Desarrollo de aplicaciones en entorno de 4º generación y con herramamienta CASE

Indice

1. Bases de datos. Definiciones y conceptos básicos

Introducción

Un **archivo** es el conjunto organizado de informaciones del mismo tipo que pueden utilizarse en un mismo tratamiento como soporte material de estas informaciones, es decir, es una colección de información (datos relacionados entre sí) localizada o almacenada como una unidad en algún medio de almacenamiento.

Sistemas de archivos frente a bases de datos

Cualquier base de datos está formada a fin de cuentas por ficheros, integrados y relacionados de cierta forma para facilitar el trabajo a los programadores, de forma que se puedan olvidar de muchos de los detalles del almacenamiento y la gestión de esos ficheros. Usar sistemas de archivos o bases de datos para nuestra aplicación, es algo que dependerá en gran medida del tipo de aplicación, ya que cada opción tiene sus características propias, que aportan ventajas e inconvenientes según qué tipo de problemas queramos resolver.

Sistemas de archivos

Un sistema de ficheros o archivos es un conjunto de programas que prestan servicio a los usuarios finales. Cada programa define y maneja sus propios datos. Se crea la estructura específica para dar solución a una demanda concreta, en lugar de establecer un sistema centralizado en donde almacenar todos los datos de la organización o empresa, se escoge un modelo descentralizado en el que cada sección o departamento almacena y gestiona sus propios datos. Se encontran diferentes programas, diferentes programadores, diferentes formatos y diferentes lenguajes.

Las **organizaciones de archivos** representan la forma en la que se disponen los datos para su almacenamiento en un fichero en un dispositivo de almacenamiento secundario. Tenemos tres tipos:

- **Secuencial:** Los registros se almacenan y graban en el mismo orden en el que fueron introducidos (uno detrás de otro). Para acceder a un registro hay que leer previamente todos los anteriores. Para insertar hay que hacerlo al final y no es posible borrarlo. Es posible utilizar archivos auxiliares para poder hacer estas operaciones aunque son muy complejas. Se pueden utilizar tanto cintas de almacenamiento secuencial como discos de acceso directo. Son muy lentas y complejas. Hay que evitarla siempre que se pueda.
- **Directo o Aleatorio:** Un registro se almacena en una posición determinada del dispositivo de almacenamiento (disco) calculada a partir de la aplicación de un algoritmo al campo clave del registro. Para acceder al registro sólo hay que conocer su clave, calcular su posición y acceder al registro directamente. Sólo puede usarse en un dispositivo de acceso directo (no cintas). Acceso directo o secuencial. Tratamiento de registros muy rápido, tanto en acceso como en inserción y borrado.
- **Indexado:** Utilizamos un fichero de índices (sólo con el campo clave, de acceso secuencial) que nos indica la posición de los datos buscados en otro fichero de datos (acceso directo).

Los inconvenientes son:

- Redundancia de información. Un mismo dato puede estar repetido varias veces, en varios ficheros
- Inconsistencia de los datos. Debido a la redundancia. Consiste en que tenemos

almacenadas dos copias del mismo dato con valores distintos, debido a que se ha actualizado en un caso pero no en otro.
- Dificultad en el acceso a los datos. En general requiere más tiempo que el acceso a las bases de datos.
- Dificultad en la reestructuración de la información. Resulta complicado añadir nuevos campos en ficheros existentes.
- Tratamiento ineficaz de los datos. No se pueden modificar ni actualizar todos los datos al mismo tiempo.
- Dificultad del tratamiento de los ficheros en su conjunto debido a la incompatibilidad de los lenguajes de programación.
- Dependencia excesiva del formato (aislamiento de datos). La información en los ficheros está en varios formatos no estándares. Sólo se pueden realizar las consultas que se han tenido en cuenta a la hora de escribir los programas de aplicación
- Descentralización de los datos (por no estar todos integrados en la misma colección).
- Las actualizaciones y modificaciones de los ficheros no son compartidas por toda la organización lo cual lleva a grandes confusiones.

Podemos afirmar que en el procesamiento de datos mediante sistemas de archivos existe una fuerte dependencia entre las aplicaciones y la organización física de los datos.

Sistemas de bases de datos

Para solventar los problemas de los sistemas de archivos surge a mediados de los años 60 un concepto nuevo, las bases de datos. Una **base de datos** es una colección de archivos relacionados que almacenan tanto una representación abstracta del dominio del problema como los datos correspondientes a la información acerca de ese sistema, sujetos todos a una serie de restricciones. Podemos afirmar que una base de datos es un conjunto o depósito de datos relacionados entre sí, almacenados en soporte informático, que permite el acceso directo a los mismos, junto a un conjunto de programas que manipulan esos datos.

Base de datos es un conjunto exhaustivo no redundante de datos estructurados organizados independientemente de su utilización y su implementación en máquina, accesibles en tiempo real y compatibles con usuarios concurrentes con necesidad de información diferente.

No es redundante, es independientemente de su utilización y su implementación en máquina y los datos son compatibles con usuarios concurrentes con necesidad de información diferente. Las bases de datos requieren básica y fundamentalmente un software de gestión que facilite las operaciones y las interfaces con los usuarios. Esto es el **Sistema de Gestión de Bases de Datos**

Ventajas aportan los sistemas de bases de datos respecto a los sistemas de archivos.

- Independencia de los datos respecto de los procedimientos. El usuario tiene una visión abstracta de los datos, sin necesidad de ningún conocimiento sobre la implementación de los ficheros de datos, índices, etc. Sin ella, el mantenimiento de la base de datos ocuparía el 50% de los recursos humanos dedicados al desarrollo de cualquier aplicación.
- Disminución de las redundancias
- Disminución de la posibilidad de que se produzca inconsistencia de datos
- Mayor integridad de los datos
- Mayor disponibilidad de los datos
- Mayor seguridad de los datos
- Mayor privacidad de los datos
- Mayor eficiencia en la recogida, codificación y entrada en el sistema.
- Interfaz con el pasado y futuro: una base de datos debe estar abierta a reconocer información organizada físicamente por otro software.
- Compartición de los datos. Los datos deben poder ser accedidos por varios usuarios

simultáneamente, teniendo previstos procedimientos para salvaguardar la integridad de los mismos.

Podríamos destacar las siguientes desventajas:

- Desventajas relativas a la implantación:
    - Instalación costosa en equipos y software.
    - Ausencia de estándares que facilite su uso.
    - Instalación larga y difícil.
    - Falta de rentabilidad a corto plazo.
- Desventajas relativas a los usuarios:
    - Necesidad de formación de un personal especializado.

Usaremos el sistema de archivos cuando la cantidad de datos a guardar sea tan reducida que no justifique las desventajas del uso de los sistemas de bases de datos.

Lenguajes de definición y manipulación de datos

Los SGBD proporcionan a los usuarios diferentes tipos de lenguajes de bases de datos para interactuar con ellos. Existen distintas clasificaciones de este tipo de lenguajes. Nosotros vamos a escoger la que los divide en lenguajes de definición de datos y lenguajes de manipulación de datos.

Los **Lenguajes de definición de datos**, más conocidos por su acrónimo anglosajón **DDL** (Data Definition Language) permiten: Especificar el esquema de la base de datos, Modificar la estructura del esquema, Especificar las condiciones de integridad, Hacer consultas a la totalidad de los datos y Mejorar el acceso a la información. En muchos SGBD el DDL es también utilizado para definir esquemas internos y esquemas externos (vistas). En algunos SGBD se separa el lenguaje de definición de almacenamiento (ADL) y el lenguaje de definición de vistas (VDL) que son usados para definir esquemas internos y externos respectivamente.

Los **Lenguajes de manipulación de datos**, más conocidos por su acrónimo anglosajón **DML** (Data Manipulation Language), o por otros nombres como lenguajes de acceso de datos, de gestión de datos o de consulta, permiten la Consulta, Actualización, Inserción y Borrado de datos.

Según la **forma de indicar la información** que queremos obtener:

- Lenguajes procedimentales. Permiten especificar en el momento de recuperar información no sólo qué información se desea recuperar sino también cómo se desea hacer la operación. Ej. IMS y resto de SGBD jerárquicos o en red.
- Lenguajes declarativos o no procedimentales. Sólo permiten especificar qué información se desea recuperar, pero no el modo de hacerlo. Ej. SQL en los SGBD relacionales.

Según la **forma de utilizar las sentencias** del DML:

- Modo Conversacional. Si el usuario puede ejecutar las sentencias de modo interactivo a través de un intérprete
- Modo Diferido. Cuando las sentencias quedan pospuestas en el tiempo, incluyendo todo un grupo de ellas en algún fichero para ejecutarlas por lotes.

Según la **forma que las sentencias tienen de recuperar la información**:

- De bajo nivel o "de procedimiento":
    - Las instrucciones recuperan registro a registro y se necesita procesar los datos usando bucles.

        - Deben estar embebidos en un lenguaje de programación.
- De alto nivel o "de no procedimiento":
    - Las instrucciones recuperan un conjunto de registros.
    - Permite introducir instrucciones interactivamente.
    - Las instrucciones también pueden estar embebidas en un lenguaje de programación. Un lenguaje de este tipo es SQL, en el cual profundizaremos a lo largo de este módulo.

Otros términos referidos a DML habitualmente usados son:

- Lenguajes de consulta: Así se llama a los DML que se usan interactivamente.
- Lenguaje anfitrión: El que permite tener embebidas instrucciones del DML
- Sublenguaje de datos, sería el DML embebido en otro lenguaje.

Abstracción de la información. Arquitectura de una BD

Un objetivo importante de un sistema de base de datos es proporcionar a los usuarios una visión abstracta de los datos, es decir, el sistema esconde ciertos detalles de cómo se almacenan y mantienen los datos. Existen diferentes niveles de abstracción para simplificar la interacción de los usuarios con el sistema:

- **Nivel Físico o Interno,** el del almacenamiento físico. Es la representación del nivel más bajo de abstracción. En éste se describe en detalle la forma de almacenar los datos en los dispositivos de almacenamiento, es decir, los datos y sus relaciones a nivel físico del almacenamiento secundario (por ejemplo, mediante señaladores o índices para el acceso aleatorio a los datos)

- **Nivel Conceptual**, el del usuario. Es el siguiente nivel más alto de abstracción. Describe qué datos son almacenados realmente en la base de datos y las relaciones que existen entre los mismos, describe la base de datos completa en términos de su estructura de diseño. Consta de las siguientes definiciones:

    - Definición de los datos: Se describen el tipo de datos y la longitud de campo de todos los elementos direccionables en la base de datos. Los elementos por definir incluyen artículos elementales (atributos), totales de datos y registros conceptuales (entidades).
    - Relaciones entre datos: Se definen las relaciones entre datos para enlazar tipos de registros relacionados para el procesamiento de archivos múltiples.

- **Nivel Externo,** el del programador. Es el nivel más alto de abstracción. Es lo que el usuario final puede visualizar del sistema terminado, describe sólo una parte de la base de datos al usuario acreditado para verla. El sistema puede proporcionar muchas visiones para la misma base de datos. Los únicos datos que existen realmente están en el nivel físico.

El proceso de transformar solicitudes y resultados de un nivel a otro se denomina **correspondencia o transformación (mapping)**

Independencia de los datos

Las Bases de Datos tienen un formato estándar que es conocido por los programadores, además, este formato es totalmente independiente de los procesos para los que vayan a ser utilizados dichos datos. Conseguimos almacenar los datos en un formato estándar independiente de los

procesos y de la manera en que se desee presentar la información. Tampoco es necesario tener distintos archivos para distintos formatos. Los programadores no tienen por qué preocuparse del formato en el que están almacenados los datos, sólo programar los accesos.
Como hemos visto en la abstracción de la información, el nivel conceptual describe la base de datos completa en términos de su estructura de diseño. Esta definición nos permite dividirlo en otros dos niveles:

- **Conceptual**. Corresponde a la visión del sistema global desde un punto de vista organizativo independiente, no informático.
- **Lógico**. Correspondería a la visión de la base de datos expresada en términos del sistema que se va a implantar con medios informáticos.

Definimos **independencia de datos** como la capacidad para modificar el esquema de un nivel del sistema de la base de datos sin tener que modificar el esquema del nivel inmediato superior. Podemos de definir dos tipos de independencia de los datos.

- **Por independencia lógica** de los datos se entiende que los cambios en el esquema lógico no deben afectar a los esquemas externos que no utilicen los datos modificados. Es decir, es la capacidad de modificar el esquema conceptual sin tener que alterar los esquemas externos ni los programas de aplicación.
- **Por independencia física** de los datos se entiende que el esquema lógico no se vea afectado por cambios realizados en el esquema interno, correspondientes a modos de acceso, etc. Es decir, es la capacidad de modificar el esquema interno sin tener que alterar el esquema conceptual (o los externos).

Sistemas de gestión de bases de datos (SGBD)

Las bases de datos requieren básica y fundamentalmente un Software de Gestión que facilite las operaciones y las interfaces con los usuarios. Esto es el Sistema de Gestión de Bases de Datos (S.G.B.D.). Según, De Miguel, 1985, el **S.G.B.D.** es un conjunto coordinado de programas, procedimientos, lenguajes, etc... que suministra, tanto a los usuarios no informáticos, como a los analistas programadores, o al administrador, los medios necesarios para describir y manipular los datos contenidos en la base de datos, manteniendo su integridad, confidencialidad y seguridad.

Las operaciones típicas que debe realizar un SGBD son las siguientes:

- Aquéllas que afectan a la totalidad de los datos. Creación, Reestructuración y Consultas a la totalidad
- Las que tienen lugar sobre registros concretos, que suelen llamarse operaciones de Actualización: Altas o inserciones, Bajas o borrados, Modificacionesy Consultas selectivas.

Funciones de un SGBD

- **Función de descripción o definición.** Permite al administrador de la BD especificar los elementos de datos que la integran, su estructura y las relaciones que existen entre ellos, las reglas de integridad semántica, los controles a efectuar antes de autorizar el acceso a la BD, etc., así como las características de tipo físico y las vistas lógicas de los usuarios. Esta función, realizada por el lenguaje de descripción o definición de datos (DDL) propio de cada SGBD, debe suministrar los medios para definir las tres estructuras de datos o vistas:
    - A nivel interno, se ha de indicar el espacio de disco reservado para la base de datos, la longitud de los campos, su modo de representación (lenguaje para la definición de la estructura externa).
    - A nivel conceptual se proporcionan herramientas para la definición de las entidades y su identificación, atributos de las mismas, interrelaciones entre ellas, restricciones de integridad, etc.; es decir el esquema de la base de datos (lenguaje para la

definición de estructura lógico global).
    - A nivel externo, se deben definir las vistas de los distintos usuarios a través del lenguaje para la definición de estructuras externas.

- **Función de manipulación.** Permite a los usuarios buscar, añadir, suprimir o modificar los datos de la base de datos, siempre de acuerdo con las especificaciones y las normas de seguridad dictadas por el administrador. Se llevará a cabo por medio de un lenguaje de manipulación de datos (DML)

- **Función de utilización.** Reúne todas las interfaces que necesitan los diferentes usuarios para comunicarse con la BD y proporciona un conjunto de procedimientos para el administrador. Incluye funciones de servicio como cambiar la capacidad de los ficheros, obtener estadísticas de utilización, cargar archivos, etc. y los relacionados con la seguridad física (copias de seguridad, etc). Esto lo realiza el LCD, Lenguaje de Control de Datos.

Los datos forman siempre un conjunto estructurado que pretenden ser una representación del mundo real y que son utilizados indistintamente por todas las aplicaciones, sean eventuales o periódicas. En ambos casos, las aplicaciones se apoyan en las facilidades del SGBD, y estos, a su vez, se apoya en los métodos del acceso del SO. Para las aplicaciones eventuales, el SGBD proporciona facilidades complementarias, como lenguajes autocontenidos mientras que para atender las aplicaciones periódicas se suelen escribir procedimientos embebidos en un lenguaje de programación o escritos en un lenguaje de 4ª generación (4GL o L4G).

Para realizar todas las funciones descritas anteriormente, es necesario que el SGBD cuente con una serie de **componentes**.

- **Lenguajes de la base de datos.** Los distintos lenguajes que se utilizan en todas las tareas relacionadas con la creación, mantenimiento y uso de la base de datos: El lenguaje de definición de datos (DDL), El lenguaje de manipulación de datos (DML) y El lenguaje de control de datos (DCL).
- **Diccionario de datos.** Es un conjunto de archivos que contienen información acerca de los datos que se almacenan en la base de datos. Se trata de una "metabase de datos", es decir, una base de datos que contienen información sobre la base de datos (datos acerca de los datos). Se encuentra almacenada: La representación de los datos a los tres niveles de abstracción (esquema lógico, físico y subesquemas externos de la base de datos), Las restricciones de privacidad y acceso a los datos definidas por el DDL y DCL, Las reglas, normas o restricciones referentes a la seguridad de los datos y otras informaciones referentes a garantizar la integridad de los datos.
- **El gestor de la base de datos.** Es un componente software encargado de garantizar el correcto, seguro, íntegro y eficiente acceso y almacenamiento de los datos. Este componente es el encargado de proporcionar una interfaz entre los datos almacenados y los programas de aplicación que los manejan. Toda operación que se quiere realizar "contra" la base de datos debe ser previamente autorizada por el gestor de la misma. Los principales componentes del gestor de la base de datos son los siguientes:
    - Control de autorización.
    - Procesador de comandos. Una vez comprobado los permisos se pasa el control al procesador de comandos.
    - Control de la integridad. Cuando se cambian los datos de la base, este módulo comprueba que la operación satisface todas las restricciones de integridad.
    - Optimizador de consultas. Determina la estrategia óptima para la ejecución de las consultas.
    - Gestor de transacciones. Realiza el procesamiento de las transacciones.
    - Planificador (scheduler). Es el responsable de asegurar que las operaciones que se realizan concurrentemente sobre la base de datos tienen lugar sin conflictos.
    - Gestor de recuperación. Garantiza que la base de datos permanece en un estado

consistente en caso de que se produzca algún fallo.
    - Gestor de buffers. Es el responsable de transferir los datos entre memoria principal y los dispositivos de almacenamiento secundario. A este módulo también se le denomina gestor de datos.

  El gestor de la base de datos es responsable de garantizar La privacidad de los datos, Su seguridad, Su integridad, El acceso concurrente sin perdida de integridad y La interacción con el sistema operativo
- **El administrador de la base de datos.** Es una persona o grupo de personas encargadas de la función de administración de la base de datos.
- **Los usuarios de la base de datos.** Existen varios:
    - **Usuarios normales o terminales**: interactúan con la base de datos a través de programas de aplicaciones
    - **Usuarios técnicos:** son profesionales informáticos que desarrollan los programas de aplicación que van a ser utilizados por los usuarios normales de la base de datos.
    - **Usuarios directivos**: que fijarán al administrador de la base de datos los objetivos de la base de datos para que respondan a los objetivos generales de la empresa.

El criterio principal que se utiliza para clasificar los SGBD es el **modelo lógico** en que se basan. Son tres:

- El **modelo relacional** se basa en el concepto matemático denominado "relación", que gráficamente se puede representar como una tabla. Es percibida por el usuario como un conjunto de tablas. Esta percepción es sólo a nivel lógico
- El **modelo de red** representa los datos como colecciones de registros y las relaciones entre los datos se representan mediante conjuntos, que son punteros en la implementación física. Los registros se organizan como un grafo: los registros son los nodos y los arcos son los conjuntos. El SGBD de red más popular es el sistema IDMS.
- El **modelo jerárquico** es un tipo de modelo de red con algunas restricciones. De nuevo los datos se representan como colecciones de registros y las relaciones entre los datos se representan mediante conjuntos. Sin embargo, en el modelo jerárquico cada nodo puede tener un solo padre. Los registros son los nodos, también denominados segmentos, y los arcos son los conjuntos. El SGBD jerárquico más importante es el sistema IMS.

La mayoría de los SGBD comerciales actuales están basados en el modelo relacional, mientras que los sistemas más antiguos estaban basados en el modelo de red o el modelo jerárquico. Estos dos últimos modelos requieren que el usuario tenga conocimiento de la estructura física, mientras que el modelo relacional proporciona una mayor independencia de datos. Se dice que el modelo relacional es **declarativo** (se especifica qué datos se han de obtener) y los modelos de red y jerárquico son **navegacionales** (se especifica cómo se deben obtener los datos).

