referencia a la tabla Clasificacion.
id_titulardefamilia	Identificador único del titular de familia de la persona, utilizado como clave foránea que hace referencia a la tabla Familia.
id_estadocivil	Identificador único del estado civil de la persona, utilizado como clave foránea que hace referencia a la tabla Estado_civil.
id_clasificacion	Identificador único de la clasificación de la persona, utilizado como clave foránea que hace referencia a la tabla Clasificacion. Esta columna es utilizada para categorizar a la persona en una de las siguientes categorías: trabajo formal, trabajo informal, trabajo independiente, desempleo o estudio.

/*insert datas*/

```
insert into clasificacion (clasificacion) values
('Trabajo formal'),
('Trabajo informal'),
('Trabajo independiente'),
('Desempleo'),
('Estudio');
```

```
insert into trabajo (trabajo) values
('Abogado'),
('Ingeniero'),
('Médico'),
('Docente'),
('Programador');

insert into provincia (nombre_provincia) values
('Buenos Aires'),
('Córdoba'),
('Santa Fe'),
('Mendoza'),
('Entre Ríos');

insert into ciudad (nombre_ciudad, id_provincia) values
('La Plata', 1),
('Rosario', 3),
('Mendoza', 4),
('Córdoba', 2),
('Paraná', 5);

insert into genero (genero) values
('Masculino'),
('Femenino'),
('Otro'),
('No especificado'),
('Prefiero no decirlo');

insert into nivel_educativo (nivel_educativo, nivel_superior) values
('Educación Primaria', NULL),
('Educación Secundaria', NULL),
```

('Carrera Técnica', NULL),

('Licenciatura', 'Maestría'),

('Posgrado', 'Doctorado');

insert into familia (residentes, domicilio, cantidad_menores) values

(4, 'Av. Siempreviva 1234', 2),

(2, 'Calle Falsa 123', 0),

(3, 'Av. Corrientes 2468', 1),

(5, 'San Martín 567', 3),

(1, 'La Rioja 321', 0);

insert into estado_civil (estadocivil) values

('Soltero/a'),

('Casado/a'),

('Viudo/a'),

('Divorciado/a'),

('Separado/a');

insert into identificacion (dni, edad, ingresos, id_genero, id_ciudad, id_nivel_educativo, id_trabajo, id_ingresos, id_titulardefamilia, id_estadocivil, id_clasificacion) values

(34567890, 25, 30000, 1, 1, 4, 2, 1, 1, 1, 1),

(40123456, 43, 50000, 2, 2, 5, 1, 2, 2, 2, 1),

(27123456, 32, 80000, 1, 3, 5, 3, 3, 3, 1, 2),

(35678901, 51, 120000, 1, 4, 5, 4, 4, 4, 2, 3),

(11223344, 19, 0, 2, 5, 2, 5, 5, 5, 3, 1);

Este script SQL contiene una serie de inserciones de datos en diferentes tablas de una base de datos. Cada tabla representa una entidad distinta, como género, nivel educativo, ciudad, estado civil, entre otros. Los datos que se están insertando representan ejemplos de registros que podrían ser almacenados en cada una de estas tablas.

Por ejemplo, la tabla de clasificación contiene diferentes tipos de clasificaciones laborales, mientras que la tabla de trabajo representa diferentes tipos de trabajos. Además, se insertan datos de personas en la tabla de identificación, incluyendo su identificación, edad, ingresos, género, ciudad, nivel educativo, trabajo, estado civil y clasificación.

