# DISEÑO DE LA BASE DE DATOS DEL PARQUE-ACUARIO $H_2$OLANDA


**Asignatura:**
Bases de datos

**Participantes:**
Carlos Gallego Andreu
Daniel Oliver Belando
Daniel Romero Alvarado

**Curso:**
2CD1

**ÍNDICE**

# 1. ANÁLISIS

Nuestra base de datos está orientada al parque-acuario $H_2$OLANDA (es un nombre inventado por nosotros). En este parque existen diversos espacios, que tienen un código que los identifican, un nombre único, una situación en el parque [abierto, cerrado, en mantenimiento, en construcción] y una superficie. De los espacios, unos son peceras, con su respectiva información: número de animales que contienen y capacidad (en litros), y otros tienen diferentes funciones ['Restaurante','Punto de información','Expositor','Seguridad','Atención al cliente','Almacén','Centro médico'].

La información sobre los animales que ocupan el acuario se hace por especies. De cada una se conoce necesariamente, además del código que la identifica, el nombre científico (no se repite), el lugar de origen, la alimentación que deben seguir, la clase animal a la que pertenece (entendemos por clase la categoría de la taxonomía situada entre el filo y el orden. Ejemplo: –Mamífero–), el número de ejemplares que hay, y en qué pecera del parque está alojada cada especie (todos los ejemplares de una especie están en la misma).

De las especies consideradas 'grandes', hay una cantidad reducida de ejemplares, y se mantiene información individualizada de cada uno. Se identifican dentro de su especie por un número, tienen un nombre (no repetido), se señala si es importado o ha nacido en la pecera además de su fecha de incorporación y el peso.

De las personas que trabajan en el parque se guarda el código del empleado, que será el identificador y, necesariamente, el nombre, NIF (que es único) y sueldo. El personal está organizado según una estructura jerárquica, la base de datos se ha creado de forma que la estructura jerárquica tiene forma de árbol, es decir no hay empleados que sean jefes de alguien superior a ellos (no hay ciclos jerárquicos)de manera que cada uno de los empleados es subordinado de otro empleado (su jefe) y sólo de uno. Se debe guardar la información sobre esta jerarquía, junto con la fecha en la que se asignó el jefe a cada uno. Lógicamente, una persona no puede ser subordinada y jefe de la misma persona. Todos los empleados tienen jefe (excepto el jefe superior que no tiene jefe). Entre el personal, de los que tienen como cargo 'cuidador' se debe guardar la información sobre qué especie o especies cuida cada uno. También se ha guardado información sobre los empleados que tienen cargo 'encargado', concretamente qué espacios funcionales tienen asignados.

Otra información que se debe almacenar es la tarifa de precios de visita al parque. Esta tarifa incluirá el precio según el tipo [jubilado, adulto o niño] y el día [festivo o laborable].

**REQUISITOS**

Los que nos hemos establecido para el proyecto son, en resumidas cuentas:

- Poder dar de alta/baja a nuevos trabajadores, en el caso de dar de baja tendría que cumplir unas pautas como que este trabajador no sea jefe de nadie.

- Ser capaces de añadir las nuevas adquisiciones del parque-acuario a la base de datos, ya sea un ejemplar cuya especie es nueva en esta DB[1] o únicamente añadir un ejemplar (porque ya hay ejemplares de dicha especie) a su correspondiente espacio (que será una pecera).

- En cuanto a las tarifas, tener disponibles diversas en función de los parámetros dados en la base de datos.

- Conocer las diferentes funciones que pueden desempeñar los trabajadores del parque-acuático dependiendo del espacio en el que trabaje. Estas funciones pueden estar orientadas al cuidado de las diferentes especies o, por el contrario, son funciones relacionadas con el espacio en el que se encuentren (como pueden ser 'Atención al cliente', 'Almacén', 'Centro Médico'...)

**Restricciones más importantes:**

- El número total de especies en una pecera es un dato derivado que se mantiene automáticamente por el sistema

- El número de ejemplares de una especie grande debe coincidir con el número de ejemplares que hay de esa especie en el parque, la DB[1] lo mantiene.

- Los empleados no pueden ser jefes de alguien superior a ellos. Los empleados tampoco pueden ser borrados si tienen algún subordinado, es decir, si son jefes de alguien (borrado restrictivo).

- Los cuidadores que cuidan especies y los empleados que se encargan de algún espacio funcional no pueden ser borrados sin antes dejar de hacer ese trabajo. (Sólo se pueden borrar empleados que no tengan un trabajo asociado). La DB[1] lo mantiene.

