|  |  |
| --- | --- |
|  | **Universidad Nacional Autónoma de México** |
|  | **Facultad de Ingeniería** |
|  | **División de Ingeniería Eléctrica** |
|  | **Laboratorio de**  **Bases de Datos** |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Practica 10 | Lenguaje de manipulación de datos y transacciones | |
|  |  |  |  | |
|  | Profesor: | Lugowski Rivero Czeslaw Kristofer |  | |
|  | Grupo: | 06 |  | |
|  | Alumnos: | Guerrero López Uriel Ivan 320046045  Muñoz San Agustin Victoria Monserrat 320094187 | | |
|  |  |  | Calificación | |
|  |  | Fecha de entrega | Jueves 17 de abril de 2025 | |
|  |  |  | Semestre 2025-2 | |

**Objetivo:**

* El alumno comprenderá y confirmará el uso del lenguaje SQL para realizar la administración de los datos empleando las instrucciones INSERT, UPDATE y DELETE. Comprenderá las principales características de una transacción, comprobará y configurará los diferentes niveles de aislamiento empleados por un RDBMS.

**Introducción**

Lenguaje de manipulación de datos

El lenguaje de manipulación de datos (DML) nos permite acceder, crear, modificar o eliminar los datos de un esquema de base de datos. Las órdenes que se utilizan son: insertar (INSERT), actualizar (UPDATE), borrar (DELETE) y seleccionar (SELECT) los datos en una base de datos.

Insertar valores a una tabla

INSERT

Una vez que se han creado las tablas, utilizamos el comando INSERT para agregar información a las tablas, su sintaxis es la siguiente:

INSERT INTO <nombreTabla> [(campo1 [,campo2, …])]

VALUES (valor1, valor2,…);

O bien,

INSERT INTO <nombreTabla>

VALUES (valor1, valor2,…);

En esta opción debemos tener cuidado de colocar los valores según el orden correspondiente en la tabla.

Los valores de tipo DATE o CHAR se deben encerrar entre comillas simples. Los valores insertados deben ser de un tipo compatible con el de las columnas de la tabla.

También se puede realizar un INSERT combinado con un SELECT

INSERT INTO <nombreTabla> [(campo1 [,campo2, …])]

SELECT <columna1, columna2, …>

FROM <nombreTabla>

Modificar datos de una tabla

UPDATE

Se emplea para modificar los datos existentes en una tabla, su sintaxis es la siguiente:

UPDATE <nombre tabla> SET

{ <columna> = <expresión> [, ...]

| {(<lista-de columnas>) | \*} = (<lista de expresiones>) }

[WHERE <condición>]

Una expresión puede estar formada por una subconsulta SELECT entre paréntesis, cuyo resultado es un único dato de una sola columna (un único valor simple). La lista de columnas está formada por aquellas columnas a las cuales se les modificará su valor. Si se omite la cláusula WHERE entonces se actualizan todos los datos de la tabla.

Eliminar datos de una tabla

DELETE

Permite eliminar datos de una tabla. Su sintaxis es la siguiente:

DELETE [FROM] <nombre tabla> [WHERE <condición>]

Si no se pone condición, se borran todos los datos de la tabla.

Seleccionar datos de una tabla

SELECT

Para recuperar la información guardada en una base de datos utilizamos el comando SELECT.

La sintaxis de la orden SELECT consta básicamente de las cláusulas SELECT y FROM como obligatorias y la cláusula WHERE como opcional, su sintaxis es la siguiente:

SELECT [ALL | DISTINCT | UNIQUE ]

[<tabla|aliasTabla>.] <columna1> [AS <nombreColumna>]

[, [<tabla|aliasTabla>.]< columna2> [AS <nombreColumna>]…., |\*]

FROM [<nombreBasedeDatos>] Tabla [<aliasTabla>]

[WHERE <condición>]

Las columnas ambiguas (iguales) se preceden del nombre de la tabla:

<tabla>.<columna> o de un sobrenombre corto para la tabla <aliasTabla>.<columna>

AS Se usa para cambiar el nombre de una columna del conjunto de resultados o para asignarle un nombre a una columna derivada.

El \* hace referencia a todas las columnas de la tabla.

Cláusula FROM

Permite indicar las tablas que contienen los datos.

FROM {<nombre de tabla> [alias de tabla] } [,...]

<alias de tabla>: Es un sinónimo activo sólo en el ámbito de la sentencia SELECT.

Cláusula WHERE

Sirve para indicar la condición que deben cumplir las filas resultantes.

WHERE <condición>

Una condición está formada por una o varias expresiones condicionales conectadas por los operadores lógicos AND, OR y NOT. Una expresión condicional tiene la forma siguiente:

Expresión de comparación

<expresión1> <operador relacional> <expresión2>

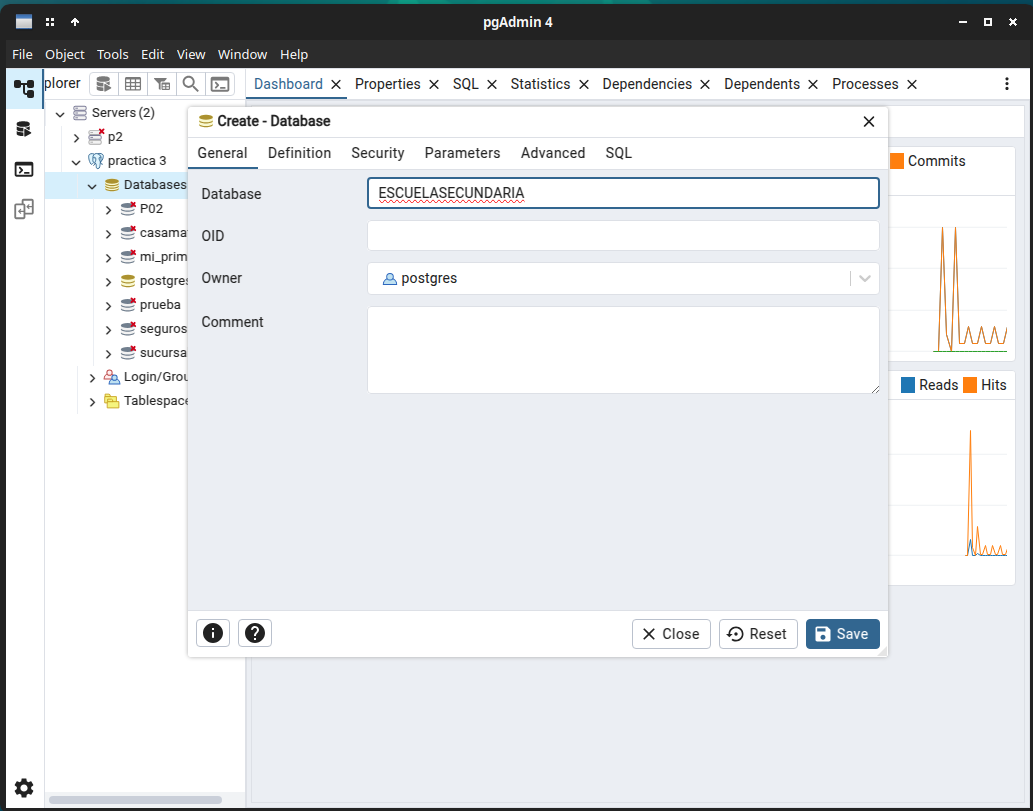
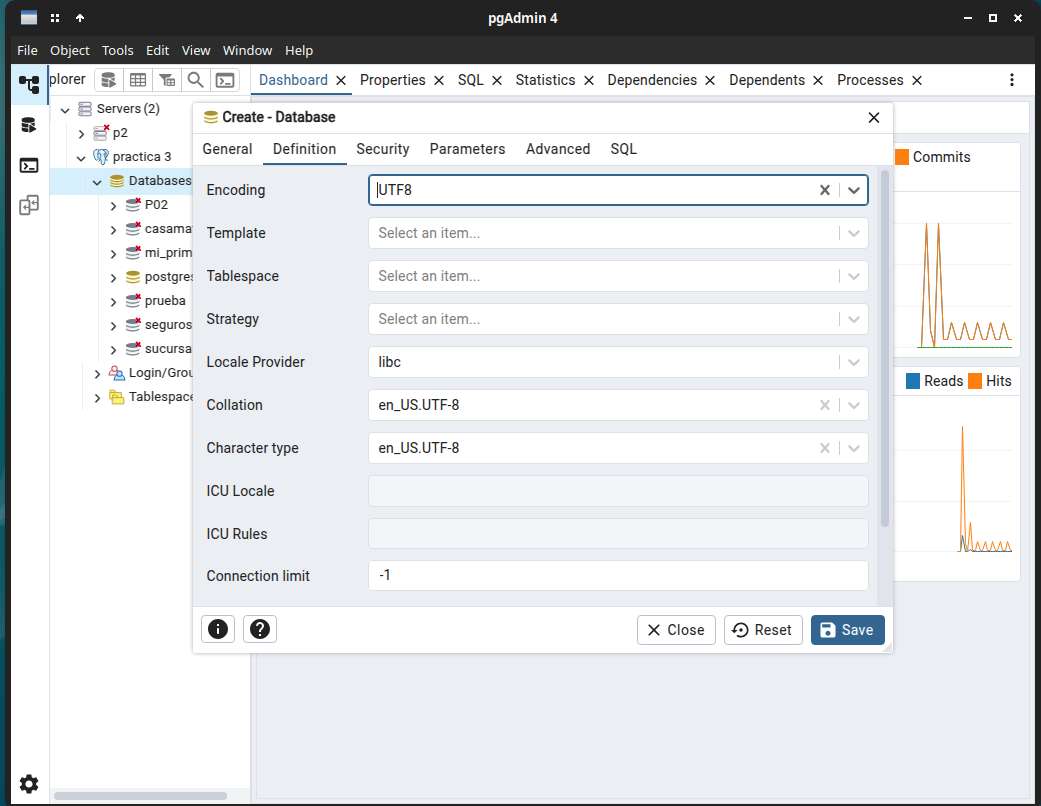
Verifica si las dos expresiones satisfacen la comparación.

**Actividades propuestas por la academia**

Carga de B.D.

Descargue el material de trabajo de la practica 10.

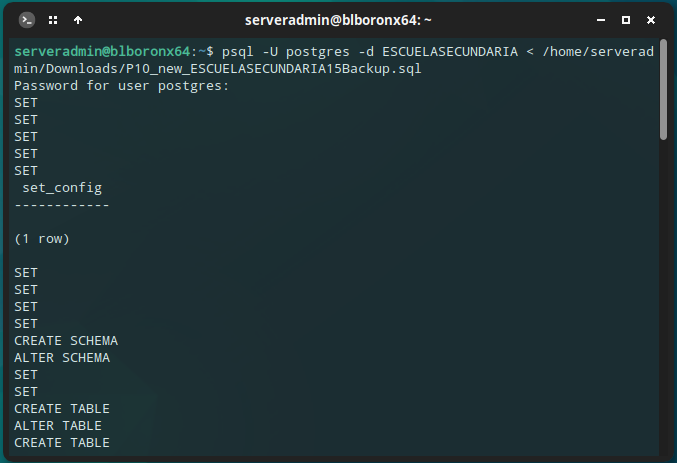
* De modo gráfico cree una base de datos llamada ESCUELASECUNDARIA con codificación UTF-8.

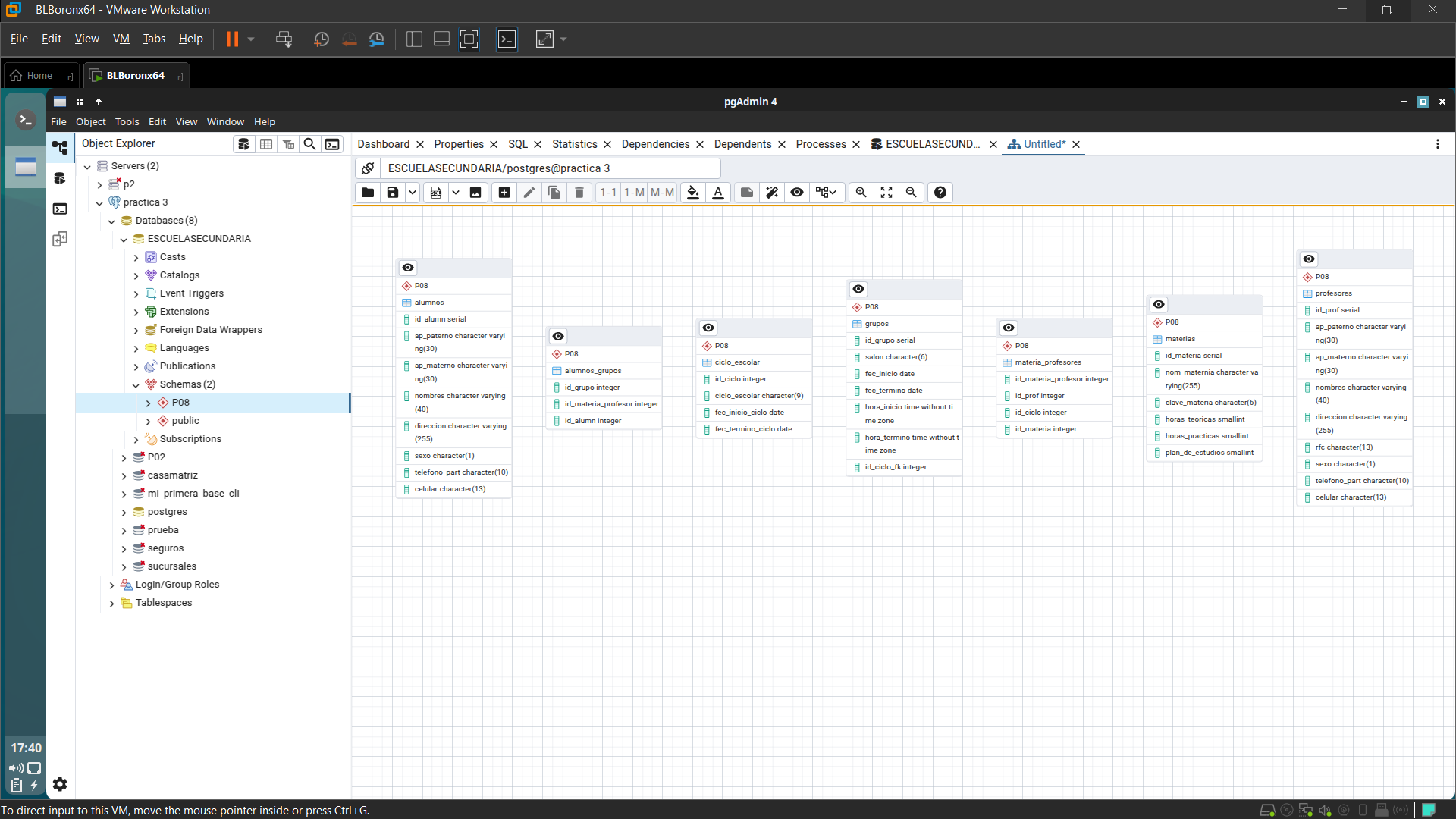
* En modo terminal entre al directorio donde está el binario psql de postgres con un usuario que tenga permisos y ejecute el siguiente comando en la terminal (Si es necesario cambiar la ruta cámbiela de acuerdo a donde se encuentra el archivo descargado o bien de modo grafico restaure “restore" la B.D. con el archivo P10\_new\_ESCUELASECUNDARIA15DUMP

psql -U postgres -d ESCUELASECUNDARIA < /home/serveradmin//Downloads/P10\_new\_ESCUELASECUNDARIA15Backup.sql

Vía terminal



Donde accedemos a la restauración mediante el archivo de Backup y el comando mostrado en la imagen. Además vía gráfica podemos acceder a las tablas y sus requerimientos de cada una, dando lugar a que así conozcamos si al restaurar la base lo hicimos de forma correcta.