/*views*/

create view facts_ingresos as select edad, nivel_educativo, trabajo, ingresos from identificacion, trabajo, nivel_educativo;

La vista "facts_ingresos" muestra los datos de edad, nivel educativo, trabajo e ingresos de la tabla "identificacion, trabajo, ingresos, nivel_educativo". La vista se crea utilizando la sentencia "create view" y el nombre de la vista es "facts_ingresos". La vista combina los datos de las tablas "identificacion, trabajo, ingresos, nivel_educativo" mediante una consulta SELECT.

create view mayores_edad_edu as select edad, nivel_educativo from identificacion, nivel_educativo where edad>"18";

La vista "mayores_edad_edu" muestra la edad y el nivel educativo de la tabla "identificacion, ingresos, nivel_educativo" donde la edad es mayor a "18". La vista se crea utilizando la sentencia "create view" y el nombre de la vista es "mayores_edad_edu". La vista utiliza una consulta SELECT con una cláusula WHERE para filtrar los datos de la tabla.

create view promedio_hijos_por_domicilio as

select f.domicilio, avg(f.residentes - f.cantidad_menores) as promedio_hijos

from familia f

group by f.domicilio;

La vista "promedio_hijos_por_domicilio" muestra el promedio de hijos por domicilio, utilizando los datos de la tabla "familia". La vista se crea utilizando la sentencia "create view" y el nombre de la vista es "promedio_hijos_por_domicilio". La vista utiliza una consulta SELECT con una función de agregación AVG() para calcular el promedio de hijos por domicilio.

create view vista_identificacion_info as

select identificacion.dni, identificacion.edad, genero.genero, ciudad.nombre_ciudad, provincia.nombre_provincia, trabajo.trabajo, identificacion.ingresos

from identificacion

```
join genero on identificacion.id_genero = genero.id_genero

join ciudad on identificacion.id_ciudad = ciudad.id_ciudad

join provincia on ciudad.id_provincia = provincia.id_provincia

join trabajo on identificacion.id_trabajo = trabajo.id_trabajo;
```

La vista "vista_identificacion_info" combina información de las tablas "identificacion, genero, ciudad, provincia y trabajo" para mostrar información detallada de la identificación, incluyendo el DNI, la edad, el género, la ciudad y la provincia donde vive la persona, el trabajo que desempeña y los ingresos que recibe. La vista se crea utilizando la sentencia "create view" y el nombre de la vista es "vista_identificacion_info". La vista utiliza una consulta SELECT con cláusulas JOIN para combinar los datos de las diferentes tablas.

```
create view promedio_ingresos_por_nivel_educativo as

select ne.nivel_educativo, avg(i.ingresos) as promedio_ingresos

from identificacion i

join nivel_educativo ne on i.id_nivel_educativo = ne.id_nivel_educativo

group by ne.nivel_educativo;
```

La vista "promedio_ingresos_por_nivel_educativo" realiza una operación de agregación de la tabla "identificacion" y "nivel_educativo". En particular, se une la tabla "identificacion" con la tabla "nivel_educativo" utilizando la cláusula "join" y luego se agrupa por el nivel educativo utilizando la cláusula "group by". La vista calcula el promedio de los ingresos de las personas agrupados por su nivel educativo. La vista se crea utilizando la sentencia "create view" y el nombre de la vista es "promedio_ingresos_por_nivel_educativo". La vista utiliza una consulta SELECT con una función de agregación AVG() para calcular el promedio de ingresos y la cláusula GROUP BY para agrupar los datos por nivel educativo.

/*triggers*/

```
create table log_identificacion

(dni int,

edad int,

provincia varchar(30),

fecha_hora datetime);

DELIMITER $$
```

```
create trigger guardar_log_identificacion
before insert on identificacion
for each row
begin
  declare v_provincia varchar(30);
  set v_provincia = (select nombre_provincia from provincia where id_provincia = (select id_provincia from ciudad where id_ciudad = new.id_ciudad));
  if v_provincia is null then
    set v_provincia = '';
  end if;
  insert into log_identificacion (dni, edad, provincia, fecha_hora)
    values (new.dni, new.edad, v_provincia, NOW());
end$$
DELIMITER ;
```

