ARTEFACTO 9 MODELADO DE DATOS

9. Modelo de Datos.

Este artefacto depende directamente del artefacto 4 de la iteración anterior en el que definíamos el dominio del problema. Partiendo lo realizado en él crearemos los nuevos diagramas con los que modelaremos los datos a manejar y finalmente los elementos mediante los que manejaremos dichos datos.

9.1 Paso a diagrama de tablas.

Con el diagrama de tablas pretendemos modelar la estructura física de los datos con los que vamos a trabajar en nuestra base de datos. Dicho diagrama se crea de forma directa en Toad Data Modeler Convirtiendo el modelo de tipo Entidad Relación realizado en la iteración anterior a uno Físico de Tablas.

9.1.1 Decisiones tomadas.

En modelo Entidad Relación hemos tenido que tomar algunas decisiones de diseño del diagrama de Tablas antes de convertirlo.

9.1.1.1 Respecto de las herencias.

Explicamos aquí como hemos diseñado las herencias pensando tanto en optimizar las futuras consultas sobre la Base de Datos como en reducir el tamaño de esta en disco.

*9.1.1.1.1 Herencia de **Trabajador**.*

Hemos decidido convertir la entidad Trabajador en cinco tablas físicas ya que prevemos que cuando se quieran listar los trabajadores se querrán mostrar solo los de un tipo concreto. Adicionalmente hacer esto tiene otra ventaja pues deja a los técnicos informáticos en una tabla propia de modo que será más fácil comprobar las restricciones de integridad cuando a un técnico informático le asignemos una petición de trabajo pues solo se deberá comprobar en esa tabla si el técnico existe o no en vez de en una tabla con todos los trabajadores mezclados diferenciados por el valor de un atributo.

Tener cada tipo de trabajador en una tabla distinta también compartimentaliza la información que en este caso es algo que nos interesa pues crea distintas vistas sobre los datos que facilitan la tarea de elegir qué usuarios verán qué información. Por ejemplo, el **responsable del almacén** podrá ver solo la tabla de los **ayudantes de almacén** y no la de los **ayudantes técnicos**.

Esta decisión de crear solo tablas hija y prescindir de la tabla padre la hemos podido tomar ya que todos los trabajadores serán siempre de un tipo y solo de un tipo, es decir, la herencia es completa y exclusiva.

*9.1.1.1.2 Herencia de **Pieza**.*

Para la tabla de piezas hemos optado por crear también dos tablas hija sin tener la tabla padre. Por las mismas razones que con la herencia anterior podemos hacer esto pues esta relación de herencia es completa y exclusiva de modo que todas las piezas serán de un tipo, básicas o especiales y solo de uno.

Nos interesa realizar la herencia de este modo pues cada tipo de pieza tendrá atributos propios distintos de los otros tipos, de modo que si todas las piezas estuvieran en una misma tabla esta estaría llena de nulos. Cada pieza tendría mínimo dos atributos a nulo y adicionalmente nos haría falta un atributo adicional que las separara. Almacenar toda esa información innecesaria podría tanto ralentizar las consultas como gastar excesivo espacio en disco.

Por lo general accederemos o a una tabla de piezas de un tipo o a las de otro según lo que queramos hacer. Se prevé un mayor acceso a la tabla con piezas especiales pues tendrán que ser buscadas con frecuencia por los **responsables de almacén** para realizar los pedidos especiales lo cual es otro argumento a favor de tener separas las **piezas** en dos tablas distintas pues las búsquedas serán más rápidas.

*9.1.1.1.3 Herencia de **Pedido**.*

Los pedidos al heredar quedarán en una única tabla diferenciados por un atributo que nos indicará si contienen piezas especiales o no. Esto lo hacemos a pesar de que la herencia de los pedidos sea completa y exclusiva pues nos aportará las siguientes ventajas.