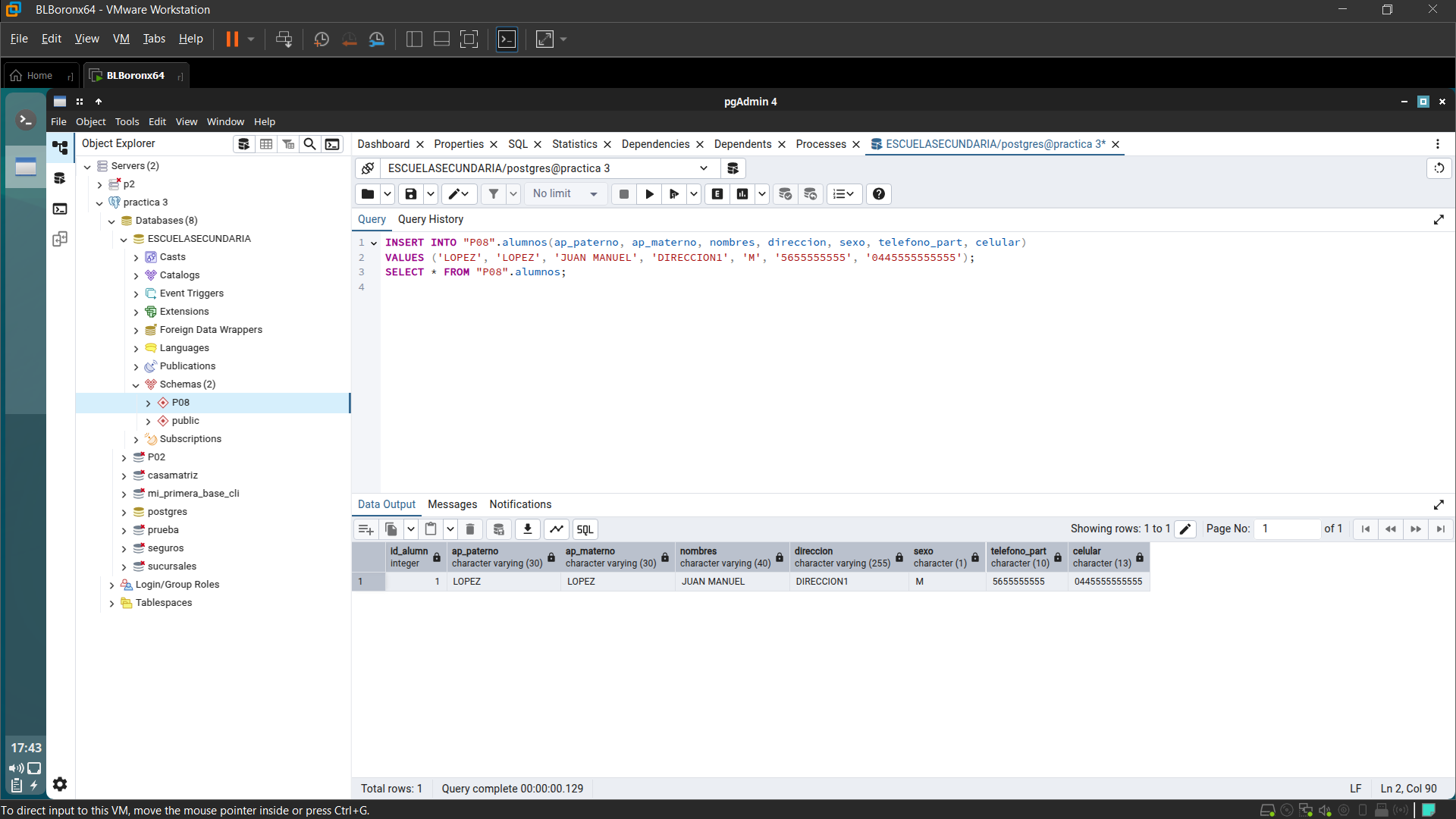


**Inserciones de renglones con y sin encabezados de columnas.**

* Ejecute los siguientes query’s.
  + *INSERT INTO “P08”.alumnos(ap\_paterno, ap\_materno, nombres, direccion, sexo, telefono\_part, celular)*

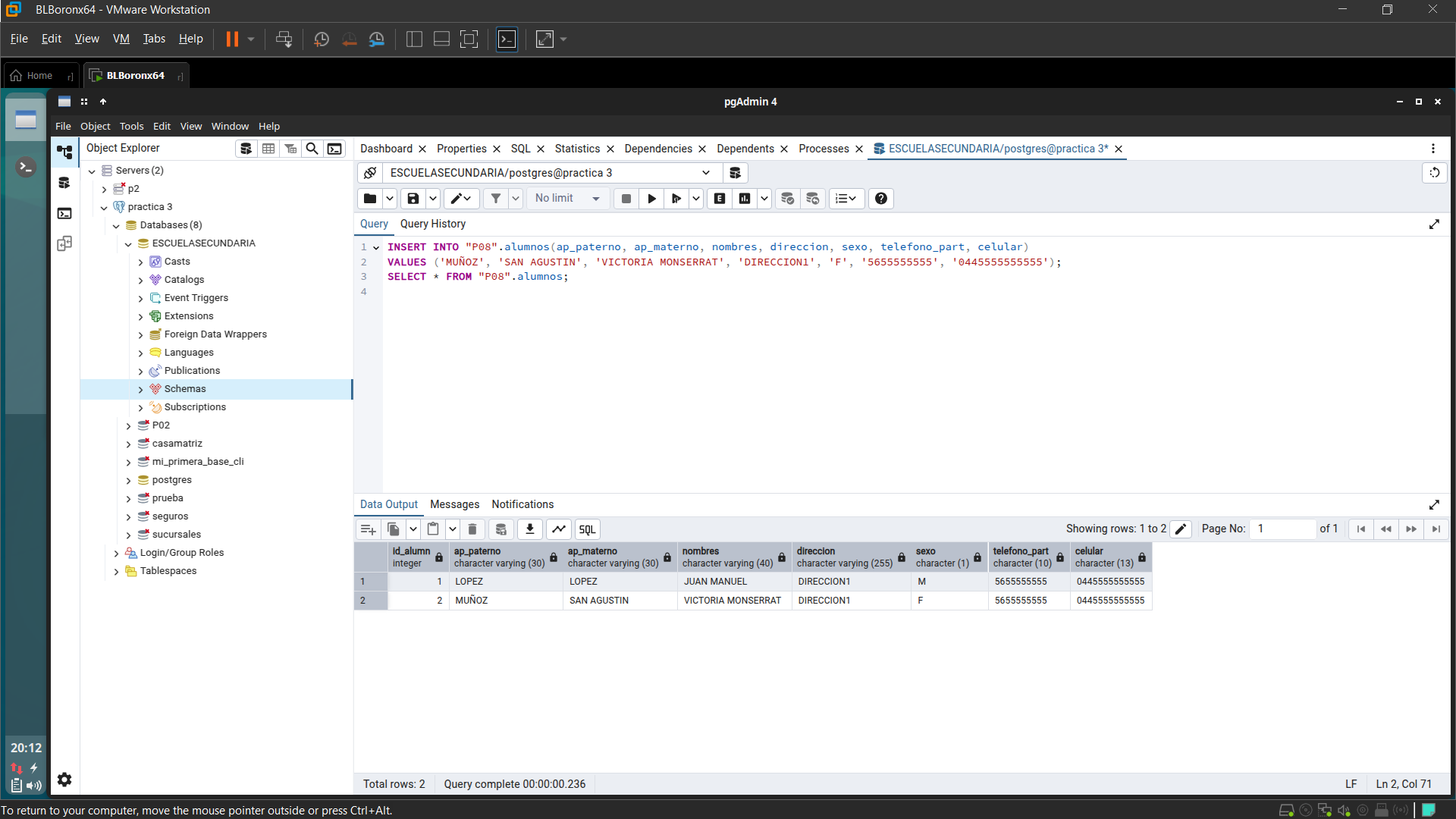
*VALUES ('LOPEZ', 'LOPEZ', 'JUAN MANUEL', 'DIRECCION1', 'M', '5655555555', ‘0445555555555’);*

*SELECT \* FROM “P08”.alumnos;*



Aquí podemos ver una simple inserción de unos datos de forma random.

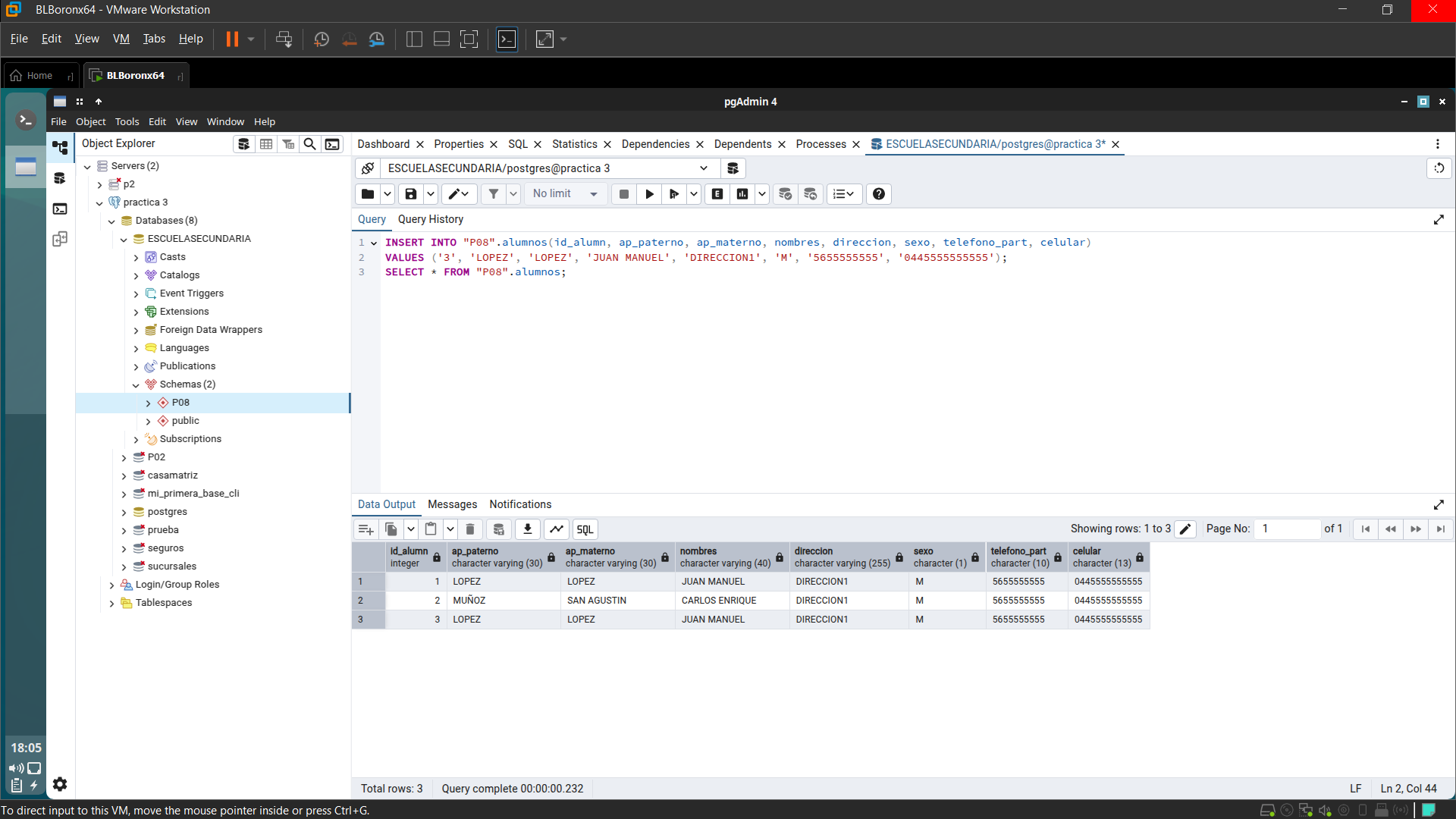
* Intente insertar otro alumno con datos diferentes.



* Ejecute el siguiente query.
  + *INSERT INTO “P08”.alumnos(****id\_alumn****, ap\_paterno, ap\_materno, nombres, direccion, sexo, telefono\_part, celular)*

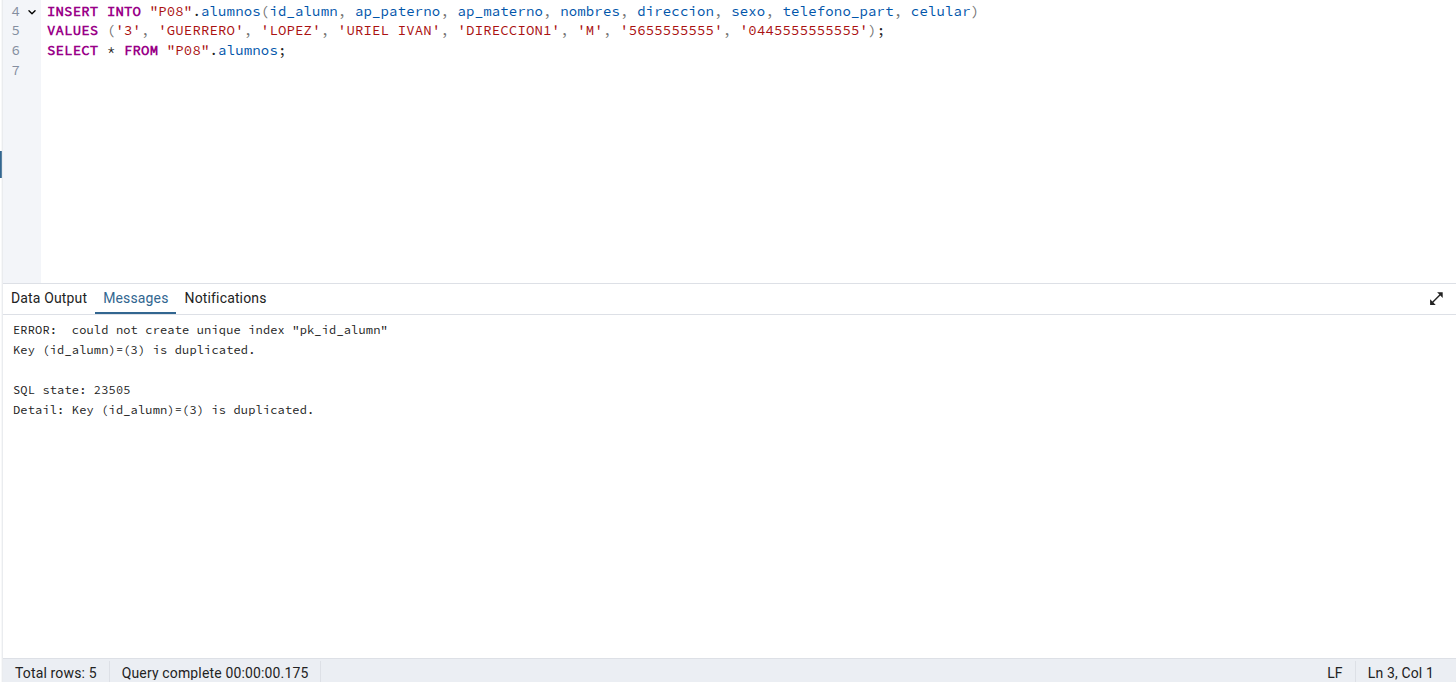
*VALUES (****'3',****' LOPEZ', 'LOPEZ', 'JUAN MANUEL', 'DIRECCION1', 'M', '5655555555', ‘0445555555555’);*

*SELECT \* FROM “P08”.alumnos;*



Ahora, que diferencia este query con el anterior res el uso del ‘id\_alumn’ pues con ese podemos hacer el uso de un id definido por nosotros mismos y no unos predefinido por la base de datos.