En este trigger, se crea una variable para almacenar los valores de provincia, obtenidos de las tablas correspondientes utilizando las claves foráneas. Luego, se verifica si alguno de estos valores es nulo y, en caso afirmativo, se cambia el valor a una cadena vacía. Finalmente, se insertan los valores de dni, edad, provincia y fecha_hora en la tabla registro_identificacion.

```
#-------------------------------------------------------------------------#

create table log_novedad_educacion
(DNI int not null,
edad int not null,
nivel_educactivo varchar(30) not null,
fecha_ingreso varchar (50),
hora_ingreso varchar (50));

create trigger registro_edu
after insert on identificacion
for each row
insert into log_novedad_educacion (DNI, edad, fecha_ingreso, hora_ingreso)
```

```
values (new.DNI, new.edad, current_date, current_time);


create trigger registro_nivel_educativo

after insert on nivel_educativo

for each row

insert into log_novedad_educacion (nivel_educativo, fecha_ingreso, hora_ingreso)

values (new.nivel_educativo, current_date, current_time);
```

Este es un conjunto de comandos SQL que crean una tabla llamada "log_novedad_educacion" y dos disparadores o triggers que registran información sobre la educación de las personas en una base de datos.

La tabla "log_novedad_educacion" tiene cinco columnas: DNI, edad, nivel educativo, fecha de ingreso y hora de ingreso. Cada vez que se inserta una nueva fila en la tabla "identificacion" o "nivel_educativo", se activa un trigger que inserta una nueva fila en la tabla "log_novedad_educacion" con la información relevante.

El primer trigger, "registro_edu", se activa después de que se inserta una nueva fila en la tabla "identificacion". Este trigger inserta la información de la persona recién insertada en la tabla "log_novedad_educacion", incluyendo el DNI, la edad, la fecha y la hora de ingreso.

El segundo trigger, "registro_nivel_educativo", se activa después de que se inserta una nueva fila en la tabla "nivel_educativo". Este trigger inserta el nivel educativo recién insertado en la tabla "log_novedad_educacion", junto con la fecha y hora de ingreso.

/*funciones*/

```
DELIMITER $$

create function ingresos_dolarizados (dolar float , id int) returns float

reads SQL data

begin

declare resultado float;

 set resultado = (select ingresos from ingresos where id = id_ingresos)/dolar;

  return resultado;

end $$

DELIMITER ;
```

La función comienza con la declaración "DELIMITER $$", que establece el delimitador para el cuerpo de la función. Luego, se declara la función con la sintaxis "create function [nombre de la función] ([parámetros]) returns [tipo de dato]". En este caso, la función se llama "ingresos_dolarizados", tiene dos parámetros de entrada (el tipo de cambio de dólares y el ID de ingresos) y devuelve un valor de tipo float.

Dentro del cuerpo de la función, se declara una variable "resultado" para almacenar el valor del ingreso dolarizado. Luego, se utiliza la sentencia "select" para recuperar el valor de ingresos de la tabla "ingresos" utilizando el ID proporcionado. Este valor se divide por el tipo de cambio de dólares proporcionado, y el resultado se asigna a la variable "resultado".

Finalmente, la función devuelve el valor de "resultado" utilizando la sentencia "return".

La sentencia "DELIMITER ;" restablece el delimitador a su valor predeterminado para que pueda continuar escribiendo otras consultas o sentencias de SQL.

######

```
DELIMITER $$

create function cant_nivel_edu (nivel varchar(20)) returns varchar(40) charset utf8mb4

reads SQL data

begin

declare resultado int;

if nivel = incompleto then

        set resultado = (select count(nivel_educativo) from identificacion where nivel_educativo =
(1));

elseif nivel = primario then

        set resultado = (select count(nivel_educativo) from identificacion where nivel_educativo = (2
, 3 , 4));

elseif nivel = secundario then

  set resultado = (select count(nivel_educativo) from identificacion where nivel_educativo = (3, 4));

elseif nivel = superior then

  set resultado = (select count(nivel_educativo) from identificacion where nivel_educativo = (4));

else

        set resultado = ("Ingresa un nivel educativo");

  end if;

return resultado;

end $$

DELIMITER ;
```

La función "cant_nivel_edu" recibe como parámetro una cadena de caracteres que representa el nivel educativo. Dependiendo del nivel recibido, cuenta el número de registros en una tabla llamada "identificacion" y almacena el resultado en una variable llamada "resultado". Si el nivel no coincidieron ninguna de las opciones especificadas, la función devuelve un mensaje indicando que se debe ingresar un nivel educativo válido. Finalmente, la función devuelve el valor de la variable "resultado".