- Todas las especies deben ser cuidadas por algún cuidador. La DB[1] lo mantiene.

[1] DB = DataBase

## 2. DISEÑO CONCEPTUAL

El diagrama de clases es un elemento muy importante para el diseño de base de datos relacional, y su realización determina el esquema lógico y la implementación final en el software de gestión de bases de datos.

Dada la descripción de la base de datos en el apartado anterior, el esquema conceptual resultante es el indicado en la Figura 1.

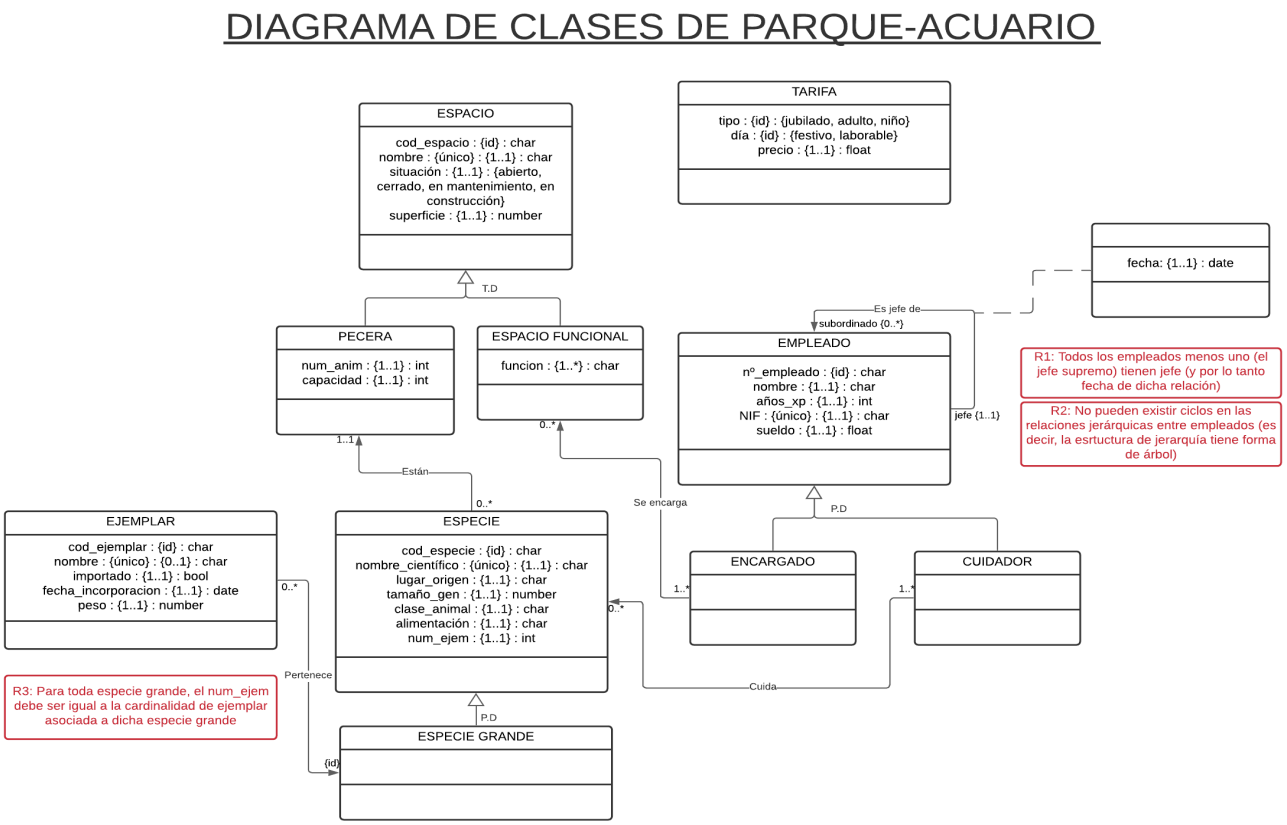


Figura 1: Diagrama de clases de Parque-Acuario. En rojo se indican las restricciones externas de la base de datos
Fuente: *Elaboración propia a partir de Lucidchart*

Las restricciones externas y necesarias para garantizar la funcionalidad de la base de datos son la siguientes:

- RI1: Todos los empleados menos uno (el jefe ejecutivo, el jefe superior de la empresa) tiene asignado un superior o jefe directo.
- RI2: La jerarquía de empleados debe seguir forma de árbol (sin ciclos)
- RI3: El nº de ejemplares de una especie grande debe coincidir con el número de filas en la clase ejemplar asociadas a dicha especie.