* Intente insertar otro alumno con datos diferentes. Explique qué sucedió y por qué marco el error



En este caso nos genere un error por lo que anteriormente se explico en el uso del query, pues al hacer uso de una definición por nuestra cuenta al intentar agregar uno igual nos genera un error al existir una duplicación de elementos y que esto mismo nos genere errores en el momento de querer generar búsquedas dentro de la base de datos. Es por eso que se nos pide que la llave sea única para evitar problemas de identificación a futuro.

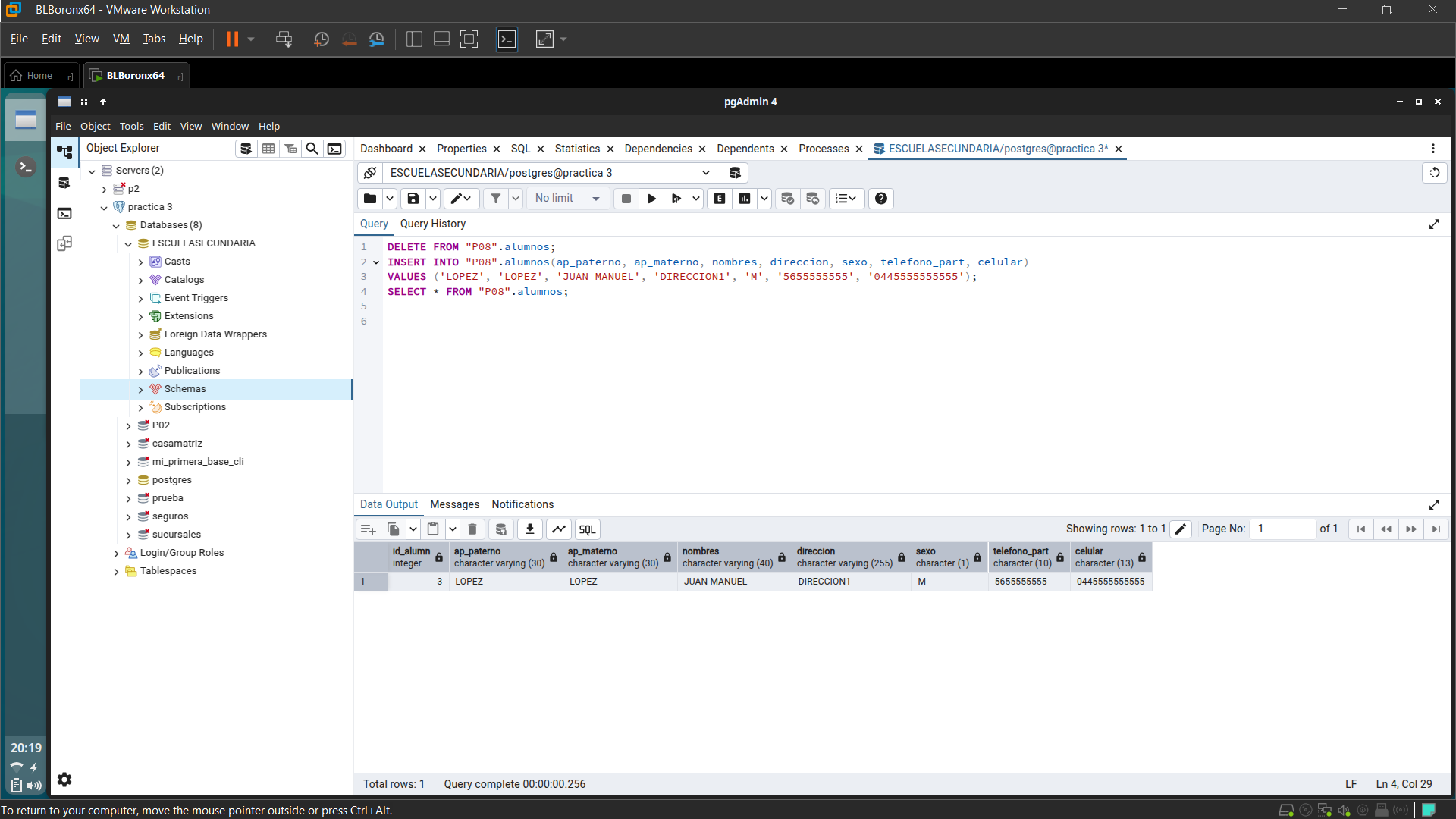
**Inserción con parámetros & Eliminar todos los renglones**

* Ejecute el siguiente query.

*DELETE FROM alumnos;*

*INSERT INTO “P08”.alumnos(ap\_paterno, ap\_materno, nombres, direccion, sexo, telefono\_part, celular) VALUES ('LOPEZ', 'LOPEZ', 'JUAN MANUEL', 'DIRECCION1', 'M', '5655555555', ‘0445555555555’);*

*SELECT \* FROM “P08”.alumnos;*



* Explique por qué es posible realizar el query y que haría para resolverlo.

La instrucción de DELETE FROM borra todos los registros de la tabla alumnos dentro del esquema P08, dejando la tabla vacía pero sin eliminar su estructura (columnas, tipos de datos, constraints, etc.). Luego, puedes insertar un nuevo registro sin problemas, porque la tabla sigue existiendo y está lista para recibir datos. Además, los datos que insertas respetan el formato y los tipos de datos de cada columna.

* Ejecute el siguiente query.

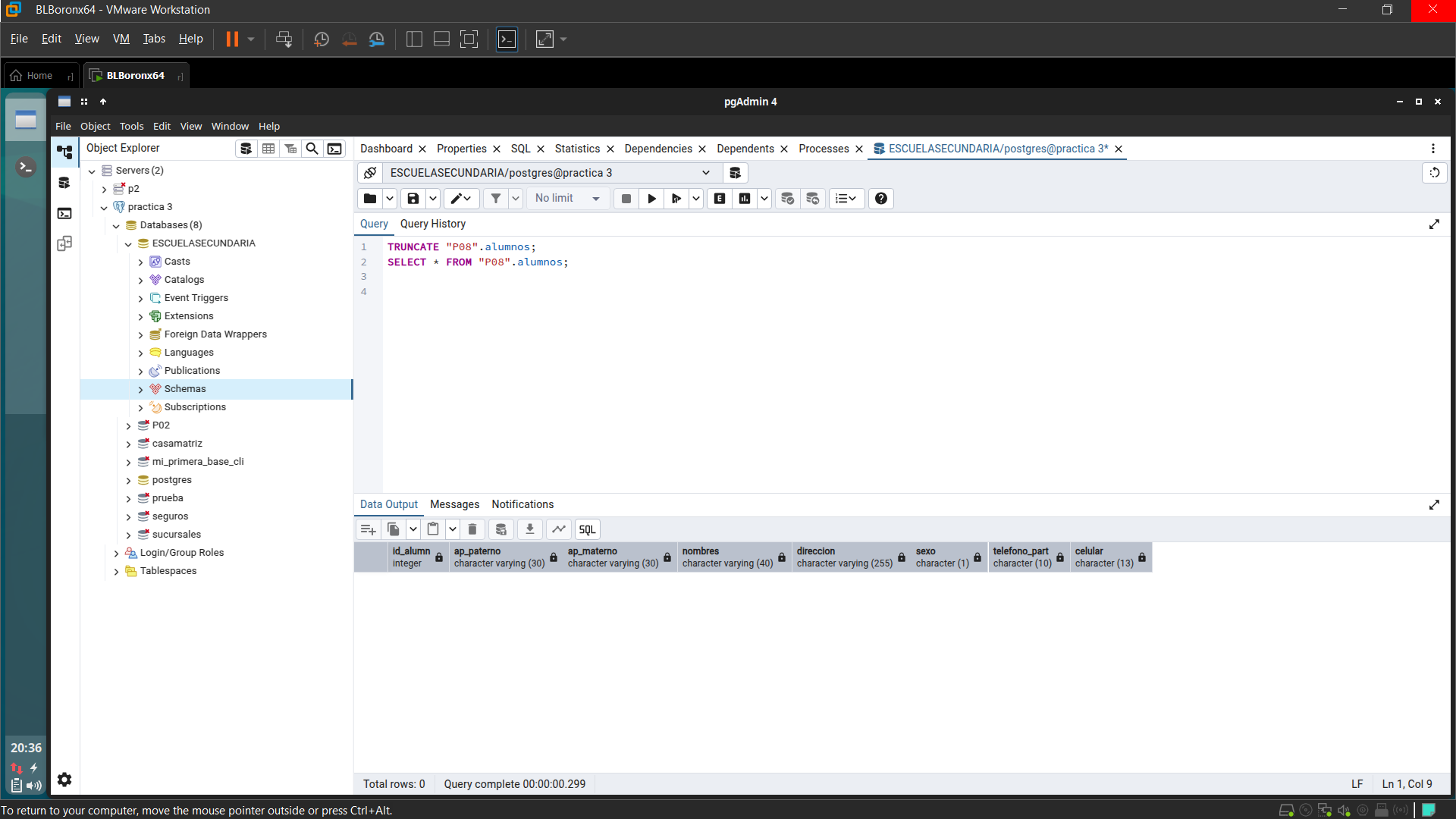
*TRUNCATE alumnos;*

*SELECT \* FROM alumnos;*

*INSERT INTO public.alumnos(ap\_paterno, ap\_materno, nombres, direccion, sexo, telefono\_part, celular)*

*VALUES ('LOPEZ', 'LOPEZ', 'JUAN MANUEL', 'DIRECCION1', 'M', '5655555555', ‘0445555555555’);*

*SELECT \* FROM alumnos;*



* Explique entonces cual es la diferencia entre el TRUNCATE y el DELETE.

|  |  |  |
| --- | --- | --- |
| **Características** | **DELETE** | **TRUNCATE** |
| ¿Qué hace? | Elimina filas una por una de la tabla. | Elimina todas las filas de golpe, sin recorrerlas una por una. |
| Velocidad | Es más lento, porque guarda registro de cada fila eliminada. | Es mucho más rápido, porque borra todo directamente sin registrar cada fila. |
| Uso de WHERE | Sí permite usar WHERE, puedes eliminar solo algunas filas. | No permite WHERE, siempre elimina todas las filas de la tabla. |
| Transacciones | Puedes hacer ROLLBACK si está dentro de una transacción normal. | Puedes hacer ROLLBACK, pero se comporta como una operación más fuerte. |
| Espacio en disco | Libera espacio fila por fila. | Libera el espacio de una vez, reseteando la tabla internamente. |
| Reseteo de identidad | No afecta contadores automáticos. | Puede resetear los contadores (opcional con RESTART IDENTITY). |

**Eliminar todos los renglones, utilizar opcionalmente TRUNCATE**

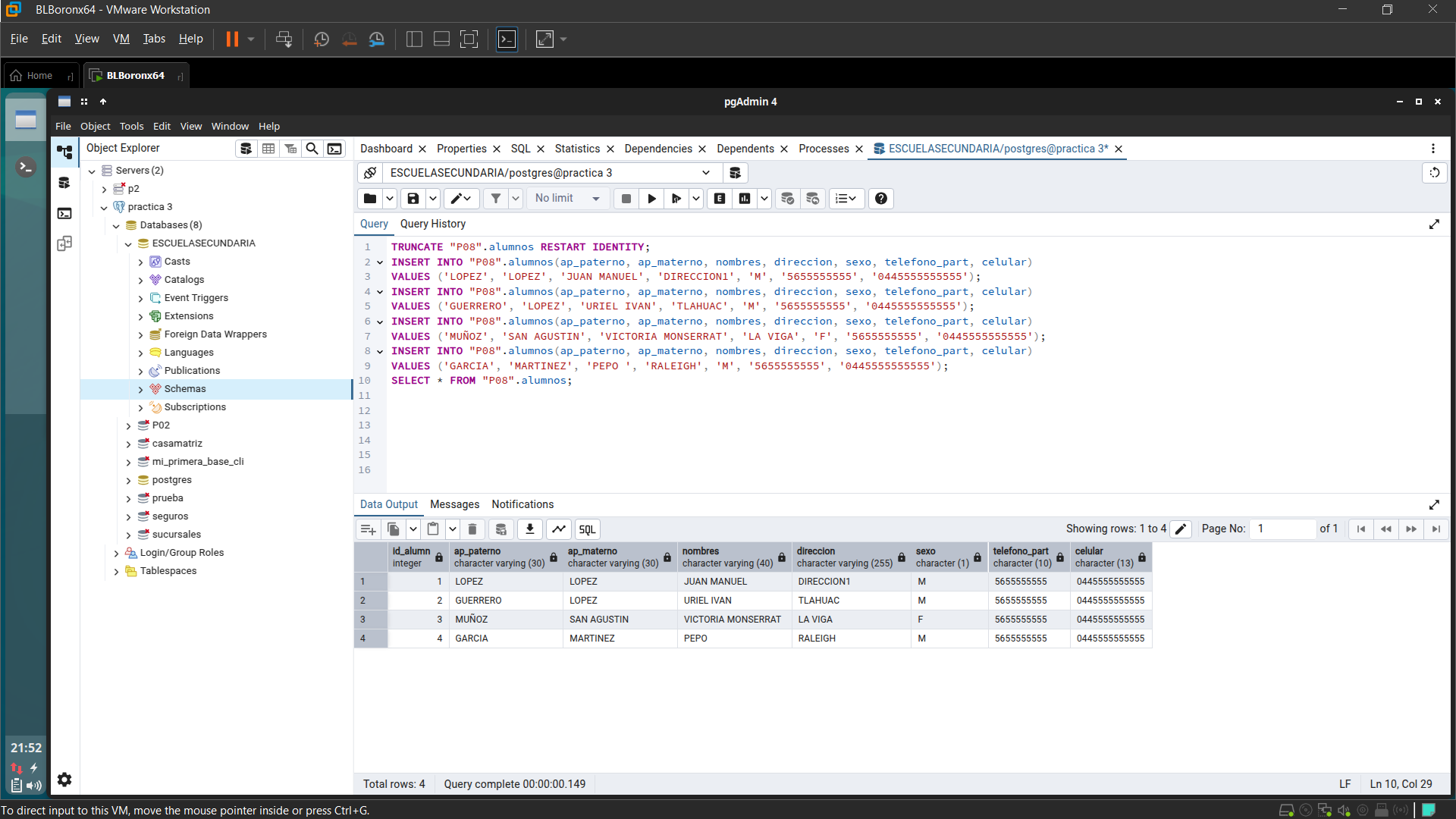
*TRUNCATE “P08”.alumnos RESTART IDENTITY;*

*INSERT INTO public.alumnos(ap\_paterno, ap\_materno, nombres, direccion, sexo, telefono\_part, celular)*

*VALUES ('LOPEZ', 'LOPEZ', 'JUAN MANUEL', 'DIRECCION1', 'M', '5655555555', ‘0445555555555’);*

*SELECT \* FROM “P08”.alumnos;*

* Inserte al menos tres registros más para alumnos.

