EVALUACIÓN
BACKEND
PL/SQL

REGLAS

Reglas

- Debe utilizar **Oracle 11gR2 o superior**
- Definir los índices que considere necesarios.
- No utilizar **cursores implícitos**.
- Todos los procedimientos, funcionalidades, records y tablas PLSQL que se construyan deben estar contenidas o pertenecer a un **paquete** de base de datos.
- Las sentencias **DML** y **DDL** utilizadas deben ser consignadas en un archivo con extensión .sql.

HABILIDADES

SQL
100 %

PL/SQL
100 %

Best Practices
80 %

GIT
100 %

ENTREGA

Si construyo múltiples scripts, estos deben estar organizados y numerados en el orden que deben ser ejecutados, con el objetivo de facilitar la replicación de los objetos de BD en el ambiente de calificación.

Debe enviar la ruta del repositorio público utilizado (GitHub, GitLab, Bitbucket, etc.) al correo electrónico que le suministro esta prueba.

Las pruebas que no sean enviadas dentro del tiempo establecido o cuya ruta del repositorio no sea accesible, no serán calificadas.

Tiempo: 1 Día

CONTEXTO

Problemática

En el creciente mundo empresarial y comercial, las empresas son los nodos medulares de la economía, por eso es imprescindible lograr disponer de los datos suficientes para analizar el patrimonio fluctuante del mercado, razón por la cual el gremio de comercio desea tener a su disposición una herramienta que les permita conocer de forma rápida y centralizada la información de los comerciantes y sus respectivos establecimientos, con el objetivo de analizar el mercado y tomar decisiones orientadas al progreso colectivo.

En este punto es donde entramos nosotros, para este lograr esta gran hazaña nos han solicitado construir una aplicación para condensar la información de los comerciantes y establecimientos con el objetivo de apoyar los procesos operativos esenciales de la agremiación nacional de comercio.

RETO 01

Modelo de Datos

Construir un modelo de datos donde se pueda organizar y almacenar la siguiente información:

- **Comerciante:** Nombre o razón social, Departamento, Municipio, Teléfono, Correo Electrónico, Fecha de Registro, Estado (Registrado, Activo, Inactivo, Suspendido y Cancelado)
- **Establecimiento:** Nombre del Establecimiento, Ingresos (debe permitir valores con dos decimales) y Número de Empleados

Notas:

- Debe normalizar el modelo, si y solo si, lo considera necesario
- Las entidades deben incluir los campos de auditoría (Fecha de actualización y Usuario)
- Se debe incluir en la entrega el modelo entidad-relación (MER)

RETO 02

Secuencias, Identificadores y Auditoría

- El identificador único de cada tabla debe ser generado por una **secuencia** de Base de Datos y cada secuencia de Base de Datos debe ser gestionada por un **trigger**.
- Los campos de auditoría (Fecha de actualización y Usuario) de cada entidad deben ser actualizados mediante un **trigger** por cada sentencia DML de inserción y actualización que se realice sobre la entidad respectiva

RETO 03

Datos Semilla

Generar los datos semilla para las entidades de Comerciantes y Establecimientos, respetando las siguientes cantidades:

Comerciantes: 10 registros
Establecimientos: 30 registros

Notas:

- Procure que sean datos aleatorios y diferentes para cada registro
- La cantidad de establecimientos por comerciante debe ser aleatoria

RETO 04

Paquete de Primer Nivel

Construir un paquete con los procedimientos o funciones necesarias para realizar las operaciones **CRUD** de la entidad **comerciante**, las condiciones son las siguientes:

- Las operaciones de **consultar por id** y **consultar** deben construirse utilizando **funciones**
- Las operaciones de **consultar por id** y **consultar** deben retornar la siguiente estructura: Nombre o razón social, Departamento, Municipio, Teléfono, Correo Electrónico, Fecha de Registro, Estado, Total Activos y Cantidad de Empleados
- Las operaciones de **consultar por id** y **consultar** deben incluir dos campos calculados (**Total Activos** y **Cantidad de Empleados**) en su respuesta, los cuales deben corresponder a la sumatoria de los valores diligenciados en los campos **Ingresos** y **Número de Empleados** respectivamente en base a cada uno de los establecimientos asociados al comerciante
- La operación de **consultar** debe implementar filtros por los campos **Nombre o razón social**, **Municipio**, **Fecha de Registro** y **Estado**
- La operación de **consultar** debe implementar **paginación**
- Las operaciones de **creación**, **actualización** y **eliminación** deben construirse utilizando procedimientos
- Las operaciones de **creación** y **actualización** deben solicitar los campos: Nombre o razón social, Departamento, Municipio, Teléfono **(Opcional)**, Correo Electrónico **(Opcional)**, Fecha de Registro y Estado
- Implementar validaciones a los parámetros de entrada (numérico, alfanumérico, estructura correo y obligatoriedad)
- Los procedimientos deben disponer de dos parámetros de salida (**Código de Error** y **Mensaje de Error**).
- Si la ejecución se realiza de forma correcta debe retornar el valor cero como **código de error** y nulo en el **mensaje de error**, de lo contrario, debe retornar el código y mensaje de error capturados en el bloque de excepciones.
- Puede construir las funciones auxiliares que considere necesarias