Cuando creamos un pedido este no es enviado directamente al proveedor a no ser que dicha acción se fuerce por los usuarios, si no que es enviado por un daemon del sistema de forma periódica con la intención de que piezas pedidas a un mismo proveedor vayan lleguen en un mismo paquete con la intención de reducir los gastos de envío derivados. Al tener todos los pedidos en la misma tabla dicho daemon solo tendrá que realizar una búsqueda sobre una única tabla para encontrar los pedidos con una fecha superior a la última en la que realizó pedidos para encontrar los que sean nuevos y solicitarlos a sus proveedores correspondientes. Dicha acción podría incrementar su velocidad de forma sencilla simplemente manteniendo la tabla ordenada por su PK que ya es la fecha y agrupando las piezas por proveedor.

9.1.1.2 Respecto de las PK/PFK.

*9.1.1.2.1 PK de **Oficina**.*

Hemos decido que la PK de las **oficinas** sea PFK en las tablas con las que se relaciona. Es decir, **Responsable de almacén** tendrá la PK de una **oficina** y **Pedido** también la tendrá. Esto nos permite comprobar si un pedido está en una oficina de forma sencilla sin tener que acceder a la tabla **Oficina** o listar los pedidos que pertenezcan a nuestra oficina con una reunión entre solo dos tablas y no entre tres lo que reduce el coste de dichas consultas que son las que más veces se realizarán en la base de datos.

Puesto que las relaciones con **Oficina** son todas 1:N situar las PFK en las relaciones del lado de la N es lo habitual pues nos evita complejidad dentro de la tabla **Oficina**. Poner la PK de las otras tablas en oficina obligaría a esta tabla a tener atributos complejos de tamaño variable o tablas intermedias donde almacenarlas lo cual son soluciones a evitar.

*9.1.1.2.2 PK de **Proveedor**.*

La PK del **Proveedor** la situaremos tanto en la tabla **Pedido** como en las tablas **Pieza**. Esto nos permite acceder a los proveedores de una forma más sencilla pues prevemos que se querrá buscar más veces el proveedor concreto que nos proporciona la pieza que deseamos solicitar o el proveedor que nos realiza un pedido antes que listar las piezas o pedidos de un proveedor ya que por lo general será el sistema el que busque el proveedor al que los pedidos pertenecen para realizarlos y solo querremos recuperar un proveedor del que de este modo tendremos ya su PK. Con un índice sobre las PK de los proveedores podríamos agilizar este tipo de consultas.

Además esto nos permite realizar reuniones entre **Piezas** y **Pedidos** sobre este atributo en común para obtener datos estadísticos que podrían ser de utilidad como la cantidad de piezas que se suelen realizar por pedido para un proveedor.

Al igual que ocurría con las relaciones de **Oficina** las de **Proveedor** son 1:N por lo que situar la PFK en el lado de la N a parte de las ventajas mencionadas nos evita lidiar con soluciones complejas poco recomendables derivadas de querer situar la PK en el lado del 1.

9.1.1.2.3 PK en las relaciones N:M.

Para transmitir la PK en este tipo de relaciones utilizamos una tabla intermedia que almacene y enlace las PK de unas tablas con las de otras tal como es habitual. Cabe destacar una ventaja notable de esta solución y es que nos permite almacenar atributos de la relación como explicaremos en el apartado 9.1.2 de este documento.

9.1.1.2.4 PFK en Petición de Trabajo.

Hemos decidido que la tabla **Petición de Trabajo** contenga como PFK las PK de las tablas con las que se relaciona de forma N:1 para simplificar la búsqueda en la base de tatos. En estas relaciones será la tabla **Petición de Trabajo** la que está en el lado de la N y la que por tanto absorbe las PK, en este caso de las tablas **Técnico Informático** y **Cliente** (también **Oficina**, pero es ya se ha explicado).

Con esta configuración nos será fácil tanto buscar el **Cliente** y el **Técnico Informático** asignados a una petición como comprobar que la dirección del técnico encaja con la del parte.

Finalmente, las tablas con las que **Petición de Trabajo** se relaciona de forma 1:1 tendrán ellas la PK de la **Petición de Trabajo**. Esto lo realizamos pues la cardinalidad mínima es 0, de modo que comprobar si existe o no un **Parte de Trabajo**, **Factura** o **Presupuesto** consistirá en buscar y fallar sobre un índice en la PK. Haciendo esto también nos evitamos la existencia de nulos ya que si no creamos un **Presupuesto** por ejemplo este simplemente no existirá sin que haga falta almacenar un nulo en el **Parte de Trabajo**.