El **modelo orientado a objetos** define una base de datos en términos de objetos, sus propiedades y sus operaciones. Los objetos con la misma estructura y comportamiento pertenecen a una clase, y las clases se organizan en jerarquías o grafos acíclicos. A estos SGBD se les conoce como sistemas **objeto-relacionales**

Clasificación de los SGBD: según el número de usuarios y según el número de sitios

- Clasificación según el **número de usuarios** a los que se da servicio:
    - Los sistemas **monousuario** sólo atienden a un usuario a la vez, y su principal uso se da en los ordenadores personales.
    - Los sistemas **multiusuario** atienden a varios usuarios al mismo tiempo.
- Clasificación según el **número de sitios** en los que está distribuida la BD.
    - Casi todos los SGBD son **centralizados**: sus datos se almacenan en un solo

ordenador.
    - En los SGBD **distribuidos** la base de datos real y el propio software del SGBD pueden estar distribuidos en varios sitios conectados por una red.
    - Los SGBD **distribuidos homogéneos** utilizan el mismo SGBD en múltiples sitios. Una tendencia reciente consiste en crear software para tener acceso a varias bases de datos autónomas preexistentes almacenadas en SGBD distribuidos heterogéneos. Esto da lugar a los SGBD federados o sistemas multibase de datos en los que los SGBD participantes tienen cierto grado de autonomía local.

Clasificación de los SGBD: según el coste y según el propósito

- Clasificación de los SGBD **según el coste**:
    - La mayor parte de los paquetes de SGBD cuestan entre 10.000 y 100.000 euros.
    - Los sistemas monousuario más económicos para microcomputadores cuestan entre 100 y 3.000 euros.
    - En el otro extremo, los paquetes más completos cuestan más de 100.000 euros.
- Clasificación de los SGBD **según el propósito**:
    - De propósito **específico**. Cuando el rendimiento es fundamental, se puede diseñar y construir un SGBD de propósito especial para una aplicación específica, y este sistema no sirve para otras aplicaciones. Muchos sistemas de reservas de líneas aéreas son SGBD de propósito especial y pertenecen a la categoría de sistemas de procesamiento de transacciones en línea (OLTP), que deben atender un gran número de transacciones concurrentes sin imponer excesivos retrasos.
    - Los SGBD de propósito **general** están diseñados para atender a cualquier tipo de aplicaciones. A cambio, nunca pueden ofrecer la máxima eficiencia para cada caso

Clasificación de los Usuarios

- **Administrador de la base de datos (ABD) o Data Base Administrator (BDA)**. Tiene el control centralizado de la base de datos y es el responsable de su buen funcionamiento. Es el encargado de autorizar el acceso a la base de datos, de coordinar y vigilar su utilización y de adquirir los recursos software y hardware que sean necesarios.

- **Diseñadores de bases de datos.** Se encargan de identificar los datos que se almacenarán en la base de datos y de elegir las estructuras apropiadas para almacenar dichos datos.

- **Operadores y personal de mantenimiento.** Forman parte del personal del ABD y son los responsables del funcionamiento y mantenimiento reales del entorno software y hardware del sistema de base de datos.

- **Usuarios finales**. Son las personas cuyo trabajo requiere acceder a la base de datos para consultarla, actualizarla y generar informes. La base de datos existe para que ellos la utilicen. Existen las siguientes categorías de usuarios finales:
    - Usuarios finales ocasionales. Estos acceden de vez en cuando a la base de datos pero es muy probable que necesiten información diferente en cada ocasión
    - Utilizan un lenguaje de consulta de bases de datos avanzado para especificar sus solicitudes y suelen ser gerentes de nivel medio o alto
    - Usuarios finales simples o paramétricos. Suelen ser la porción más considerable de la totalidad de los usuarios finales. La función principal de su trabajo gira en torno a consultas y actualizaciones constantes de la base de datos, utilizando tipos estándar de consultas y actualizaciones.
    - Usuarios finales avanzados. Suelen ser personal altamente cualificado que está

suficientemente familiarizado con los recursos del SGBD como para implementar sus aplicaciones de forma que cumplan sus complejos requerimientos.
    - Usuarios finales autónomos. Mantienen bases de datos personales mediante la utilización de paquetes de programas comerciales que cuentan con interfaces de fácil uso basados en menús o en gráficos.

- **Analistas de sistemas y Programadores de aplicaciones.** Aunque no son considerados como usuarios en sí de una base de datos, sí es conveniente conocer la figura.
    - Los analistas de sistemas determinan los requerimientos de los usuarios finales, sobretodo de los simples o paramétricos, y desarrollan especificaciones para transacciones programadas que satisfagan dichos requerimientos.
    - Los programadores de aplicaciones implementan esas especificaciones en forma de programas y luego prueban, depuran, documentan y mantienen estas transacciones programadas.

2. Modelo entidad-relación

Introducción

Gracias al modelo conceptual Entidad-Relación, creado por Peter Chen en los años setenta, podemos representar el mundo real a través de una serie de símbolos y expresiones determinados. El objetivo de este modelo es simplificar el diseño de bases de datos partiendo de las descripciones textuales de la realidad, que establecen los requerimientos del sistema

Lo primero que tenemos que hacer cuando vamos a crear una base de datos es analizar el problema sobre el papel y pensar qué tipo de información necesitamos guardar.

Entidades y relaciones

Este modelo hace uso básicamente de tres componenes: Entidades, Relaciones entre dichas entidades y Atributos. Además, para potenciar la capacidad expresiva del modelo Entidad-Relación se tiene en cuenta la definición de atributos compuestos y los objetos especializados (o generalizados)

Los componentes básicos de un sistema de información son los **objetos o entidades** de los que se quiere almacenar la información. Todos los objetos de una misma clase se representan con un tipo de Entidad concreto que se diferencia de otra entidad porque posee ciertas características que la hacen única. Cada entidad tendrá una serie de instancias, que no son más que objetos, ocurrencias concretas de ese tipo de entidad.

Una entidad representa cualquier persona, suceso, evento o concepto (en otras palabras, cualquier "cosa") sobre el que queramos almacenar información.

Una entidad cumple las siguientes **propiedades**:

- **Tiene existencia propia**. La entidad existe como un elemento que interviene en el comportamiento global del sistema.
- Es **diferente** del resto de las entidades (objetos)
- Las entidades de un mismo tipo tienen características y propiedades similares.

Las entidades las podemos clasificar en:

- **Entidades Fuertes, o regulares.** Son aquellas entidades que existen por sí mismas. La existencia de una instancia en la entidad no depende de la existencia de otras instancias en otra entidad. En el modelo E/R una entidad fuerte se representa con un rectángulo nominado, es decir con un rectángulo en cuyo interior aparece el 'nombre' de la entidad.

- **Entidades Débiles.** Son aquellas entidades en las que se hace necesaria la existencia de instancias de otras entidades distintas para que puedan existir instancias en esta entidad. Por tanto, la existencia de una instancia de una entidad débil depende de la existencia de una instancia de la entidad fuerte con la que se relaciona. En el modelo E/R una entidad débil se representa con un rectángulo doble nominado. Existen dos tipos de dependencia:
    - Dependencia en existencia (entre entidades). Si desaparece una instancia del tipo de entidad fuerte deben desaparecer las instancias de la entidad débil que dependen de ella. Se nota con una "E" en la relación débil.

- Dependencia en identificación. Se produce cuando además de la dependencia en existencia, una instancia del tipo de entidad débil no se puede identificar por sí misma, y debe hacerse mediante la clave de la entidad fuerte asociada. Su clave es (clave_entidad_fuerte, clave_parcial). Se nota con una "ID" en la relación débil.

Tanto las entidades fuertes como las débiles se nombran habitualmente con sustantivos en singular.

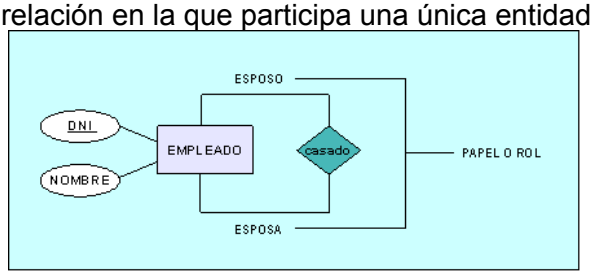
Se entiende por **relación** aquella asociación o correspondencia existente entre entidades. Para definir una relación debemos tener en cuenta los siguientes elementos:

- **Nombre de la relación.** Cada relación tiene un nombre que la distingue claramente del resto y mediante el cual ha de ser referenciada. Se utiliza un verbo en forma singular.
- **Grado de la relación.** Es el número de entidades que participan en una relación.
- **Cardinalidad de la relación**. Es el número máximo de instancias de cada entidad que pueden intervenir en una instancia de relación que se está tratando. Puede ser 1:1, 1:N, N:1 o N:M, que se leen respectivamente como uno a uno, uno a muchos, muchos a uno y muchos a muchos.
- **Cardinalidades de las entidades.** Se definen como el número máximo y mínimo de instancias de una entidad que pueden estar relacionadas con una instancia de otra u otras entidades que participan en la relación. Su representación gráfica es una etiqueta del tipo (0,1), (1,1), (0,N) o (1,N), según corresponda.

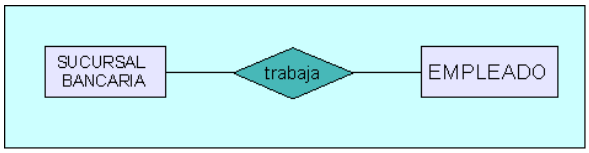
Al igual que las entidades, las relaciones se clasifican también en: fuertes, asocian dos entidades fuertes y débiles, asocian una entidad débil con otra fuerte.

Podemos distinguir los siguientes **tipos de relaciones según su grado**:

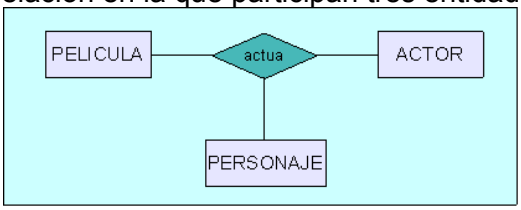
- **Unaria:** Es aquella relación en la que participa una única entidad.



- **Binaria:** Es aquella relación en la que participan dos entidades, es el tipo más habitual de relación.



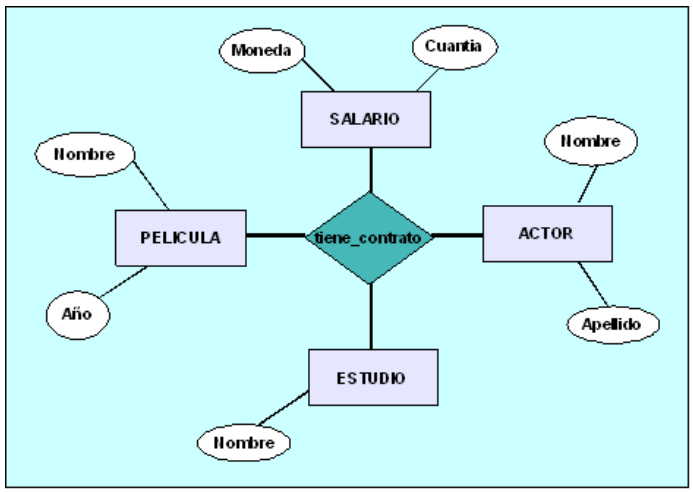
- **Ternaria:** Es aquella relación en la que participan tres entidades al mismo tiempo.



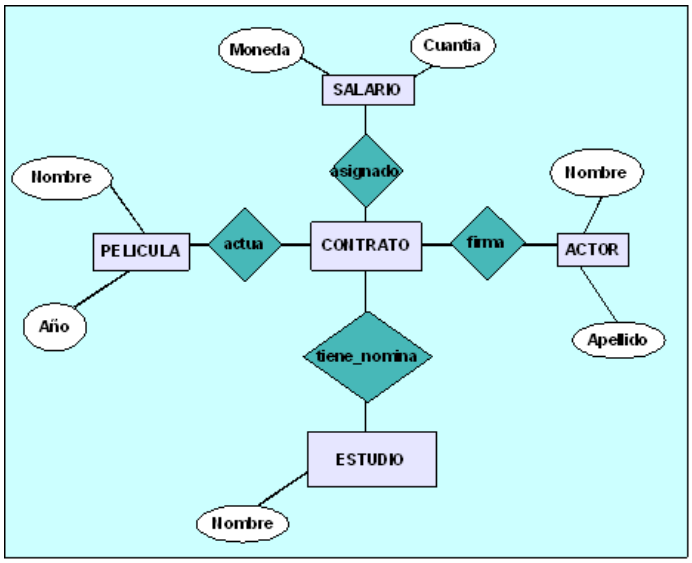
- **N-aria:** Es aquella relación en la que participan n conjuntos de entidades. Es muy poco

frecuente su aparición y debe disminuirse el grado de la relación para hacer más intuitivo el modelado de nuestro sistema a representar.

Para disminuir el grado de la siguiente relación:



Es muy sencillo. Sustituimos la relación 'tiene_contrato' por una entidad nueva llamada CONTRATO y convertimos todas las relaciones en binarias de la manera que puede apreciarse en la figura:



Gracias al modelo **Entidad-Relación Extendido** podemos considerar ciertas restricciones acerca de las relaciones que el modelo Entidad-Relación no contemplaba y que nos permite representar con mayor exactitud determinados comportamientos de las entidades, y sus correspondientes relaciones.

- **Relaciones con restricciones de Exclusividad.** Se dice que dos (o más) relaciones tienen una restricción de exclusividad con respecto a una entidad que participa en ambas relaciones cuando cada instancia de dicha entidad sólo puede pertenecer a una de las relaciones, pero en el momento en que pertenezca a uno ya no podrá formar parte del otro. Ejemplo: Un profesor puede impartir o recibir cursos, pero no ambas cosas. Si el profesor no es doctor podrá recibir cursos de doctorado y en caso contrario impartirlos.

- **Relaciones con restricciones de Exclusión.** Se permite a un profesor ya doctor

matricularse en cursos aunque él, a su vez, esté impartiendo otros cursos. En este caso la restricción que debemos imponer es que un profesor no esté impartiendo y recibiendo el mismo curso.

- **Relaciones con restricciones de Inclusividad.** Supongamos ahora que se desea imponer la restricción de que sólo pueden impartir clases en nuestro programa de doctorado aquellos profesores que hayan realizado al menos un curso dentro de este mismo programa, aunque no tiene porqué ser el mismo que él imparte. Aplicamos entonces una restricción de inclusividad entre dos relaciones (o más) con respecto a una de las entidades que participa en ambas relaciones, por lo cual toda instancia de dicho tipo de entidad que participa en una de las relaciones tiene necesariamente que participar en la otra.

- **Relaciones con restricciones de Inclusión.** Si un profesor imparte un curso es porque previamente ha tenido que recibir dicho curso. Aplicamos pues una restricción de inclusión

Atributos

Un **atributo** es cualquier detalle que sirve para calificar, identificar, clasificar, cuantificar o expresar el estado de algo, en nuestro caso de una entidad, es decir un atributo es cualquier descripción de una característica de importancia. Los atributos de una entidad se representan mediante elipses o círculos etiquetados, que se conectan por una línea recta a la entidad que califica, cada uno de los cuales tiene que tener un nombre único y que haga referencia a su contenido. Los nombres de los atributos deben ir en minúsculas.

Cada atributo tiene un conjunto de valores asociados denominado **dominio**. El dominio define todos los valores posibles que puede tomar un atributo. Generalmente, los dominios nos sirven para limitar el tamaño de los atributos.

Primera clasificación que podemos hacer de los atributos.

- **Un atributo obligatorio (identificador)** es aquél que siempre debe estar definido para la entidad. Está claro que si tenemos una entidad EMPLEADO, un atributo obligatorio de esa entidad debe ser 'DNI'. Se pone la palabra subrayada.
- **Un atributo opcional** puede quedar sin definir para algunas de las instancias de la entidad. En el caso de la entidad EMPLEADO un atributo opcional podría ser 'edad', que es un atributo que no es imprescindible para la identificación de las instancias de la entidad.

Otra clasificación que se puede realizar de los atributos es si son simples o compuestos.

- Un **atributo simple** es un atributo que tiene un solo componente, que no se puede dividir en partes más pequeñas que tengan un significado propio, un ejemplo claro sería el DNI de una persona.
- Un **atributo compuesto** es un atributo con varios componentes, cada uno con un significado por sí mismo, por ejemplo si consideramos la dirección de una persona como la unión de la calle donde vive, el número y la población. Un grupo de atributos se representa mediante un atributo compuesto cuando tienen afinidad en cuanto a su significado, o en cuanto a su uso.

Los atributos también pueden clasificarse según los valores que toman en monovalentes o polivalentes.

- Un **atributo monovalente** es aquél que tiene un solo valor para cada instancia de la

entidad o relación a la que pertenece, como es el caso del DNI en nuestro ejemplo.

- Un **atributo polivalente** es aquél que tiene varios valores para cada instancia de la entidad o relación a la que pertenece, como es el caso del atributo 'teléfono' en nuestro ejemplo podríamos estar interesado en guardar la información del teléfono de casa, el móvil y el que tenemos en el trabajo.

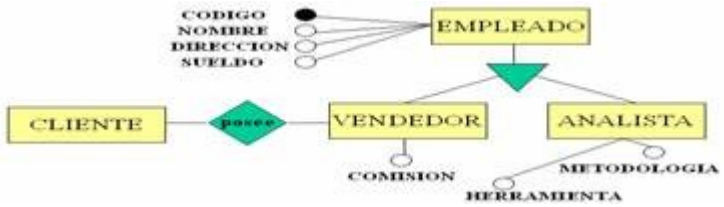
La **cardinalidad** de un atributo indica el número mínimo y el número máximo de valores que puede tomar para cada ejemplar de la entidad o relación a la que pertenece. La **cardinalidad mínima** indica la cantidad de valores del atributo que debe existir para que la entidad sea válida. Este número casi siempre es 0 (no se requiere que el atributo tenga un valor) o 1 (el atributo debe tener un valor). La **cardinalidad máxima** indica la cantidad máxima de valores del atributo que puede tener la entidad. Por lo general es 1 o N. Si es 1, el atributo no puede tener más que un valor, si es N, el atributo puede tener múltiples valores y no se especifica la cantidad absoluta.

Por último, los atributos pueden ser derivados, aunque habitualmente este tipo de atributo no se considera en esta fase del diseño del modelo Entidad-Relación. Un **atributo derivado** es aquél que representa un valor que se puede obtener a partir del valor de uno o varios atributos, que no necesariamente deben pertenecer a la misma entidad o relación. Por ejemplo, la edad es un atributo derivado, ya que puede obtenerse a partir de la fecha de nacimiento del empleado.

Las relaciones también pueden tener atributos asociados. Se representan igual que los atributos de las entidades.

La **generalización** no es más que la reunión en un "supertipo" de entidad de una serie de "subtipos" de entidades, que tienen ciertos aspectos en común, pero que también se diferencian en algunos otros. El concepto de **especialización** es algo parecido al de generalización, pero considerado justo desde el punto de vista contrario. Si se considera de arriba hacia abajo se considera como especialización. Si se considera de abajo hacia arriba se considera como generalización. Las entidades de bajo nivel heredan todos los atributos de las entidades de mayor nivel.

La descomposición de entidades en varios subtipos es una necesidad muy habitual. La relación que se establece entre un supertipo de entidad y sus subtipos corresponde a la noción de "ES UN", más conocida por sus siglas inglesas "ISA" o, más exactamente, "ES UN TIPO DE". Se representa mediante un triángulo invertido



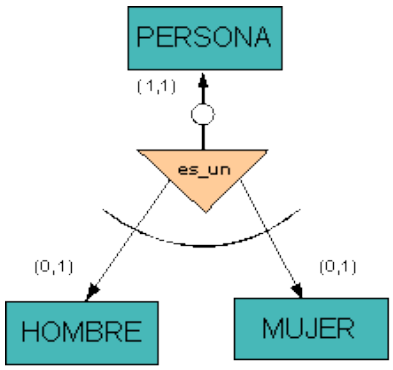
Un subtipo de entidad es un tipo de entidad que mantiene una de relación jerárquica con otro tipo de entidad supertipo, y que cumple que:

- Las propiedades y el comportamiento de los subtipos son heredados del tipo de entidad con el cual mantienen una relación jerárquica.
- Los subtipos añaden a sus propias propiedades las del supertipo del que heredan.
- Un tipo de entidad puede ser un subtipo para más de un tipo de entidad con las que puede mantener diferentes relaciones jerárquicas. Esta característica, denominada **herencia múltiple,** permite que una entidad herede propiedades y comportamiento de más de una entidad diferente.

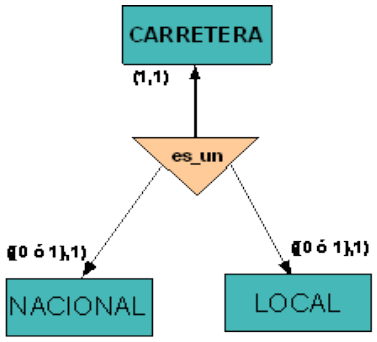
Existen los siguientes tipos de especialización según aparezca solapamiento de instancias de una

entidad o no:

- Una especialización **exclusiva**, denominada **especialización sin solapamiento** representa el hecho de que una instancia u ocurrencia del tipo de entidad más general sólo puede pertenecer o estar asociada a una y sólo una instancia u ocurrencia de los subtipos de entidad especializados. La especialización exclusiva se representa mediante un arco que une los subtipos como se muestra en la imagen siguiente:

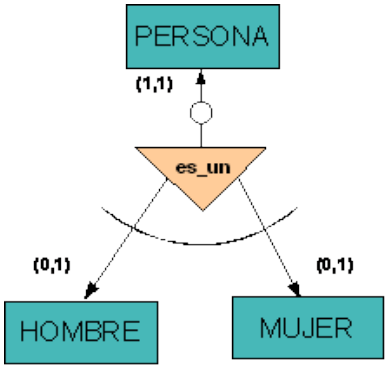


- Una especialización **inclusiva**, denominada **especialización con solapamiento**, representa el hecho de que una instancia del tipo de entidad más general puede tener asociadas instancias de cualquiera de los subtipos. La especialización inclusiva se representa sin ningún arco que una los subtipos.