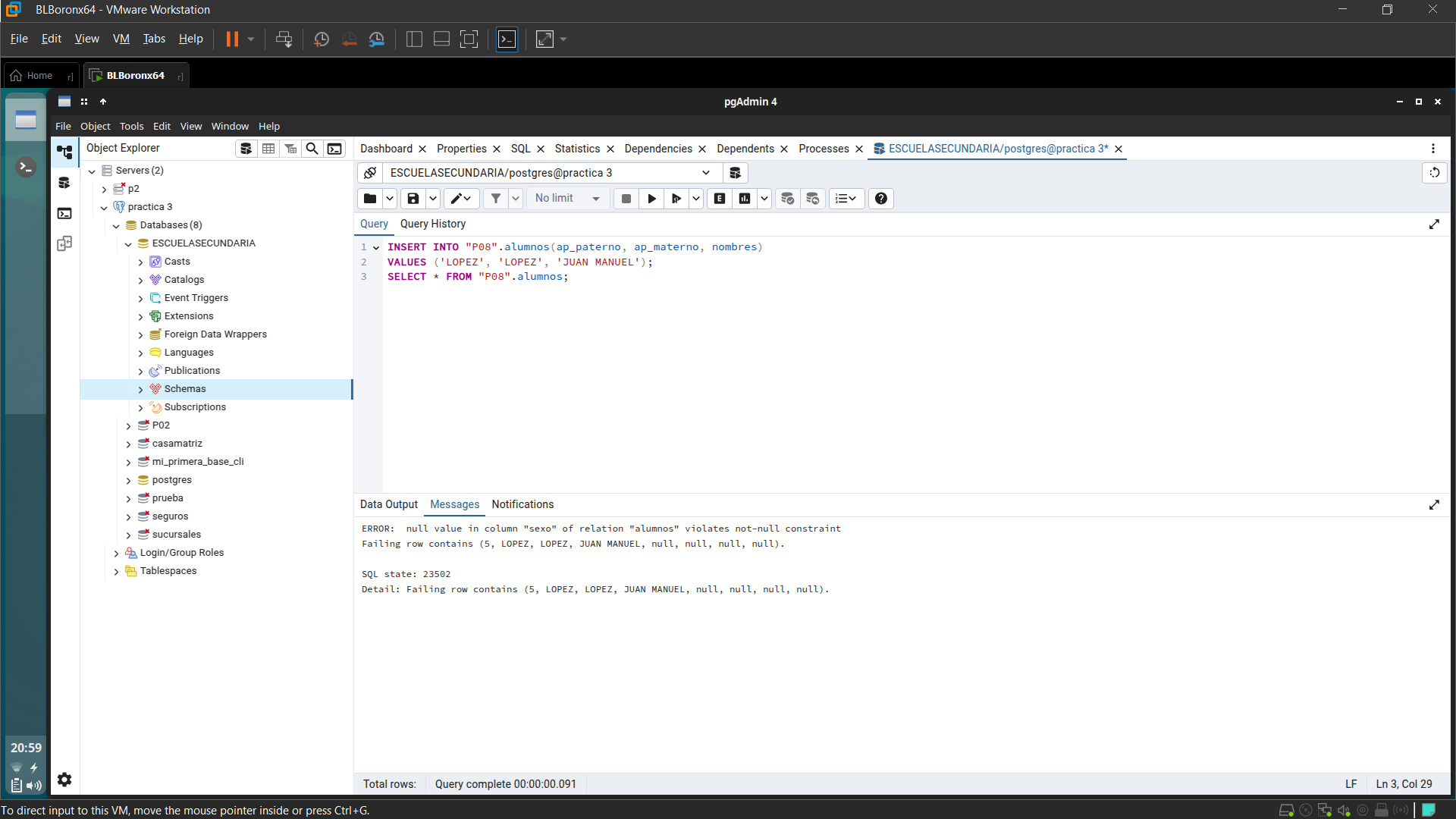


**Inserción con parámetros**

*INSERT INTO “P08”.alumnos(ap\_paterno, ap\_materno, nombres)*

*VALUES ('LOPEZ', 'LOPEZ', 'JUAN MANUEL');*

*SELECT \* FROM “P08”.alumnos;*



* Explique el error y comente brevemente la importancia de los CONSTRAINTS.

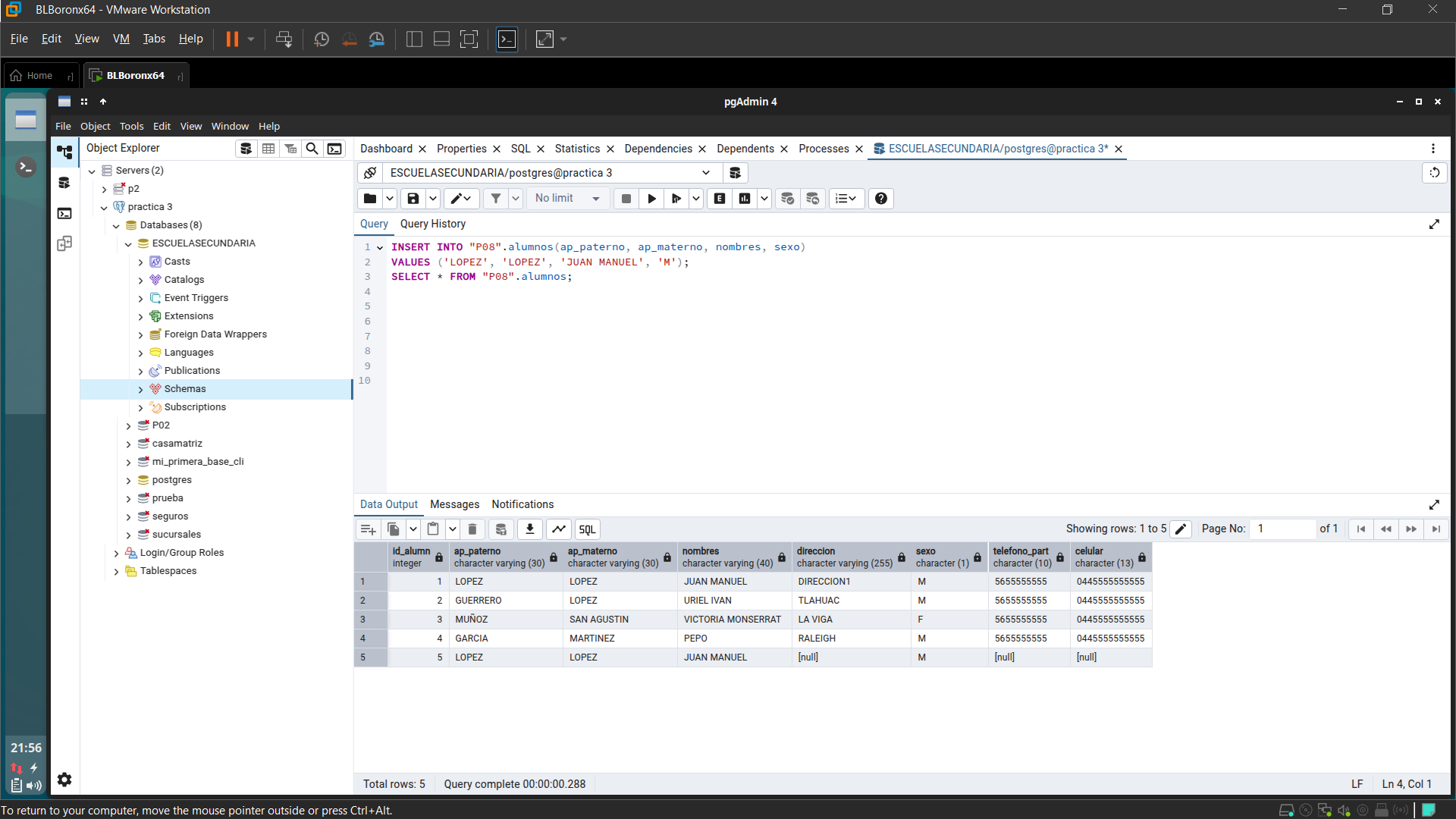
El error generado se da porque a las otras columnas como dirección, sexo, teléfono\_part y celular están definidas como NOT NULL, dando lugar a que estas no puedan ser omitidas ya que la inserción de valores es obligatoria.

Sabemos que los CONSTRAINTS son reglas que protegen la identidad de los datos en una tabla, esto evita que haya datos incorrectos, incompletos o duplicados. Además, mantienen la lógica y coherencia de las relaciones entre tablas. Y, ayudan a proteger la base de datos de errores de programación o inserciones accidentales. Es por eso que su uso es muy importante dentro de la definción de una base de datos.

*INSERT INTO “P08”.alumnos(ap\_paterno, ap\_materno, nombres, sexo )*

*VALUES ('LOPEZ', 'LOPEZ', 'JUAN MANUEL', ‘M');*

*SELECT \* FROM “P08”.alumnos;*



* Explique por qué fue posible insertar datos vacíos y comente brevemente la importancia de manejar DEFAULTS.

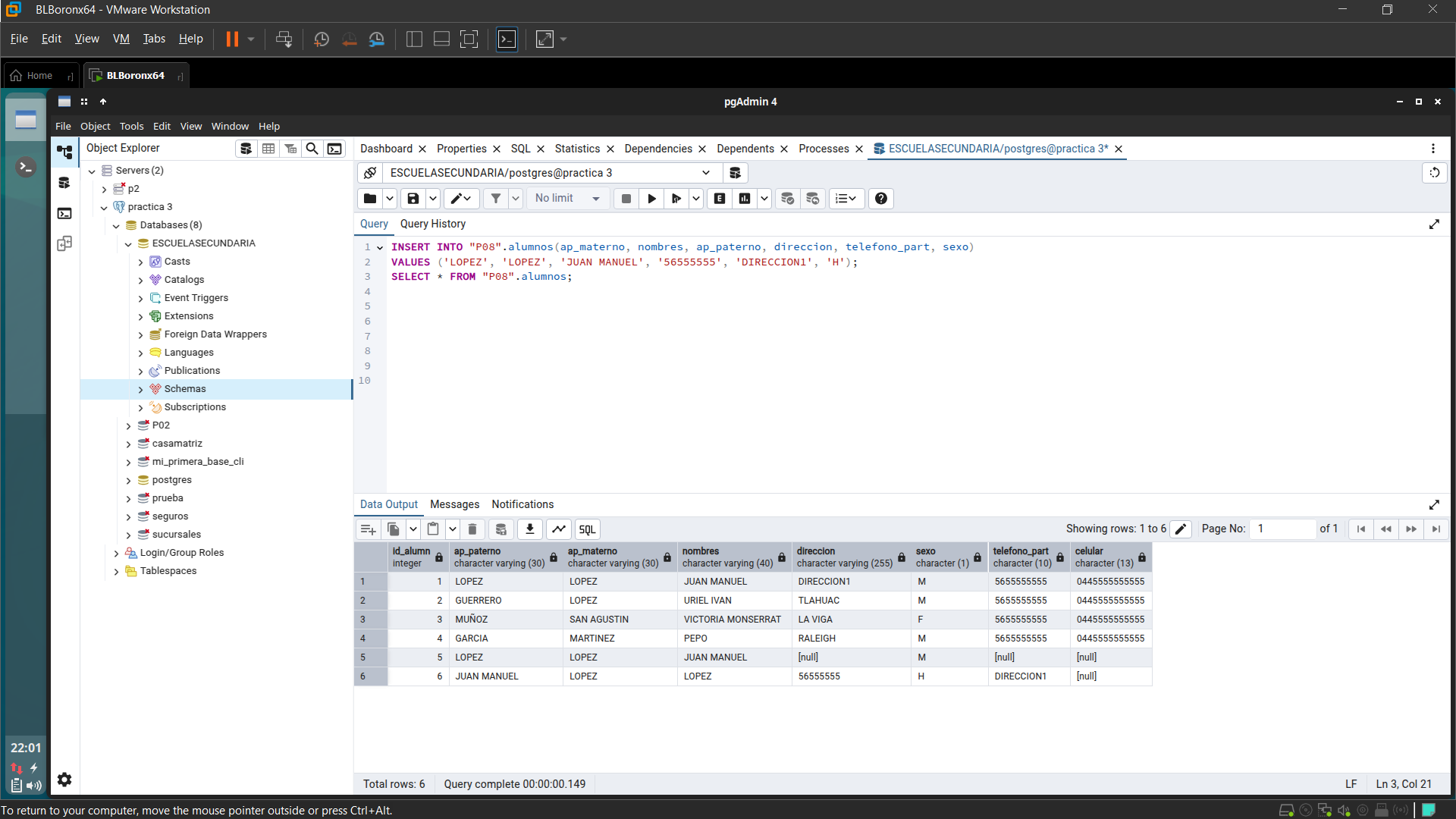
En este caso fue posible la inserción porque las columnas sin datos perminen el acceso a valores nulos (NULL), por lo que PostgreSQL las llena automáticamente con NULL cuando no se inserta nada en ellas.

Conocemos que los DEFAULTS son valores automáticos que se insertan cuando no se proporciona un valor en un INSERT. Y son importantes porque:

* Evitan tener valores NULL innecesarios.
* Facilitan la inserción de datos: no necesitas siempre escribir todos los campos manualmente.
* Aseguran coherencia en la base de datos: siempre hay un valor razonable en vez de quedar campos vacíos.

*INSERT INTO public.alumnos(ap\_materno, nombres, ap\_paterno, direccion, telefono\_part, sexo) VALUES ('LOPEZ', 'LOPEZ', 'JUAN MANUEL', '56555555', 'DIRECCION1', ‘H’);*

*SELECT \* FROM alumnos;*



* Explique que está mal con la consulta anterior y la importancia de tener cuidado.

Existen los siguientes problemas:

1. Orden de las columnas vs. orden de los valores: El orden de las columnas no corresponde al tipo de dato de los valores, pues ‘nombres’ tiene asignado un apellido.
2. Datos mezclados: Se esta insertando un teléfono ('56555555') donde debería ir una dirección (direccion), y viceversa.

