Para preparar la base de datos

Compilamos los procedimientos

01\_DROP\_TABLE

02\_CREATE\_TABLE

Y

03\_INSERT\_DATA

Una vez compilados estos 3, compilamos

04\_INITIALIZE

Por ultimo ejecutamos el “script\_InsertTodadTABLAS”

Afirmaciones provisionales, pueden cambiar o se tienen que especificar u explicar en el glosario.

**BASE DE DATOS**

La base de datos de este proyecto esta compuesta por un total de 8 tablas

Todas las tablas tienen un clave primaria compuesta por un numero

(Campos/columnas y filas es una palabra a definir en el glosario)

-**SHIP.** Esta tabla tiene los datos de las naves del juego, tiene un total de 10 campos o columnas:

-**ID\_Defense**, la clave primaria de la nave.

-**name**, nombre de la nave, formato VARCHAR, este es unico.

-**metal\_Cost**, coste de metal de construir la nave, en formato INTEGER

-**crystal\_Cost**, coste de cristal de construir la nave, en formato INTEGER, si bien los datos de cristal estan presentes, no se han implementado en el proyecto

-**deuterium\_Cost**, coste de deuterio de construir la nave, en formato INTEGER

-**initial\_Armor**, armadura inicial con la que cuenta la nave, en formato INTEGER , con este dato junto con el nivel tecnologico se calculara la armadura final

-**armor**, nivel de armadura final, en formato INTEGER (? la armadura no deberia ser necesario guardarla, pues con la tecnologia y la inicial se puede calcular la armadura final)

-**baseDamage**, el daño base que inflije la nave, en formato INTEGER , el daño final estar también influido por el nivel tenológico

-s**peed,** la velocidad, en formato INTEGER

-**generateWasting,** la probabilidad de generar residuos, en formato INTEGER con un valor de entre 0 y 100

-**DEFENSE,** esta tabla tiene los datos de las estructuras defensivas. Sus columnas de datos son exactamente las mismas que las de SHIP con las mismas propiedades

-**USER.** Esta tabla tiene el objetivo de guardar datos de los usuarios (explicar en glosario que usuario = jugador). Tiene un total de 4 campos:

-**ID\_User**. La id del usuario.

-**username**. El nombre del usuario, es un VARCHAR y tiene la propiedad de que es unico y obligatorio, por lo tanto no podran haber dos usuarios con el mismo nombre.

-**password**. La contraseña que habra registrada para el usuario, en formato VARCHAR y obligatoria.

-**birth\_date**. La fecha de nacimiento del usuario, en formato DATE, también es un campo obligatorio.

-**PLANET.** Datos de los planetas, estos pertenecen a los usuarios y almacenan recursos, naves y estructuras defensivas y tienen un nivel tecnológico, que influira en la fuerza de las naves y las estructuras. Tiene un total de 19 campos o columnas:

**-ID\_Panet**. La id del planeta.

**-User\_ID\_User**. La id del usuario al que pertenecera el planeta, es una clave foranea que liga la tabla PLANET con la tabla USER, es obligatoria y esta en formato INTEGER, como todas las IDs.

**-planet\_Name**. Nombre del planeta, en formato VARCHAR, obligatorio y también unico para que no se repitan nombres.

**-quantity\_Metal**. Cantidad de metal que hay en este planeta, formato INTEGER, si no se especifica un valor por defecto se pone a 0.

**-quantity\_Crystal**. Cantidad de cristal que hay en este planeta, formato INTEGER, si no se especifica un valor por defecto se pone a 0.

**-quantity\_Deuterium**. Cantidad de cristal que hay en este planeta, formato INTEGER, si no se especifica un valor por defecto se pone a 0.

**-num\_LighHunter**. El numero de naves de tipo LightHunter que hay en el planeta, en formato INTEGER, si no se le especifica valor por defecto se pone a 0.

**-num\_HeavyHunter**. El numero de naves de tipo HeavyHunter que hay en el planeta, en formato INTEGER, si no se le especifica valor por defecto se pone a 0.

**-num\_BattleShip**.El numero de naves de tipo BattleShip que hay en el planeta, en formato INTEGER, si no se le especifica valor por defecto se pone a 0.

**-num\_ArmoredShip**. El numero de naves de tipo ArmoredShip que hay en el planeta, en formato INTEGER, si no se le especifica valor por defecto se pone a 0.

**-num\_MissileLauncher**. El numero de esturcturas defensivas de tipo MissileLauncher que hay en el planeta, en formato INTEGER, si no se le especifica valor por defecto se pone a 0.

**-num\_IonCannon**. El numero de esturcturas defensivas de tipo IonCannon que hay en el planeta, en formato INTEGER, si no se le especifica valor por defecto se pone a 0.