Por otro lado, la especialización de un tipo de entidad en un conjunto de subtipos puede ser total o parcial:

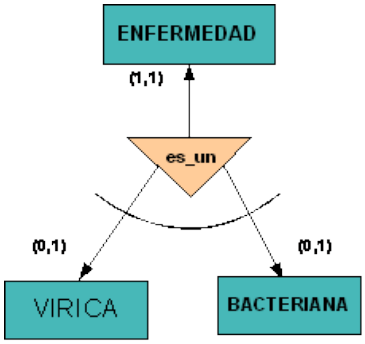
- Una **especialización total** representa el hecho de que las entidades que son reconocidas en el problema que se está representando son alguno de los subtipos especializados, no existiendo entidades que no pertenezcan a alguno, varios o todos estos subtipos de entidad. La especialización total se representa mediante un círculo superpuesto en la línea que une el supertipo con el triángulo que indica la especialización



Las personas no pueden ser ninguna otra cosa, y cualquier persona estará encuadrada en

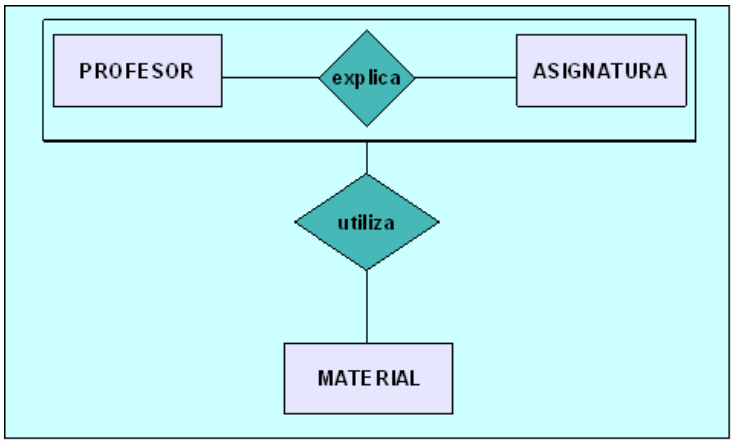
una de esos dos subtipos.

- Una **especialización parcial** representa el hecho de que pueden existir entidades que pertenezcan al tipo de entidad y no a ninguno de los subtipos en los cuales este tipo de entidad está especializado. Una especialización parcial describe un refinamiento incompleto del problema



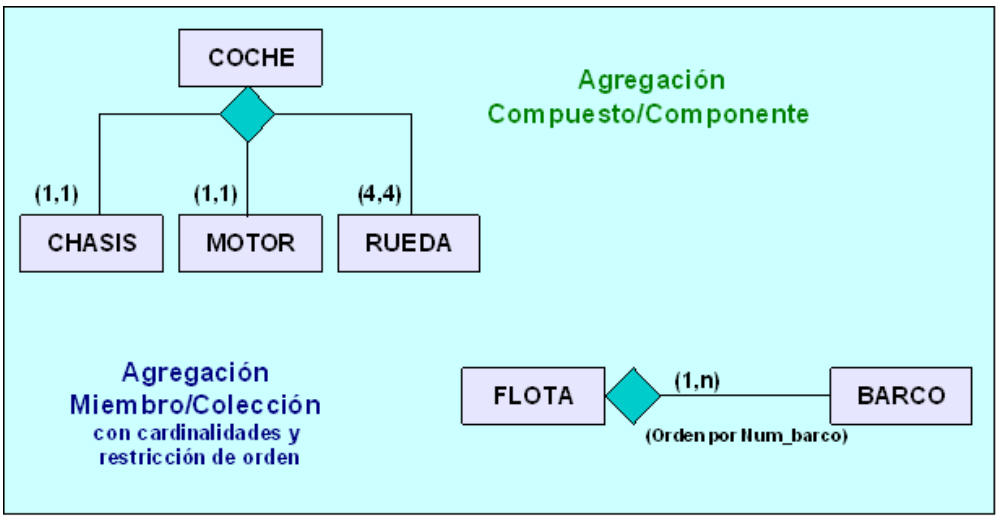
Entonces, se pueden presentar cuatro tipos de relaciones jerárquicas que pueden ser representadas mediante el modelo EE/R: total sin solapamiento, o total exclusiva, parcial sin solapamiento, o parcial exclusiva, total con solapamiento, o total inclusiva y parcial con solapamiento, o parcial inclusiva.

Las relaciones entre relaciones en el modelo E/R provocan redundancias de información, y si tratamos de simplificar el diagrama obtenido, es muy posible que no quede claro lo que pretendemos expresar. La mejor forma de modelar esta situación es usando una **agregación**. La agregación es una abstracción a través de la cual las relaciones se tratan como entidades de nivel más alto. Un ejemplo de agregación



Existen dos clases de agregaciones:

- **Compuesto/Componente:** Abstracción que permite representar que un todo o agregado se obtiene como la unión de diversas partes o componentes que pueden ser tipos de entidades distintas y que juegan diferentes roles en la agregación.
- **Miembro/Colección:** Abstracción que permite representar un todo o agregado como una colección de miembros, todos de un mismo tipo de entidad y todos jugando el mismo rol.



Cardinalidad de entidades y relaciones

La **cardinalidad** con la que una **entidad** participa en una relación especifica el número mínimo y el número máximo de correspondencias en las que puede tomar parte cada ejemplar de dicha entidad. La participación de una entidad en una relación es obligatoria (total) si la existencia de cada una de sus instancias requiere la existencia de, al menos, un ejemplar de la otra entidad participante. Si no, la participación es opcional (parcial).

La **cardinalidad de las relaciones** se obtiene de considerar el máximo número de instancias con las que puede participar cada una de las entidades en la relación, es decir con el máximo de las cardinalidades de cada una de las entidades que participan en la relación. Dependiendo del número de instancias que aparezcan, podemos tener:

- Relaciones uno a uno. Se notan por 1:1
- Relaciones uno a muchos. Se notan por 1:N.
- Relaciones muchos a uno. Se nota por N:1.
- Relaciones muchos a muchos. Se nota por N:M.

Claves primarias de entidades y relaciones

La **clave primaria** es la clave elegida por el usuario para identificar las instancias de la entidad, de entre todas las claves que tiene dicha entidad. No pueden tomar valores nulos.

Clave (Llave): Aquel atributo o conjunto de atributos que identifican a una entidad. Por ejemplo el DNI identifica claramente un ejemplar de cualquier otro dentro de la entidad EMPLEADO

Superclave (Superllave): Conjunto de atributos no vacío, que identifica en forma única a cada ejemplar dentro de una entidad. Una superclave puede tener atributos no obligatorios. El conjunto de todos los atributos de una entidad es una superclave.

- **Clave candidata (Llave candidata):** Es una superclave para la cual ningún subconjunto es superclave, excepto el mismo, es decir, al eliminar cualquiera de los atributos que la componen deja de ser superclave.
- **Clave primaria (Llave Primaria o Primary Key):** Es la clave candidata escogida por el

diseñador, de entre todas las posibles. Por lo tanto, además de ser el atributo o conjunto de atributos que permiten identificar en forma única una instancia en la entidad y ningún subconjunto de ella posee esta propiedad, podemos decir que es la única que efectivamente se usa con ese fin en la base de datos, de entre todas las posibles

Las claves primarias se representan subrayando el nombre del atributo o atributos que las constituyen en el caso de haberse representado éstos con elipses etiquetadas, y se representan con un círculo negro en el caso de representarse los atributos con círculos vacíos.

Diagrama E-R

Un **diagrama E-R** consiste en representar mediante las figuras geométricas vistas a lo largo de la unidad un modelo completo del problema, proceso o realidad a describir, de forma que se definan tanto las entidades que lo componen, como las interrelaciones (relaciones) que existen entre ellas.

Las tareas a realizar para la creación del diagrama E/R son las siguientes:

- Identificar las entidades. Se buscan los nombres o sustantivos que se mencionan que hacen referencia a objetos. Hay que diferenciar entre entidades débiles y fuertes.
- Identificar las relaciones, una vez definidas las entidades, se deben definir las relaciones existentes entre ellas. Una vez identificadas todas las relaciones, hay que determinar la cardinalidad mínima y máxima con la que participa
- Identificar los atributos y asociarlos a entidades y relaciones. Son atributos los nombres que identifican propiedades, cualidades, identificadores o características de entidades o relaciones. Al identificar los atributos, hay que tener en cuenta si son simples o compuestos.
- Identificar los atributos derivados o calculados, que son aquellos cuyo valor se puede calcular a partir de los valores de otros atributos.
- Determinar las claves candidatas y elegir las claves primarias.
- Determinar las jerarquías de generalización. En este paso hay que estudiar detenidamente las entidades que se han identificado hasta el momento. En cada jerarquía hay que determinar si es total o parcial y exclusiva o solapada.
- Dibujar el diagrama entidad-relación.
- Revisar el resultado con la información sobre el problema que tenemos.

De cada atributo se debe tener la siguiente información: nombre y descripción del atributo, si el atributo es compuesto y, en su caso, qué atributos simples lo forman y si el atributo es derivado y, en su caso, cómo se calcula su valor.

Existen cuatro estrategias a la hora de construir un esquema E/R que se resumen a continuación, y en la tabla que sigue:

- La **estrategia descendente** es análoga a la descomposición en niveles de DFD, partiendo de una única entidad que describe el universo del discurso (por ejemplo, EMPRESA) que se va descomponiendo sucesivamente con mayor nivel de detalle. Ventaja: No aparecen sorpresas de última hora. Inconveniente: Requiere un alto nivel de abstracción
- En la **estrategia ascendente**, por el contrario, se parte del nivel más bajo, es decir, los atributos, que se van agrupando en entidades. Posteriormente se crean relaciones entre

las entidades y las jerarquías de generalización hasta obtener el esquema completo. Vetanaj: Facilita las decisiones de diseño. Inconveniente: Necesidad de reestructuración conforme se progresa

- La más usual es, sin embargo, la estrategia de **"mancha de aceite"**, también llamada **Inside-Out**. Se empieza creando un esquema E/R en una parte del papel, completándose, a medida que se examina el esquema percibido en lenguaje natural y el resto de entidades y relaciones hasta "ocupar" todo el papel, de forma análoga a la manera en la que se extiende el aceite en un mantel. Ventaja: Facilita describir nuevos conceptos relacionados. Inconveniente: Sólo se tiene una visión global al final del proceso
- La **estrategia mixta**. Cuando el dominio de aplicación es muy complejo, el diseñador divide los requerimientos en subconjuntos, que más tarde se consideran por separado. Ventaja: "Divide y vencerás". Inconveniente: Requiere decisiones importantes acerca del esqueleto inicial

Tenemos que evitar repetir la información mostrada en nuestro diagrama, es decir, tenemos que **evitar la redundancia**. Para que una relación pueda ser eliminada por redundante se tiene que cumplir: que exista un ciclo, que las relaciones que componen el ciclo sean equivalentes semánticamente, que después de eliminar la relación se puedan seguir asociando las instancias de las dos entidades que estaban relacionadas, y que la relación no tenga atributos o que éstos puedan ser transferidos a otro elemento del esquema a fin de no perder su semántica. Aunque, la existencia de un ciclo no implica la existencia de relaciones redundantes.

Se definen varios criterios para evaluar la calidad de un **diagrama ERE**, o esquema conceptual, y son los siguientes:

- Ser Completo.
- Correción. Un diagrama ERE es correcto si usa apropiadamente el modelo Entidad/Relación. Hay dos tipos de corrección: sintáctica y semántica.
- Minimalidad. Un esquema es mínimo cuando cada aspecto de los requerimientos aparece una sola vez en el esquema
- Expresividad. Un esquema es expresivo cuando representa los requerimientos de una manera natural, no forzada.
- Legibilidad. Legible significa que se puede leer con facilidad,
- Autoexplicación
- Capacidad de Extensión (Flexibilidad)

3. Modelo relacional

Introducción

La arquitectura relacional se puede expresar en términos de tres niveles de abstracción: Nivel interno, Nivel conceptual y Nivel de visión.

Veamos cómo se relacionan los distintos componentes de la arquitectura relacional con los distintos niveles de abstracción:

- **Modelo relacional de datos:** En el nivel conceptual, el modelo relacional de datos está representado por una colección de relaciones (tablas) almacenadas.
- **Submodelo de datos:** En el nivel de visión, los esquemas externos de un sistema relacional se llaman submodelos relacionales de datos. Cada uno consta de una o más vistas para describir los datos requeridos por una aplicación dada. Una vista puede incluir datos de una o más tablas de datos. Cada programa de aplicación está provisto de un buffer ("Área de trabajo de usuario") donde el SGBD puede depositar los datos recuperados de la base para su procesamiento, o puede guardar temporalmente sus salidas antes de que el SGBD las escriba en la base de datos.
- **Esquema de almacenamiento:** En el nivel interno, cada tabla base se implanta como un archivo almacenado. Para las recuperaciones sobre las claves principal o secundaria se pueden establecer uno o más índices para indexar un archivo almacenado.

A finales de los años sesenta **Edgar F. Codd** introdujo la teoría de las relaciones en el campo de las bases de datos. De esta manera los datos se estructuran lógicamente en forma de relaciones. Los objetivos que buscaba Codd con el modelo relacional van encaminados a obtener: Independencia física, Independencia lógica, Flexibilidad, Uniformidad y Sencillez.

El modelo relacional representa la segunda generación de los SGBD. En él, todos los datos están estructurados a nivel lógico como tablas formadas por filas y columnas, aunque a nivel físico pueden tener una estructura completamente distinta. Un punto importante del modelo relacional es la sencillez de su estructura lógica.

El modelo relacional: Estructura

El modelo relacional se basa en el concepto matemático de relación, que se representa gráficamente mediante una tabla. La **estructura del Modelo Relacional** está formada por tres partes claramente diferenciadas:

- **La parte estructural:** Utiliza una estructura de datos muy sencilla, la relación.
- **La parte manipulativa**: Compuesta por **un conjunto de operadores** que permiten manejar la estructura anterior, **el Álgebra Relacional**
- **La Teoría de las Dependencias Funcionales y de la Normalización:** Compuesta por un conjunto de definiciones que permite estudiar las dependencias entre los atributos de las relaciones y proporciona métodos para un correcto diseño de las bases de datos relacionales.

El modelo relacional: Relaciones

La relación es el elemento básico del modelo relacional y está compuesta por dos partes:

- **Cabecera (intensión).** La cabecera está formada por un conjunto fijo de atributos. Es la parte invariante de la relación y se le denomina también **esquema de la relación**. Está constituida por: El nombre del conjunto (tabla), El nombre de los atributos (columnas de la tabla) y Los dominios de los que toman valores.
- **Cuerpo (extensión).** El cuerpo está formado por un conjunto de tuplas.

Podemos nombrar una relación con el nombre de tabla, la cual está compuesta por filas y columnas, donde cada fila (tupla) representa un conjunto de valores relacionados entre sí (hechos del mundo real), y las columnas (atributos) tienen la función de ayudar a interpretar el significado de los valores que están en cada fila de la tabla.

El número de columnas que tiene una relación recibe el nombre de **grado de la relación**, y el número de filas recibe el nombre de **cardinalidad de la relación**.

Una relación es una especie abstracta de objeto y una tabla es una representación concreta de ese objeto abstracto.

Estas tablas poseen ciertas propiedades, todas ellas consecuencia inmediata de la relación:

- **No existen tuplas repetidas:** en matemáticas por definición los conjuntos no incluyen elementos repetidos. Esto se traduce en que dos registros de una misma relación deben diferir, al menos, en el valor de un campo.
- **Las tuplas no están ordenadas**
- **Los atributos no están ordenados**
- **Todos los valores de los atributos son atómicos.** Es decir, un atributo sólo puede tomar un valor en cada tupla. La **tabla es plana**, es decir, en el cruce de un atributo y una tupla sólo puede haber un valor.

Una **Base de datos** es un conjunto de tablas entrelazadas entre sí a través de los campos con información común. En un SGBD relacional pueden existir varios tipos de relaciones, aunque no todos manejan todos los tipos.

- **Relaciones Base**. Son relaciones reales que tienen nombre y forman parte directa de la base de datos almacenada (son autónomas).
- **Vistas.** También denominadas relaciones virtuales, son relaciones con nombre y relaciones derivadas. Se representan mediante su definición en términos de otras relaciones con nombre y no poseen datos propios almacenados.
- **Relaciones Instantáneas.** Son relaciones con nombre y derivadas, pero a diferencia de las vistas, son reales, no virtuales. Están representadas no sólo por su definición en términos de otras relaciones con nombre, sino también por sus propios datos almacenados. Son relaciones sólo de lectura y se refrescan periódicamente.
- **Resultados de consultas**. Son las relaciones resultantes de alguna consulta especificada. Pueden o no tener nombre y no persisten en la base de datos.
- **Resultados intermedios**. Son las relaciones que contienen los resultados de las

subconsultas. Normalmente no tienen nombre y tampoco persisten en la base de datos.

- **Resultados temporales**. Son relaciones con nombre, similares a las relaciones base o a las instantáneas, pero la diferencia es que se destruyen automáticamente en algún momento apropiado.

Tablas, tuplas y columnas

Las columnas son los campos o atributos que definen la tabla. Cada **campo está definido por:**

- **Nombre:** que describe los datos almacenados en él.
- **Dominio:** que indica el tipo de valores que contendrá dicho campo. Es un conjunto finito de valores homogéneos (son todos del mismo tipo) y atómicos (son indivisibles en cuanto a lo que representan se refiere).

De esta manera la **cabecera** de una relación está constituida por un conjunto de "m" campos, y es expresada de la siguiente manera:

relacion (nombre_campo_1:dominio_1, ..., nombre_campo_m: dominio_m)

A esta expresión se le conoce también como **intensión de la relación.**

Las **tuplas** constituyen los registros de la tabla. Un **registro** es un conjunto de m valores, correspondientes a los m campos de la relación. Cabe destacar que todos los registros de una relación deben tener el mismo número de campos, aunque alguno esté vacío (se admite el valor NULO).

Clave primaria y claves ajenas

Los distintos tipos de claves que existen en el Modelo Relacional:

- **Clave Candidata**: Es un conjunto mínimo y no vacío de atributos que identifica unívocamente cada registro de una relación.
- **Clave Primaria**: Es la clave candidata que elige el usuario para identificar los registros de una relación. Se dice que una **clave primaria es compuesta cuando está formada por más de un atributo**. Se representa en el Modelo Relacional marcándola en *negrita y con subrayado continuo*.
- **Clave Alternativa**: Es cualquiera de las claves candidatas que no han sido elegidas como clave primaria.
- **Clave Ajena**: Es un conjunto no vacío de atributos de una relación R1 cuyos valores han de coincidir con los valores de la clave primaria de otra relación R2. R1 y R2 no tienen que ser necesariamente distintas (es el caso de las relaciones reflexivas). Se representa en el Modelo Relacional marcándola en *negrita y subrayado discontinuo*.

La integridad en el modelo relacional

Existen una serie de reglas, llamadas **reglas de integridad**, que al cumplirse nos garantizan que los datos almacenados en nuestra base de datos son correctos, es decir, son una serie de normas que mantienen la corrección semántica de la base de datos.

Detrás del concepto de relación hay una fuerte base matemática, lo que implica una serie de restricciones que son: No hay dos tuplas iguales, El orden de las tuplas no es significativo, El orden de los atributos (columnas) no es significativo y cada atributo sólo puede tomar un único valor del dominio, es decir, los atributos son atómicos y homogéneos

Es importante conocer el **atributo es NULO (null).** Un atributo es nulo cuando dicho atributo es desconocido. Un atributo nulo no se representa por el valor cero, ni por la cadena vacía, estos valores tienen significado. Se utiliza el valor null en un atributo cuando dicho atributo no tiene asociado ningún valor o que "su valor es desconocido". Null puede asignarse a atributos de cualquier tipo

Además tenemos que añadir dos reglas que completan el conjunto de **restricciones inherentes**:

- **Regla de integridad de entidad:** es el mecanismo que garantiza la identificación y unicidad de las tuplas en una relación. Ningún atributo que forme parte de la clave primaria de una relación puede tomar un valor nulo.
- **Regla de integridad referencial:** controla los casos de representación de interrelaciones (no tablas) en el universo del discurso, de modo que permite representar vínculos existentes entre tablas, evitando referencias no permitidas. Los valores de una clave ajena, o bien coinciden con los de la clave primaria a la que referencian o bien son nulos.

