**Informe – Proyecto Entrega #1**

Juan Pablo Hernández – 202122707

**1. Introducción:** La implementación de una base de datos se debe construir a partir de los requerimientos funcionales y no funcionales del negocio. Con estos, se garantiza que la base de datos diseñada pueda responder a lo que el negocio pide y, además, facilita modificaciones en el futuro en caso de que los requerimientos cambien o sean alterados. En esta iteración, la base de datos se diseñó con el fin de responder a las consultas y modificaciones descritas por los requerimientos funcionales, aunque fácilmente puede ser expandida o modificada para mejorar las respuestas a cada requerimiento. Adicionalmente, se tuvieron en cuentas los requerimientos no funcionales en la medida de lo posible.

**2. Modelo UML:**

****

**3. Modelo Entidad/Relación**

****

**4. Modelo Relacional**

****

**5. Nivel de Normalización**

Al analizar el modelo relacional, se identifica lo siguiente en relación con el nivel normalización.

1NF: El modelo sí se encuentra en 1NF, ya que todos los atributos del modelo son atómicos. Es decir, son datos simples y no compuestos.

2NF: El modelo también se encuentra en 2NF. Esta forma normal se logra si no hay dependencias parciales desde los atributos primos de la base de datos. Como se ve en el modelo, todos los atributos no primos (aquellos no pertenecientes a una llave primaria) dependen por completo de la llave primaria de su relación. Es decir, las llaves primarias definen por completo todos los demás atributos de la tabla.

3NF: En cuanto a la 3NF, el modelo no presenta dependencias transitivas entre atributos no primos de una relación. Es decir, no existe un atributo no primo que se defina por otro atributo no primo. La llave primaria de una relación es capaz de definir sin transitividad a todos los demás atributos de la relación.

BCNF: Por lo general, cuando se está en 3NF, también se está en BCNF. En este caso, esto sí es cierto, ya que no hay dependencias triviales entre atributos primos. En otras palabras, no hay dependencias parciales entre exclusivamente **atributos primos**.

**6. Creación y Borrado de Tablas e Inserción de Datos**

Para crear y borrar las tablas, se utilizaron los archivos SQL de CreateTables.SQL y DropTables.SQL. Estos se crearon usando Data Modeler y consultas sobre la base de datos.

Para la insercíon de datos, se utiliza el archivo TableData.SQL, que fue construido manualmente a partir de la estructura de la base de datos.

**7. Requerimientos Funcionales (RF)**

**RF1:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **SQL** | **Before** | **After / Results** |
| **CREATE** | NSERT INTO TIPOSUSUARIO (ID, TIPO)  VALUES (1, 'Cliente');  INSERT INTO TIPOSUSUARIO (ID, TIPO)  VALUES (2, 'Recepcionista');  INSERT INTO TIPOSUSUARIO (ID, TIPO)  VALUES (3, 'Empleado');  INSERT INTO TIPOSUSUARIO (ID, TIPO)  VALUES (4, 'Administrador');  INSERT INTO TIPOSUSUARIO (ID, TIPO)  VALUES (5, 'Gerente'); |  |  |
| **READ** | SELECT ID, TIPO  FROM TIPOSUSUARIO  WHERE TIPO = 'Empleado'; | **-** |  |
| **UPDATE** | UPDATE TIPOSUSUARIO  SET TIPO = 'Empleado'  WHERE TIPO = 'Cliente'; |  |  |
| **DELETE** | DELETE  FROM TIPOSUSUARIO  WHERE TIPO = 'Recepcionista'; |  |  |

**RF2:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **SQL** | **Before** | **After / Results** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**RF3:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **SQL** | **Before** | **After / Results** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**RF4:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **SQL** | **Before** | **After / Results** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**RF5:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **SQL** | **Before** | **After / Results** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**RF6:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **SQL** | **Before** | **After / Results** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**RF7:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **SQL** | **Before** | **After / Results** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**RF8:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **SQL** | **Before** | **After / Results** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**RF9:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **SQL** | **Before** | **After / Results** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**RF10:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **SQL** | **Before** | **After / Results** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**RF11:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **SQL** | **Before** | **After / Results** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**8. Requerimientos No Funcionales (RNF)**

RNF1: Privacidad: En el estado actual de la aplicación, todavía no se chequea que ciertos tipos de usuario puedan realizar ciertas operaciones. Por ahora, se garantiza que las consultas necesarias sí se puedan realizar.