RETO 05

Reporte de Comerciantes

Construir una función que retorne un **cursor referenciado o por referencia** con la información de los comerciantes **registrados** y **activos**, bajo las siguientes condiciones:

- La estructura del **cursor referenciado o por referencia** debe contener el Nombre o razón social, Departamento, Municipio, Teléfono, Correo Electrónico, Fecha de Registro, Estado, Cantidad de Establecimientos, Total Activos y Cantidad de Empleados
- Los campos **Cantidad de Establecimientos**, **Total Activos** y **Cantidad de Empleados** deben ser calculados en base a los establecimientos asociados al comerciante
- Los **comerciantes** deben estar ordenados de forma **descendente** por la **cantidad de establecimientos** que tengan asociados
- Puede construir las funciones auxiliares que considere necesarias

EVALUACIÓN BACKEND

SPRING BOOT

REGLAS

Reglas

- Debe utilizar **Java 11 o superior**
- Debe utilizar **Spring Boot**
- Debe construir mínimo **tres pruebas unitarias** que considere necesarias utilizando **JUnit**
- Debe implementar **Swagger** o compartir la colección de **Postman** utilizada
- Utilizar arquitectura limpia y principios **SOLID**
- Se debe construir el **Docker File** para implementar el despliegue contenerizado utilizando **Docker**

HABILIDADES

- Java — 100 %
- Spring Boot — 100 %
- JUnit — 80 %
- Best Practices — 80 %
- GIT — 100 %

ENTREGA

Debe enviar la ruta del repositorio público utilizado (GitHub, GitLab, Bitbucket, etc.) al correo electrónico que le suministro esta prueba.

Las pruebas que no sean enviadas dentro del tiempo establecido o cuya ruta del repositorio no sea accesible, no serán calificadas.

Tiempo: 1 Día

CONTEXTO

Problemática

En el creciente mundo empresarial y comercial, las empresas son los nodos medulares de la economía, por eso es imprescindible lograr disponer de los datos suficientes para analizar el patrimonio fluctuante del mercado, razón por la cual el gremio de comercio desea tener a su disposición una herramienta que les permita conocer de forma rápida y centralizada la información de los comerciantes y sus respectivos establecimientos, con el objetivo de analizar el mercado y tomar decisiones orientadas al progreso colectivo.

En este punto es donde entramos nosotros, para este lograr esta gran hazaña nos han solicitado construir una aplicación para condensar la información de los comerciantes y establecimientos con el objetivo de apoyar los procesos operativos esenciales de la agremiación nacional de comercio.

RETO 06

Web API - Seguridad

- Crear un **endpoint** para realizar la **autenticación (login)** del usuario, recibiendo en el cuerpo de la petición los siguientes datos:
  - Correo Electrónico
  - Contraseña
- Implementar **JWT**, de tal forma que al realizar el proceso de **autenticación (login)** se genere un **token** cuyo tiempo de expiración sea de **1 hora**.
- Implementar **Autorización** por roles (**Administrador** y **Auxiliar de Registro**) para controlar el acceso a cada endpoint del API.