También tenemos las **restricciones semánticas** que son restricciones que dependen de la semántica del problema y sirven para reflejar de la forma más fiel posible el mundo real que se modela. Algunas son:

- **Clave primaria.** Permite declarar un atributo o un conjunto de atributos como clave primaria de una relación, por lo que sus valores no se podrán repetir ni se admitirán valores nulos.
- **Unicidad.** Mediante la cual se indica que los valores de un conjunto de atributos (uno o más) no pueden repetirse en una relación. Esta restricción permite la definición de claves candidatas.
- **Obligatoriedad** (NOT NULL) de uno o más atributos, indicando de esta manera que ese conjunto de atributos no admite valores nulos.
- **Verificación**, comprueba, en toda operación de actualización de la tabla, si el valor a introducir es correcto y en caso de que no lo sea, rechaza la operación. Una restricción de verificación se define sobre un único atributo y puede o no ser nombrada. Cada relación puede tener tantas restricciones de verificación como atributos tenga.
- **Disparador**, restricción en la que el usuario puede especificar libremente la respuesta del SGBD ante una determinada condición. Los disparadores son reglas de integridad procedimentales, siendo necesario que el usuario escriba el procedimiento que ha de aplicarse en caso de que se cumpla la condición.

La integridad ha de ser mantenida en todo momento por el sistema y se logra tal objetivo considerando las dos reglas de integridad más importantes:

- **Integridad de entidad**: Se debe comprobar que los atributos que forman parte de una clave primaria son no nulos y que el valor de dichos atributos no se repite en los procesos de **inserción** y **actualización**.
- **Integridad referencial**:
    - **En inserción**, comprobar que el valor de la clave ajena es nula o coincide con un valor existente
    - **En actualización**, si se actualiza la clave ajena, comprobar las condiciones
    - **En borrado**, si se borra la clave primaria, borrar en cascada o poner a null la clave ajena que hace referencia a dicha clave primaria

Transformación de diagramas E-R al Modelo Relacional

La transformación de un diagrama E/R al Modelo Relacional está basado en los siguientes principios:

- Toda entidad se convierte en una relación (tabla).
- Toda interrelación (relación) N:M se transforma en una relación (tabla)
- Toda interrelación (relación) 1:N se traduce en el fenómeno de "propagación de clave" (se crea una clave ajena)

Transformación de las entidades y sus atributos

- **Transformación de las Entidades.** Cada entidad que aparezca en el diagrama E/R se convierte en una tabla.
- **Transformación de Entidades Débiles.** Las entidades débiles se transforman en una tabla, propagando la clave de la entidad fuerte, que pasa a formar parte de la clave primaria de la entidad débil.
- **Transformación de los Atributos de las entidades.** Cada atributo de una entidad se transforma en una columna en la relación a la que ha dado lugar la entidad.
- **Transformación de Atributos Compuestos.** El modelo relacional no permite representar atributos compuestos, por lo que se busca una alternativa:
    - Considerar el atributo compuesto como un atributo simple.
    - Eliminar el atributo compuesto y considerar cada uno de sus atributos simple

Transformación de las relaciones y sus atributos

- **Transformación de las relaciones (interrelaciones):** Dependiendo del tipo de relación: N:M (se crea una nueva tabla), 1:N (se propaga o se crea una nueva tabla) y 1:1
- **Transformación de los atributos de relaciones.** Si la relación se transforma en una tabla, todos sus atributos pasan a ser columnas de la tabla.
- **Transformación de relaciones exclusivas.** Para soportar relaciones exclusivas debemos definir las restricciones pertinentes en cada caso.

Transformación en casos especiales

- **Transformación de relaciones ternarias (grado 3).**
    - Relaciones muchos a muchos a muchos: Este tipo de relación se transforma en una tabla cuya clave primaria es la concatenación de las claves primarias de las tablas surgidas al transformar las entidades que forman parte de la relación.
    - Relaciones muchos a muchos a uno: Este tipo de relación se transforma en una tabla cuya clave primaria es la concatenación de las claves primarias de las tablas que corresponden a la cardinalidad M surgidas al transformar las entidades que forman parte de la relación.
- **Transformación de la generalización (tipos y subtipos).** Los tipos y subtipos no son objetos que se puedan representar en el modelo relacional estándar. Existen pues, varias posibilidades: Crear una tabla para el supertipo, y tantas tablas como subtipos existan o Crear sólo tablas para los subtipos, añadiendo en cada una de ellas los atributos pertenecientes al supertipo.
- **Transformación de la agregación.** Se trata de transformar primero el nivel más alto de la agregación (aplicando las reglas adecuadas) y trataremos la relación resultante como si fuera "una nueva entidad a relacionar" con el nivel más bajo (aplicando las normas anteriores).

Las 12 reglas de Codd

Las doce reglas de Codd se han convertido en la definición teórica de una base de datos relacional.

- **Regla de información.** Toda la información de una base de datos relacional está representada explícitamente a nivel lógico y exactamente de un modo: Mediante valores en tablas.
- **Regla de acceso garantizado.** se garantiza que sean lógicamente accesibles recurriendo a una combinación de nombre de tabla, valor de clave primaria y nombre de columna
- **Tratamiento sistemático de valores nulo.** Los valores nulos se soportan en los SGBD relacionales para representar la falta de información y la información inaplicable
- **Catálogo en línea dinámico basado en el modelo relacional.** La descripción de la base de datos se representa a nivel lógico del mismo modo que los datos ordinarios, de modo que los usuarios autorizados puedan aplicar a su consulta el mismo lenguaje relacional que aplican a los datos regulares.
- **Regla de sublenguaje completo de datos.** Un sistema relacional puede soportar varios lenguajes, sin embargo, debe haber al menos un lenguaje cuyas sentencias sean expresables mediante alguna sintaxis bien definida
- **Regla de actualización de vista.** Todas las vistas que sean teóricamente actualizables son también actualizables por el sistema.
- **Inserción, actualización y supresión de alto nivel.** La capacidad de manejar una relación de base de datos o una relación derivada como un único operando se aplica, no solamente a la recuperación de datos, sino también a la inserción, actualización y supresión de los datos.
- **Independencia física de los datos.** Los programas de aplicación y las actividades terminales permanecen lógicamente inalterados cualquiera que sean los cambios efectuados, ya sea a las representaciones de almacenamiento o a los métodos de acceso.
- **Independencia lógica de los datos.** Los programas de aplicación y las actividades terminales permanecen lógicamente inalterados cuando se efectúen, sobre las tablas de base, cambios preservadores de la información de cualquier tipo que teóricamente permita alteraciones.
- **Independencia de integridad.** Las restricciones de integridad específicas para una base de datos relacional particular deben ser definibles en el sublenguaje de datos relacional y almacenables en el catálogo, no en los programas de aplicación.
- **Independencia de distribución**.
- **Regla de no subversión**. Si un sistema relacional tiene un lenguaje de bajo nivel, ese bajo nivel no puede ser utilizado para subvertir o suprimir las reglas de integridad y las restricciones expresadas en el lenguaje relacional de nivel superior

Álgebra relacional

Se define como la parte dinámica del Modelo Relacional y se basa en una serie de operadores matemáticos. Clasificación de operadores primitivos:

- **Operadores Unarios.** Son los operadores que sólo involucran una tabla. Son: **proyección (π)**, devuelve columnas completas; **selección (σ)**, devuelve filas completas
- **Operadores Binarios.** Son los operadores que involucran dos tablas. Son: **unión (U)**, tuplas que pertenezcan a R1, a R2 o a ambas; **diferencia (-)**, tuplas que pertenecen a R1 y no pertenecen a R2 y **producto cartesiano (X)** tuplas formadas concatenando cada tupla de la primera relación con cada una de las tuplas de la segunda relación.

Operadores Derivados: Estos operadores se obtienen como combinación de los operadores primitivos. Principalmente se consideran los tres siguientes, aunque existen algunos más: la combinación, la intersección y la división.

4. Normalización

Introducción

El diseño de una base de datos se puede realizar de dos maneras diferentes:

- Obteniendo el esquema relacional directamente
- Realizando el diseño en dos fases, la primer obteniendo el diagrama E/R y en la segunda fase transformando éste en un esquema relacional

El resultado final es un esquema relacional que debe verificar en todo momento un principio básico en todo diseño: **Hechos distintos se deben almacenar en objetos distintos.**

Para evitar anomalías y conseguir los objetivos aplica la **Teoría de la Normalización** que consiste en aplicar una serie de reglas para asegurarnos la eliminación de redundancias e inconsistencias en las tablas que constituyen el diseño de nuestra base de datos. Las reglas en las que se basa la Teoría de la Normalización son conocidas con el nombre de **Formas Normales**, que son un conjunto de restricciones sobre las tablas que evitan los problemas de redundancia y anomalías de modificación, inserción y borrado de datos.

Existen **seis formas normales.** Las tres primeras reglas de normalización fueron perfiladas por el Dr. E.F.Codd en su escrito de 1972, "Further Normalization of the Data Base Relational Model" (Referente a la normalización de las Bases de Datos Relacionales).

Ventajas y objetivos de la normalización

Las principales **ventajas** del uso de la normalización son las siguientes:

- **Facilidad de uso y claridad**, las tablas recogen los datos de manera clara y sencilla de entender incluso para usuarios poco expertos.
- **Flexibilidad y facilidad de gestión**, la información que necesitan los usuarios se puede obtener directamente de las tablas o mediante operaciones de álgebra y cálculo relacional.
- **Precisión**, las relaciones entre las tablas hacen que los datos de distintas tablas se relacionen con toda exactitud.
- **Mínima redundancia**, no se guarda información duplicada de manera innecesaria.
- **Máximo rendimiento de las aplicaciones**, sólo se utiliza la información que va a servir de utilidad a cada aplicación.

Los principales **objetivos** que busca un diseño normalizado son los siguientes:

- Eliminar anomalías de actualización, inserción y borrado.
- Conservar la información original (descomposición sin pérdida de información).
- Conservar las dependencias funcionales originales (descomposición sin pérdida de dependencias funcionales).
- No crear dependencias nuevas o relaciones inexistentes.
- Facilidad de uso y modificación de tablas creadas.
- Eficiencia.

¡¡A veces no es posible conseguir todos esos objetivos a la vez!!

Dependencias funcionales

Una relación se compone de un conjunto de atributos y dependencias, R (A, DF). Las dependencias funcionales son propiedades de la semántica de los atributos y no pueden obtenerse de manera automática en una relación determinada.

Las dependencias funcionales son:

- **Restricciones de integridad** establecidas por el usuario que permiten conocer las relaciones que existen entre los atributos del problema planteado.
- **Propiedades inherentes** a la semántica del problema.
- **Se han de cumplir para todos los registros de una relación.**

Resumiendo, las dependencias funcionales reflejan enlaces semánticos permanentes entre los datos en un diseño concreto.

Consideramos un esquema de relación general R(A), con A el conjunto de sus atributos, y consideramos X, Y, y Z subconjuntos de A llamados **descriptores**. Nos encontramos con que podemos definir los siguientes **tipos de dependencias**:

- **Dependencia funcional.** Se dice que un conjunto de atributos (Y) depende funcionalmente de otro conjunto de atributos (X), si para cada valor de X existe un único valor posible para Y. Se nota por X(Y. También se dice que X determina a Y o que X implica a Y. A X se le conoce como determinante o implicante y a Y se le conoce como implicado. Las dependencias funcionales vienen definidas por la semántica del problema.

  > *Por ejemplo, si consideramos la relación EMPLEADO (DNI, NOMBRE, DIRECCION, TELEFONO, DEPARTAMENTO) es evidente que el nombre de una persona depende funcionalmente del DNI, es decir, para un DNI determinado existe sólo un nombre que le corresponda. Sin embargo, si consideramos el ejemplo al revés, DNI no depende funcionalmente de NOMBRE*

- **Dependencia funcional plena o completa.** Si el descriptor X es compuesto, X(X1, X2), se dice que Y tiene dependencia funcional completa o plena de X, si depende funcionalmente de X, pero no depende funcionalmente de ningún subconjunto de X, es decir: Y depende funcionalmente de X, pero Y no depende funcionalmente de X1, ni tampoco de X2. Una dependencia funcional completa se denota como X⇒Y

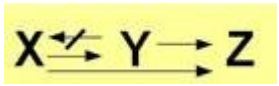
  > *Si consideramos la relación: PROYECTO (COD_PROYECTO, DNI_EMPLEADO, FECHA_INCORPORACION, HORAS_DEDICADAS) está claro que los atributos FECHA_INCORPORACION y HORAS_DEDICADAS tienen dependencia funcional completa respecto de COD_PROYECTO y DNI_EMPLEADO. Es evidente que la FECHA_INCORPORACION a un proyecto de un empleado depende tanto del proyecto al que se incorpora como del empleado que se incorpora.*

- **Dependencia funcional trivial.** Una dependencia funcional X --> Y se dice que es trivial si Y es un subconjunto de X (Y c X). No debe haber dependencias funcionales triviales en las relaciones de las bases de datos.

- **Dependencia funcional elemental.** Una dependencia funcional X-->Y es elemental si Y es un atributo único no incluido en X, y no existe ningún subconjunto de X, llamémosle X1, tal que X1 (Y, es decir, la dependencia funcional elemental es una dependencia completa no trivial en la que el implicado es un atributo único.

Si consideramos la relación PROYECTO (COD_PROYECTO, DNI_EMPLEADO, FECHA_INCORPORACION) con FECHA_INCORPORACION dependiendo funcionalmente de COD_PROYECTO y DNI_EMPLEADO, vemos claramente que esta dependencia es una dependencia funcional elemental ya que el implicado está constituido únicamente por el atributo FECHA_INCORPORACION.

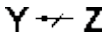
- **Dependencia funcional transitiva.** Consideramos la relación R(X, Y, Z) con las siguientes dependencias funcionales: Z depende funcionalmente de X y de Y, Y depende funcionalmente de X, pero X no depende funcionalmente de Y. O sea:



y se representa:

En estas condiciones se dice que Z tiene una dependencia funcional transitiva respecto a X, a través de Y.

- **Dependencia funcional transitiva estricta.** Una dependencia funcional transitiva se dice que es estricta si se verifica que



- **Dependencia funcional multivaluada.** Dada una relación R (X, Y, Z) con los atributos X, Y y Z, la dependencia multivaluada R.X -->--> R.Y se cumple en R si y sólo si el conjunto de valores de Y correspondiente a un par dado (valor de X, valor de Z) en R depende sólo del valor de X y es independiente del valor de Z.

Si consideramos la relación AGENDA (DNI, DIRECCION, FECHA_NACIMIENTO, TELEFONO) podemos observar que por cada DNI de la AGENDA puedo tener varios números de TELEFONO independientemente de los valores que tomen dirección y FECHA_NACIMIENTO. Podemos decir que TELEFONO es un atributo multivaluado y que existe la siguiente dependencia multivaluada DNI -->--> TELEFONO

Formas Normales 1FN, 2FN, 3FN y FNBC

Existen distintos "grados de normalización", que se van diferenciando entre sí, fundamentalmente por el grado de redundancia de la información que permiten. Mientras más alto sea el grado de normalización, menos redundancia de información habrá, y menos anomalías o problemas. Esos "grados de normalización" se conocen como **Formas Normales**.

1FN (Primera Forma Normal)

Una relación está en primera forma normal (1FN) si y sólo si todos sus atributos son atómicos (cada algunas tuplas tiene un sólo valor).

2FN (Segunda Forma Normal)

Una relación está en segunda forma normal (2FN) si y sólo si está en 1FN y todos los atributos no clave dependen por completo de la clave primaria.

Ejemplo:

Consideramos la siguiente relación:

ALUMNO (DNI, COD_ASIGNATURA, NOMBRE, APELLIDOS, NOTA, CURSO, AULA)

donde se aprecian las siguientes dependencias funcionales:

DNI --> NOMBRE, APELLIDOS

DNI, COD_ASIGNATURA --> NOTA

COD_ASIGNATURA --> CURSO, AULA

Dicha relación NO se encuentra en 2FN

Los **inconvenientes** son: redundancia de datos, anomalías en la inserción de tuplas, anomalías en el borrado de tuplas, anomalías en la actualización de tuplas, posible inconsistencia de datos en las actualizaciones e imposibilidad de almacenar ciertos datos

La **transformación** a 2FN se realiza de la siguiente manera:

Dada una Relación R con atributos X, Y, Z simples o compuestos, con la clave (X, Y) si en la relación existe una dependencia funcional incompleta Y --> Z (Z depende de parte de la clave), entonces:

- De R se elimina Z.
- Se construye una nueva relación R' con:
    - Z como atributo no clave.
    - Y como clave primaria de R'.

En nuestro caso concreto, la descomposición sería la siguiente:

ALUMNO (DNI, NOMBRE, APELLIDOS)

CALIFICACION (DNI, COD_ASIGNATURA, NOTA)

ASIGNATURA (COD_ASIGNATURA, CURSO, AULA)

3FN (Tercera Forma Normal)

Una relación está en tercera forma normal (3FN) si y sólo si esta en 2FN y todos los atributos no clave dependen de manera no transitiva de la clave primaria.

Ejemplo:

ASIGNATURA (COD_ASIGNATURA, CURSO, AULA)

En la que existen las siguientes dependencias funcionales:

COD_ASIGNATURA --> CURSO

CURSO --> AULA

La **descomposición** se realiza de la siguiente manera:

Dada una relación R con atributos X, Y, Z simples o compuestos, con la clave X y atributos no

claves Y, Z tal que en la relación existen las dependencias funcionales:

X --> Y

Y <-- Z

... es decir, una dependencia funcional transitiva X --> Z, entonces la relación R se descompone como sigue:

- De R se elimina Z (el atributo que tiene la dependencia funcional transitiva en la relación).
- Se construye una nueva relación R'con:
    - Z como atributo, pero que no sea clave principal.
    - Y como clave principal de R' .
    - Y --> Z como dependencia funcional.

El resultado sería:

ASIGNATURA (COD_ASIGNATURA, CURSO)

CURSO (CURSO, AULA)

FNBC (Forma Normal de Boyce-Codd)

Una relación está en forma normal Boyce/Codd (BCNF) si y sólo si todo determinante es una clave candidata.

Un determinante es un descriptor del que depende funcionalmente un atributo o conjunto de atributos de una relación.

En esta ocasión vuelven a aparecer **anomalías** de inserción, actualización y borrado en la Base de Datos

Por ejemplo:

Si consideramos la siguiente relación:

CALLEJERO (CALLE, CIUDAD, COD_POSTAL)

con las siguientes dependencias funcionales:

CALLE, CIUDAD --> COD_POSTAL

COD_POSTAL --> CIUDAD

COD_POSTAL es un determinante que no es clave candidata, por lo que la relación no está en FNBC.

La **descomposición** se realiza de forma genérica como sigue:

Dada una relación R con atributos X, Y, Z simples o compuestos, con clave {X, Y} y tal que en la relación existen las siguientes dependencias funcionales:

{X, Y} --> Z

Z --> Y

es decir, existe un determinante no clave, y por lo tanto la relación no está en forma normal de Boyce/Codd. Dicha relación R se descompone como sigue:

- En R se deja la parte de la clave que es independiente y todos los atributos no primarios, es decir, R (X, Z)
- Se crea otra tabla R' con la parte de la clave restante y el atributo secundario del que depende, siendo éste último la clave de la nueva tabla. Es decir, R' (Y, Z).

La solución:

CALLEJERO' (CALLE, COD_POSTAL)

CODIGO_POSTAL (COD_POSTAL, CIUDAD)

Otras Formas Normales

Alcanzar la 4FN o la 5FN es posible únicamente en un número muy reducido de casos, y siempre hay que observar con detenimiento que se conservan todas las dependencias funcionales que definen el sistema que estamos modelando.

Una relación R (X, Y, Z) está en **cuarta forma normal** si para cada dependencia multivaluada no trivial X -->--> Y, X es una clave primaria de R. Esto es equivalente a decir que R está en 4FN si está en FNBC y todas las dependencias multivaluadas en R son de hecho dependencias funcionales.

5. Lenguaje SQL

SQL - Structured Query Language (Lenguaje de Consulta Estructurado)

SQL es un lenguaje que nos permite interactuar con los SGBDRelacionales para especificar las operaciones que deseamos realizar sobre los datos y su estructura. Es un lenguaje declarativo, en él se especifica lo qué queremos obtener. Es un lenguaje no procedimental porque no necesitamos especificar el procedimiento para conseguir el objetivo, sino el objetivo en sí. No es un lenguaje de programación

En 1986 el ANSI (Instituto Americano de estándares) publicó la primera versión estándar del lenguaje, hoy conocido como SQL-86. Al año siguiente este estándar es adoptado por la ISO

SQL no es propiedad de ningún fabricante, sino que es una norma a seguir

¿Cómo usarlo?