## 3. DISEÑO LÓGICO

Una vez expuesto el diseño conceptual, el diagrama lógico es el siguiente:

| | |
|---|---|
| **TARIFA** (tipo:*char*, día: *char*, precio: *number*)<br><br>CP : {tipo, día}<br>VNN : {precio} | **ESPACIO** (cod_espacio: *char*, nombre: *char*, superficie: *number,* situación: *char*)<br><br>CP : {cod_espacio}<br>VNN : {nombre, situación, superficie}<br>Único : {nombre}<br><br>*RI1: Todo valor del atributo cod_espacio de **Espacio**, o aparece como valor del atributo cod_pec en **Pecera**, o como valor del atributo cod_esp de **Esp_Funcional**, pero no en los dos a la vez (Especialización total disjunta)* |
| **PECERA** (cod_pec: *char*, num_anim: *number*, capacidad: *number*)<br><br>CP : {cod_pec}<br>VNN : {num_anim, capacidad}<br>CAj : {cod_pec} → **Espacio**(cod_espacio)<br><br>*RI2: El valor del atributo num_anim para cada tupla de la tabla pecera debe coincidir con la suma de los valores del atributo nº_ejemplares de las especies asociadas a dicha pecera.* | **ESP_FUNCIONAL** (cod_esp: *char*)<br><br>CP : {cod_esp}<br>CAj : {cod_esp} → **Espacio**(cod_espacio) |

**ESPACIO FUNCIONAL - FUNCIÓN** (cod_espacio: *char*, función : *char*)

CP : {cod_espacio, función}
CAj : {cod_espacio} → **Esp_Funcional**(cod_esp)

*RI3: Restricción de cardinalidad {1..*} de **Espacio Funcional-Función** Todo valor que aparezca en el atributo cod_esp de **Esp_Funcional** debe aparecer al menos una vez como valor en el atributo cod_espacio de **Espacio Funcional - Función.***

**ESPECIE** (cod_especie: *char*, nombre_científico: *char*, lugar_origen: *char*, tamaño_gen: *number*, clase: *char*, nº_ejemplares: *number,* alimentación: *char*, cod_pec: *char*)

CP : {cod_especie}
VNN : {nombre_científico, lugar_origen, tamaño_gen, clase, alimentación, nº_ejemplares, cod_pec)
CAj : {cod_pec} → **Pecera**(cod_pec) Están

*RI4: Para aquellas especies cuyo valor del atributo cod_especie aparezca en **Especie_Grande**, el valor del atributo nº_ejemplares debe coincidir con la cantidad de tuplas en **Ejemplar** cuyo valor en el atributo cod_especie sea el mismo que en **Especie** (el nº de ejemplares asociados a una especie grande debe ser igual al valor del atributo nº_ejemplares en dicha especie grande)*

**ESPECIE_GRANDE** (cod_especie: *char*)

CP : {cod_especie}
CAj : {cod_especie} → **Especie**(cod_especie)

**EJEMPLAR** (cod_especie: *char*, cod_ejemplar: *char*, nombre: *char*, importado: *number*, fecha_inc: *date*, peso: *number*)

CP : {cod_especie, cod_ejemplar}
VNN : {nombre, importado, fecha_inc, peso}
Único : {nombre}
CAj : {cod_especie} → **Especie**(cod_especie) Pertenece

**EMPLEADO** (nº_empleado: *char*. nombre: *char*, anyo_exp: *number*, NIF: *char*, sueldo: *number,* nº_emp_jefe: *char*, fecha: *date*)

CP : {nº_empleado}
VNN : {nombre, anyo_exp, NIF, sueldo}
Único : {NIF}
CAj : {nº_emp_jefe} →
**Empleado**(nº_empleado)
Es jefe de

*RI5: Solo hay una tupla en la tabla* ***Empleado*** *que tiene como valor nulo los atributos nº_emp_jefe y fecha, que se corresponde con el jefe ejecutivo*

*RI6: El mismo valor del atributo nº_empleado no puede aparecer a la vez como valor del atributo nº_empleado en* ***Encargado*** *y del atributo nº_empleado en* ***Cuidador*** *(especialización disjunta)*

**ENCARGADO** (nº_empleado: *char*)

CP : {nº_empleado}
CAj : {nº_empleado} →
**Empleado**(nº_empleado)