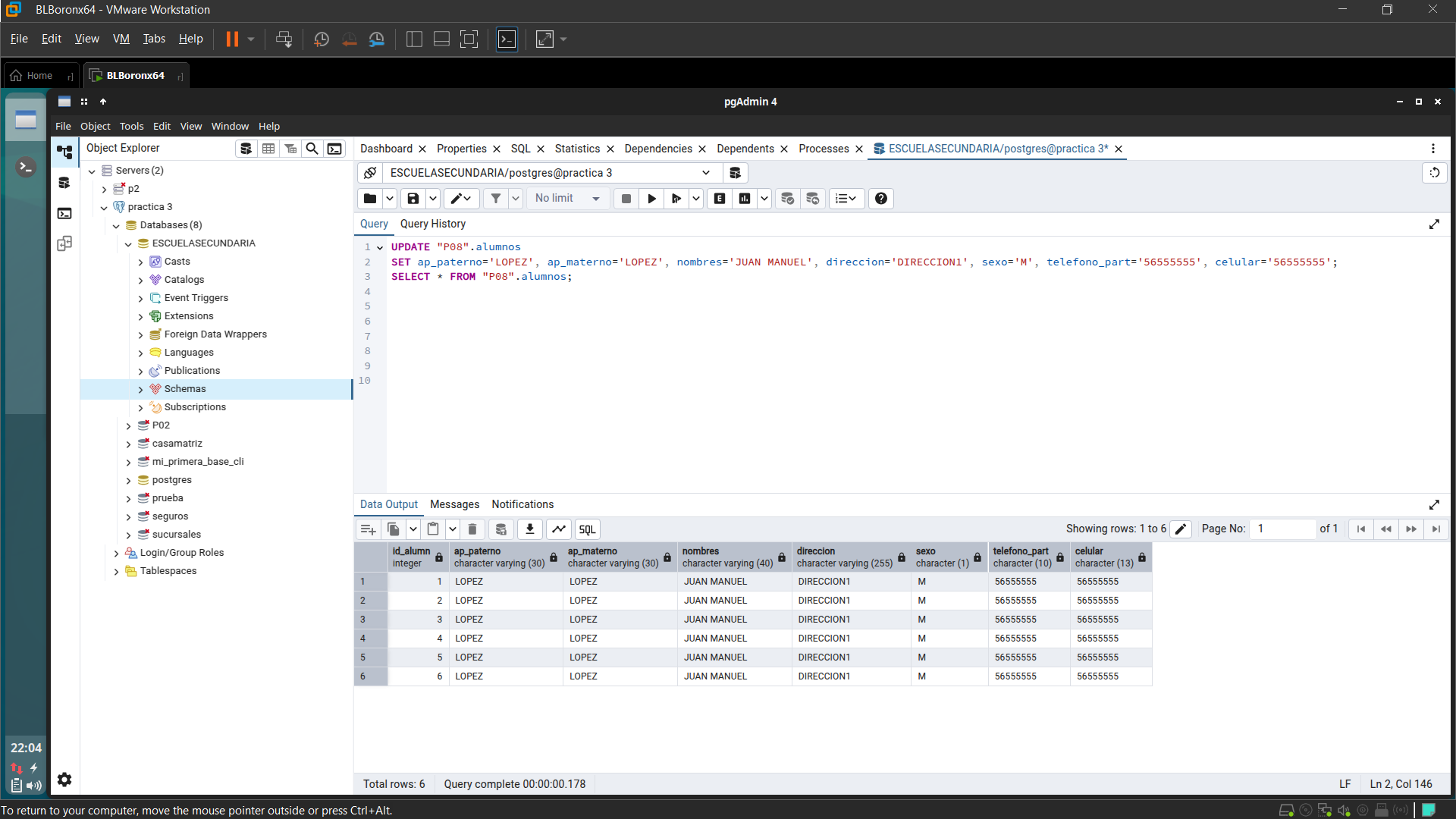
Lo anterior da hincapié a la importancia de que un pequeño error puede causar:

* Que los datos queden mal almacenados.
* Que la calidad de la información se pierda (y sea difícil de corregir luego).
* Que los reportes o consultas posteriores den resultados equivocados (porque los datos están en los campos incorrectos).

**Actualizar una columna con una condición**

UPDATE “P08”.alumnos SET ap\_paterno='LOPEZ', ap\_materno='LOPEZ', nombres='JUAN MANUEL', direccion='DIRECCION1', sexo='M', telefono\_part='56555555', celular=‘56555555’;

SELECT \* FROM "P08”.alumnos;

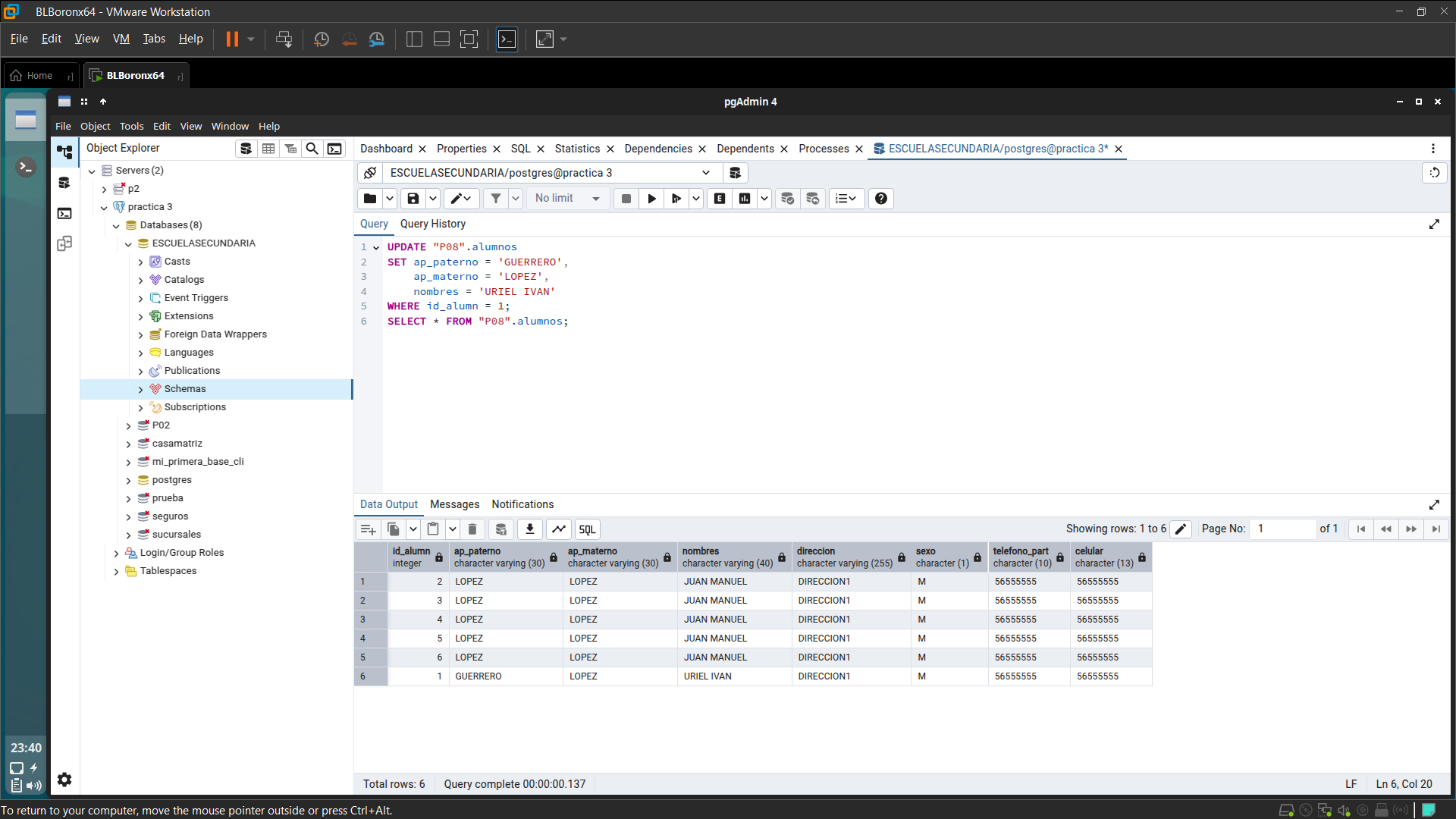


* Explique qué sucedió y como evitarlo.

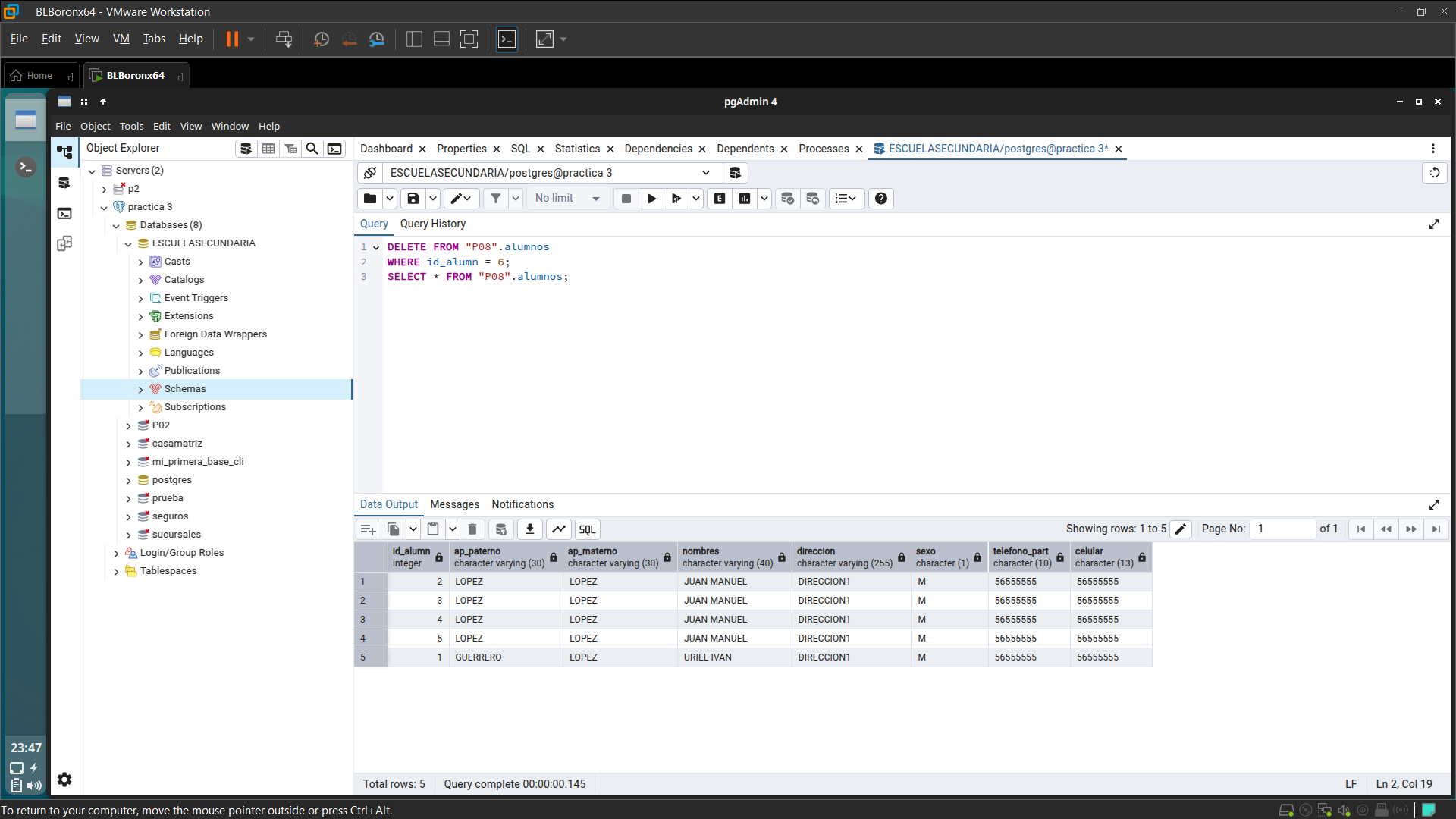
En este caso al usar UPDATE se actualizaron TODAS las filas de la tabla. Y como no usamos un WHERE, el UPDATE afectó a todos los registros de "P08".alumnos. Dando lugar a que todas las filas ahora tienen los mismos valores: 'LOPEZ', 'LOPEZ', 'JUAN MANUEL', etc.

La forma de evitarlo es siempre usando una condición WHERE para limitar que registros se quieren utilizar, esto es importante pues podemos dar lugar a un daño en toda la base de datos o la dificultad de deshacerlo si no se estaba dentro de una transacción.

* Genere un query para actualizar solo los apellidos paternos, maternos y nombres de un solo alumno.

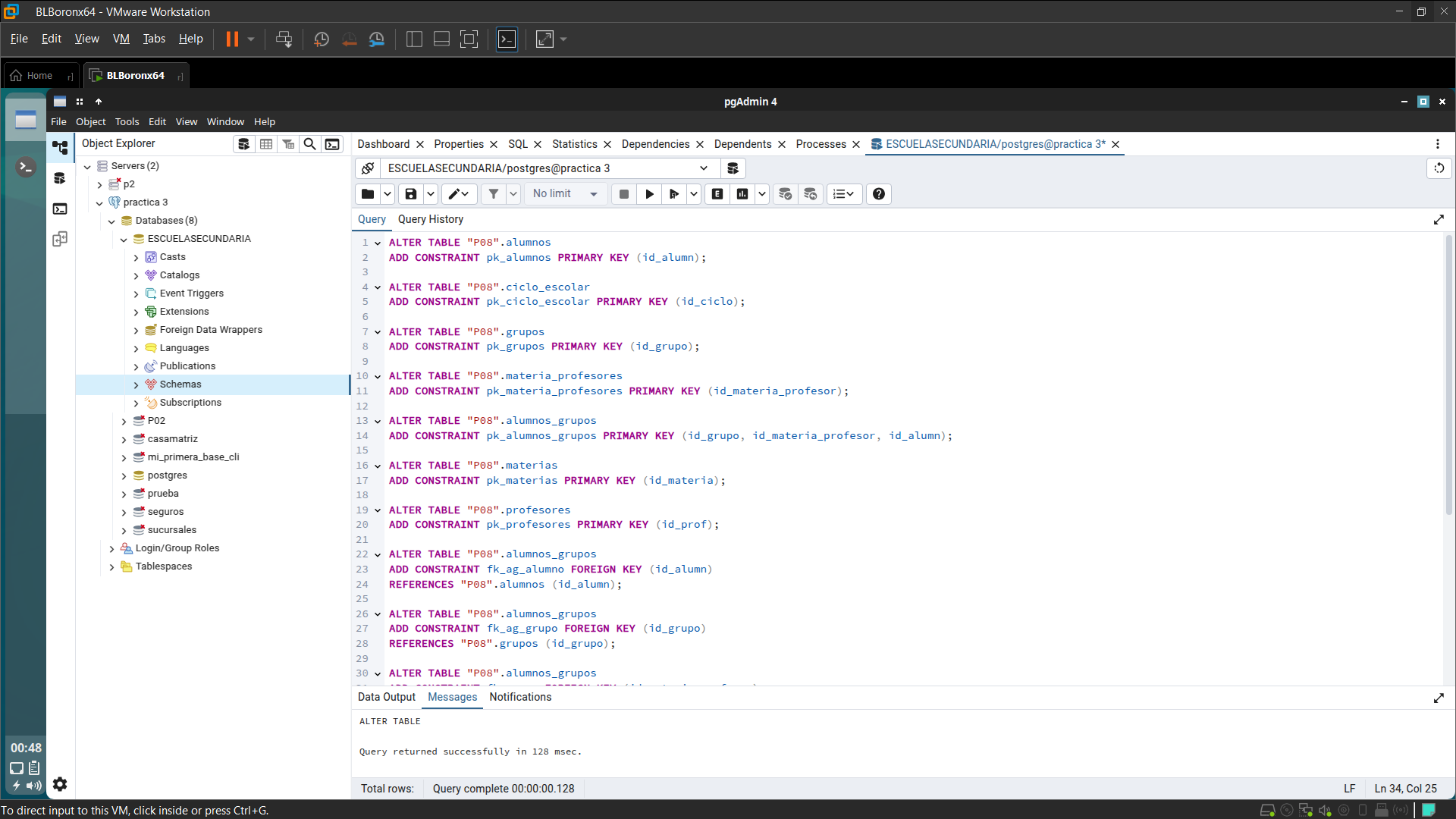


* Genere un query para eliminar los datos de un solo alumno.