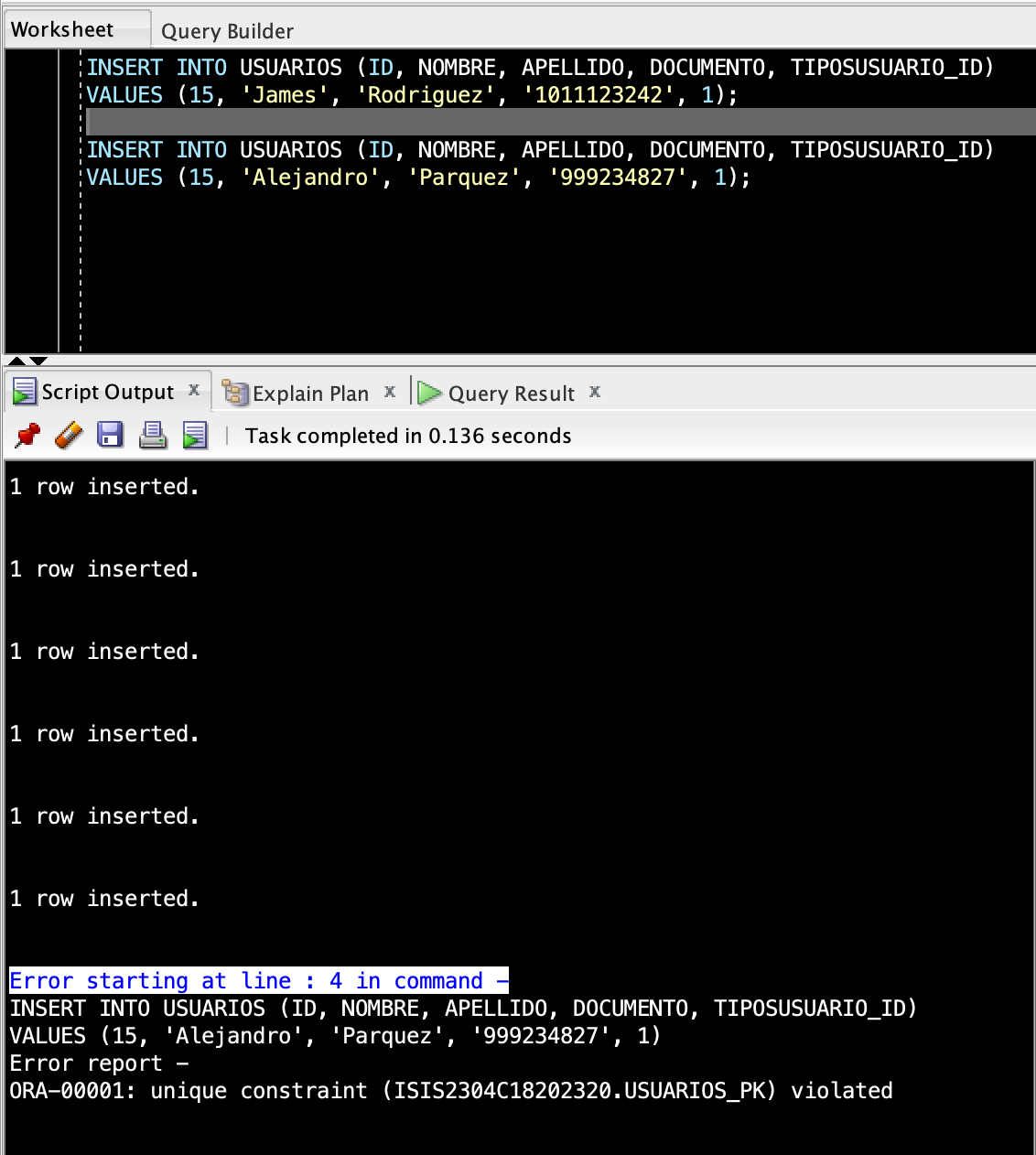
RNF2: Persistencia: La aplicación si persiste la información que es insertada según las especificaciones de la base de datos. Esto permite que se hagan consultas tiempo después de que las tablas sean creadas y llenadas de datos.