**ENCARGO** (nº_empleado: *char*, cod_espacio: *char*)

CP : {nº_empleado, cod_espacio}
CAj : {nº_empleado} →
**Encargado**(nº_empleado)
CAj : {cod_espacio} →
**Esp_Funcional**(cod_espacio)
Encargarse

*RI7: Restricción de cardinalidad {1..*} de* ***Encargo****: Todo valor del atributo cod_espacio de la tabla* ***Espacio Funcional*** *debe aparecer al menos una vez como valor del atributo cod_espacio de la tabla* **Encargo**

**CUIDADOR** (nº_empleado: *char*)

CP : {nº_empleado}
CAj : {nº_empleado} →
**Empleado**(nº_empleado)

| | |
|---|---|
| **CUIDADO** (nº_cuid: *char*, cod_especie: *char*)<br><br>CP: {nº_cuid, cod_especie}<br>CAj : {nº_cuid} → **Cuidador**(nº_cuid)<br>CAj : {cod_especie} → **Especie**(cod_especie) Cuida<br><br>*RI8: Restricción de cardinalidad {1..*} de **Cuidado**: Todo valor del atributo cod_especie de la tabla **Especie** debe aparecer al menos una vez como valor del atributo cod_especie de la tabla* **Cuidado** | |

Figura 2. Esquema lógico de Parque-Acuario. En rojo se han indicado las restricciones que deben implementarse en el software de gestión de base de datos, y en azul se han indicado las relaciones entre clases expuestas en el diagrama conceptual
Fuente: *Elaboración propia*

En cuanto a la normalización, como ya hemos señalado las restricciones anteriormente, únicamente faltaría mostrar los diferentes diagramas que muestran que todas las tablas están en 3ª FN:

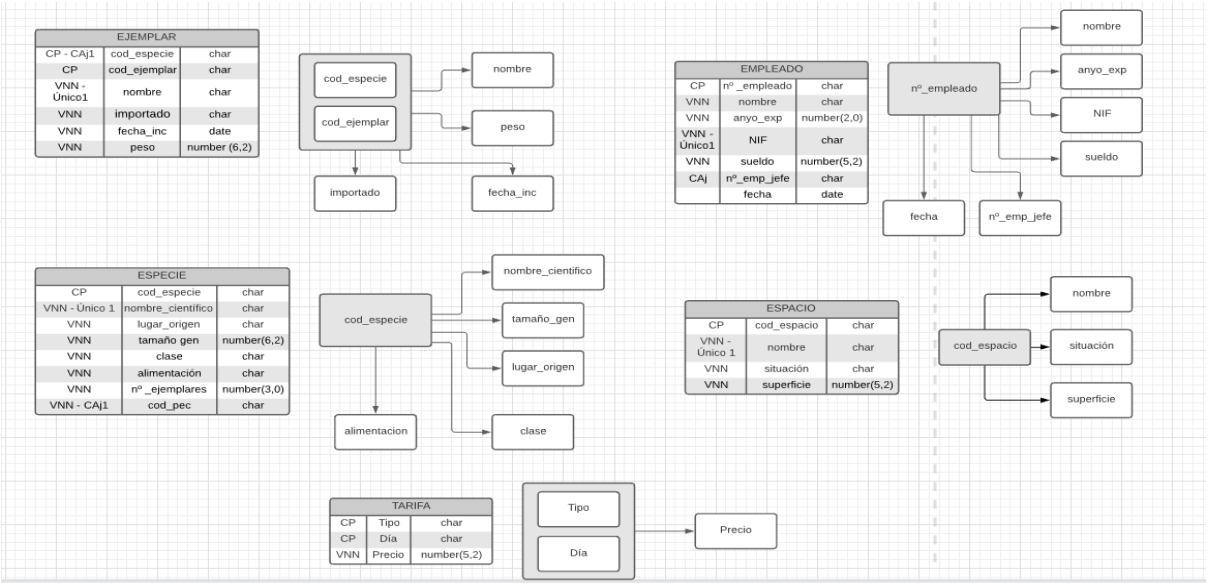


Figura 3. Tablas de la base de datos en Tercera Forma Normal
Fuente: *Elaboración propia a partir de Lucidchart*

Las tablas que faltan en la imagen no las hemos realizado porque están compuesta de uno o dos atributos que conforman la clave primaria.

# 4. IMPLEMENTACIÓN

La base de datos está a nombre de Carlos Gallego — *CGALAND*. Para que todos pudiéramos trabajar en ella en un inicio Carlos dió permisos tanto a Daniel Oliver como a Daniel Romero, pero al tener tantas tablas finalmente decidimos compartir la contraseña de ORACLE de Carlos y trabajar todos desde ahí.