Los SGBDR permiten dos modos de acceso a las bases de datos:

- Modo interactivo, destinado principalmente a los usuarios finales, avanzados u ocasionales, en el que las diversas sentencias SQL se escriben y ejecutan en línea de comandos, utilizando un intérprete de órdenes.
- Modo embebido, destinado al uso por parte de los programadores. En este caso las sentencias SQL se introducen en lenguajes de programación, llamados lenguajes anfitrión

Componentes del lenguaje SQL

El lenguaje SQL está compuesto por sentencias que se clasifican en tres grupos:

- **Sentencias DDL (Lenguaje de Definición de Datos):** Sirven para crear, modificar y borrar elementos estructurales en los SGBDR, como: bases de datos, tablas, índices, etc.
- **Sentencias DML (Lenguaje de Manipulación de Datos):** Nos permiten indicar al sistema las operaciones que queremos realizar con los datos almacenados en las estructuras creadas por medio de las sentencias DDL.
- **Sentencias DCL (Lenguaje de Control):** Un conjunto de sentencias orientado a gestionar todo lo relativo a: usuarios, permisos, seguridad, etc.

Las sentencias SQL a su vez se construyen a partir de:

- **Cláusulas:** Que modifican el comportamiento de las sentencias. Constan de palabras reservadas y alguno de los siguientes elementos.
- **Operadores lógicos y de comparación:** Sirven para ligar operandos y producir un resultado booleano (verdadero o falso).
- **Funciones de agregación:** Para realizar operaciones sobre un grupo de filas de una tabla.
- **Funciones:** Para realizar cálculos y operaciones de transformación sobre los datos.
- **Expresiones:** Construidas a partir de la combinación de operadores, funciones, literales y nombres de columna.

Lenguaje de Definición de Datos (DDL)

Cuando se crea una base de datos, en el SGBD ocurren muchos procesos internos que culminan con la asignación de espacio físico de almacenamiento para contener las estructuras de datos y acceso a los mismos de la base de datos, y también el registro en el diccionario de datos del SGBD de las características de la base de datos creada.

Para **crear una base de datos** en SQL:

CREATE DATABASE [IF NOT EXISTS] <nombre_bd>
[[DEFAULT] CHARACTER SET <juego_caracteres>]
[[DEFAULT] COLLATE <modo_ordenación>]

SHOW DATABASES

DROP DATABASE [IF EXISTS] <nombre_bd>

Para **crear una tabla** en SQL:

CREATE TABLE [IF NOT EXISTS] nombre_tabla
(definicion_elemento_de_tabla,...)

definicion_elemento_de_tabla:
definición_columna
| PRIMARY KEY (nombre_columna,...)
| FOREIGN KEY (nombre_columna,...) [definicion_referencia]

definicion_columna:
nombre_columna tipo_dato [NOT NULL | NULL] [DEFAULT valor_defecto]
[AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]

tipo_dato:
BIT[(longitud)]
| INT[(longitud)] [UNSIGNED] [ZEROFILL]
| INTEGER[(longitud)] [UNSIGNED] [ZEROFILL]
| REAL[(longitud,decimales)] [UNSIGNED] [ZEROFILL]
| DOUBLE[(longitud,decimales)] [UNSIGNED] [ZEROFILL]
| FLOAT[(longitud,decimales)] [UNSIGNED] [ZEROFILL]
| DECIMAL(longitud,decimales) [UNSIGNED] [ZEROFILL]
| NUMERIC(longitud,decimales) [UNSIGNED] [ZEROFILL]
| DATE | TIME | TIMESTAMP | DATETIME | YEAR
| CHAR(longitud) | VARCHAR(longitud)
| BINARY(longitud)
| BLOB
| TEXT [BINARY]
| ENUM(valor1,valor2,valor3,...)
| SET(valor1,valor2,valor3,...)

definicion_referencia:
REFERENCES nombre_tabla [(columna1,...)]
[ON DELETE opcion_referencia]
[ON UPDATE opcion_referencia]

opcion_referencia:
RESTRICT | *CASCADE* | *SET NULL* | *NO ACTION*

En MySQL, para que tengan efecto las definiciones de claves externas, debe utilizarse el modo de almacenamiento de tablas **InnoDB** (ENGINE=INNODB).

Los **FOREIGN KEY** funcionan de la siguiente manera: las columnas indicadas en FOREIGN KEY hacen referencia a las columnas indicadas en REFERENCES. Se puede indicar la acción a realizar cuando se modifican (ON UPDATE) o se borran (ON DELETE) los valores especificados en REFERENCES, esta acción puede ser: impedir, arrastrar el cambio, poner a nulo o no hacer nada (respectivamente: RESTRICT, CASCADE, SET NULL, NO ACTION).

Ejemplo:

```
FOREIGN KEY (cddep)
  REFERENCES departamento(cddep)
  ON UPDATE CASCADE ON DELETE RESTRICT
```

Lo que se interpreta como: La columna cddep de la tabla empleado hace referencia (FOREIGN KEY (cddep)) a la columna cddep de la tabla departamento (REFERENCES departamento(cddep)). No puede existir una fila en empleado que contenga un valor de cddep que no exista previamente en la tabla departamento. Si se modifica el cddep de una fila en la tabla departamento (ON UPDATE) el cambio se transmite automáticamente a todas las filas de la tabla empleado que tuviesen ese departamento (CASCADE). Si se intenta borrar una fila de la tabla departamento (ON DELETE) , la operación se cancelará si existen filas en la tabla de empleados con ese valor de departamento (RESTRICT).

Para modificar o borrar la estructura de una tabla:

ALTER TABLE <nombre_tabla>
DROP TABLE <nombre_tabla>

Los **índices** son unas estructuras gestionadas por el SGBDR para agilizar el acceso a datos y los cálculos. El uso de índices es gestionado automáticamente por el motor del SGBDR, en concreto por el optimizador de consultas, encargado de decidir en cada momento qué índice usar y cómo hacerlo. Cuando creamos una tabla y definimos columnas como PRIMARY KEY o utilizamos la cláusula FOREIGN KEY el SGBDR crea de forma automática los índices adecuados. También podemos crear índices de manera explícita utilizando la siguiente sintaxis:

CREATE INDEX <índice> ON <tabla>(<columna1>,<columna2>,...)

Por ejemplo:

```
CREATE INDEX empleado_nombre ON empleado (nombre)
```

A partir de ese momento el motor de datos del SGBDR usará el índice para agilizar las operaciones sobre la tabla empleado que impliquen a la columna nombre. Hay que hacer notar que en tablas pequeñas el uso de índices puede tener un impacto negativo en el rendimiento.

SHOW INDEX FROM <tabla>.
DROP INDEX <índice> ON <tabla>.

Lenguaje de Manipulación de Datos (DML)

Para **insertar** filas:

INSERT [INTO] <tabla> [(<columna>,...)]
VALUES ({<expresión> | DEFAULT},...)

Añadir filas a una tabla procedentes de otra tabla:

INSERT INTO trabaja (cdemp,cdpro)
SELECT cdemp,'AEE' FROM empleado WHERE
cddep='02'

Para **modificar** datos de una tabla:

UPDATE <tabla>
SET <columna_1>=<expresión_1> [, <columna_2>=<expresión_2> ...]
WHERE <criterio>]

Es posible especificar expresiones como nuevo valor de columnas. Ejemplo:

UPDATE trabaja SET nhoras=nhoras+5 WHERE cdpro='02'

Para **borrar** datos de una tabla:

DELETE FROM <tabla>
[WHERE <criterio>]

Para **hacer consultas** de datos de una tabla:

SELECT <lista_de_expresiones>
FROM <tabla>

La sentencia SELECT admite otras cláusulas que la hacen más potente y versátil. Estas son las siguientes:

- **WHERE:** va seguida de una condición lógica
  - Operadores de comparación: <, <=, =, >=, >, <>.
  - Operador de rango: BETWEEN ... AND
  - Operador de pertenencia a un conjunto: <expresión> IN (<conjunto>)
  - Operador de correspondencia con un patrón: <expresión> LIKE <patrón>. Para construir el patrón se utilizan caracteres comodín, los más usados son: "%" que se sustituye por cualquier secuencia de 0 o más caracteres, y "_" que se sustituye por un único carácter.
  - Condición de valor nulo o no nulo: <expresión> IS NULL y <expresión> IS NOT NULL
  - Operadores lógicos para enlazar más de un criterio: AND y OR.
  - Operador de negación lógico: NOT.

  Las expresiones lógicas pueden agruparse con paréntesis para indicar el orden de evaluación o aclarar las expresiones complicadas.
- **DISTINCT:** suprime los resultados repetidos de la consulta, de forma que se muestren sólo los resultados distintos, es decir, cada resultado aparecerá en la consulta una sola vez.
- **ORDER BY:** consegue un orden de filas dado en función del criterio. Se puede decidir si el orden será ascendente (ASC) o descendente (DESC).

- **GROUP BY:** las funciones de agregación. Son funciones que realizan cálculos sobre expresiones basadas en los datos de la tabla y resumen en un solo dato (numérico) el resultado.
    - Suma SUM(<expresión>) : Calcula la suma de los valores de expresión de cada fila.

        SELECT SUM(nhoras) FROM trabaja

    - Media aritmética AVG(<expresión>): AVG es la abreviatura de "Average", que significa media, calcula la media aritmética

        SELECT AVG(nhoras) FROM trabaja

    - Valor mínimo MIN(<expresión>): Produce como resultadoel mínimo
    - Valor máximo MAX(<expresión>): Obtiene el máximo
    - Cuenta del número de filas de una consulta COUNT(*)
    - Cuenta del número de filas que no producen el valor NULL COUNT(<expresión>)
    - Cuenta del número de filas distintas de una consulta COUNT(DISTINCT <expresión>): Análogo al anterior pero sin contar las filas repetidas.

  No se pueden combinar funciones de agregación con otro tipo de expresiones, esto provocaría un error de SQL. Aunque sí se puede combinar varias funciones de agrupamiento en la misma consulta. Es muy útil combinar estas funciones de agregación con la cláusula **GROUP BY**, de esta forma se pueden calcular subtotales de grupos de filas con alguna característica en común.

  SELECT cdpro, SUM(nhoras) FROM trabaja GROUP BY cdpro

  En caso de utilizar GROUP BY sí es posible mezclar expresiones simples con funciones de agregación, siempre que las expresiones simples formen parte de la cláusula ORDER BY. También es posible establecer varios niveles de agrupamiento.

- **HAVING:** la cláusula HAVING puede ser utilizada para seleccionar grupos de filas. El formato de la cláusula HAVING es análogo al de la cláusula WHERE, consistiendo en la palabra clave HAVING seguida del criterio lógico de selección.

  SELECT cdpro, SUM(nhoras) FROM trabaja GROUP BY cdpro HAVING SUM(nhoras)>0

- **LIMIT:** En ocasiones resulta muy necesario poder limitar el número de filas que devuelve una consulta.

  LIMIT <desplazamiento>,<número de filas>

- **Subconsulta:** En muchas ocasiones el criterio lógico para seleccionar las filas de una consulta en la cláusula WHERE viene dado por los resultados de otra consulta previa. A esta consulta previa se la denomina subconsulta o consulta anidada. Aparece dentro de la cláusula WHERE o HAVING de otra sentencia SQL.

  SELECT cdemp,nhoras FROM trabaja
  WHERE nhoras > (SELECT AVG(nhoras)FROM trabaja)

- En la cláusula HAVING las subconsultas sirven para seleccionar los grupos de filas del

resultado. Una subconsulta debe producir una única columna de datos como resultado. La cláusula ORDER BY no tiene sentido en una subconsulta. Es posible referirse a nombres de columnas de la consulta principal desde dentro de la subconsulta. Se pueden usar operadores de comparación (<, <=, =, >=, >, <>), operador de pertenencia a un conjunto (<expresión> IN (<subconsulta)), operador de existencia (EXIST(<subconsulta>)), es posible utilizar alias para los nombres de tabla en el caso de que la tabla de la consulta principal y de la subconsulta sean la misma, operadores de comparación cuantificada (<operador de comparación>[SOME|ALL])

Es muy habitual que los datos que queremos obtener por medio de una consulta se encuentren en tablas diferentes. En estos casos hay que combinar las tablas necesarias para construir una única consulta. Existen dos tipos de combinaciones de tablas, la unión (UNION) y la composición (JOIN).

La **unión de tablas** en una consulta produce como resultado la unión, en el sentido algebraico de la teoría de conjuntos, de las filas de ambas tablas, es decir, produce una nueva tabla que tiene todas las filas de la primera tabla y a continuación también todas las filas de la segunda tabla. Las filas de ambas tablas deben tener el mismo número de columnas y del mismo tipo.

```
(SELECT * FROM empleado WHERE cddep='04')
UNION
(SELECT * FROM empleado WHERE cdjefe='A11')
```

En la **composición de tablas** las filas de una tabla se concatenan a las filas de la otra tabla. A esto se le llama JOIN de tablas. Esto es el producto cartesiano

```
SELECT empleado.nombre,departamento.nombre
FROM empleado, departamento
WHERE empleado.cddep=departamento.cddep
```

Las consultas combinadas suelen relacionar tablas por medio de una columna común que es clave primaria en una tabla y externa en la otra. Siempre que exista ambigüedad será necesario utilizar la notación <tabla>.<columna>. Se pueden componer filas de una tabla con filas de esa misma tabla. En este caso deberemos emplear alias de tabla

Existen otros tipos de composición

- **Composición interna INNER JOIN:** Básicamente funciona igual que el producto cartesiano sólo que utiliza índices para agilizar el tiempo en obtener el resultado.

  ```
  SELECT <lista_de_expresiones>
      FROM <tabla1> INNER JOIN <tabla2> ON
      <expresión_composición>
  ```

  En donde <expresión_composición> es la combinación por medio de un operador de comparación (<, >, <=, >= ,=, <>) de dos columnas procedentes de cada tabla.

  ```
  SELECT a.nombre, b.nombre
      FROM empleado a INNER JOIN departamento b ON a.cddep=b.cddep
  ```

- **Composiciones externas LEFT JOIN y RIGHT JOIN:** En el caso de las composiciones producto cartesiano e INNER JOIN se concatenan filas de dos tablas que constan de una columna y valores comunes. Pero si alguna de las filas de las tablas a componer tienen valores nulos (NULL) en alguna de sus filas, o el valor que contienen no coincide con ninguno de los de la otra tabla, se pierden esas filas en la composición resultado, lo cual

puede hacer aparecer resultados erróneos. Para corregir este comportamiento se utilizan las composiciones LEFT JOIN y RIGHT JOIN. En ambos casos las filas con valores nulos en las columnas de composición, o con valores no coincidentes en ambas tablas de composición son tenidos en cuenta. LEFT JOIN añade al resultado del INNER JOIN las filas de la tabla izquierda que no tienen correspondencia en la tabla de la derecha, RIGHT JOIN hace lo propio con las filas de la tabla derecha.

Una **vista** no es más que una sentencia SQL almacenada que puede ser lanzada (ejecutada) sin tener que volver a escribirla, y que funcionalmente actúa cómo si fuera una tabla más.

CREATE VIEW <nombre_vista> (<columnas>) AS <sentencia_select>
DROP VIEW <nombre_vista>

Funciones SQL

SQL también dispone de una extensa gama de funciones que pueden ser utilizadas en las expresiones:

- Funciones de control de flujo.
  - IF(<expresión1>,<expresión2>,<expresión3>): Devuelve <expresión2> o <expresión3> en función de si <expresión1> es VERDAD o FALSO respectivamente.
  - IFNULL(<expresión1>,<expresión2>): Devuelve <expresión1> si <expresión1> es distinto de NULL, en otro caso devuelve <expresión2>.

- Funciones de caracteres
  - CHAR_LENGTH(<cadena>): Devuelve un entero que es la longitud de la cadena de caracteres.
  - CONCAT(<cadena1>,<cadena2>,...) : Concatena las cadenas de caracteres.
  - INSTR(<cadena>,<subcadena>): Devuelve la posición de la primera ocurrencia de la subcadena dentro de la cadena.
  - LOWER(<cadena>): Pasa la cadena a minúsculas.
  - UPPER(<cadena>): Pasa la cadena a mayúsculas.
  - RTRIM(<cadena>) y LTRIM(<cadena>) : Devuelve la cadena sin espacios en blanco a la derecha (RTRIM) o izquierda (LTRIM).
  - SUBSTR(<cadena>,<posición>,<longitud>): Devuelve la subcadena de longitud <longitud> a partir de la posición <posición> de la cadena <cadena>.

- Funciones numéricas
  - ABS(<número>) : Devuelve el valor absoluto de un número.
  - CEIL(<número>): Devuelve el entero más pequeño no menor que número.
  - MOD(<numero1>,<número2>): Devuelve el resto de la división entera entre <numero1> y <numero2>.
  - ROUND(<número>): Devuelve el redondeo al entero más cercano.
  - TRUNCATE(<número>,<Posiciones_decimales>): Devuelve el número con las posiciones decimales dadas sin redondeo.

- Funciones de fecha y hora
  - ADDDATE(<fecha>,<numero_dias>): Devuelve la fecha proporcionada, incrementada en el número de días indicado.
  - CURDATE(): Devuelve la fecha actual del sistema.
  - CURTIME(): Devuelve la hora actual del sistema.
  - DATEDIFF(<fecha1>,<fecha2>): Devuelve el número de días entre las dos fechas.

        - DATE_FORMAT(<fecha>,<formato>): Devuelve la fecha formateada según el formato especificado. Ejemplo: SELECT DATE_FORMAT('2005-04-19', '%d %m %Y')
        - DAY(<fecha>): Devuelve el día del mes de una fecha.
        - MONTH(<fecha>): Devuelve el mes de un fecha.
        - NOW(): Devuelve la fecha y hora del sistema.
        - YEAR(<fecha>): Devuelve el año de un fecha.

- Funciones de conversión:
    - CONVERT(<expresion>,<tipo_dato>): Devuelve la expresión convertida al tipo de dato suministrado.
    - CAST(<expresión> AS <tipo_dato>): Análoga a la anterior.

- Otras funciones:
    - PASSWORD(<cadena>): Devuelve la cadena encriptada en el formato de encriptación usado por MySQL.
    - MD5(<cadena>): Análoga a la anterior utilizando el algoritmo de encriptación MD5.

Sentencias de Control (DCL)

El Administrador de Base de Datos es una persona que se encarga de gestionar todo lo relativo a organización y seguridad del SGBD para dar servicio al resto de usuarios. El administrador del SGBD dispone para llevar a cabo su tarea de un conjunto de sentencias SQL muy específicas que gestionan todo lo relativo a usuarios y permisos. Las dos sentencias más importantes son **GRANT** para conceder permisos y **REVOKE** para quitarlos.

GRANT <tipo_privilegio> ON <base_de_datos>.<tabla> TO <usuario>
[IDENTIFIED BY '<password>']

Si el usuario no existe se crea en ese momento y se le asigna el password dado. Para poder conceder (GRANT) permisos hay que ser el propietario de la tabla o tener el permiso adecuado sobre ella. Se pueden utilizar el comodín (*) para indicar las bases de datos y las tablas. Si queremos conceder todos los permisos posibles podemos utilizar ALL como tipo de privilegio.

Para quitar permisos concedidos a un usuario utilizaremos la sentencia REVOKE.

REVOKE <tipo_privilegio> ON <base_de_datos>.<tabla> FROM <usuario>

Transacciones

Existen ocasiones en las que interesa agrupar una serie de sentencias SQL para que se lleven a cabo como un todo, de forma que se ejecuten todas, o no se ejecute ninguna. A esa agrupación de sentencias SQL se le llama transacción, y cuando termina es posible grabarla de forma efectiva (COMMIT) o deshacerla (ROLLBACK).

Para iniciar una transacción se utiliza START TRANSACTION. A continuación podemos iniciar la secuencia de sentencias SQL para terminar grabando con COMMIT o deshaciendo con ROLLBACK.

6. Diseño de Base de Datos relacionales

Introducción

Empezamos aplicando el modelo Entidad-Relaciónpara obtener un modelo lógico de datos, que posteriormente se traducía a un esquema de tablas siguiendo el modelo relacional, que a su vez eran normalizadas, y que por último se implementaban en el sistema gestor de bases de datos por medio de un lenguaje estándar cómo es SQL.

Un sistema CASE (Computer Aided Software Engineering) es una herramienta informática que permite a los desarrolladores de software concentrarse en las tareas que les son propias, aumentando su productividad y mejorando la calidad de su trabajo. El objetivo máximo que persiguen es la generación automática de programas de ordenador a partir de las especificaciones de diseño.