/*stores procedures*/

```
DELIMITER $$
create procedure order_by_dependiendo_del_campo(in tabla varchar(50), in campo char(20), in orden char(20))
begin
if campo <> '' and orden = 'ascendente' then
        set @orden = concat('order by ', campo, ' asc');
elseif campo <> '' and orden = 'descendente' then
        set @orden = concat('order by ', campo, ' desc');
elseif campo = '' and orden = 'ascendente' then
        set @orden = 'order by dni asc';
elseif campo = '' and orden = 'descendente' then
        set @orden = 'order by dni desc';
else
        set @orden = 'error: campos incorrectos';
end if;
set @clausula = concat('select * from ', tabla, ' ', @orden);
prepare ejecutarSQL from @clausula;
execute ejecutarSQL;
deallocate prepare ejecutarSQL;
end $$
DELIMITER ;
```

Este código crea una procedimiento almacenado que permite ordenar los registros de una tabla en función de un campo determinado. El procedimiento toma tres parámetros de entrada: el nombre de la tabla, el nombre del campo y la dirección del ordenamiento.

En la primera parte del procedimiento, se comprueba si se ha especificado un campo de ordenamiento y si el orden es ascendente o descendente. Luego, se construye una cadena de texto que contiene la cláusula "order by" con el campo y el orden especificados. Si no se especifica un campo, se ordena por defecto por el DNI. Si los valores de entrada no son válidos, el procedimiento devuelve un mensaje de error.

En la segunda parte del procedimiento, se utiliza la cláusula "prepare" para preparar una consulta SQL dinámica a partir de la cadena de texto generada en la primera parte. Luego, se ejecuta la consulta utilizando la cláusula "execute" y se liberan los recursos utilizando la cláusula "deallocate".

```
DELIMITER $$

create procedure eliminar_nulos(in tabla varchar(50),in columna varchar(50))

begin

   set @sql = concat('delete from ', log_novedad_identificacion, ' where ',DNI, ' is null or',genero,' is null or',edad,' is null or',localidad,' is null or',provincia,' is null');

   prepare stmt FROM @sql;

   execute stmt;

   deallocate prepare stmt;

end $$

DELIMITER ;
```

Este código de SQL es la definición de un procedimiento almacenado llamado "eliminar_nulos". Su función es eliminar filas de una tabla donde alguna de las columnas especificadas en el parámetro "columna" tenga un valor nulo. La tabla de la cual se eliminarán los nulos es especificada en el parámetro "tabla".

El procedimiento utiliza una variable @sql para construir una sentencia DELETE dinámica que utiliza el operador IS NULL para evaluar cada columna especificada en el parámetro "columna" y determinar si su valor es nulo. Una vez que se construye la sentencia DELETE, se prepara y ejecuta dinámicamente utilizando los comandos PREPARE, EXECUTE y DEALLOCATE.

Este procedimiento puede ser útil en proyectos que involucren grandes cantidades de datos y que requieran mantener una base de datos limpia y consistente. La eliminación de filas con valores nulos puede ayudar a mejorar la eficiencia en la recuperación de datos y en la optimización de consultas en la base de datos.