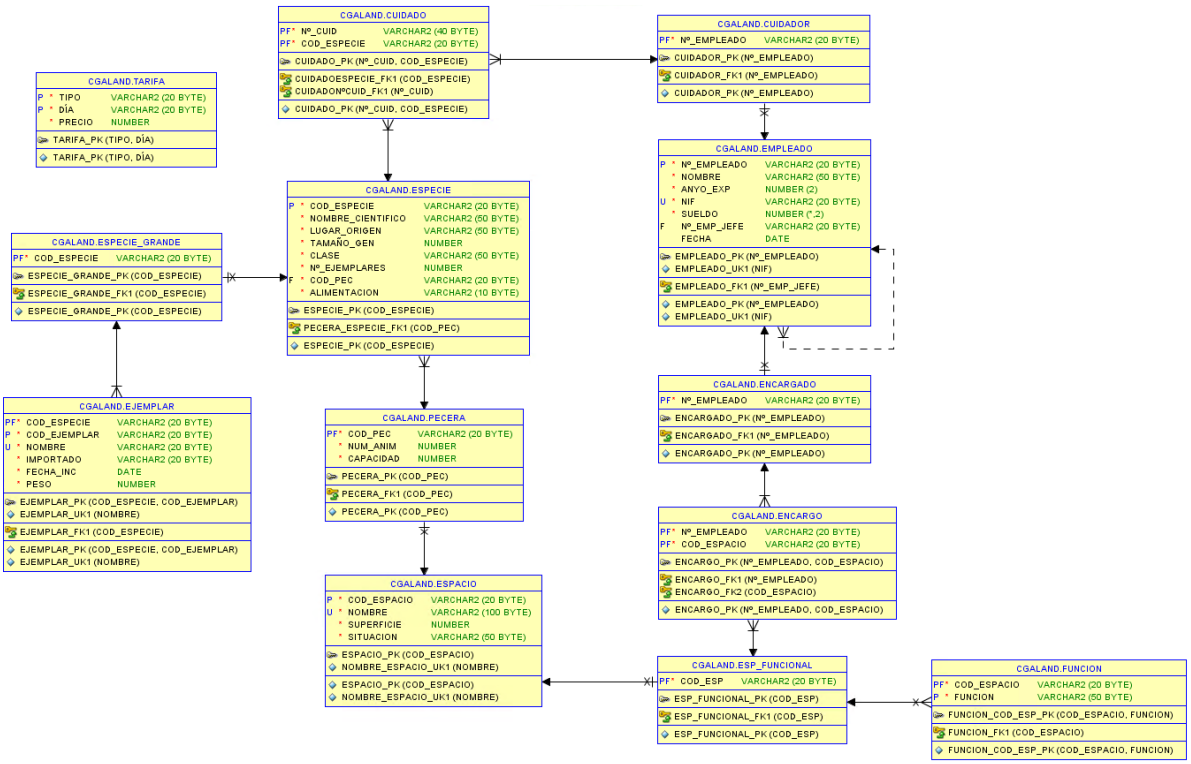


Figura 4. Esquema gráfico relacional en Oracle. Obtenido con 'data modeler' de SQL.
Fuente: *Elaboración propia a partir de Oracle*

Sobre la carga de datos

Inicialmente se pensó en realizar un pequeño programa que extrayera datos de páginas web mediante técnicas de Web Scraping, pero algunos datos (como el NIF de los trabajadores y potencialmente su sueldos) no serían fáciles de encontrar, con lo que finalmente decidimos generar nosotros los datos de forma relativamente aleatoria[2]. A continuación resumimos el proceso, realizado en Python:

[2] Con "relativamente aleatoria" nos referimos a que los valores de muchos atributos se han generado mediante números aleatorios, pero algunos atributos específicos o ciertas clase como Empleado han requerido un tratamiento especial para asegurarse de cumplir las restricciones de la base de datos

- Datos aleatorios de bibliotecas externas: muchos atributos de las diferentes clases se han podido obtener simplemente usando bibliotecas de Python como random (generación de números aleatorios, entre otras cosas) y names (generación de nombres aleatorios). Esto, aunque no viola ninguna integridad de la base de datos, puede llevar a resultados "de poco sentido común" (que un empleado con un año de experiencia cobre 3000 € mientras un empleado con diez años de experiencia cobra 1500 €)

- Datos aleatorios creados por nosotros mismos: ciertos atributos (la mayoría de los identificadores) han sido generados secuencialmente y de forma específica (por ejemplo, las especies tienen códigos como "esp001", "esp002") para poder identificar los datos de manera visual más fácilmente en la base de datos.