Diseño de BD usando herramientas CASE de modelado

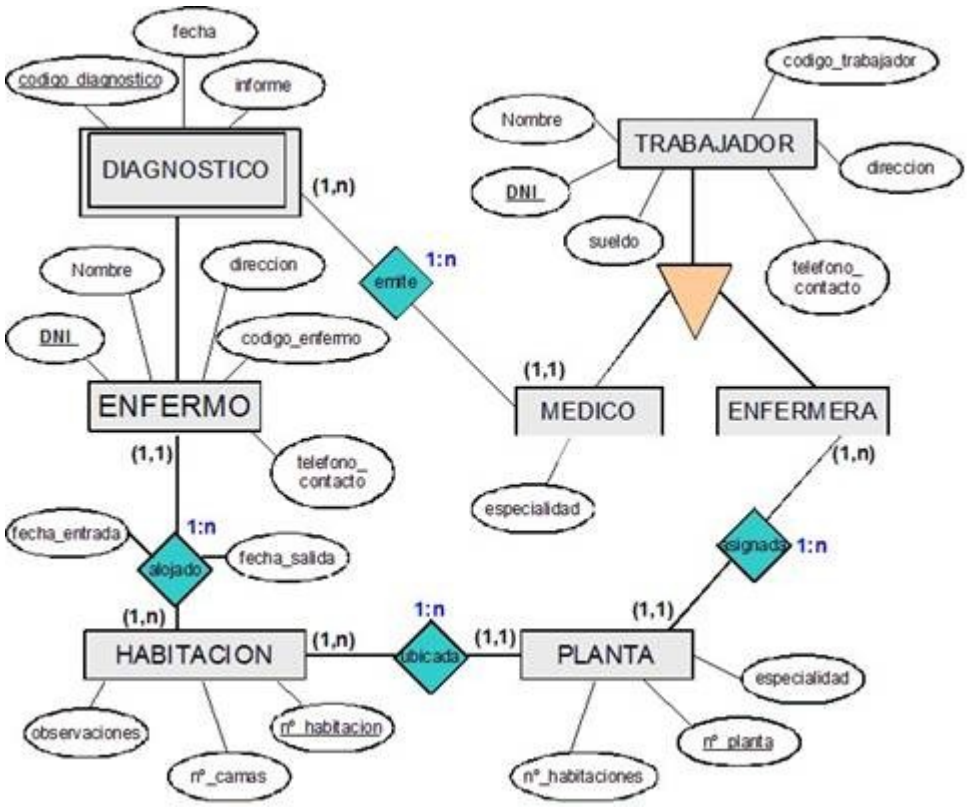
Se puede decir que todas comparten las siguientes características:

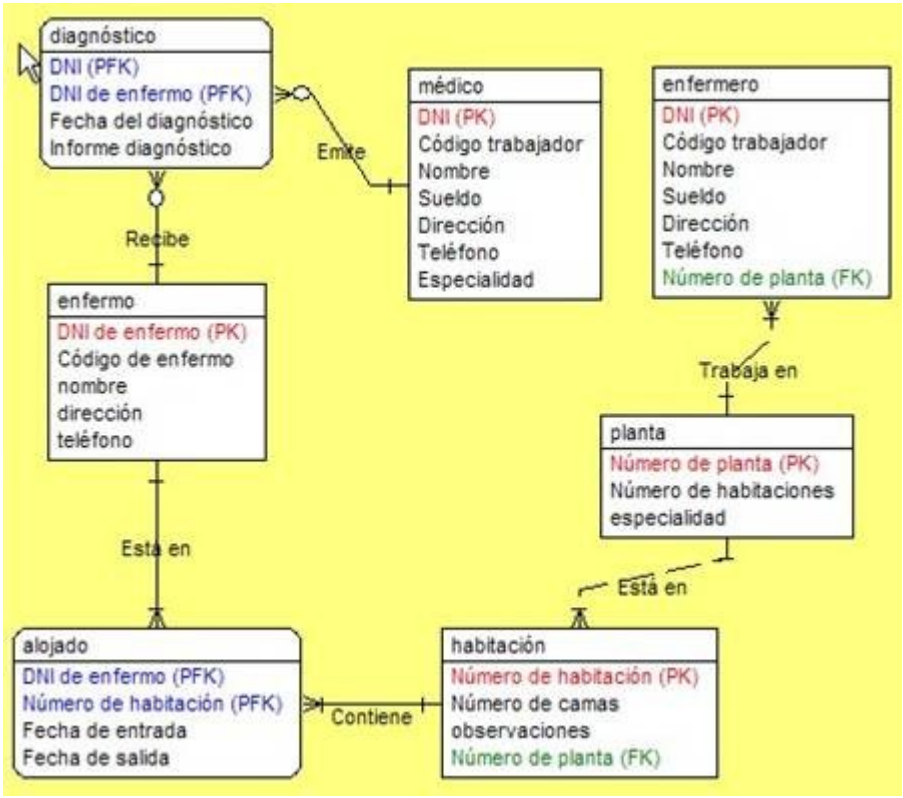
- Implementación de las reglas de modelado según los modelos E/R y relacional.
- Gestión de un diccionario de datos donde se almacenan los elementos creados para el diseño de la base de datos. Permite crear y mantener nuestros propios tipos de datos
- Comunicación con uno o más sistemas de gestión de base de datos para exportar de forma cómoda el modelo conceptual creado. En ocasiones se puede realizar ingeniería inversa para importar el modelo conceptual a partir del esquema físico existente en la base de datos real.
- Comprobación y optimización del modelo diseñado.
- Generación automática de la documentación relativa al diseño realizado.
- Interfaz de usuario cómoda y visual para presentar de forma clara el diseño de la base de datos.

La principal **ventaja** de utilizar estas herramientas es conseguir la mayor productividad posible en el proceso, reduciendo los costes de esfuerzo y tiempo de diseño. Este objetivo se consigue a través de diversos frentes.

- Facilitan la aplicación práctica de metodologías de diseño estructuradas (como el modelo E/R), para agilizar y sistematizar el trabajo de los diseñadores de aplicaciones.
- Permiten el rápido desarrollo de prototipos de aplicaciones, sobre todo en proyectos de gran envergadura.
- Simplifican enormemente el mantenimiento ordenado y estructurado de las aplicaciones, y en concreto de las bases de datos utilizadas.
- Ayudan a los diseñadores en la producción y estandarización de la documentación de las bases de datos.
- Aumentan la portabilidad entre sistemas de las bases de datos diseñadas.
- Aseguran la reutilización de componentes software
- Representan una ayuda inestimable en la planificación de bases de datos

Ejemplo:


fecha
codigo_diagnostico
informe
DIAGNOSTICO
(1,n)
emite
1:n
Nombre
DNI
direccion
codigo_enfermo
ENFERMO
(1,1)
telefono_
contacto
fecha_entrada
1:n
alojado
fecha_salida
(1,n)
HABITACION
(1,n)
observaciones
nº_habitacion
nº_camas
codigo_trabajador
Nombre
TRABAJADOR
direccion
DNI
sueldo
telefono_
contacto
(1,1)
MEDICO
ENFERMERA
especialidad
(1,n)
asignada
1:n
1:n
ubicada
(1,1)
(1,1)
PLANTA
especialidad
nº_planta
nº_habitaciones


diagnóstico
DNI (PFK)
DNI de enfermo (PFK)
Fecha del diagnóstico
Informe diagnóstico
Emite
médico
DNI (PK)
Código trabajador
Nombre
Sueldo
Dirección
Teléfono
Especialidad
enfermero
DNI (PK)
Código trabajador
Nombre
Sueldo
Dirección
Teléfono
Número de planta (FK)
Recibe
enfermo
DNI de enfermo (PK)
Código de enfermo
nombre
dirección
teléfono
Trabaja en
planta
Número de planta (PK)
Número de habitaciones
especialidad
Está en
Está en
alojado
DNI de enfermo (PFK)
Número de habitación (PFK)
Fecha de entrada
Fecha de salida
Contiene
habitación
Número de habitación (PK)
Número de camas
observaciones
Número de planta (FK)

7. Sistemas de gestión de bases de datos relacionales

Características de un SGBD Relacionales (SGBDR)

Recordemos que en 1982 Codd publicó un artículo en el que establecía que para que un SGBD pueda ser considerado relacional debe poseer las dos características siguientes:

- Los usuarios deben percibir las bases de datos gestionadas como tablas, y nada más que como tablas. Se asegura que el sistema implementa el modelo relacional basado en tablas
- El SGBD debe manejar las operaciones de manipulación de datos sin requerir definiciones previas de rutas de acceso físico a los datos. Dota al sistema de la necesaria independencia entre la estructura lógica de los datos y su almacenamiento físico.

Funciones de los SGBD

Además de las 12 reglas de Cood, un SGBD debe proporcionar a los usuarios una serie de funciones y servicios para cubrir las necesidades de uso del sistema. A continuación te presentamos las funciones que el propio Codd estableció como necesarias:

- El SGBD debe proporcionar a los usuarios la capacidad de almacenar datos en la base de datos, acceder a ellos y actualizarlos, y esto debe hacerse ocultando al usuario la estructura física interna de la base de datos.
- El SGBD debe proporcionar un catálogo en el que se almacene la estructura de la base de datos y que sea accesible por los usuarios con el suficiente nivel de acceso. Este catálogo es lo que se denomina diccionario de datos. Este almacena:
    - Nombre, tipo y tamaño de los datos.
    - Relaciones entre los datos.
    - Restricciones de integridad sobre los datos.
    - Información sobre los usuarios autorizados a acceder a la base de datos y sus niveles de acceso.
    - Estadísticas de utilización, tales como la frecuencia de las transacciones y el número de accesos realizados a los objetos de la base de datos
- El SGBD debe proporcionar un mecanismo que garantice que las actualizaciones sobre los datos (inserción, borrado y modificación) correspondientes a una determinada transacción se realicen todas, o no se realice ninguna.
- El SGBD debe asegurar la utilización simultánea de la base de datos por varios usuarios al mismo tiempo (concurrencia)
- El SGBD debe incorporar los mecanismos necesarios para recuperar la base de datos en el caso de que ocurra algún suceso que la dañe.
- El SGBD debe garantizar que sólo los usuarios autorizados puedan acceder a la base de datos, y además debe incorporar algún sistema que permita establecer permisos y niveles de acceso diferentes
- El SGBD debe ser capaz de integrarse con software de comunicaciones para que usuarios alejados físicamente de la instalación puedan acceder a ella.
- El SGBD debe proporcionar los medios necesarios para garantizar que tanto los datos de la base de datos, como los cambios que se realizan sobre estos datos, sigan ciertas reglas
- El SGBD debe facilitar la independencia entre la estructura de la base de datos y las aplicaciones que acceden a ella. Se debería poder alterar la estructura física Manossin por ello tener que cambiar el software de aplicación.
- El SGBD debe proporcionar una serie de herramientas que permitan administrar la base de datos de modo efectivo y versátil.

- Herramientas para importar y exportar datos
- Herramientas para monitorizar el uso y el funcionamiento de la base de datos
- Programas de análisis estadístico
- Herramientas para optimizar el espacio
- Aplicaciones de gestión y acceso que faciliten sus tareas a administradores, programadores y usuarios en general.
- Sistema de copias de seguridad.

Componentes de un SGBD

Los SGBD son aplicaciones informáticas muy complejas con un elevado número de componentes. No es posible generalizar esos componentes puesto que cada fabricante diseña y desarrolla su producto de forma diferente, pero sí que es posible definir de manera general algunas de las partes que suelen estar presentes con frecuencia en la mayoría de los actuales SGBD.

Los más comunes son:

- **El procesador de consultas.** Es el encargado de transformar las consultas proporcionadas por los usuarios en el lenguaje (normalmente SQL) de definición (DDL) y manipulación (DML), en un conjunto de instrucciones de bajo nivel para obtener los resultados solicitados.
- **El gestor de ficheros.** Es el encargado de manejar los ficheros en disco en donde se almacena la base de datos, es decir el almacenamiento físico.
- **El gestor de la base de datos.** Es el interface entre los programas de aplicación que acceden a la base de datos y la base de datos en sí. El gestor de la base de datos acepta consultas y examina los esquemas externo y conceptual para determinar qué filas son requeridas y que operaciones hay que realizar con ellas.

Los principales componentes del gestor de la base de datos son los siguientes:

- **Control de autorización.** Este módulo comprueba que el usuario tiene los permisos necesarios para llevar a cabo la operación que solicita.
- **Procesador de comandos.** Una vez que el sistema ha comprobado los permisos del usuario, se pasa el control al procesador de comandos.
- **Control de la integridad.** Cuando una operación cambia los datos de la base de datos, este módulo debe comprobar que la operación a realizar satisface todas las restricciones de integridad necesarias.
- **Optimizador de consultas.** Este módulo determina la estrategia óptima para la ejecución de las consultas con el menor costo de tiempo y esfuerzo posible.
- **Gestor de transacciones.** Este módulo realiza el procesamiento de las transacciones.
- **Planificador (scheduler).** Este módulo es el responsable de asegurar que las operaciones que se realizan concurrentemente sobre la base de datos tienen lugar sin conflictos.
- **Gestor de recuperación.** Este módulo garantiza que la base de datos permanece en un estado consistente en caso de que se produzca algún fallo.
- **Gestor de buffers.** Este módulo es el responsable de transferir los datos entre memoria principal y los dispositivos de almacenamiento secundario.

Lenguajes de los SGBD

La herramienta fundamental para gestionar, configurar, acceder y manipular una base de datos son los lenguajes de los SGBD.

Los **lenguajes de cuarta generación (4GL)** son aquellos lenguajes en los que el programador define qué debe hacerse, y no cómo debe hacerse. Son lenguajes no procedurales, puesto que no indican el procedimiento para conseguir el objetivo, sino el objetivo en sí. SQL puede ser considerado un lenguaje de 4GL al no ser procedural, pero habitualmente los lenguajes de 4GL se presentan en forma de herramientas que permiten abordar los problemas a resolver por medio de asistentes en los que el usuario va definiendo los detalles que llevarán al resultado final. Mejoran de la productividad de los programadores, pero por contra adolecen de esquemas de trabajo más o menos rígidos

Las herramientas 4GL más comunes son:

- **Generador de formularios.** Es una herramienta interactiva que permite crear rápidamente formularios de pantalla para introducir o visualizar datos. Los generadores de formularios permiten que el usuario defina el aspecto de la pantalla, qué información se debe visualizar y en qué lugar de la pantalla debe visualizarse.
- **Generador de informes.** Es una herramienta para crear informes a partir de los datos almacenados en la base de datos. Se parece a un lenguaje de consultas en que permite al usuario hacer preguntas sobre la base de datos y obtener información de ella para un informe. Tiene un mayor control sobre el aspecto de la salida.
- **Generador de aplicaciones:** Es una herramienta para crear programas que hagan de interface entre el usuario y la base de datos. Puede reducir el tiempo que se necesita para diseñar un programa de aplicación. El usuario especifica qué debe hacer el programa y el generador de aplicaciones es quien determina cómo realizar la tarea.

SGBDR Oracle

Oracle Express se ejecuta en servidores de base de datos con un solo procesador y hasta 4 Gb de almacenamiento.

Toda la estructura de seguridad se encuentra almacenada en tablas del diccionario de datos, propiedad del usuario SYSTEM (DBA). En Oracle existen varios **niveles de seguridad**:

- **Seguridad de cuentas,** para la validación de usuarios. Para acceder a los datos de una base de datos Oracle: se debe tener acceso a una cuenta de usuario en esa base datos con contraseña. La base de datos almacena una versión encriptada de la contraseña en una tabla del diccionario de datos llamada dba_users.

- **Seguridad en el acceso** a los objetos de la base de datos. El acceso a los objetos de la base de datos (tablas, vistas, secuencias, disparadores, procedimientos, etc) se realiza vía privilegios que establecen que comandos son permitidos según el usuario. Se pueden especificar privilegios de forma individual con el comando GRANT y retirarlos utilizando el comando REVOKE. Los privilegios se pueden agrupar formando lo que se conoce por **roles**. Los roles pueden ser protegidos con contraseñas, y pueden activarse y desactivarse dinámicamente

- **Seguridad a nivel de sistema**, para la gestión de privilegios globales. Los **roles de sistema** se pueden utilizar para gestionar de forma sencilla los privilegios asignados a las cuentas de usuario. Se pueden agrupar varios privilegios bajo el mismo rol y asignar este rol a una cuenta de usuario de esta forma se puede aligerar el trabajo del DBA. Si se hace algún cambio en el rol se propagará a todos los usuarios.

  Existen varios roles prediseñados por el sistema, todos los usuarios que quieran acceder a la base de datos deben tener el rol CONNECT; aquellos que tengan que crear objetos

necesitarán el rol RESOURCE; y los usuario con rol DBA tendrán privilegios de administrador. Es posible crear nuevos roles, pero hay que hacerlo utilizando comandos SQL, puesto que no está disponible esta opción en las páginas de administración de Oracle Express vía web.

Los privilegios que pueden otorgarse sobre objetos son los siguientes:

- SELECT: Puede consultar a un objeto.
- INSERT: Puede insertar filas en una tabla o vista. Puede especificarse las columnas donde se permite insertar dentro de la tabla o vista.
- UPDATE: Puede actualizar filas en una tabla o vista. Puede especificarse las columnas donde se permite actualizar dentro de la tabla o vista.
- DELETE: Puede borrar filas dentro de la tabla o vista.
- ALTER: Puede alterar la tabla.
- INDEX: Puede crear índices de una tabla.
- REFERENCES: Puede crear claves ajenas que referencien a esta tabla.
- EXECUTE: Puede ejecutar un procedimiento, paquete o función.

El sistema de **copias de seguridad** más sencillo consiste en utilizar los accesos directos incluidos en el grupo de programas de Oracle Express "Realizar Copia" y "Restaurar Copia". Se puede refinar el proceso utilizando la herramienta **RMAN de Oracle**. Ésta es una aplicación especializada en el proceso de copias de seguridad en bases de datos Oracle.

Una **instancia de base de datos** es el conjunto de procesos del servidor Oracle que tienen su propio área global de memoria y una base de datos asociada a ellos.

Los parámetros que determinan el tamaño y composición de una instancia están almacenados en un fichero llamado **init.ora.** Este fichero es leído durante el arranque de la base de datos y puede ser modificado por el DBA.

La instancia se compone del Área Global del Sistema (SGA) y el Área Global de Programas (PGA).

- **Área Global del Sistema (SGA):** La SGA es un área de memoria compartida que se utiliza para almacenar información de control y de datos de la instancia. Se crea cuando la instancia es arrancada y se borra cuando ésta se deja de usar (cuando se cierra la instancia). La información que se almacena en esta área consta de los siguientes elementos:
  - Buffer de caché (database buffer cache): Almacena los bloques de datos utilizados recientemente. Se reducen las operaciones de entrada/salida y mejora el rendimiento del sistema.
  - Buffer de redo log: Guarda los cambios efectuados en la base de datos. El archivo redo log se utiliza para recuperar la base de datos ante eventuales fallos del sistema.
  - El área shared pool: Esta área almacena estructuras de memoria compartida, tales como las áreas de código SQL compartido e información interna del diccionario.

  La instancia tiene una serie de procesos asociados, con las siguientes funciones:
  - DBWR (database writer): Es el responsable de la escritura en disco de toda la información almacenada en los buffers.
  - LGWR (log writer): Es el responsable de escribir información desde el buffer de log hacia el archivo redo log.
  - PMON (process monitor): Su misión es monitorizar los procesos del servidor y tomar acciones correctoras cuando alguno de ellos se interrumpe de forma abrupta, limpiando la caché y liberando los posibles recursos

    - SMON (system monitor):Arranca la instancia cuando se le da la instrucción de partida.

- **Área Global de Programas (PGA):** La PGA es un área de memoria que contiene datos e información de control para los procesos que se ejecutan en el servidor de Oracle. El tamaño y contenido de la PGA depende de las opciones del servidor que se hayan instalado.

La base de datos de Oracle tiene una capa lógica y otra física. La **capa física** son los archivos que residen en el disco, y los componentes de la **capa lógica** son estructuras que mapean los datos lógicos hacia estos componentes físicos. En el fondo, debajo de todas las capas lógicas que conforman una base de datos Oracle nos encontramos archivos físicos del sistema operativo.

Una base de datos Oracle se compone de **varios archivos físicos**:

- Uno o más datafiles. Los datafiles almacenan toda la información de la base de datos.
- Dos o más archivos redo log. En ellos se almacenan las operaciones efectuadas sobre la base de datos
- Uno o más control files. Estos archivos contienen información que se utiliza cuando se arranca una instancia, tal como la información de dónde se encuentran ubicados los datafiles y los archivos redo log. Estos archivos de control deben encontrarse siempre protegidos.