**-num\_PlasmaCannon**. El numero de esturcturas defensivas de tipo PlasmaCannon que hay en el planeta, en formato INTEGER, si no se le especifica valor por defecto se pone a 0.

**-current\_LevelDefense**. El nivel de tecnologia defensiva actual que hay en el planeta, esta en formato INTEGER y por defecto, en caso de que no se especifique un valor, es 1, el primer nivel.

**-cost\_DefenseUp**. El coste de deuterio de subir un nivel a tecnologia de defensa del planeta, es una variable en formato INTEGER y obligatoria.

**-current\_LevelAttack**. El nivel de tecnologia ofensiva actual que hay en el planeta, esta en formato INTEGER y por defecto, en caso de que no se especifique un valor, es 1, el primer nivel.

**-cost\_AttackUp**. El coste de deuterio de subir un nivel a tecnologia de ataque del planeta, es una variable en formato INTEGER y obligatoria.

(Constantes en el glosario)

-**CONSTANTS.** Tabla para guardar los valores de las constantes que regularan varios aspectos del balance del juego. Esta tabla solo tiene 3 campos:

**-ID\_Constant**. La clave primaria de la constante.

**-name**. El nombre de la constante, esta en formato VARCHAR y es unico y obligatorio.

**-value**. El valor de la constante, en formato VARCHAR y también obligatorio.

-**UNITS**. Cada nave y cada estructura es unica y por lo tanto estara registrada en esta tabla, que guardara de que tipo es la unidad y a quien pertenece. Tiene 7 campos o columnas:

**-ID\_Unit**.

**-defense\_Level**.

**-attack\_Level**.

**-Defense\_ID\_Defense**.

**-Ship\_ID\_Ship**.

**-Enemy\_ID\_Enemy**.

**-Planet\_ID\_Planet**.

-**ENEMY.** para guardar datos de los enemigos, estos son como una mezcla simplificada del jugador y el planeta, tienen naves, niveles tecnologicos y recursos para hacer naves, por si deseamos implementar una mecanica completa de posesión de unidades y uso de recursos como la del jugador.

-**BATTLE**, para guardar las batallas, esta es la mas compleja, se ata con llaves foráneas a un usuario, un planeta (aunque con unirse al planeta bastaría) y un enemigo, guarda el reporte step by step en forma de conjunto de caracteres, y tiene dos booleanos para guardar quien gana, si el de usuario es true y el de enemy es false, gana el usuario, en caso contrario gana el enemigo. Tiene datos de residuos generados y registros del numero de unidades con el que empiezan y terminan tanto jugador como enemigo.

SHIP y DEFENSE entiendo que sus datos son constantes y solo se deben insertar una vez al iniciar la BBDD para posteriormente consultarlos, sin posibilidad de hacer cambios posteriores

Las tablas USER, CONSTANTS, PLANET, UNITS, ENEMY, BATTLE les he creado un conjunto de procedimientos con una estructura común para trabajar con los datos. No todos son necesarios en todas las tablas, pero para abrir distintas posibilidades los he creado y dar flexibilidad. Permiten sacar datos, crear nuevos registros, actualizarlos y eliminarlo.

Los 4 procedimientos comunes (TABLA = Nombre de la tabla sobre la que queramos trabajar) son

-**GET\_TABLA(id IN, columna OUT , columna OUT , columna OUT, …)**

Este es el procedimiento de ejemplo en la conexión, se le entra una ID en forma de numero y nos devuelve el resto de variables, que están como OUT en el procedimiento, por lo que deben ser recogidas en Java.

Para recoger valores de variables y operar con ellas en el código de Java.

**-SET\_TABLA(id IN, columna IN , columna IN , columna IN, …)**

Este procedimiento es para modificar un registro que ya existe, el primer dato es una id, que usara para buscar la fila y modificar insertando el resto de parámetros.

Si alguno de los parámetros es nulo, es decir tiene las comillas sin ningún espacio **(**SET\_TABLA(id, 400, ‘’, ‘pepe’, …)el segundo campo no lo actualizaría y lo dejaría como esta**)**, el procedimiento esta diseñado para dejar el antiguo valor y no modificar, por lo que se pueden hacer modificaciones parciales, salvo de la ID.

**-INSERT\_TABLA(id IN, columna IN , columna IN , columna IN, …)**

Procedimiento para crear un nuevo registro, incluso admitiendo especificar la ID (aunque no admitirá IDs repetidas) y especificando el resto de datos. Inserta una nueva fila. Importante diferenciar entre SET, para actualizar una fila ya existente y INSERT para crear una nueva fila.

Si no se quiere insertar un valor para todas las columnas, mirar en el PDF con el modelo relacional, los datos que tienen un asterisco rojo son datos obligatorios que no pueden estar nulos.