RNF3: Distribución: La base de datos actualmente está centralizada ya que almacena y administra todos los datos en un único servidor central, lo que simplifica la gestión y reduce la redundancia de datos. Ofrece control centralizado y seguridad, pero puede carecer de escalabilidad y ser vulnerable a fallos en el servidor central.

**9. Escenarios de Prueba**

1. **Unicidad de Tuplas**

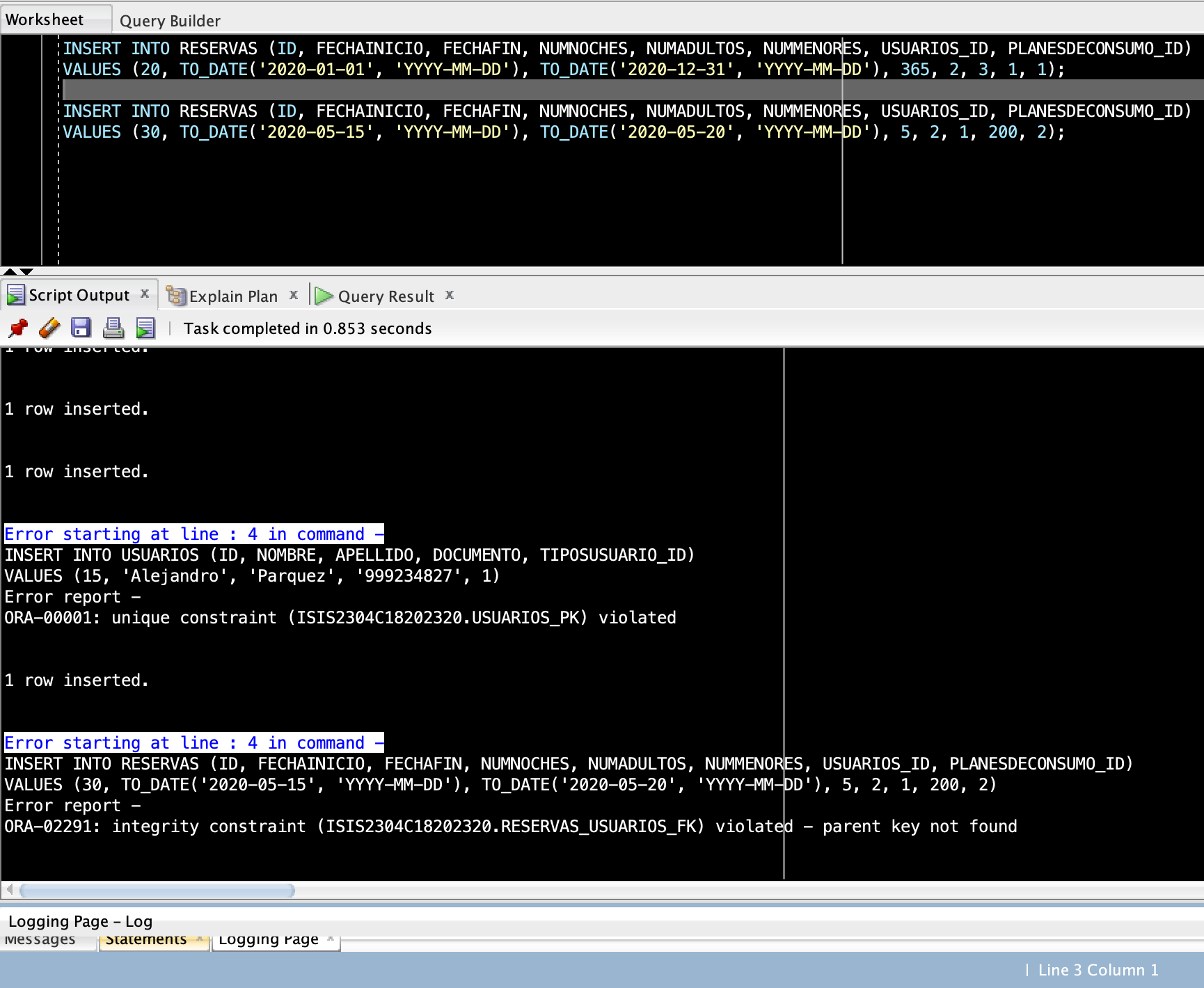
Ejemplo: Tabla Usuarios:

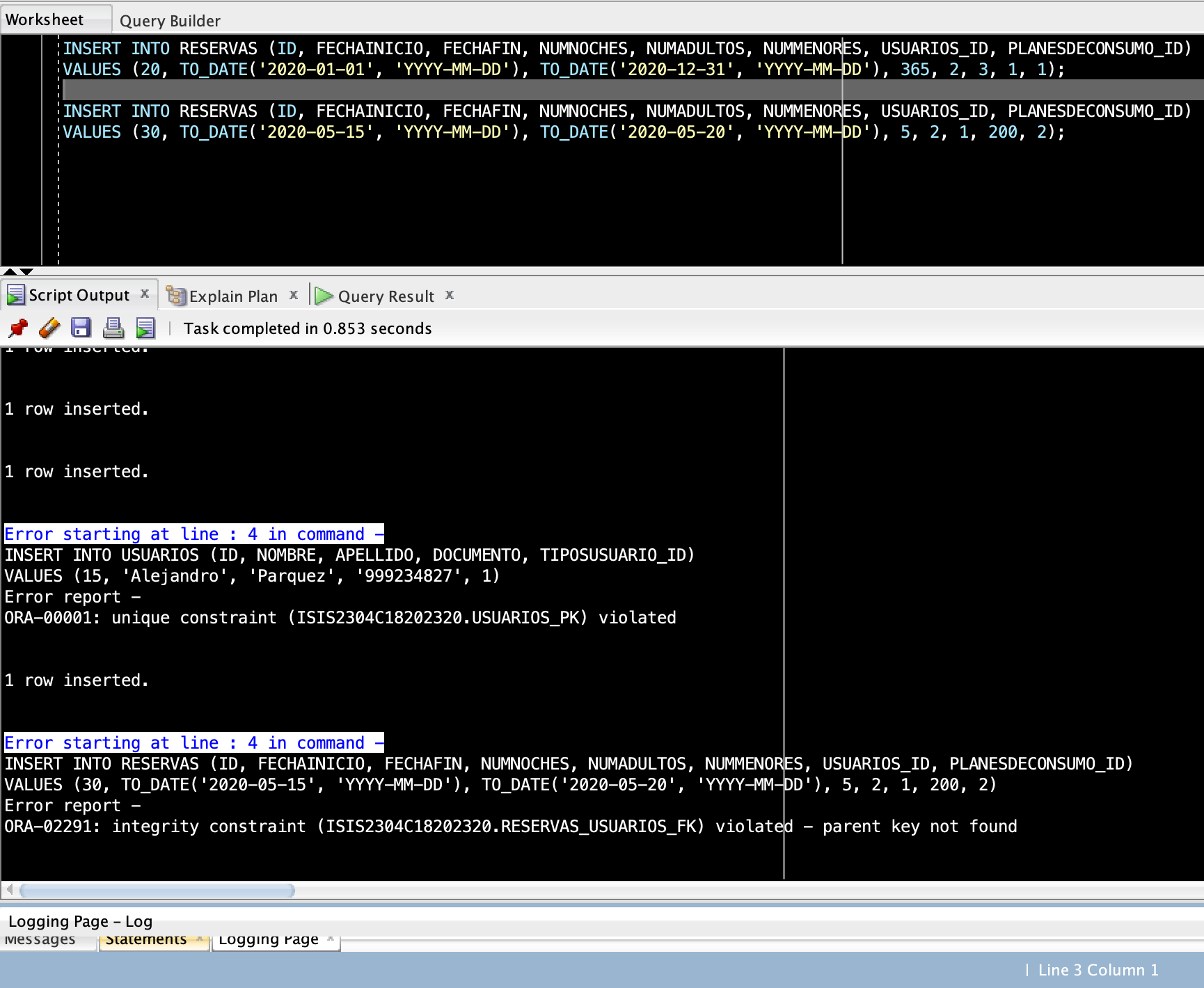


Se intentó insertar dos usuarios con el mismo ID (15). El primero se inserta exitosamente, pero el segundo viola la condición de PK y no es añadida. Esto garantiza la unicidad de las tuplas para toda tabla que tenga una condición de PK bien establecida.