Notas:

- El **endpoint** de **login** debe ser **público**, es decir, no debe solicitar un **token JWT** para su ejecución
- Debe implementar una política de **CORS** para asegurar el consumo controlado de las APIs

RETO 07

Web API - Listas de Valores

- Crear un **endpoint** para cada una de las listas de valores de los campos **Departamento** y **Municipio**, las cuales serán utilizadas en el **CRUD** de la entidad - **Comerciantes**
- Implementar **Cache en Memoria** para evitar accesos adicionales a la base de datos
- Se debe utilizar el mapeo de entidades estándar o un ORM (Hibernate o Spring Boot JPA)

Notas:

- Todos estos **endpoints** deben ser **privados**, es decir, deben solicitar un **token JWT** para su ejecución
- Implementar estandarización de la respuesta HTTP de los endpoints

RETO 08

Web API - CRUD - Comerciante

- Crear un **endpoint** para cada una de las operaciones **CRUD** de la entidad - **Comerciante**
  - Consulta Paginada (5 registros por página por defecto)
  - Consultar por Id
  - Crear
  - Actualizar
  - Eliminar
- Las operaciones de **crear** y **actualizar** deben solicitar los campos: Nombre o razón social, Departamento, Municipio, Teléfono **(Opcional)**, Correo Electrónico **(Opcional)**, Fecha de Registro y Estado
- La consulta **paginada** debe permitir filtrar por **Nombre o razón social**, **Municipio**, **Fecha de Registro** y **Estado**
- Se deben utilizar o consumir los procedimientos y/o funciones construidas en el **Reto 4** para este propósito
- Crear un **endpoint** para modificar el estado del comerciante (Activar/Inactivar) utilizando el método HTTP **PATCH**

Notas:

- Implementar validaciones respectivas (tipos de datos, obligatoriedad y formato de correo electrónico) para verificar la información suministrada en los **endpoints** de **creación** y **actualización**
- Todos estos **endpoints** deben ser **privados**, es decir, deben solicitar un **token JWT** para su ejecución
- Implementar estandarización de la respuesta HTTP de los **endpoints**

RETO 09

Web API - CRUD - Establecimiento

- Crear un **endpoint** para cada una de las operaciones **CRUD** de la entidad - **Establecimiento**
  - Consulta por Comerciante (Establecimientos asociados a un comerciante)
  - Crear
  - Actualizar
  - Eliminar
- Las operaciones de **crear** y **actualizar** deben solicitar los campos: Nombre del Establecimiento, Ingresos y Numero de Empleados
- Se debe utilizar el mapeo de entidades estándar o un ORM (Hibernate o Spring Boot JPA)

Notas:

- Implementar validaciones respectivas (tipos de datos y obligatoriedad) para verificar la información suministrada en los **endpoints** de **creación** y **actualización**
- Todos estos **endpoints** deben ser **privados**, es decir, deben solicitar un **token JWT** para su ejecución
- Implementar estandarización de la respuesta HTTP de los **endpoints**

RETO 10

Web API - Reporte Comerciantes

Crear un **endpoint** que permita generar un **archivo plano (.csv)** con la información de los comerciantes **registrados** y **activos**, bajo las siguientes condiciones:

- La estructura del **archivo plano (.csv)** debe contener el Nombre o razón social, Departamento, Municipio, Teléfono, Correo Electrónico, Fecha de Registro, Estado, Cantidad de Establecimientos, Total Activos y Cantidad de Empleados
- Para la generación del archivo plano se debe utilizar como separador el carácter **pipe (|)**, es decir, la barra vertical
- Se deben utilizar o consumir la función construida en el **Reto 5** para este propósito

Notas:

- El **endpoint** debe ser **privado**, es decir, debe solicitar un **token JWT** para su ejecución
- Solo debe ser accesible a usuarios con el rol - **Administrador**
- Implementar estandarización de la respuesta HTTP del **endpoint**

RETO 11

Web API - Ficha del Comerciante

Crear un **endpoint** que permita generar un **archivo PDF** con la información detallada del comerciante de acuerdo al identificador suministrado, bajo las siguientes condiciones:

- La estructura del **archivo PDF** debe contener el Nombre o razón social, Departamento, Municipio, Teléfono, Correo Electrónico, Fecha de Registro, Estado, Cantidad de Establecimientos, Total Activos y Cantidad de Empleados
- El **endpoint** debe tener un **parámetro de entrada** donde se especifique el identificador del comerciante
- Se debe utilizar el mapeo de entidades estándar o un ORM (Hibernate o Spring Boot JPA)
- El diseño u organización de la información dentro del PDF queda a libre elección del desarrollador

Notas:

- El **endpoint** debe ser **privado**, es decir, debe solicitar un **token JWT** para su ejecución
- Solo debe ser accesible a usuarios con el rol - **Administrador**
- Implementar estandarización de la respuesta HTTP del **endpoint**