Los **elementos de la capa lógica** son:

- Uno o más tablespaces. Un tablespace es una unidad lógica de almacenamiento que está constituido por uno o más datafiles, que son los archivos físicos que están de forma efectiva en el disco duro. Cuando se crea una base de datos, hay que crear al menos un tablespace
- El esquema de la base de datos o conjunto de objetos de un usuario (schema). Consiste en una colección de objetos que conforman el modelo lógico relacional que se desea implementar. En el esquema puede haber:
    - Tablas: Son la unidad lógica básica de almacenamiento.
    - Índices: Son estructuras creadas para ayudar a recuperar datos de una manera más rápida y eficiente.
    - Vistas: Las vistas reúnen una selección de varias columnas de una o diferentes tablas. Una vista no almacena datos; sólo los presenta en forma dinámica.
    - Procedimientos y funciones almacenados. Son programas escritos en PL/SQL que permiten independizar el manejo de datos desde una aplicación y efectuarla directamente desde el motor de base de datos,
    - Paquetes (Packages): Son agrupaciones de procedimientos y funciones.
    - Disparadores (Triggers): Un trigger es un procedimiento que se ejecuta en forma inmediata cuando ocurre un evento especial.
    - Secuencias: El generador de secuencias de Oracle se utiliza para generar números únicos y utilizarlos, por ejemplo, como claves de tablas. La principal ventaja es que libera al programador de la tarea de obtener números secuencialesque no se repitan
    - Sinónimos: Para identificar completamente un objeto dentro de una BD se necesita especificar el nombre de la máquina, el nombre del servidor, el nombre del propietario y el nombre del objeto

8. Diseño y desarrollo de aplicaciones cliente-servidor

Características de la programación orientada a entornos cliente-servidor

Las características que distinguen a la arquitectura cliente-servidor de otros entornos son:

- El término cliente-servidor apareció a principios de los años 80 haciendo **referencia a los ordenadores personales (PC´s) en una red**. El modelo cliente-servidor comenzó a ganar aceptación a finales de los 80 hasta llegar a ser hoy en día el modelo de arquitectura más extendido

- La arquitectura cliente-servidor, llamada modelo cliente-servidor o servidor-cliente, es una forma de dividir y especializar programas y equipos de cómputo a fin de que la tarea que cada uno de ellos realiza se efectúe con la **mayor eficiencia** y permita simplificarla.

- La arquitectura del software del cliente-servidor es una infraestructura versátil, y modular que se desarrolla para **mejorar la utilidad, flexibilidad, interoperabilidad, y ser escalable** con respecto a la arquitectura centralizada o la arquitectura de servidor de archivos

- La arquitectura cliente-servidor, el objetivo que se pretende alcanzar es que **la capacidad de proceso esté repartida entre el servidor y los clientes**.
    - El cliente es quien recibe los servicios que ofrece un servidor. El término se usó inicialmente para dispositivos que no eran capaces de ejecutar programas por sí mismos, pero podían interactuar con ordenadores remotos por red
    - El servidor es una aplicación informática o programa que realiza algunas tareas en beneficio de otras aplicaciones llamadas clientes.

Tipos de arquitectura

Los distintos tipos de arquitectura son:

- **Arquitectura centralizada:** Este modelo de arquitectura se basa en la existencia de una máquina servidora que almacena los datos y las aplicaciones que los procesan. En este modelo de arquitectura, los clientes se comportan como terminales y sólo sirven para introducir datos desde teclado.

    - Ventajas: gran nivel de seguridad y facilidad de administración.
    - Desventajas: alto coste y máquina servidora sobrecargada

- **Arquitectura de servidor de archivos:** Este modelo de arquitectura se basa en una o más máquinas servidoras que almacenan datos y estaciones de trabajo que ejecutan aplicaciones. Los clientes son activos.

    - Ventajas: bajo coste y escalable
    - Desventajas: alta dependencia de las comunicaciones y alto tráfico de red

    Cuando el equipo n lanza una consulta al fichero A (por ejemplo: select * from empleados where departamento='contabilidad') el servidor de ficheros no devuelve el resultado de la consulta, sino el fichero A completo, y es en el equipo n donde se ejecuta la consulta sobre el fichero A.

- **Arquitectura cliente-servidor**: Este modelo de arquitectura se basa en la existencia de dos tipos de aplicaciones ejecutándose independientemente: una de las aplicaciones actúa como servidora y otra como cliente. El trabajo se reparte entre dos ordenadores. De acuerdo con la distribución de la lógica de la aplicación hay dos posibilidades: **cliente ligero (o cliente fino)** si el cliente sólo se hace cargo de la presentación (capa de presentación) y **cliente pesado (o cliente grueso)** si el cliente asume también la lógica del negocio.

    - Ventajas: el servidor no necesita tanta potencia de procesamiento, parte del proceso se reparte con los clientes y se reduce el tráfico de red considerablemente. Idealmente, el cliente se conecta al servidor cuando es necesario obtiene los datos y cierra la conexión dejando la red libre para otra conexión.
    - Desventajas: alta dependencia de las comunicaciones y instalación de nuevas aplicaciones (una nueva aplicación debe ser instalada en cada uno de los clientes)

  Cuando el cliente n lanza una consulta sobre la base de datos (por ejemplo: select * from empleados where departamento='contabilidad') el SGBD devuelve únicamente el resultado de la consulta siendo recibido el mismo por el cliente.

  La **arquitectura n-capas o n-tier** surge como evolución de la arquitectura cliente-servidor. Podemos afirmar que es una especialización de la arquitectura cliente-servidor. En la arquitectura cliente-servidor el software reparte su carga de cómputo en dos partes independientes pero sin reparto claro de funciones mientras que hablaremos de arquitectura n-capas cuando exista un reparto claro de funciones. El modelo n-capas está especialmente indicado para el desarrollo en paralelo.

  El **término capa** hace referencia a la forma como una solución es segmentada desde el punto de vista lógico: Presentación/ Lógica de Negocio/ Datos. El **término nivel**, corresponde a la forma en que las capas lógicas se encuentran distribuidas físicamente.
    - Una solución de tres capas (presentación, lógica, datos) que residen en un solo ordenador (Presentación+lógica+datos). Se dice, que la arquitectura de la solución es de tres capas y un nivel.
    - Una solución de tres capas (presentación, lógica, datos) que residen en dos ordenadores (presentación+lógica, lógica+datos). Se dice que la arquitectura de la solución es de tres capas y dos niveles.
    - Una solución de tres capas (presentación, lógica, datos) que residen en tres ordenadores (presentación, lógica, datos). La arquitectura que la define es: solución de tres capas y tres niveles.

  Los tipos de capa y en qué consisten:
    - **Capa de presentación**: es la que ve el usuario, el interface de usuario, presenta el sistema al usuario, le comunica la información y captura la información del usuario dando un mínimo de proceso. Es comunica con la capa de negocios
    - **Capa de negocio o lógica de negocio**: es donde residen los programas que se ejecutan, recibiendo las peticiones del usuario y enviando las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) pues es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación y con la capa de datos
    - **Capa de datos**: es donde residen los datos. Está formada por uno o más gestor de bases de datos que realizan todo el almacenamiento de datos, y reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

  Tenemos varios modelos:

    - **Arquitectura de 2 Capas.** La lógica está partida en dos capas, hechas

generalmente partiendo de la lógica del acceso de los datos. Esto da lugar a la estructura mostrada en el siguiente cuadro:

Podemos cambiar la lógica a cualquier de las capas así que hemos de seleccionar según los requisitos, si usaremos clientes ligeros o pesados. Si es un cliente pesado (como la figura) presenta la ventaja de que evita todas las complejidades de comunicarse con la base de datos a una capa separada.

- **Arquitectura de 3 Capas.** Este modelo divide cada una de las tres áreas lógicas en su propia capa. Para que esta estructura trabaje con eficiencia deben desarrollarse interfaces bien definidos entre cada una de las capas. Esto debe permitir que se puedan modificar componentes de una capa sin requerir cambios a cualquiera de los componentes de otras capas. La ventaja principal de esta estructura sobre el modelo 2-capas es que toda la lógica del negocio está contenida en su propia capa y es compartida por muchos componentes en la capa de presentación.

Estudio de alternativas actuales

- **Servidor Web (WEB Server) y Servidor de Aplicaciones (Application Server):** Un web server o **servidor web** es un programa que, usando el modelo del cliente-servidor y el protocolo de transferencia de hipertexto (http) del World Wide Web, sirve los archivos que forman páginas web a los usuarios de la Web. Cada computador en Internet que contiene un sitio web debe tener un programa web server. Los **servidores de aplicaciones** son software que nos ayudan a construir la separación lógica (a veces también física) entre la capa de datos y la lógica o capa de negocio. Desarrollar un servidor de aplicaciones implica el ocuparse de muchas tareas complicadas: gerencia de la conexión de red, diseñar una consola de la gerencia, balanceo de carga, etc. Los sitios Web construidos usando el modelo del servidor de aplicaciones contienen por lo menos tres "subcapas" en el back-end. Éstas son: Capa del servidor web, Capa del servidor de aplicaciones y Capa de datos.

- **Peer-to-Peer:** una red informática entre iguales se refiere a una red que no tiene clientes y servidores fijos, sino una serie de nodos que se comportan simultáneamente como clientes y como servidores de los demás nodos de la red. En la comunicación P2P cualquier nodo puede iniciar, detener o completar una transacción compatible. Se basa un sistema enteramente meritocrático en donde "el que más comparta, más privilegios tiene y más acceso dispone de manera más rápida a más contenido". Aquellos usuarios que no comparten contenido en el sistema y con ello no siguen la filosofía propia de esta red, se les denomina "**leechers**". Estos muchas veces representan una amenaza para la disponibilidad de recursos en una red P2P debido a que únicamente consumen recursos sin reponer lo que consumen, por ende podrían agotar los recursos compartidos y atentar contra la estabilidad de la misma. Debido a que la mayoría de los ordenadores domésticos no tienen una IP fija, la solución habitual es realizar una conexión a un servidor (o servidores) con dirección conocida (normalmente IP fija), que se encarga de mantener la relación de direcciones IP de los clientes de la red

Aplicación práctica sobre un sistema real

PL/SQL es la extensión o ampliación del lenguaje procedural del SQL. EL lenguaje ofrece un entorno de programación robusto que te posibilitará programar de forma procedural y/o mediante técnicas de programación orientada a objetos como la encapsulación, la ocultación de la información, y la sobrecarga de procedimientos o funciones. Ofrece construcciones procedurales como variables, constantes y tipos. Las mayores ventajas de la utilización de PL/SQL son :

- Integración de construcciones procedurales con SQL. Es decir, el PL/SQL integra declaraciones de control y condicionales con SQL.
- Reducción de congestión de red. PL/SQL te permite combinar declaraciones SQL, de forma lógica como una unidad. La aplicación puede enviar el bloque completo a la base de datos
- Desarrollo modularizado o modular. PL/SQL te permite agrupar declaraciones lógicamente relacionadas, en bloques. Puedes anidar bloques dentro de otros bloques más grandes para construir programas potentes
- Integración con herramientas. La máquina PL/SQL está integrada en herramientas de Oracle como HTML DB, Oracle Databese XE, Oracle Forms, Oracle Reports, etc.
- Portabilidad. Programas PL/SQL pueden correr en cualquier sitio. Oracle Server corre independientemente del sistema operativo y de la plataforma.
- Gestión de excepciones. Una excepción es un error en PL/SQL que surge durante la ejecución de un bloque. PL/SQL te permite gestionar excepciones eficientemente.

La construcción básica en PL/SQL es un **bloque**. Un bloque se compone de un conjunto de declaraciones SQL y/o PL/SQL, combinados y pasados a la máquina Oracle. Un bloque PL/SQL **se compone de tres secciones**:

- **Declarativo** (opcional). Esta sección empieza por la palabra clave “DECLARE” y termina al empezar tu sección ejecutable. Contiene declaraciones de todas las variables, constantes, cursores y excepciones definidas por usuario
- **Ejecutable** (obligatorio) Esta sección empieza por la palabra clave “BEGIN” y termina por “END;”. Contiene declaraciones SQL para rescatar datos de la base de datos y declaraciones PL/SQL para manipular datos en el bloque.
- **Gestión de excepciones** (opcional). La sección excepción está anidada en la sección ejecutable. Esta sección empieza por la palabra clave EXCEPTION. Especifica las acciones a realizar al surgir errores

Un programa PL/SQL se compone de uno o más bloques. Estos bloques pueden estar totalmente separados o anidados dentro de otro bloque. Existen **tres tipos de bloques** que componen un programa PL/SQL:

- **Bloques anónimos:** Estos son bloques PL/SQL sin nombre, encuadrados en una aplicación o emitidos interactivamente.
- **Procedimientos:** Estos son los llamados bloques PL/SQL. Estos bloques aceptan parámetros de entrada pero no devolverán valores explícitos.
- **Funciones:** Estos son los llamados bloques PL/SQL. Estos bloques aceptan parámetros de entrada y siempre devolverán un valor.

Las características de los **bloques anónimos** son: no se almacenan en la base de datos y se pasan a la máquina PL/SQL para su ejecución en el momento de la ejecución, no podrás invocar o llamar el bloque que escribiste anteriormente porque los bloques son anónimos y no existen tras haber sido ejecutados y pueden utilizarse como bloques dentro de procedimientos, funciones y otros bloques anónimos.

La sintaxis para la creación de un bloque anónimo en PL/SQL es la siguiente:

[DECLARE
variables, constantes, excepciones de usuario...]
BEGIN
<órdenes SQL>
<ódenes PL/SQL>
[EXCEPTION
acciones a realizar cuando llegue una excepción]
END

Triggers o disparadores

Un trigger o disparador es un bloque PL/SQL o un procedimiento PL/SQL asociados a una tabla, vista, esquema o base de datos que se ejecuta siempre que se produce un evento específico. Los triggers se utilizan para asegurar la integridad de datos, revisando datos de forma consistente.

Hay dos **tipos**:

- **Triggers de aplicación:** se dispara siempre que un evento tiene lugar con una aplicación en particular.
- **Trigger de base de datos:** se dispara siempre que tiene lugar un evento de datos (como DML) o un evento de sistema (como un logon o shutdown) sobre un esquema o base de datos.

Debemos tener en cuenta las siguientes pautas para el diseño de los triggers:

- Puedes diseñar triggers para llevar a cabo acciones relacionadas y centralizar operaciones globales.
- No debes diseñar triggers donde la funcionalidad ya está incluida en el SGBD.
- Evita diseñar triggers que dupliquen otros triggers.
- Puedes crear procedimientos almacenados e invocarlos en un trigger si el código PL/SQL es muy largo.
- Debes de tener en cuenta que un uso excesivo de triggers que se ejecutan en tablas que presenten relaciones de interdependencia complejas, pueden llegar a ser bastante difíciles de mantener en aplicaciones grandes.

La **sintaxis para crear un trigger DML** en PL/SQL es la siguiente:

CREATE [OR REPLACE] TRIGGER *trigger_name*
timing
event1 [OR event2 OR event3]
ON object_name
*[[**REFERENCING OLD AS old/NEW AS new**]*
FOR EACH ROW
[WHEN *(condition)]]*
trigger_body

Analicemos los componentes de la sintaxis:

- **Trigger_name**, identifica el trigger de forma única.
- **Timing (Momento)**. si es lanzado antes o después de realizar la operación que lo lanza. Los valores son BEFORE o AFTER(antes o después).

- **Evento.** identifica si es lanzado al insertar, al actualizar o al borrar de una tabla. Los valores son INSERT, UPDATE y DELETE.
- **Object_name**, indica la tabla o la vista asociada al trigger.
- **Referenciar**. La cláusula REFERENCING se utiliza para elegir un nombre de correlación para referenciar los valores antiguos y nuevos de la línea actual. REFERENCING nos permite asignar un alias a los valores NEW o/y OLD de las filas afectadas por la operación. Si el disparador es lanzado al insertar, el valor antiguo no tendrá sentido y el valor nuevo será la fila que estamos insertando. Para un disparador lanzado al actualizar, el valor antiguo contendrá la fila antes de actualizar y el valor nuevo contendrá la fila que vamos a actualizar. Para un disparador lanzado al borrar sólo tendrá sentido el valor antiguo. Sólo podrá ser utilizados con disparadores para filas.
- **Cuando**. La cláusula WHEN se utiliza para aplicar un predicado condicional, entre paréntesis, que es evaluado en cada línea para determinar si ejecutar el cuerpo del trigger. WHEN nos permite indicar al disparador que sólo sea lanzado cuando sea TRUE una cierta condición
- **Cuerpo del trigger.** El trigger_body es la acción llevada a cabo por el trigger.

Los tipos de triggers DML son:

- **Trigger de declaración** . Un trigger de declaración es el tipo de trigger por defecto y se ejecuta una vez por el evento disparador. Un trigger de declaración se dispara incluso si no afecta a ningún registro.
- **Trigger de fila o registro.** Un trigger de fila se ejecuta una vez por cada fila afectada por el evento disparador. Un trigger de fila no se ejecuta si el evento disparador no afecta a ninguna fila. Para indicar un trigger de fila se especifica FOR EACH ROW.

Un **trigger tiene dos estados o modos: ENABLED y DISABLED.** En el momento de crear un trigger, éste estará habilitado (enabled) por defecto. El Oracle Server revisa las restricciones de integridad en busca de triggers habilitados y garantiza que los triggers no puedan comprometer las operaciones a realizar.

ALTER TRIGGER trigger_name DISABLE
ALTER TABLE table_name DISABLE ALL TRIGGERS.
ALTER TRIGGER trigger_name COMPILE # Recompila un trigger para una tabla

Es posible que desees hacer un seguimiento de las siguientes operaciones que se realicen en una aplicación de gestión: inserción de nuevos datos, eliminación de datos existentes y modificación de datos existentes. La auditoria de aplicación se utiliza cuando deseamos tener un registro de las operaciones realizadas en alguna tabla. Los triggers están especialmente indicados para realizar esta función, aunque también podría realizarse con procedimientos almacenados. Para crear un sistema de auditoría debe existir una tabla auditora donde guardar la información deseada. Esta tabla se utiliza para el seguimiento de los cambios en los datos

Procedimientos almacenados

Los procedimientos almacenados son bloques nombrados de código que posibilitan la agrupación y organización de una serie de declaraciones SQL y PL/SQL. Tanto el código fuente como el código ejecutable se almacenan en la base de datos. Al almacenarlos en la base de datos, el código se encuentra en una ubicación centralizada y accesible y eso hace que la invocación al código almacenado sea muy eficiente.

Pueden ser llamados de forma interactiva desde aplicaciones cliente, desde otros procedimientos almacenados, y también desde los disparadores o desencadenadores. Pueden aceptar y devolver

parámetros, también diversos conjuntos de resultados, y un código de estado. Son un buen lugar donde programar las reglas de negocio de las aplicaciones

Las principales características son:

- Facilitan la reutilización y mantenimiento.
- Aceptar parámetros de entrada y devolver varios valores en forma de parámetros de salida
- Contener instrucciones de programación que realicen operaciones en la base de datos, incluidas las llamadas a otros procedimientos.
- Devolver un valor de estado que indica si la operación se ha realizado correctamente o ha habido un error
- Permiten una ejecución más rápida, ya que los procedimientos son analizados y optimizados en el momento de su creación
- Pueden reducir el tráfico de red.
- Pueden utilizarse como mecanismo de seguridad, ya que se puede conceder permisos a los usuarios

Se crean con la declaración CREATE PROCEDURE, la sintaxis es la siguiente:

CREATE [OR REPLACE] PROCEDURE *nombre_procedimiento (*
param [IN | OUT | INOUT] <tipo> [, param [IN | OUT | INOUT] <tipo> , ...]
IS/AS
*[**DECLARE**]*
    [<nombre variables locales>]
BEGIN
    <código del procedimiento>
*[**EXCEPTION**]*
END *nombre;*

La opción DECLARE se utiliza si deseamos usar variables locales en el procedimiento. Cuando se crea un procedimiento, éste se compila en primer lugar y queda almacenado en la base de datos de forma compilada. El código compilado puede ser posteriormente utilizado por cualquier bloque PL/SQL. Podemos eliminar un procedimiento mediante la orden DROP PROCEDURE nombre.

Los **parámetros** (o argumentos) se utilizan para transferir valores de datos entre el entorno invocador y el procedimiento (o subprograma). Existen tres modos de paso de parámetros:

- Un parámetro IN pasa un valor constante desde el entorno de llamada al procedimiento.
- Un parámetro OUT pasa un valor del procedimiento al entorno de llamada.
- Un parámetro INOUT pasa un valor del entorno de llamada al procedimiento y un valor posiblemente diferente del procedimiento de vuelta al entorno de llamada utilizando el mismo parámetro.

Para ejecutar un procedimiento se debe poner lo siguiente:

EXECUTE nom_procedimiento([parámetros]

Funciones PL/SQL

Una función es un bloque PL/SQL nombrado que puede aceptar parámetros, ser llamado y devolver un valor. Las funciones son iguales que los procedimientos, pero además devuelven un valor

```
CREATE [OR REPLACE] FUNCTION nombre_funcion
        [ ( parameter1 [mode1] datatype1,
        parameter2 [mode2] datatype2, .... ) ]
RETURN datatype
IS/AS
[DECLARE]
        [ local_variables_declarations; ]
BEGIN
        < codigo_procedimiento >
RETURN <valor>
[EXCEPTION]
END;
```

Una **excepción** es un error que surge durante la ejecución de un bloque. Un bloque siempre termina cuando se produce un error en la ejecución, pero gracias al uso de excepciones podemos especificar un gestor de excepción para ejecutar acciones antes que finalice el bloque.