9.1.1.3 Respecto de los índices.

Hemos creado un índice hash adicional en la tabla **Pedidos** que va sobre fechas pues prevemos que el sistema va a buscar con bastante frecuencia los pedidos a partir de la última fecha en la que comunicó los pedidos pendientes a los proveedores.

Sobre la tabla **Peticiones de trabajo** hemos creado dos índices B, uno sobre estado y otro sobre prioridad ya que prevemos que con gran frecuencia querrán ordenarse sus tuplas por esos campos para asignar los trabajos a los **Técnicos Informáticos**.

9.1.2 Atributos adicionales a los del modelo Entidad Relación.

Ya que en el diagrama de Entidad Relación la herramienta Toad no nos lo permitía ahora en el diagrama de tablas incluiremos los atributos de las relaciones N:M tal y como habíamos indicado en el modelo de datos. Estos atributos indicarán la cantidad que se presupuestan, consumen o se han traído en un pedido. Almacenar dichos atributos es importante pues formarán el histórico del consumo de piezas que entre otras cosas nos permitirán realizar el control de los gastos y del stock.

9.1.3 Restricciones de integridad.

Por seguridad para la base de datos con la intención de evitar inconsistencias por defecto hemos establecido que las actualizaciones las realizaremos en cascada y restringiremos los borrados de datos. Nuestro planteamiento ha sido que creemos que por lo general no se realizarán demasiadas actualizaciones en los datos ya que aparentemente son datos estáticos y acumulativos por lo que si de verdad se quiere hacer una modificación de alguna PK se querrá que esta modificación se extienda por todas las tablas relacionadas de modo que los datos

almacenados en la base de datos sean verídicos y consistentes a pesar de el coste de modificación adicional.
Para el borrado de las tablas hemos creído que lo mejor sería restringirlo para no perder datos en las tablas, si se hizo una venta hace tiempo impedimos que se borre pues afecta al stock y provocaría inconsistencias.
Sobre la restricción en el borrado hacemos dos excepciones y es que permitimos borrar tanto clientes como trabajadores sin restricción. Dichas tablas estarán suficientemente protegidas mediante pedidos de acceso de modo que si se desean borrar datos de ellas es porque una razón de peso. No permitir estos borrados incurriría en problemas legales sobre la privacidad de datos y realizar borrados en cascada haría que perdiéramos información valiosa para el sistema como los **partes de trabajo** realizados o las **peticiones de trabajo** de las que podemos obtener información estadística.
Adicionalmente permitimos el borrado en cascada de los pedidos pues es relativamente usual que un pedido pueda extraviarse o que se cancele una vez realizado y si lo que se pidió finalmente no llega sería un error almacenar que si lo hizo.

9.2 Paso a script de la Base de Datos.

El script de la base de datos fue generado sin problemas de forma directa mediante Toad. Obtuvimos durante la generación warnings debido a que en algunas relaciones tenemos atributos que se llaman de igual modo. No obstante, decidimos obviarlos ya que por nuestra experiencia con bases de datos eso no ha sido hasta ahora un problema si no una ventaja ya que permite realizar natural joins de forma más sencilla entre tablas y también pues el autocompletado que tienen algunas bases de datos al escribir sql se restringe solo sobre los atributos cuando les hemos antepuesto el nombre de la tabla seguido de punto.

9.3 Elementos adicionales para almacenar los datos.

Hemos decidido que para los siguientes elementos: **Facturas**, **Informes de trabajo**, **Informes de beneficios** y **resúmenes de trabajo** crearemos archivos json para almacenar su información de forma duplicada. La finalidad de dichos archivos no será la de almacenar la información, para eso está ya la base de datos, si no de compartirla. Los archivos json, muy extendidos en la actualidad son fáciles de crear y parsear, además son soportados por muchas plataformas de forma nativa como es el caso de JavaScript. Puesto que planteamos orientar nuestro sistema a la red sería un error no utilizar este tipo de archivos para comunicar la información, pues nos permite enviarla de forma tipada y segura con mayor facilidad que la mayoría de los formatos. También nos planteamos utilizar archivos xml, pero descartamos esta opción ya que xml es un formato con el que es más difícil de trabajar, ocupa más para almacenar la misma información y no es portado por las plataformas web de forma nativa, aunque si por las plataformas móviles.
Incluimos los metadatos de la estructura de dichos archivos a modo de ejemplo de cómo se organizaría la información en ellos.