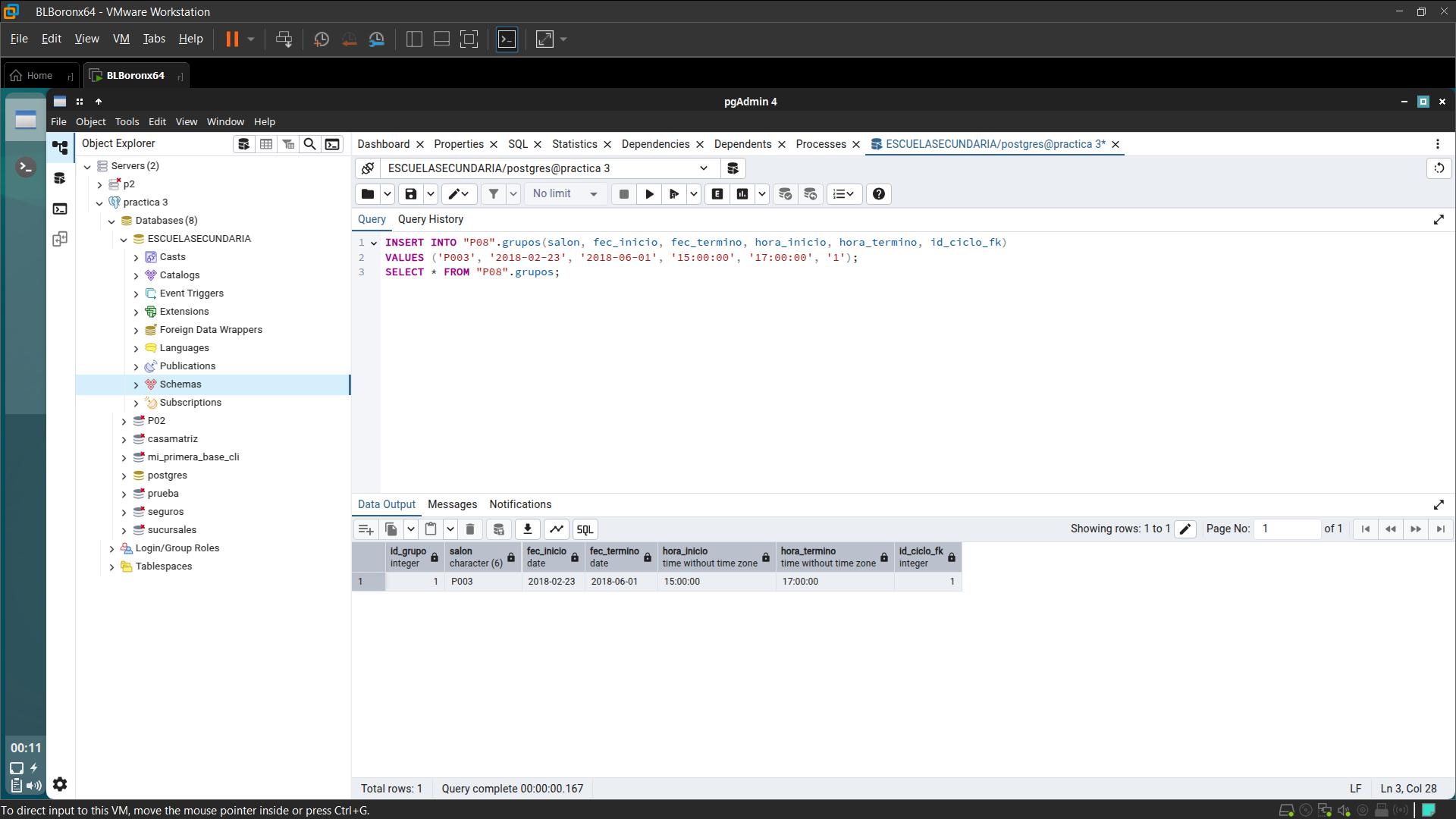
**Actividades 2,3,4**

* Genere los CONSTRAINTS necesarios en toda la base de datos para lograr integridad referencial.



INSERT INTO “P08”.grupos(salon, fec\_inicio, fec\_termino, hora\_inicio, hora\_termino, id\_ciclo\_fk) VALUES ('P003', '2018-02-23', '2018-06-01', '15:00:00', '17:00:00', ‘1’);

SELECT \* FROM “P08”.grupos;



* Explique por qué fue posible ejecutar el query y que haría para evitar que se pierda integridad.

Se pudo ejecutar porque:

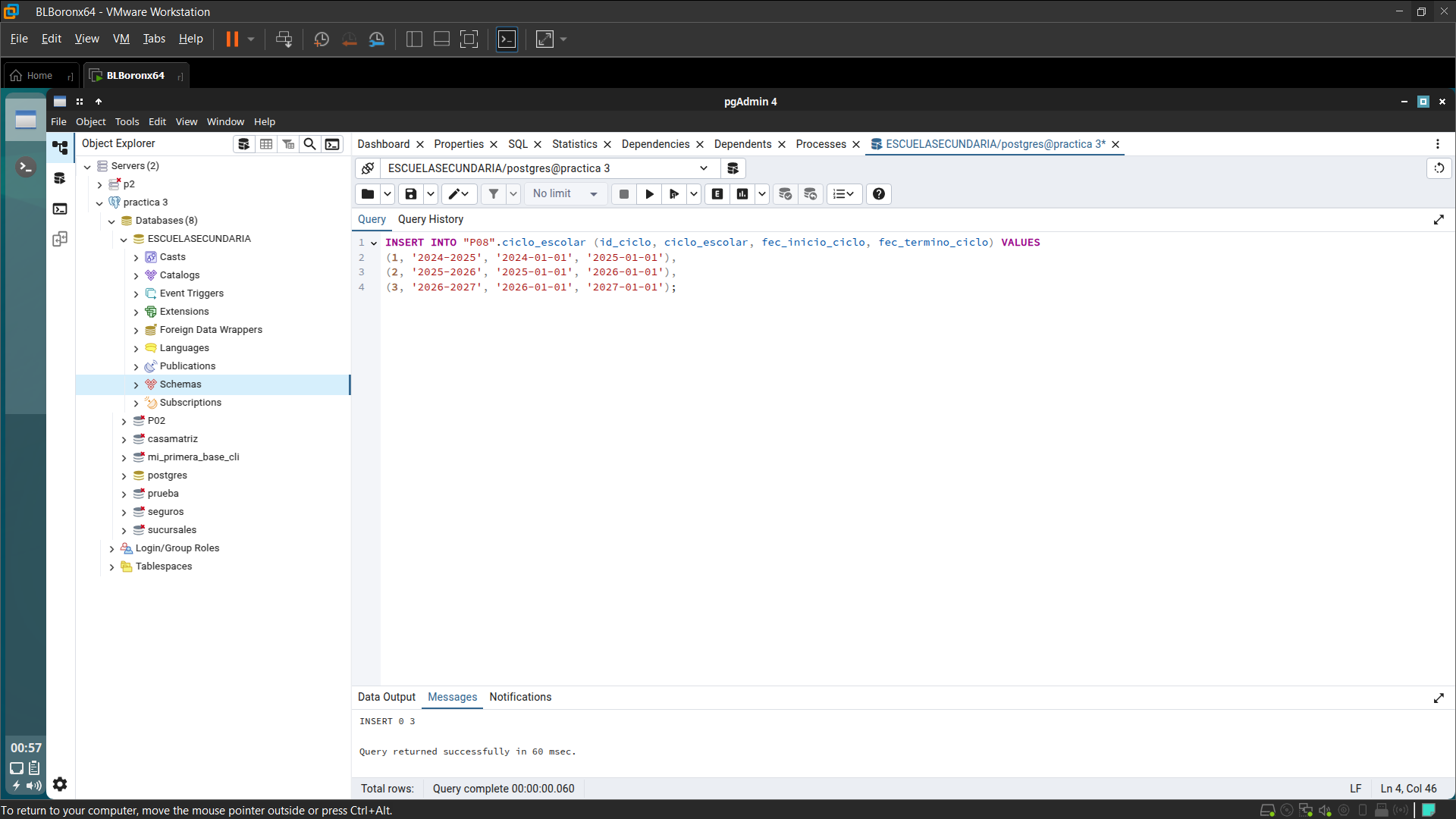
* Todos los campos obligatorios (NOT NULL) están siendo proporcionados en el INSERT.
* No hay restricciones que estén bloqueando los datos que estás insertando en el momento.
* id\_ciclo\_fk es una llave foránea, donde existe el valor 1 en la tabla referenciada. Si el valor 1 no existiera en la tabla de ciclos, PostgreSQL debería rechazar el INSERT si la llave foránea (FOREIGN KEY) está bien definida.

Para evitar problemas de integridad debemos saber que:

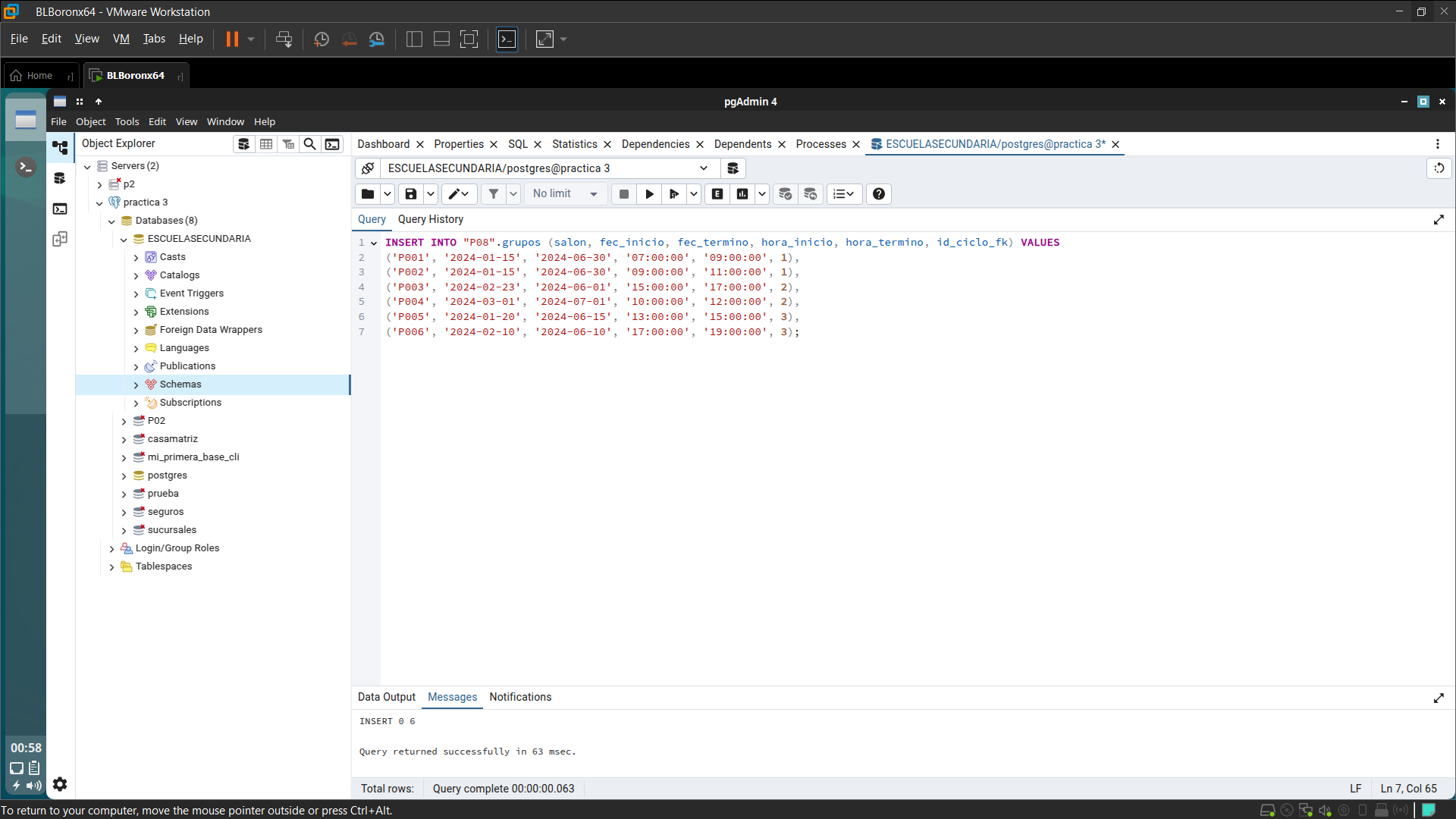
1. La llave foránea (FOREIGN KEY) esté correctamente definida.
2. Definir restricciones de NOT NULL en las columnas que siempre deban tener valor.
3. Manejar correctamente los tipos de datos.
4. Validar los datos antes de insertarlos.

* Inserte al menos 6 grupos.

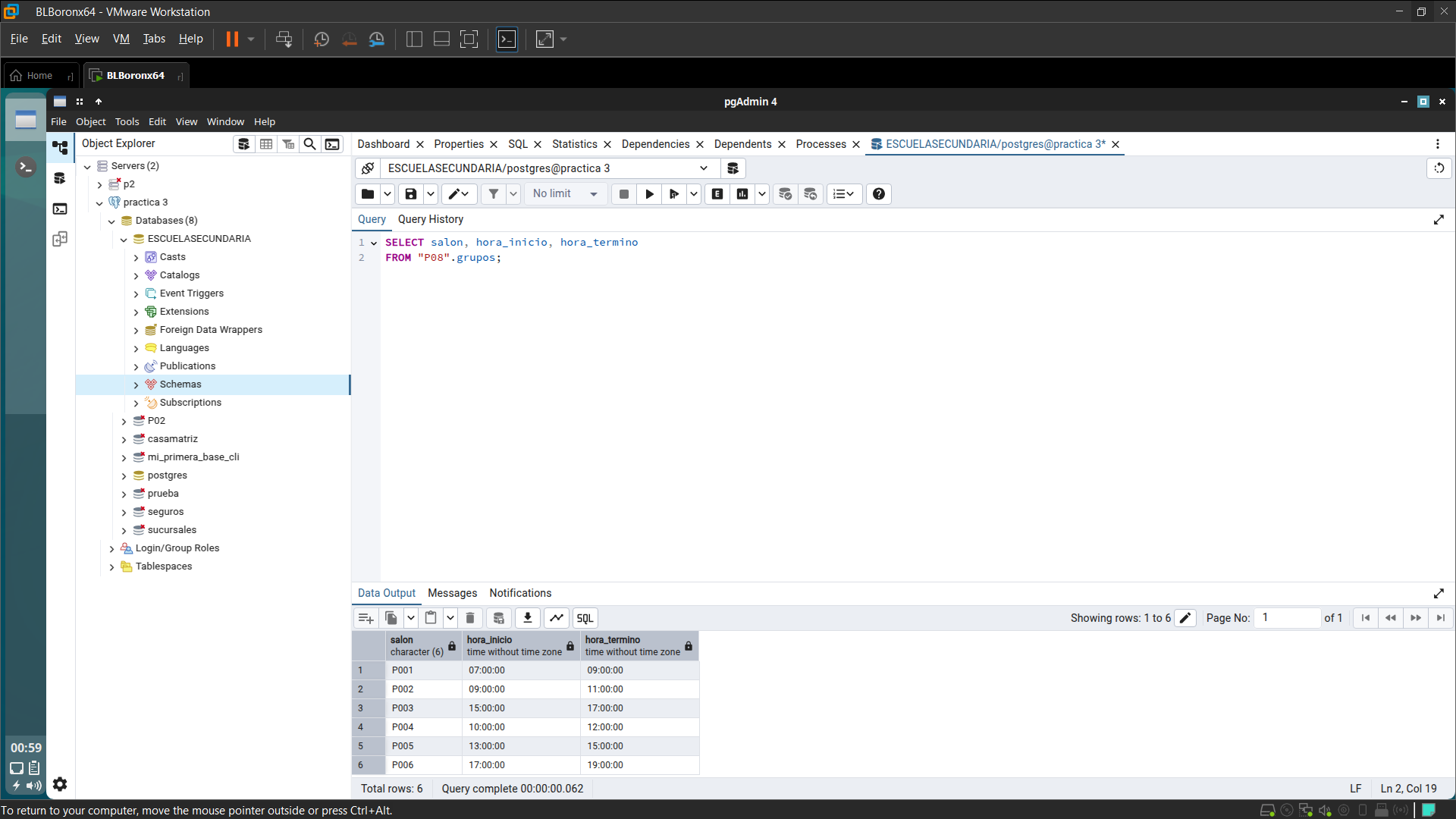
Primero debemos insertar valores a la tabla de ‘ciclo\_escolar’, pues de esta forma no nos arrojará errores al querer insertar valores a la tabla de ‘grupos’, ya que se está referenciando una llave foránea (id\_ciclo)



Una vez hecho el paso anterior donde llenamos conforme a los requerimientos, podemos hacer la inserción de valores para la tabla de grupos.