- Datos determinados manualmente: ciertos atributos como el nombre científico de las especies han tenido que ser buscados e introducidos manualmente.

- Asegurándonos de cumplir las restricciones: en todo momento nos hemos asegurado de cumplir las restricciones de los atributos (NIFs únicos, por ejemplo), como las restricciones externas (los empleados formando una estructura de árbol por ejemplo).

Por último, se han creado ficheros .csv para cada tabla y se han introducido todos los datos de golpe en la base de datos en el orden adecuado para evitar problemas de restricción de clave ajena.

# 5. PROGRAMACIÓN

A continuación, vamos a explicar en dos apartados distintos qué teníamos en mente a la hora de hacer los distintos informes y, por otro lado, los distintos disparadores, procedimientos y funciones que hemos realizado para la base de datos.

## 5.1 Informes

Primero de todo, hemos realizado hasta un total de 5 informes de distintos formatos y con objetivos muy dispares entre ellos:

Jerarquía directa de un empleado

El siguiente informe es uno parametrizado: recibe un código de empleado de una lista desplegable de empleados disponibles y devuelve cada subordinado directo y, para cada subordinado directo, sus subordinados directos. Es decir, recibe un empleado seleccionado por el usuario y devuelve sus subordinados hasta un nivel 2 de jerarquía.

[Empleado_Subordinados.pdf]

Especies grandes y sus respectivas peceras.

Este informe tiene como objetivo, primero de todo, mostrar la información de las peceras que contienen especies grandes sin importar la cantidad de ejemplares y, una vez se reconocen estas, señalar la información correspondiente a las especies que hay dentro de cada pecera.

Finalmente, se muestra un gráfico que resalta la relación que hay entre los ejemplares de especies grandes y las demás que no cumplen esta característica, en las distintas peceras.

[Pecera_EspeciesGrandes.pdf]

Peceras con más de una especie.

Con este informe queremos mostrar tanto información sobre las peceras con más de 1 especie, como de esas especies. Los datos sobre la pecera están en forma de texto, y lo acompaña una gráfica de barras apiladas con el que podemos ver sus especies y las clases de estos.

Es útil para ver cómo de diversas son las peceras y en cuáles podemos encontrar mayor variedad. No era interesante incluir peceras con solo una especie.

[Peceras_más_1_especie.pdf]

Top cuidadores

En este informe se ha querido plasmar el ranking de los 5 mejores cuidadores dependiendo del sueldo que cobran. Estos empleados aparecen por orden de mayor salario a menor.

A continuación de los datos de cada cuidador, como son su nº de empleado, nombre, sueldo y años de experiencia, aparece la información de las diferentes especies que cuidan y la cantidad de ejemplares de cada uno de ellos.

[Top_Cuidadores.pdf]

Encargados por función

El siguiente informe es uno parametrizado: recibe un código de empleado de una lista desplegable de empleados disponibles y devuelve la información general de todos los espacios de los que es encargado el empleado (indistintamente de su función).

Además, muestra quién es su superior, señalando el nivel de jerarquía en el que se encuentra, y la 'ruta' que hay desde el jefe supremo o ejecutivo hasta el empleado que nosotros hemos señalado. Con 'ruta' nos referimos a los empleados, que en relación a la jerarquía de la empresa, se encuentran entre el jefe superior y nuestro empleado.
[Encargado por función.pdf]

## 5.2 Procedimientos, Funciones y Disparadores

Hemos realizado un total de 2 procedimientos, 3 funciones y 10 disparadores

**PROCEDIMIENTOS**

**NUEVA_PECERA**

Descripción del procedimiento: El siguiente procedimiento recibe un código de pecera, un nombre, una superficie (en $m^2$), la situación de la pecera, el nº de animales en la pecera y la capacidad de la pecera (en L) e inserta una nuevo espacio en la tabla ESPACIO y una nueva pecera en la tabla PECERA.

Parámetros de entrada:

- i_cod_pec → Código de identificación de la pecera. Clave primaria en ESPACIO y PECERA, y clave ajena en PECERA.
- i_nombre → Nombre de la pecera. Ha de ser único.
- i_superficie → Superficie en $m^2$ de la pecera.
- i_situacion → Estado actual de la pecera. Debe ser uno de estos valores: 'Abierto', 'Cerrado', 'En mantenimiento', 'En construcción'.
- i_n_animales → Nº de animales alojados en la pecera. Se usarán disparadores para comprobar que el valor coincide con la suma de los valores del atributo nº_ejemplares de las especies asociadas a dicha pecera.
- i_capacidad → Capacidad en L de la pecera.