1. **Integridad de las Foreign Keys**

Ejemplo: Tabla Reservas

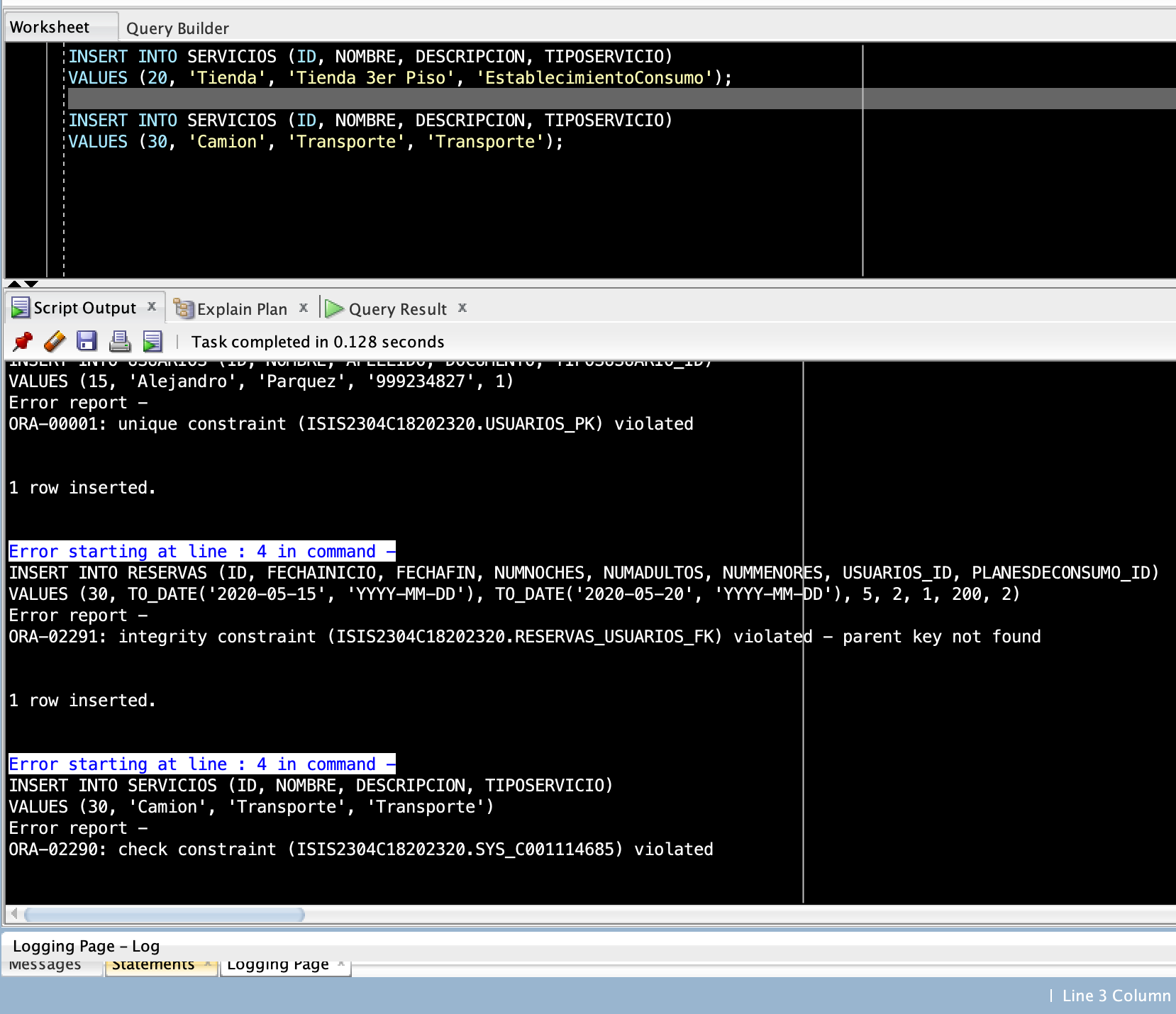


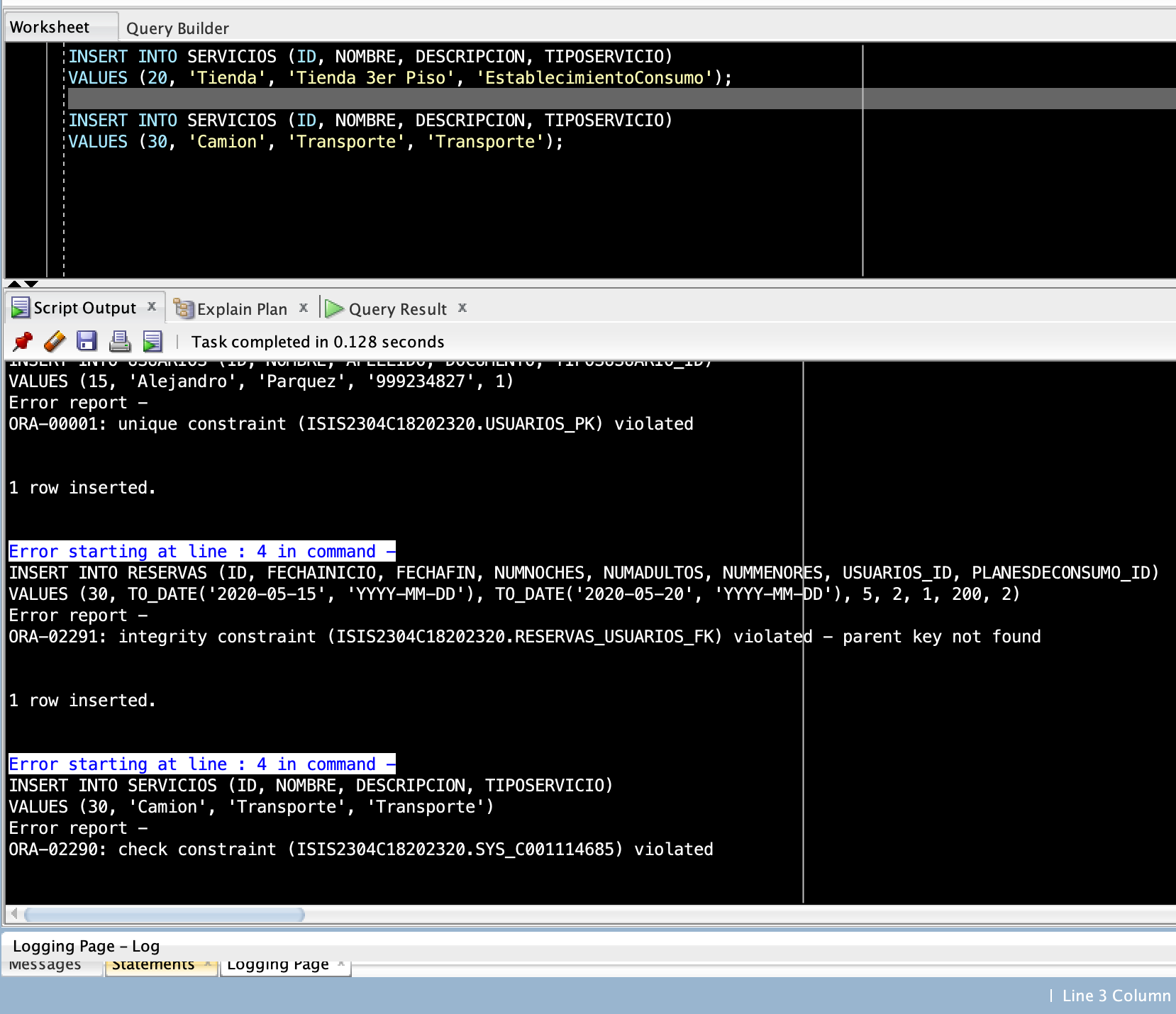


Se intentó insertar una reserva referenciando un id de usuario que no existe (200). La primera se inserta exitosamente ya que referencia un id válido, pero la segunda viola la condición de FK, ya que no encuentra la llave que está intentando referenciar y no es añadida. Esto garantiza que las relaciones se construyan correctamente y que las referencias estén bien establecidas.

1. **Restricciones de Chequeo**

Ejemplo: Servicios:





Se intentó insertar un servicio con un valor en el atributo de tiposervicio inválido (Transporte). El primero se inserta exitosamente ya que tiene un tiposervicio válido definido en el check, pero el segundo viola la condición del constraint, ya que no encuentra el valor proveído en la lista de valores válidos y no es añadida. Esto garantiza que todos los servicios insertados en la tabla sean de uno de los múltiples tipos válidos de servicio que describe el negocio.