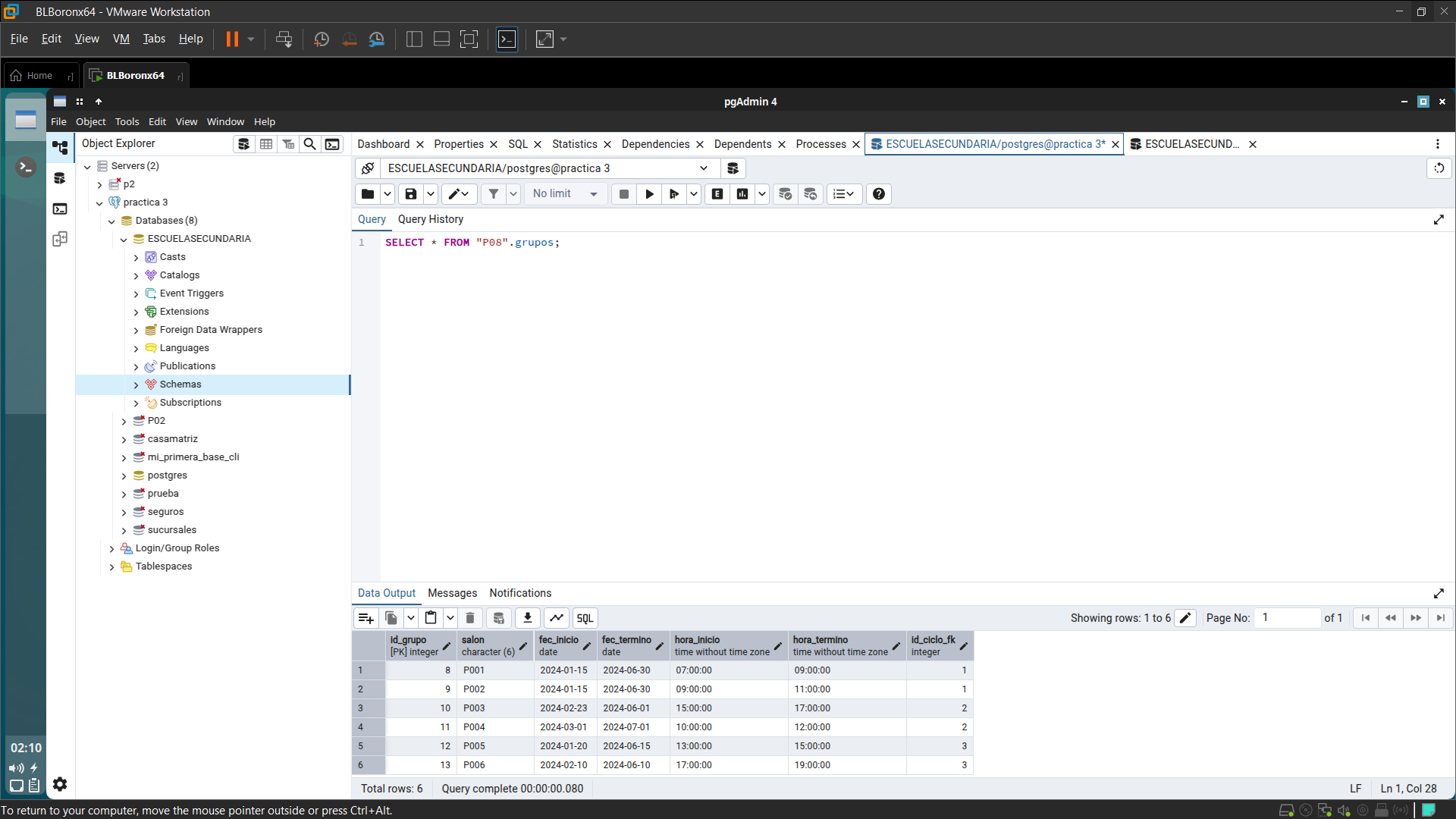


* Genere un query para seleccionar solo el salón, la hora de inicio y hora de termino de todos los grupos diponibles.

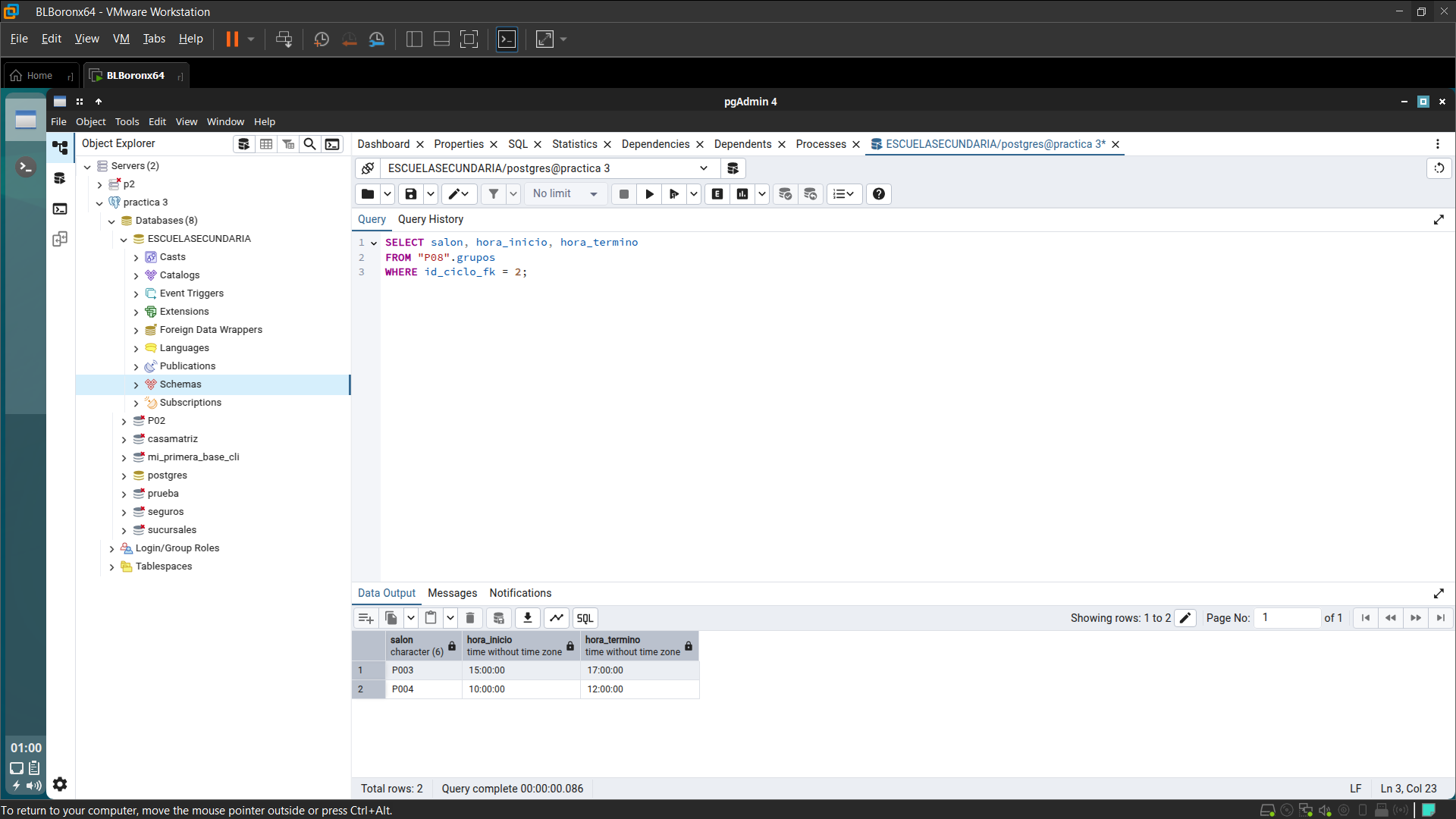


* Genere un query para seleccionar solo el salón, la hora de inicio y hora de termino de todos los grupos disponibles del ciclo 2.

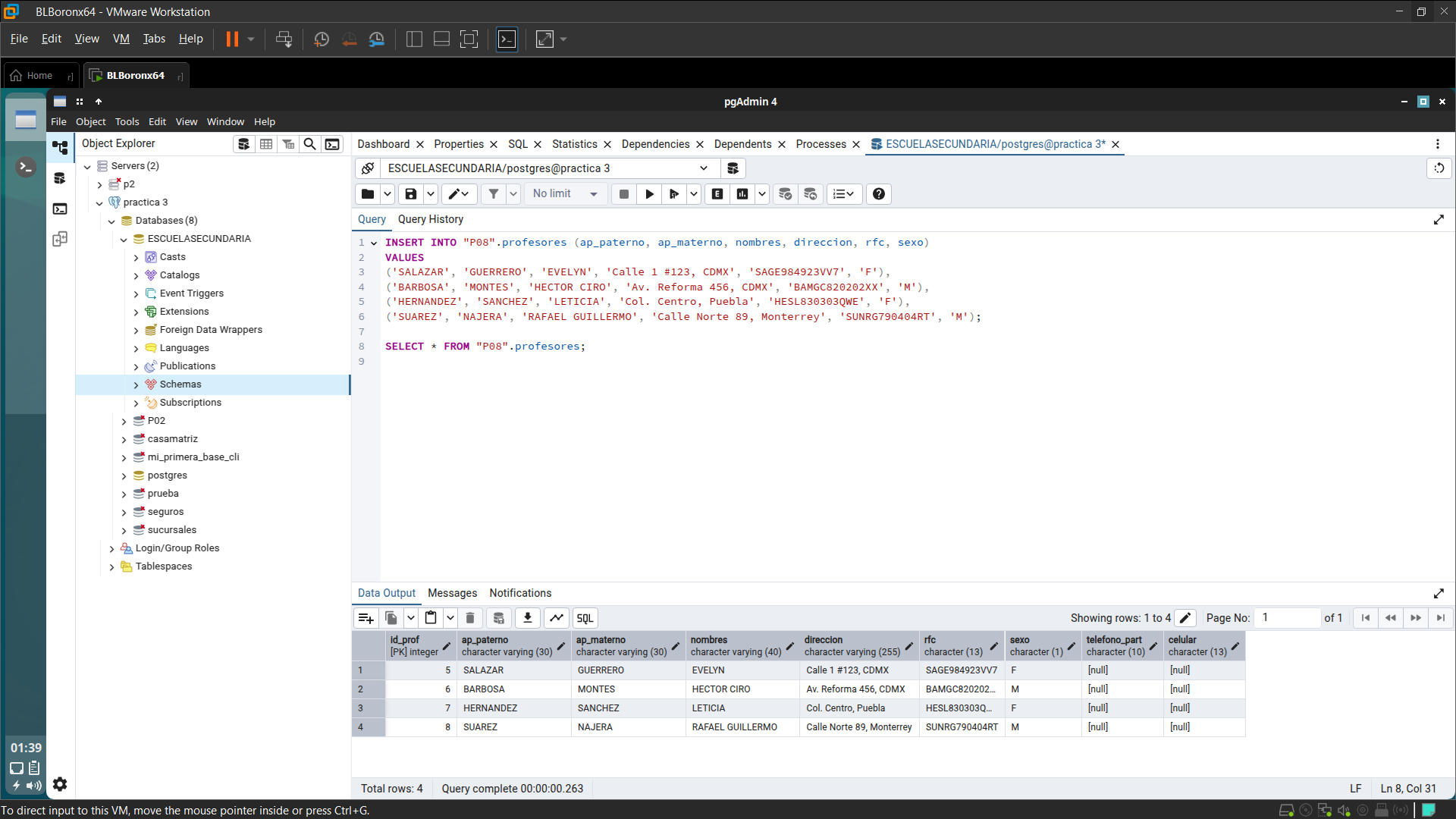
Para poder hacer este paso, primero debemos conocer el ambiente que estamos trabajando y así saber los resultados esperados para la consulta a hacer.



Ahora, para cumplir con la consulta de solo los grupos 2 hacemos uso del WHERE que nos situara en el valor que queramos que se nos muestre dentro de la tabla de ‘grupos’.



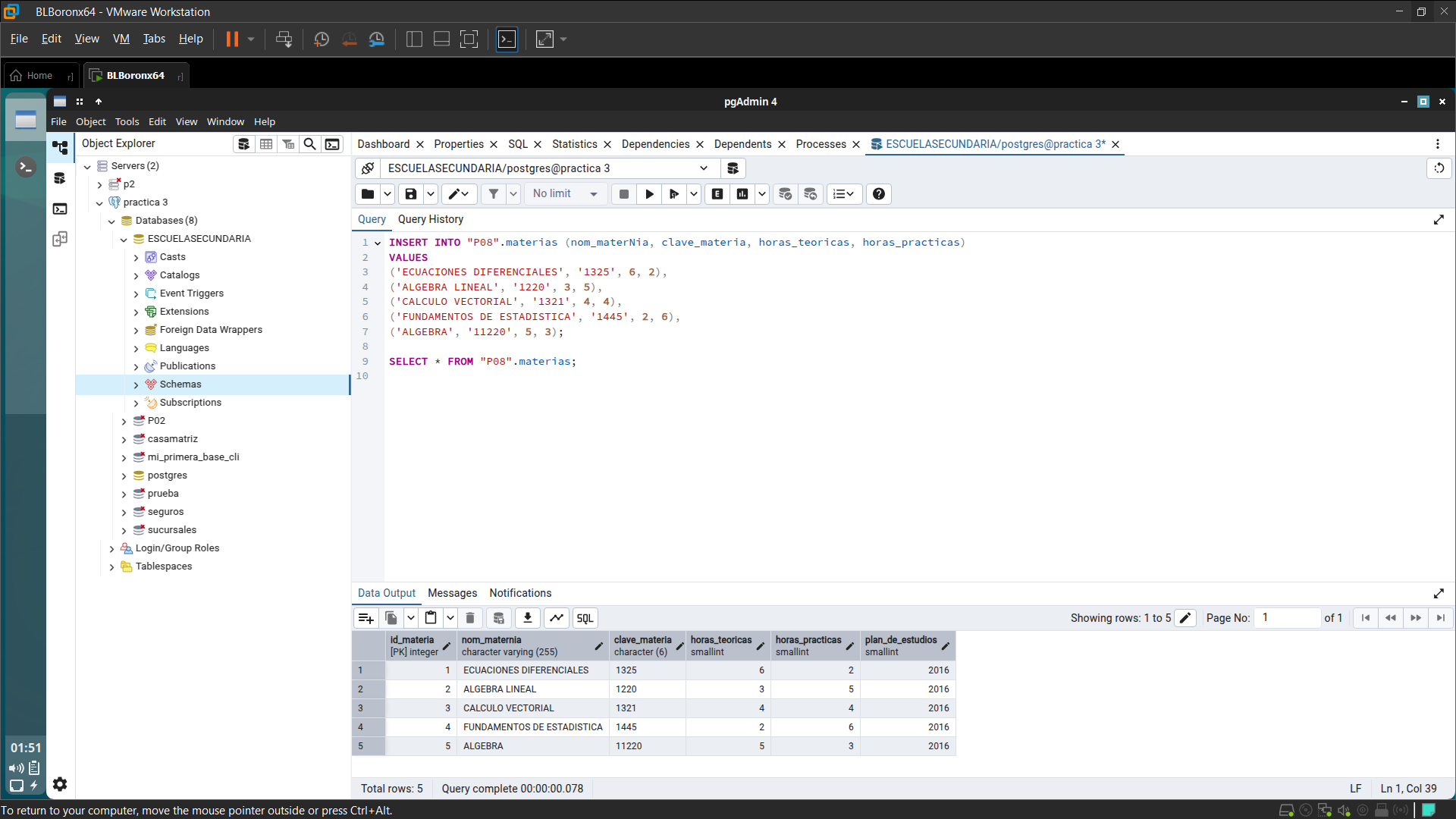
* Genere los query’s para dar de alta al menos 4 profesores.