/*SENTENCIAS*/

```
-- Crear el usuario con el nombre "ejemplo2"

create user ejemplo2@localhost;


-- Dar permisos de lectura, inserción y modificación de datos en todas las tablas de la base de datos "ejemplos2"

grant select, insert, update on *.* to ejemplo2@localhost;


-- Crear el usuario con el nombre "ejemplo1"

create user ejemplo1@localhost;


-- Dar permisos de solo lectura en todas las tablas de la base de datos "ejemplo1"

grant select on *.* to ejemplo1@localhost;


-- Para eliminar un usuario por completo, borrando todas las conexiones con las bases de datos se usa la sintaxis "drop user"

drop user ejemplo1@localhost;

drop user ejemplo2@localhost;
```

Este bloque de código relacionado con las sentencias en MySQL, se enfoca en la creación y eliminación de usuarios y sus respectivos permisos en la base de datos. En este caso, se están creando dos usuarios diferentes y asignándoles diferentes permisos en la base de datos. El primer usuario, "ejemplo2", tendrá permisos de lectura, inserción y modificación en todas las tablas de la base de datos "ejemplos2", mientras que el segundo usuario, "ejemplo1", tendrá permisos de solo lectura en todas las tablas de la base de datos "ejemplo1". La sintaxis "drop user" se utiliza para eliminar usuarios por completo y todas sus conexiones con las bases de datos.

/*TCL*/

```
start transaction;


-- insertar los registros en la tabla "trabajo"

insert into trabajo (trabajo) values

('Enfermera'),
```

```
('Ingeniero Informatico'),
('Cirujano'),
('Maestro'),
('instructor de boxeo');

-- insertar los registros en la tabla "provincia"
insert into provincia (nombre_provincia) values
('Buenos Aires'),
('Buenos Aires'),
('Mendoza');

-- Crear un punto de guardado (checkpoint1) para poder volver si es necesario
savepoint checkpoint1;
insert into provincia (nombre_provincia) values
('Santa Fe'),
('Buenos Aires'),
('Cordoba');

-- Si ocurre algún error, deshacer todas las modificaciones hasta el savepoint indicado con rollback
rollback to checkpoint1;

-- Si no hay errores, confirmar los cambios en ambas tablas con commit
commit;
```

Este bloque de código relacionado con las transacciones o sino conocido como “transactions” se enfoca en el control de transacciones en la base de datos.

Aquí, se inicia una transacción y se realizan cambios en dos tablas diferentes: "trabajo" y "provincia". Se crea un punto de guardado ("savepoint") llamado "checkpoint1" que se puede utilizar para deshacer todas las modificaciones hasta ese punto si ocurre algún error en la transacción. Luego, se insertan más registros en la tabla "provincia" y si ocurre algún error, se pueden deshacer todas las modificaciones hasta el punto de guardado utilizando "rollback".

Si no hay errores, se confirman los cambios en ambas tablas utilizando "commit". La transacción se utiliza para asegurarse de que todas las modificaciones se realicen correctamente en ambas tablas o que se deshagan completamente si ocurre algún error en el proceso.

/*Backup*/

El siguiente backup contiene una exportación de datos de diferentes tablas en la base de datos. El servidor ha generado un script que solo exporta los datos de las tablas especificadas: familia, estado_civil, identificacion, provincia, ciudad, trabajo, clasificacion, genero y nivel_educativo. Esto significa que no se incluye la estructura de la tabla, sino solamente los datos almacenados en ellas.

Este tipo de backup puede ser útil en situaciones donde se necesita transferir o importar datos de una base de datos a otra sin tener que preocuparse por la estructura de la tabla. Es importante tener en cuenta que, si se desea restaurar estos datos en otra base de datos, debe haber una tabla con la misma estructura y nombre en la base de datos de destino.

Adjunto los links donde se encuentra el script tanto del proyecto final como del propio BackUp

https://github.com/joaacoopes18/ProyectoFinal.JoaquinPessina

https://github.com/joaacoopes18/ProyectoFinal.JoaquinPessina

https://github.com/joaacoopes18/ProyectoFinal.JoaquinPessina