**NUEVO_EF**

Descripción del procedimiento: El siguiente procedimiento recibe un código de espacio funcional, un nombre, una superficie (en $m^2$), la situación del espacio funcional y una de las funciones que tenga dicho espacio (ha de tener como mínimo una función) e inserta un nuevo espacio en la tabla ESPACIO, un nuevo espacio funcional en la tabla ESP_FUNCIONAL y una nueva función asociada a dicho espacio funcional en la tabla FUNCION.

Parámetros de entrada**:**

- i_cod_ef → Código de identificación del espacio funcional. Clave primaria en ESPACIO y ESP_FUNCIONAL, y clave ajena en ESP_FUNCIONAL.
- i_nombre → Nombre del espacio funcional. Ha de ser único.
- i_superficie → Superficie en $m^2$ del espacio funcional.
- i_situacion → Estado actual del espacio funcional. Debe ser uno de estos valores: 'Abierto', 'Cerrado', 'En mantenimiento', 'En construcción'.
- i_funcion_1 --> Función que cumpla el espacio funcional. Si un espacio funcional tiene varias funciones, se crea el espacio funcional con una función y después se deberán añadir mediante inserciones en la tabla FUNCION.

## FUNCIONES

**EXISTE_EMPLEADO**

Descripción de la función: Esta función sirve para comprobar si un empleado existe en la base de datos, recibe un id de empleado y devuelve 1 (TRUE) si existe o 0 (FALSE) si no existe. La función puede ser auxiliar a otras para simplificar el código.

**TIENE_FAENA**

Descripción de la función: Esta función sirve para comprobar si un empleado (cuidador o encargado) tiene asignado un trabajo, devuelve 1 (TRUE) si el empleado tiene faena (aparece su id en la tabla cuidado o encargo) y 0 (FALSE) si no tiene faena. Si no existe lanza un error. Esta función aprovecha la función EXISTE_EMPLEADO para comprobar si el empleado que recibe existe o no.

**SITUACIÓN_ESPECIE**

Descripción de la función: Esta función averigua la situación en la que se encuentra la pecera de una especie dada, es decir, recibe una especie y devuelve la situación (abierto, cerrado, en mantenimiento, en construcción) de su pecera. Puede ser útil para saber si esa especie puede ser visitada en algún momento dado o no.

## DISPARADORES

**CHECK_EMP_JEFEMAX**

Descripción del disparador: el disparador comprueba que, tras haberse insertado, actualizado o borrado una tupla en la tabla EMPLEADO, únicamente existe una tupla (la del jefe supremo) con el valor n_emp_jefe y la fecha a nulo.

Restricción: el disparador comprueba la restricción externa de que solo existe un jefe ejecutivo (jefe máximo). El disparador devuelve diferentes mensajes de error en función de si no hay empleado máximo o de si hay más de uno.

**CHECK_EMPLEADO_CYCLES**

Descripción del disparador: el disparador comprueba que, tras haberse actualizado una tupla en la tabla EMPLEADO, no existe ningún empleado cuyo jefe tenga como empleado jefe al empleado original. Un ejemplo:

| Nº EMPLEADO | Nº EMPLEADO JEFE |
|---|---|
| emp001 | emp003 |
| emp003 | emp001 |

Figura 5. Tabla de ejemplo de una relación jerárquica con ciclos
Fuente: *Elaboración propia*

El disparador también comprueba que un empleado no es defe de sí mismo.

Restricción: el disparador comprueba que no existan ciclos de nivel 0 o 1 en la relación jerárquica de los empleados, ya que tiene una estructura de árbol. El disparador no comprueba ciclos de nivel superior a 1 (por ejemplo, que el el abuelo de un empleado tenga como jefe al propio empleado).

**AÑADIR_EJEMPLAR**

Descripción del disparador: el disparador se asegura que, al añadir o eliminar un ejemplar de una especie grande, se actualiza (sumando o restando 1) al valor nº_ejemplares de la tupla correspondiente en la tabla ESPECIE.

Restricción: el disparador se encarga de la restricción externa de mantener un dato derivado: en este caso comprobar que el valor del atributo nº de ejemplares de una especie grande coincide con las tuplas de ejemplares asociadas a dicha especie.