9.3.1 Factura.

```
"Factura" : {
        "@fecha" : "1940-10-09",
        "@oficina" : "dirección",
        "@numero" : "integer",
        "tiempo" : "integer",
        "precio" : "integer",
        "Trabajador" {
                "@dni" : "text",
                "@nombre" : "text",
        },
        "ciente" {
                "@dni" : "text",
                "@nombre" : "text",
        }
        "piezas" : [
                "pieza" {
                        "@identificador" : "integer",
                        "@proveedor" : "integer",
                        "precio" : "integer"
                }
        ]
}
```

9.3.2 Resumen de trabajo.

```
"Resumen de trabajo" : {
        "@fecha" : "1940-10-09",
        "@oficina" : "dirección",
        "Trabajos nuevos" : {
                "Totales" : "integer",
                "Media" : "integer"
        },
        "Trabajos completados" : {
                "Totales" : "integer",
                "Media" : "integer"
        },
        "Trabajos pendientes" : {
                "Totales" : "integer",
                "Acumulados" : "integer"
        },
        "Personal_extra" : "integer"
}
```

9.3.3 Informe de trabajo.

```
"Informe de trabajo" : {
        "@fecha" : "1940-10-09",
        "@oficina" : "dirección",
        "Trabajadores" : [
                "Trabajador" {
                        "@dni" : "text",
                        "@nombre" : "text",
                        "trabajos_realizados" : "integer",
                        "horas_trabajadas" : "integer",
                        "tiempo_medio" : "integer",
                        "dispersion_tiempo" : "integer",
                        "Trabajos" : [
                                "ciente" {
                                        "@dni" : "text",
                                        "tiempo" : "integer",
                                        "precio" : "integer",
                                        "piezas" : [
                                                "pieza" {
                                                        "@identificador" : "integer",
                                                        "@proveedor" : "integer",
                                                        "precio" : "integer"
                                                }
                                        ]
                                }
                        ]
                }
        ]
}
```

9.3.4 Informe de beneficios.

```
"Informe de beneficios" : {
        "@fecha" : "1940-10-09",
        "@oficina" : "dirección",
        "ingresos": "integer",
        "gastos" : {
                "compra" : "integer",
                "investigacion" : "integer",
                "extra" : "integer",
                "pagos" : {
                        "total" : "integer",
                        "media" : "integer"
                }
        },
        "beneficios" : {
                "periodo" : "integer",
                "media" : "integer"
        }
}
```

9.4 Otras consideraciones.

Quisimos desnormalizar alguna de las tablas de la base de datos, aunque fura a modo de ejemplo ilustrativo de algún caso en el que esto fuera una ventaja en nuestra base de datos. A pesar de no encontrar ningún caso claro en el que realizarlo explicamos a continuación los dos criterios principales que utilizamos para poder elegir cuando desnormalizar una tabla.

Si ciertos atributos de una tabla no deben ser accedidos por todos los usuarios por motivos de seguridad o privacidad puede ser una buena idea desnormalizar dicha tabla. Esto se haría para facilitar la gestión de permisos de modo que todos los usuarios podrían ver los atributos generales de la tabla y solo algunos con permisos suficientes los que estuvieran en la tabla más protegida.

Si una tabla es demasiado grande, cada tupla ocupas varios bloques de memoria y sobre ella detectamos atributos sobre los que no se accederá con frecuencia o que no son necesarios para la actividad base del sistema podría ser buena idea desnormalizar creando dos tablas. Una principal con los datos a los que se accede con frecuencia cuya misión será no entorpecer otras consultas para las que el tamaño de sus tuplas podría estar haciendo de cuellos de botella y otra con el resto de los atributos unidas ambas por una relación 1:1.

9.5 Captura del diagrama de Tablas.