**-DELETE\_TABLA(id IN)**

Simple, solo nos pide una ID y a partir de esta elimina la fila.

Para la tabla constants he creado un procedimiento para obtener valores de las variables a partir del nombre, que coincidirá con el nombre de variable en Java, para hacer la búsqueda mas fácil sin necesitar de poner una Query de por medio que nos transforme el nombre en ID y de esta obtengamos el valor. Si hace falta creare mas procedimientos específicos.

Estructuras de los 4 tipos de PROCEDIMIENTO básicos (Estos son lo mínimo que tendrán, común con los otros procedimientos del mismo tipo):

-**GET\_TABLA(id IN, columna OUT , columna OUT , columna OUT, …)**

Se declaran en el procedimiento una variable por columna, todas del mismo tipo que la columna a la que correspondan.

En IS se declara una variable NUMBER(1) para comprobar si el ID existe y una excepción para disparar en caso de que el id n exista.

Hace el selecto COUNT(ID) que coincida con el ID de entrada, si no lo encuentra (id\_Encontrada =0) salta un error.

En caso contrario selecciona todas las columnas de la tabla EXCEPTO la de la ID y deposita sus valores (INTO) el resto de variables de la declaración del procedimiento, las que están en OUT.

Tiene dos excepciones, la de Id, que da el mensaje de que no se ha encontrado esta ID en la tabla y una WHEN OTHERS que incluye un mensaje especifico del error.

**-SET\_TABLA(id IN, columna IN , columna IN , columna IN, …)**

El procedimiento mas complejo. Empieza en la declaración del procedimiento con una variable por cada columna, todas IN.

La declaración de variables IS añade para cada una de las variables de la declaración del proceso una variable idéntica, que tendrá el mismo nombre + “\_Insertar”, esta será la variable que se insertara, a continuación veremos el porque de esta variable, pero como avance, serán las que se usaran para insertar aunque parezca una redundancia.

El IS o declaración también incluye un NUMBER(1) y una excepción para el caso en que no encuentre la ID en la tabla y un VARCHAR de considerable tamaño para guardar la query que se ejecutara para actualizar la base de datos.

El procedimiento empieza con la típica comprobación de que la ID existe y el condicional con la excepción.

A continuación, tiene un condicional por cada columna de la tabla salvo la de la ID. Este condicional se activa si la variable entrada en el procedimiento es nula, en este caso lo que hace es que la variable insert adquiera el valor que tiene en la tabla esa columna en la ID que hemos entrado.

Esto hace que insertemos el valor que había previamente y por lo tanto no cambie ese dato, de esta forma si uno de los datos entrados en el procedimiento es nulo (‘’) no cambia. En caso contrario la variable a insertar será la que hemos entrado en el procedimiento.

He tenido que hacer eso y complicar el codigo porque con variables que se entran al procedimiento en IN no se les pueden asignar valores desde una query con INTO (con las variables en OUT, de salida, si que se puede hacer), pero he considerado que es útil para permitir modificar solo algunos campos de la fila y dejar otros intactos.

Terminados los condicionales, se prepara el script para actualizar, que es un UPDATE BATTLE VALUE() con valores para actualizar una fila especificando un WHERE que será donde la ID que hayamos entrado.

El script se ejecuta con execute immediate, mas seguro que escribirlo directamente. Tras todo esto tiene un mensaje de confirmación con las nuevas variables y un COMMIT.

De excepciones tiene la de ID y la WHEN OTHERS, destacar que las excepciones por precaución tienen un ROLLBACK para deshacer posibles cambios a medias como buena practica.

**-INSERT\_TABLA(id IN, columna IN , columna IN , columna IN, …)**

Para crear un nuevo registro, también nos pide que le entremos, IN, una variable por cada columna, incluida una ID, que será la ID de ese nuevo registro (Tiene codigo para comprobar que no se repita).

El IS tiene un NUMBER(1) y una excepción para comprobar que la ID no este ya insertada en la tabla y un VARCHAR del script con los valores que se insertaran.

Al empezar el procedimiento, se comprueba primero que no exista la ID, si existe id\_Encontrada =1 salta una excepción. Posteriormente prepara el script, INSERTO INTO TABLA VALUES() una fila. Executa el script con execute immediate y suelta un mensaje de confirmación que el insert esta correcto con los valores. Tiene un COMMIT y dos excepciones con ROLLBACK.

**-DELETE\_TABLA(id IN)**

El procedimiento mas sencillo de todos, solo le entramos una ID. Nos controla con la variable NUMBER(1) y la excepción que existe dicha ID y en caso de existir prepara un Script en forma de VARCHAR tipo DELETE FROM TABLA WHERE id = id\_Entrada y executa el script. Acaba en COMMIT o ROLLBACK en las dos excepciones típicas.