**PECERA_NUM_ANIM**

Descripción del disparador: El disparador se activa tras haberse insertado, actualizado o borrado una tupla en la tabla ESPECIE, lo que hace es actualizar el número de animales de la pecera de la especie que ha sido alterada o añadida. Esto lo hace mirando el nuevo y el antiguo valor del atributo nº_ejemplares de la especie.
Restricción: El disparador nos ayuda a mantener un dato derivado, el número de animales de la tabla PECERA, es el atributo num_anim.

**CHECK_CUIDADO**

Descripción del disparador: el disparador comprueba que, tras haberse actualizado o borrado un tupla de la tabla CUIDADO, todos los valores del atributo cod_especie de la tabla ESPECIE aparecen como valores en el atributo cod_especie de la tabla CUIDADO. Para ello vamos a comprobar que el nº de especies distintas es el mismo en ambas tablas.
Restricción: El disparador se encarga de que se cumpla la restricción de cardinalidad 1…* de la tabla CUIDADO.

**CHECK_ENCARGO**

Descripción del disparador: El disparador comprueba que, tras haberse actualizado o borrado una tupla de la tabla ENCARGO, todos los valores del atributo cod_esp de la tabla ESP_FUNCIONAL aparecen como valores del atributo cod_espacio de la tabla ENCARGO. Para ello vamos a comprobar que el nº de espacios funcionales distintos es el mismo en ambas tablas.

Restricción: Este disparador tiene como objetivo cumplir la restricción de cardinalidad 1…* de la tabla ENCARGO.

**DISJOINT_CUIDADOR**

Descripción del disparador: Este disparador comprueba si el empleado que se intenta añadir se encuentra en la tabla ENCARGADO. En dicho caso, salta un error con que existe un encargado con ese nº de empleado, sino no habría problema en añadirlo.

Restricción: El disparador se encarga de que la restricción de especialización disjunta de la clase Empleado.

**DISJOINT_ENCARGADO**

Descripción del disparador: Este disparador comprueba si el empleado que se intenta añadir se encuentra en la tabla CUIDADOR. En dicho caso, salta un error con que existe un cuidador con ese nº de empleado, sino no habría problema en añadirlo.

Restricción: el disparador se encarga de la restricción de especialización disjunta de la clase Empleado.

**CHECK_FUNCION**

Descripción del disparador: el disparador comprueba que todo código de espacio funcional que se encuentre en la tabla ESP_FUNCIONAL aparece al menos una vez como valor del atributo cod_espacio de la tabla FUNCION. Para ellos vamos a comprobar que el nº de espacios funcionales distintos es el mismo en ambas tablas.

Restricción: el disparador se encarga de que se cumpla la restricción de cardinalidad 1..* del atributo función de la clase Espacio Funcional.

**CHECK_NUM_ESPECIES_GRANDES**

Descripción del disparador: Este disparador es un poco complejo, se activa cuando se inserta o se actualiza una tupla de la tabla ESPECIE, lo que hace es insertar en la tabla ESPECIES_GRANDES la especie si se trata de una especie grande (tamaño_gen > 2.5),al actualizar, comprueba los valores old y new de tamaño_gen, si se pasa de una especie grande a una pequeña la elimina de la tabla ESPECIES_GRANDES, si se pasa de una especie pequeña a una grande la añade.

Además este disparador comprueba si hay tantos ejemplares (tabla EJEMPLARES) como lo indica en el atributo nº_ejemplares de la especie insertada o actualiza (si finalmente se trata de una especie grande).
Lo ideal al insertar una especie (grande) es indicar que no hay ningún ejemplar, hemos tenido problemas ya que nos daba error de tabla mutante, por lo que lo mejor sería que quien llevara

la base de datos inserte primero las especies con el atributo nº_ejemplar a 0 y luego se vayan añadiendo uno a uno, y se actualizará este atributo directamente gracias al disparador AÑADIR_EJEMPLAR.

Restricción: El disparador ayuda a solucionar el problema de que haya tantos ejemplares como el atributo nº_ejemplares de las especies grandes. También introduce o borra la fila correspondiente de la tabla ESPECIE_GRANDE cuando se trata de especies grandes.

# 6. BIBLIOGRAFÍA

Casamayor, Juan Carlos & Mota, Laura, *UD4: Diseño de base de datos relacionales,* (3/12/2019)

Casamayor, Juan Carlos & Mota, Laura, *UD5: Acceso y programación al servidor,* (16/12/2020)

Oracle Help Center, *Hierarchical Queries,* (no especifica fecha), https://docs.oracle.com/cd/B19306_01/server.102/b14200/queries003.htm#