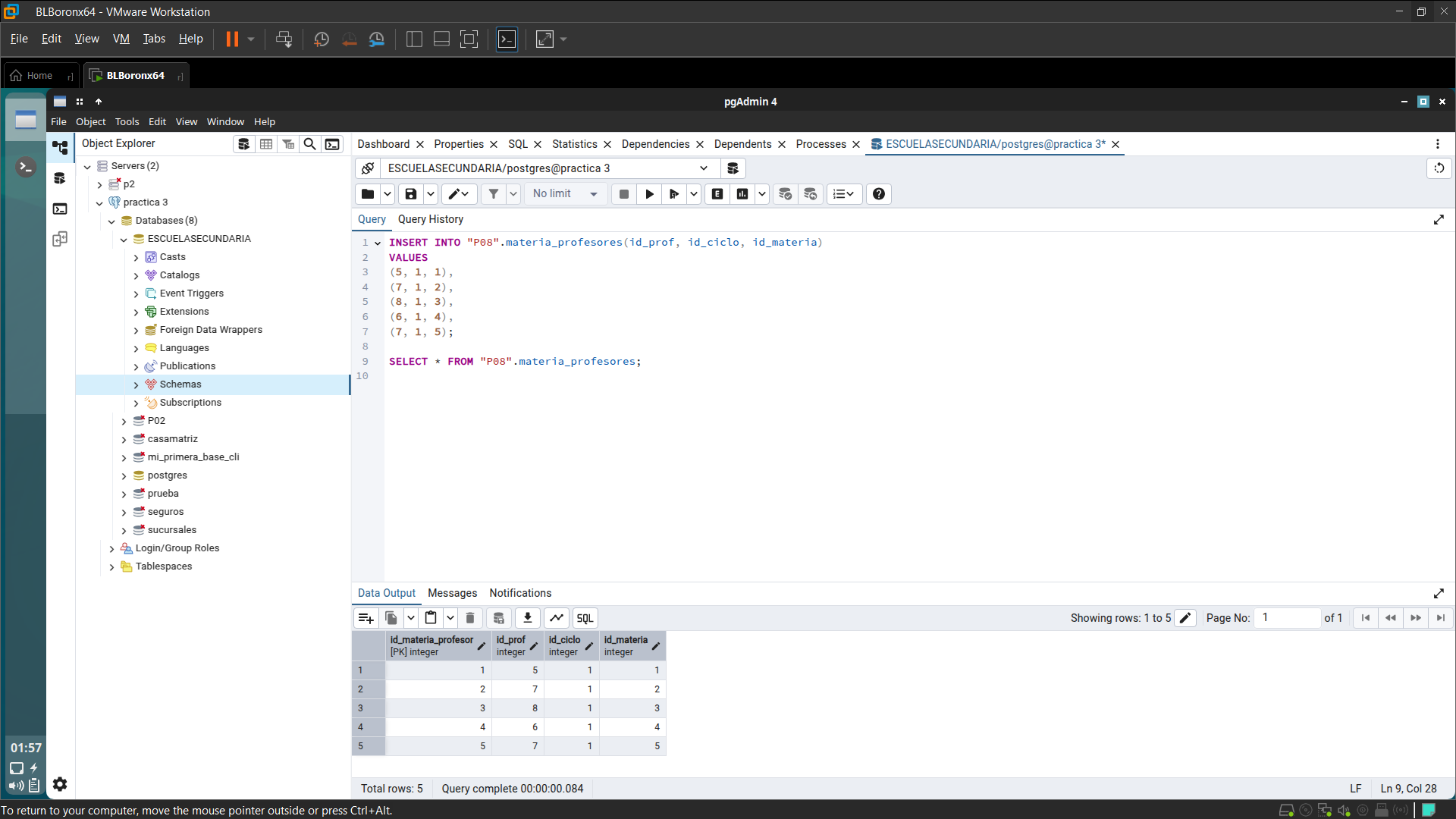
Los teléfonos son dejados vacíos pues la tabla nos permite trabajar así al generar un NULL en caso de omisión de valores en la inserción.

* Genere los query’s para dar de alta al menos esos 4 profesores en 5 materias.

Como anteriormente nos pasó en una pregunta, antes de poder insertar valores a una tabla debemos checar que no se este referenciando a un fk dentro de la misma. Si es el caso, debemos de primero crear la tabla de la fk para así poder trabajar en la inserción siguiente sin error alguno.



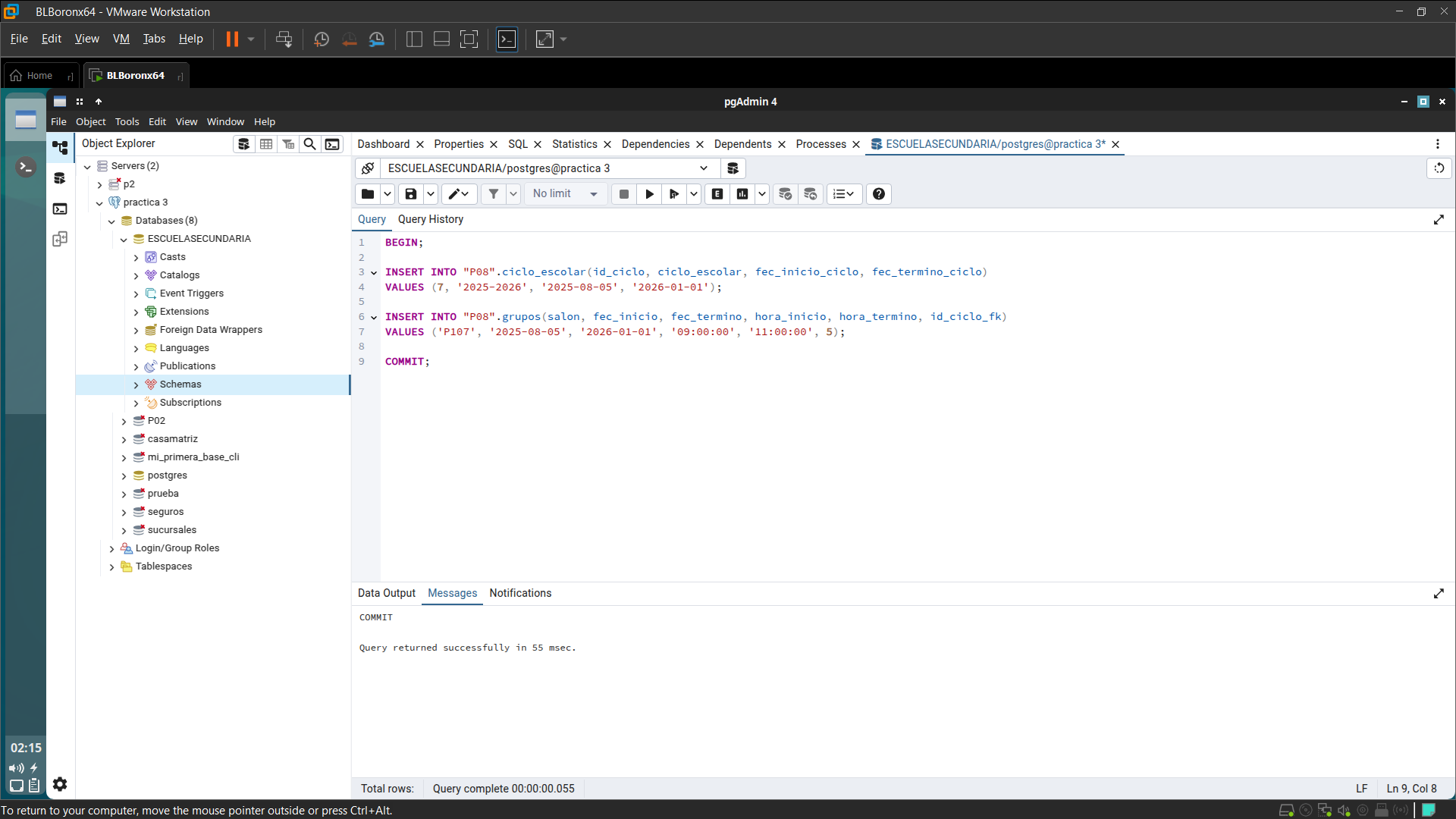
Una vez creada la anterior inserción conforme a los requerimientos necesarios podemos hacer la inserción pedida para el ejercicio sin error alguno.



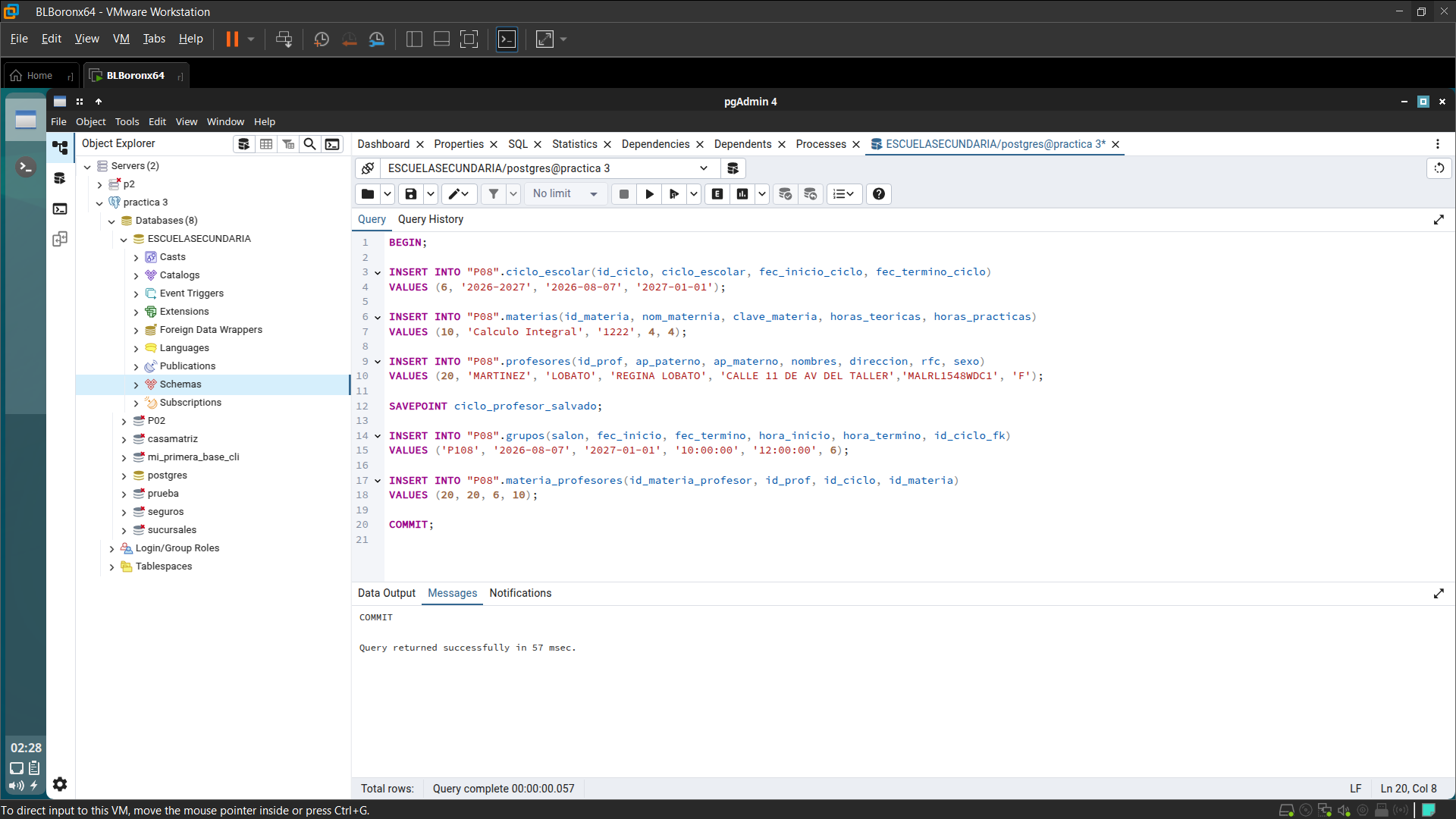
Lea como formar una transacción básica en postgres.

https://www.postgresql.org/docs/15/tutorial-transactions.html

* Genere una transacción para primero dar de alta el ciclo escolar y posteriormente dar de alta un grupo.



* Genere una transacción para dar de alta ciclo, la materia, el profesor el grupo y la materia que imparte el profesor con un punto de salvado para que en caso de falla solo se inserte el ciclo la materia y el profesor utilizando checkpoint y rollback.

****

**Conclusiones**

* Guerrero López Uriel Ivan

Durante esta práctica logré comprender de manera más clara el uso del lenguaje SQL para administrar datos dentro de un sistema de base de datos. El manejo de las instrucciones INSERT, UPDATE y DELETE me permitió interactuar directamente con las tablas, modificando y controlando la información de forma precisa. Al realizar ejercicios prácticos, reforcé los conceptos teóricos y entendí la importancia de utilizar correctamente estas sentencias para mantener la integridad de los datos.

También entendí las principales características de una transacción, observando cómo agrupa varias operaciones en una sola unidad lógica de trabajo. Comprobé que una transacción asegura que los cambios en los datos se realicen de manera correcta, y que se pueden cancelar en caso de errores.

Finalmente, configuré los diferentes niveles de aislamiento que maneja un RDBMS y verifiqué su comportamiento. Pude notar las diferencias entre cada nivel y cómo estos afectan la forma en que las transacciones acceden a los datos cuando se presentan múltiples usuarios trabajando al mismo tiempo.

* Muñoz San Agustin Victoria Monserrat

Gracias a esta práctica logre consolidar mi aprendizaje de las instrucciones INSERT, UPDATE y DELETE, pues me ha permitido comprender cómo interactúan los datos dentro de una base de datos, dándome la capacidad de agregar, modificar o eliminar registros de manera controlada. Además, la capacidad de realizar estas operaciones de manera rápida y confiable es crucial cuando se trabaja con grandes volúmenes de datos, algo que ahora valoro más que antes. La comprensión de las transacciones ha sido algo importante, ya que he logrado entender que permiten agrupar varias operaciones de manera que se asegure la atomicidad y consistencia de los datos, incluso si alguna de esas operaciones falla.

Finalmente, dentro de las mismas transacciones conocí lo que son los niveles de aislamiento y me he dado cuenta que cada nivel tiene sus propias implicaciones, y ahora soy capaz de elegir el nivel de aislamiento adecuado según las necesidades de concurrencia, rendimiento y seguridad. Esta capacidad me brinda una flexibilidad única, permitiéndome adaptar mis soluciones a diferentes contextos y asegurando que el sistema funcione de manera eficiente bajo diversas condiciones de uso.

**Referencias**

IBM. (2021). Lenguaje de manipulación de datos. Recuperado el 20 de abril de 2025 de https://www.ibm.com/docs/es/psfa/7.1.0?topic=categories-data-manipulation-language

Jürgenson, S. (2023). DML: Lenguaje de manipulación de datos. Recuperado el 20 de abril de 2025 de https://appmaster.io/es/blog/dml-lenguaje-de-manipulacion-de-datos