Podemos definir diferente bloques según el tipo de error lanzado dentro de nuestro programa. Aquí varios ejemplos:

```
WHEN NO_DATA_FOUND THEN
        ocurren una excepción de tipo
NO_DATA_FOUND

WHEN ZERO_DIVIDE THEN
        error al dividir por cero
ZERO_DIVIDE

WHEN OTHERS THEN
        se produce cualquier error
```

Una vez finalizada la ejecución del bloque de EXCEPTION no se continúa ejecutando el bloque anterior. Las excepciones definidas por el usuario deben ser alcanzadas explícitamente utilizando la sentencia RAISE. Los errores son:

- **ACCESS_INTO_NULL** El programa intentó asignar valores a los atributos de un objeto no inicializado -6530
- **COLLECTION_IS_NULL** El programa intentó asignar valores a una tabla anidada aún no inicializada -6531
- **CURSOR_ALREADY_OPEN** El programa intentó abrir un cursor que ya se encontraba abierto. Recuerde que un cursor de ciclo FOR automáticamente lo abre y ello no se debe especificar con la sentencia OPEN -6511
- **DUP_VAL_ON_INDEX** El programa intentó almacenar valores duplicados en una columna que se mantiene con restricción de integridad de un índice único (unique index) -1
- **INVALID_CURSOR** El programa intentó efectuar una operación no válida sobre un cursor -1001
- **INVALID_NUMBER** En una sentencia SQL, la conversión de una cadena de caracteres hacia un número falla cuando esa cadena no representa un número válido -1722
- **LOGIN_DENIED** El programa intentó conectarse a Oracle con un nombre de usuario o password inválido -1017
- **NO_DATA_FOUND** Una sentencia SELECT INTO no devolvió valores o el programa referenció un elemento no inicializado en una tabla indexada 100
- **NOT_LOGGED_ON** El programa efectuó una llamada a Oracle sin estar conectado -1012

- **PROGRAM_ERROR** PL/SQL tiene un problema interno -6501
- **ROWTYPE_MISMATCH** Los elementos de una asignación (el valor a asignar y la variable que lo contendrá) tienen tipos incompatibles. También se presenta este error cuando un parámetro pasado a un subprograma no es del tipo esperado -6504
- **SELF_IS_NULL** El parámetro SELF (el primero que es pasado a un método MEMBER) es nulo -30625
- **STORAGE_ERROR** La memoria se terminó o está corrupta -6500
- **SUBSCRIPT_BEYOND_COUNT** El programa está tratando de referenciar un elemento de un array indexado que se encuentra en una posición más grande que el número real de elementos de la colección -6533
- **SUBSCRIPT_OUTSIDE_LIMIT** El programa está referenciando un elemento de un array utilizando un número fuera del rango permitido (por ejemplo, el elemento "-1") -6532
- **SYS_INVALID_ROWID** La conversión de una cadena de caracteres hacia un tipo rowid falló porque la cadena no representa un número -1410
- **TIMEOUT_ON_RESOURCE** Se excedió el tiempo máximo de espera por un recurso en Oracle -51
- **TOO_MANY_ROWS** Una sentencia SELECT INTO devuelve más de una fila -1422
- **VALUE_ERROR** Ocurrió un error aritmético, de conversión o truncamiento. Por ejemplo, sucede cuando se intenta calzar un valor muy grande dentro de una variable más pequeña -6502
- **ZERO_DIVIDE** El programa intentó efectuar una división por cero -1476

PL/SQL permite al usuario definir sus propias excepciones, las que deberán ser declaradas y lanzadas explícitamente utilizando la sentencia RAISE. Las excepciones deben ser declaradas en el segmento DECLARE de un bloque, subprograma o paquete. Se declara una excepción como cualquier otra variable, asignándole el tipo EXCEPTION. Una excepción es válida dentro de su ámbito de alcance, es decir, el bloque o programa donde ha sido declarada.

Ejemplo:

```
DECLARE
        VALOR_NEGATIVO EXCEPTION;
        valor NUMBER;
BEGIN
        valor := 1
        if valor < 0 THEN
                RAISE VALOR_NEGATIVO
        END IF;
EXCEPTION
        WHEN VALOR_NEGATIVO THEN
                ....
END;
```

Al manejar una excepción es posible usar las funciones predefinidas SQLCODE y SQLERRM para aclarar al usuario la situación de error acontecida.

- SQLCODE devuelve el número del error de Oracle y un 0 (cero) en caso de éxito al ejecutarse una sentencia SQL.
- SQLERRM devuelve el correspondiente mensaje de error.

Estas funciones no pueden ser utilizadas directamente en una sentencia SQL, pero sí se puede asignar su valor a alguna variable de programa y luego usar esta última en alguna sentencia.

Ejemplo:

```
WHEN OTHERS THEN
        err_num := SQLCODE
        err_msg := SQLERRM
        DBMS_OUTPUT.put_line('Error: ' || TO_CHAR(err_num));
        DBMS_OUTPUT.put_line(err_msg);
END;
```

Propagación de excepciones. Cuando se lanza una excepción, el control se transfiere hasta la sección EXCEPTION del bloque donde se ha producido la excepción. Entonces se busca un manejador válido de la excepción (WHEN excepcionTHEN, WHEN OTHERS THEN) dentro del bloque actual. En el caso de que no se encuentre ningún manejador válida el contro del programa se desplaza hasta el bloque EXCEPTION del bloque que ha realizado la llamada PL/SQL.

9. Herramientas CASE

Estudio de las herramientas case actuales

La tecnología CASE supone la **automatización del desarrollo del software**, contribuyendo a mejorar la calidad y la productividad en el desarrollo de sistemas de información y se plantea los siguientes objetivos:

- Permitir la aplicación práctica de **metodologías estructuradas**, que al ser realizadas con una herramienta consiguen agilizar el trabajo
- Facilitar la **realización de prototipos** y el desarrollo conjunto de aplicaciones
- **Simplificar el mantenimiento** de los programas
- Mejorar y estandarizar la **documentación**
- Aumentar la **portabilidad** de las aplicaciones
- Facilitar la **reutilización** de componentes software
- Permitir un desarrollo y un **refinamiento visual de las aplicaciones**, mediante la utilización de gráficos

Permite desarrollar aplicaciones con unos recursos humanos y un tiempo de desarrollo mucho más reducidos que si tuviéramos que trabajar sin estas herramientas.

Las **características** más reseñables de una herramienta CASE cliente / servidor son:

- **Proporcionar topologías de aplicación flexibles.** La herramienta debe proporcionar facilidades de construcción que permita separar la aplicación (en muchos puntos diferentes) entre el cliente y el servidor y también, entre servidores
- **Proporcionar aplicaciones portátiles.** La herramienta debe generar código para Windows, OS/2, Macintosh, Unix y todas las plataformas de servidores conocidas.
- **Control de versión**. La herramienta debe reconocer las versiones de código que se ejecutan en los clientes y servidores, y asegurarse que sean consistentes.
- **Crear código compilado en el servidor.** La herramienta debe ser capaz de compilar automáticamente código 4GL en el servidor.
- **Trabajar con una variedad de software intermedios.** La herramienta debe adaptar sus comunicaciones cliente / servidor al software intermedio existente. Como mínimo la herramienta debería adaptar así el tráfico que se está moviendo en una LAN o WAN.
- **Soporte multiusuarios.** La herramienta debe permitir que varios diseñadores trabajen en una aplicación simultáneamente.
- **Seguridad.** La herramienta debe proporcionar mecanismos para controlar el acceso y las modificaciones a los datos que contiene. La herramienta debe, al menos, mantener contraseñas y permisos de acceso en distintos niveles para cada usuario.

Las herramientas **CASE evolucionan hacia tres tipos de integración**:

- La integración de datos permite disponer de herramientas CASE con diferentes estructuras de diccionarios locales para el intercambio de datos.
- La integración de presentación confiere a todas las herramientas CASE el mismo aspecto.
- La integración de herramientas permite disponer de herramientas CASE capaces de invocar a otras CASE de forma automática.

Usando el creador de aplicaciones del Oracle Express

Una **aplicación** (para Oracle Express) es una recopilación de páginas enlazadas entre sí mediante separadores, botones o enlaces de hipertexto. Las páginas de una aplicación comparten una definición de estado de sesión y un método común de autentificación (también se suele llamar autenticación).

La **autentificación (o autenticación)** es el proceso de identificación de una persona que generalmente se hace a partir del nombre del usuario y la contraseña. En los sistemas de seguridad, la autentificación difiere de la autorización, la cual consiste en permitir el acceso personal a los elementos de un sistema a partir de la identidad de la persona.

La herramienta del Oracle Express nos facilita tres métodos o formas de crear nuestra aplicación.

- **Crear Aplicación.** Crea una aplicación definiendo páginas, seleccionando un esquema de autenticación y especificando una interfaz de usuario. Las páginas se pueden basar en tablas, consultas o consultas desplegables.
- **Crear a partir de Hoja de Cálculo.** Crea una aplicación basada en datos de hoja de cálculo. Permite cargar o pegar datos de una hoja de cálculo para crear una tabla. La aplicación mostrará las capacidades de consulta, inserción, actualización y análisis en la tabla recién creada.
- **Aplicación de Demostración.** Permite instalar o desinstalar las aplicaciones de demostración.

Una **página** es un documento de hipertexto (html), donde la página es cada uno de los elementos que nos puede presentar un cliente Web. Las páginas contienen texto, enlaces, imágenes, y otros elementos multimedia. **Desde el punto de vista del desarrollo de la capa cliente** de una aplicación cliente-servidor, definiremos una página como cada segmento de un programa o aplicación que se carga en la memoria principal sólo si es necesario que se ejecute.

Los tipos de páginas son:

- **Página en blanco.** Crea una página sin ninguna funcionalidad incorporada.
- **Página Informe.** Crea una página que contiene el resultado con formato de una consulta SQL. Pueden incluir además **Páginas de Análisis** para crear informes y gráficos de resumen adicionales.
- **Página Pantalla.** Crea una pantalla para actualizar una única fila de una tabla de base de datos.
- **Página Pantalla Tabular.** Crea una pantalla para realizar operaciones de actualización, inserción y supresión en varias filas de una tabla de base de datos.
- **Página Informe y Pantalla.** Crea una combinación de pantalla e informe de dos páginas. En la primera página, los usuarios seleccionan la fila que se vaya a actualizar. En la segunda, los usuarios pueden actualizar la tabla o vista seleccionada.

Mediante los **separadores** el usuario puede acceder a las distintas páginas de la aplicación con sólo pulsar en la opción deseada.

Podemos definir un separador como un enlace a una página para acceder a dicha página rápida y fácilmente. Gracias al uso de separadores, el usuario conoce las funcionalidades principales de la aplicación (o de una página) con un solo golpe de vista. Es decir, el uso de separadores mejora notablemente la ergonomía de nuestra aplicación.

Oracle Express nos permite añadir hasta **dos niveles de separadores en una página**. Si deseamos añadir más niveles (lo cual no es aconsejable atendiendo a la ergonomía) deberemos, una vez creada la página, editarla para después modificarla.

Una aplicación con sólo un nivel de separadores utiliza un **juego de separadores estándar**. Un juego de separadores estándar se asocia a una página e identificador de página concretos. Se puede utilizar separadores estándar, por ejemplo, para enlazar usuarios a una página concreta. Un **juego de separadores principales** funciona como un contenedor para albergar un grupo de separadores estándar. Los separadores principales ofrecen a los usuarios otro nivel de navegación,

Oracle Express nos posibilita reutilizar los componentes que hemos desarrollado o que existen en otra aplicación mediante una de sus funcionalidades, en concreto mediante el uso de los **componentes compartidos**, que son elementos de aplicación comunes que se pueden mostrar o aplicar en varias páginas de una aplicación.

Las aplicaciones que creemos con el Creador de Aplicaciones de Oracle Express tendrán una serie de **atributos** que deben ser definidos. Los atributos son:

- La **autenticación** es el proceso de establecer las identidades de los usuarios para que puedan acceder a una aplicación. Los esquemas de autenticación disponibles son:
    - **Autenticación de Oracle Express.** Sólo tendrán acceso a la aplicación los mismos usuarios que los autenticados dentro del Oracle Express.
    - **Sin Autenticación.** Los usuarios no necesitan autenticación para acceder a la aplicación.
    - **Cuenta de base de datos.** Los usuarios que tendrán acceso a la aplicación son los "autentificados" para la base de datos sobre la que corre la aplicación.
- **Idioma.** Este atributo identifica el idioma en el que se desarrolla una aplicación. Se trata del idioma base del que se realizan todas las traducciones
- **Preferencia de Idioma de Usuario Derivada de.** Especifica el modo en que el sistema determina el idioma de la aplicación. El idioma primario de la aplicación puede ser estático (es decir, derivado del idioma del explorador Web) o determinado a partir de un elemento o de una preferencia del usuario.

La herramienta de Oracle Express nos permite seleccionar doce **temas** distintos para nuestra aplicación. Los temas son recopilaciones de plantillas que definen el diseño y el estilo de toda una aplicación.

Las **regiones** de una página las podemos definir como las partes en las que se divide la página para ofrecer funcionalidades distintas en cada una de esas partes. Existen los siguientes tipos de regiones:

- **Nueva Página:** accederemos a la pantalla de creación de páginas que ya conocemos.
- **Región en esta Página:** estamos indicando que la nueva región debe de estar incluida en la página actual. Podrá ser: HTML, Informe, Pantalla, Gráfico, Ruta de Navegación, Contenido dinámico PL/SQL, Árbol, URL, Calendario, Varios HTML y Texto de Ayuda
- **Control de Página:** estamos indicando que la nueva región se encargará de realizar funciones de control en la propia página y que pueden ser utilizados para direccionar la aplicación a otra página. Puede ser de tipo: Elemento, Botón, Bifurcación, Cálculo, Proceso y Validación.

- **Control Compartido:** estamos indicando que la nueva región será un componente que se podrá usar en las demás páginas de la aplicación. Los tipos de controles compartidos que podemos insertar son: Icono de Barra de Navegación, Separador Principal, Separador Estándar, Lista de Valores, Lista y Ruta de Navegación.

Podemos definir los **atributos de aplicación** como los identificadores de la misma. El identificador de una aplicación es un número. Los tipos de atributos son los siguientes:

- **Atributos estándar.** Los atributos estándar de aplicación. Se usan estos atributos para controlar el nombre de la aplicación y disponibilidad al igual que se pueden definir las cadenas de sustitución. También muestran opciones definidas de construcción, el tema asociado, plantillas por defecto y componentes por defecto

- **Atributos de seguridad.** Estos atributos se usan para definir valores de seguridad a nivel de aplicación. También se utilizan las páginas de definición de componentes de aplicación para gestionar valores más específicos. Los atributos son los siguientes:
    - Autenticación. La autenticación es el proceso que establece la identidad de cada usuario antes de que pueda acceder a la aplicación.
    - Autorización. Los esquemas de autorización de la aplicación controlan el acceso a todas las páginas de una aplicación. El acceso no autorizado a la aplicación, independientemente de la página que se haya solicitado, hace que se muestre una página de error.
    - Esquema de Base de Datos. Todos los comandos SQL y PL/SQL emitidos por una aplicación se ejecutan con los derechos y privilegios del esquema de base de datos definido.
    - Protección de Estado de la Sesión. La activación de la protección de estado de la sesión puede evitar que los piratas informáticos manipulen las direcciones URL de la aplicación.
    - Base de Datos Privada Virtual (VPD). La Base de Datos Privada Virtual (VPD) proporciona una interfaz de programas de aplicación (API) que permite a los desarrolladores asignar políticas de seguridad a las tablas y vistas de bases de datos. Mediante PL/SQL, los desarrolladores pueden crear políticas de seguridad con procedimientos almacenados y enlazar los procedimientos a una tabla o vista mediante una llamada a un paquete RDBMS.

- **Atributos de globalización.** Mediante los atributos de globalización de la aplicación se puede acceder a las traducciones de la aplicación siguiendo los siguientes enlaces:
    - Página Inicial de Traducción. Las aplicaciones se pueden traducir del idioma primario a otro idioma.
    - Asignaciones de Traducción. Las aplicaciones traducidas se publican como nuevas aplicaciones. Debe especificar un identificador de aplicación de idioma primario y un identificador de aplicación de idioma traducido para cada idioma que se desee traducir.
    - Exportar Traducciones. Para traducir una aplicación, primero debe extraer el texto para traducir. Al extraer la traducción, se copia todo el texto traducible en un repositorio de texto de traducción. Cuando se extrae el texto traducible, se puede iniciar el proceso de traducción de las cadenas de texto traducibles. Después de extraer el texto de traducción de la aplicación e idioma específico y copiarlo en el repositorio de texto de traducción, puede generar y exportar un archivo XLIFF para traducirlo. El proceso de extracción mantiene la aplicación en el idioma primario sincronizada con el repositorio de texto de traducción. Por tanto, debe ejecutarse cada vez que haya cambios en la aplicación en idioma origen.
    - Importar Traducciones.
    - Definición de Mensaje. Se pueden utilizar mensajes de texto para crear cadenas de texto traducibles con variables de sustitución que se pueden llamar desde

paquetes, procedimientos y funciones PL/SQL.
    - Traducción de Mensaje. Los mensajes son cadenas de texto con nombre que se pueden llamar desde el código PL/SQL escrito por el usuario. Este código PL/SQL puede constituir bloques anónimos dentro de los procesos de página y regiones de página o en paquetes y procedimientos.
    - Traducciones Dinámicas. Las traducciones dinámicas permiten que el código PL/SQL realice traducciones con la función HTMLDB_LANG.LANG. Esta tabla asigna valores del idioma primario a otro idioma nacional.

Los **componentes compartidos** son los componentes de la aplicación que se pueden utilizar en cualquier página de la aplicación. Los componentes compartidos pueden ser de tipo:

- Lógico:
    - Elementos de Aplicación. Se utilizan para mantener el estado de la sesión. Los elementos de la aplicación se pueden definir mediante cálculos o procesos, o bien transfiriendo valores en una dirección URL. Se usan para mantener el estado de la sesión que no se muestra ni es específico de ninguna página.
    - Procesos de Aplicación. Ejecutan lógica PL/SQL en puntos específicos para cada página de una aplicación. También definen las condiciones que deben cumplirse tanto para que se ejecute la página como para la forma de hacerlo.
    - Cálculos de Aplicación. Se usan para asignar valores a elementos de página y aplicación.
    - Referencias de Servicio Web. El sistema de Application Express puede utilizar las referencias de servicio Web para acceder a un servicio Web a través de la red. El servicio Web realiza una acción y, a continuación, devuelve una respuesta.
    - Opciones de Creación. Se usan para mostrar de forma condicional la funcionalidad específica en una aplicación.

- Navegación.
    - Árboles. Un árbol es un mecanismo de navegación jerárquica. Se puede crear uno a partir de una consulta que especifique una relación jerárquica mediante la identificación de la columna de identificador y de identificador principal en una tabla o vista.
    - Entradas de Barra de Navegación. Las entradas de barra de navegación proporcionan una navegación basada en enlaces de hipertexto. La ubicación se determina mediante plantillas de página.
    - Listas. Una lista es una recopilación de enlaces compartidos y controlados por plantillas. Se usan las listas para agregar navegación a la aplicación.
    - Rutas de Navegación. Las rutas de navegación proporcionan una navegación jerárquica hasta un número infinito de niveles.
    - Separadores. podemos definir un separador como un enlace a una página para acceder a dicha página rápida y fácilmente.

- Seguridad.
    - Esquemas de Autenticación. Un esquema de autenticación es una configuración guardada que se puede aplicar a la aplicación.
    - Esquemas de Autorización. La autorización para aplicaciones, páginas y para la mayor parte del resto de los componentes se gestiona mediante los esquemas de autorización. Se asocia a una aplicación, página o cualquier otro componente en la página de atributos del componente. Los tipos de esquema de autorización comunes son: consultas SQL Existe y No Existe y Función PL/SQL que devuelve valor booleano.
    - Protección de Estado de la Sesión. La activación de la protección de estado de la sesión puede evitar que los piratas informáticos manipulen las direcciones URL de la aplicación.

    - Editar Atributos de Seguridad. Estos atributos son también componentes compartidos.

- Interfaz de usuario.
    - Temas. Un tema es una recopilación con nombre de plantillas utilizadas para definir la interfaz
    - Plantillas. El sistema Application Express crea el aspecto de cada página de una aplicación mediante plantillas. Las plantillas definen la forma en que se van a mostrar las páginas, los controles de página o los componentes de páginas.
    - Valores por Defecto de Interfaz de Usuario. Los valores por defecto de interfaz de usuario se utilizan para rellenar con valores iniciales las propiedades de región y elemento, a fin de proporcionar consistencia entre páginas de una aplicación o de varias aplicaciones.
    - Listas de Valores. Tanto los elementos de página como los campos de informe pueden hacer referencia a la lista de valores. La lista de valores controla los valores mostrados y limita la selección del usuario. Las listas de valores pueden ser estáticas (basadas en los valores introducidos) o dinámicas (basadas en la consulta SQL).
    - Métodos Abreviados. Se utilizan los métodos abreviados para escribir una vez el código que se utiliza con frecuencia y hacer referencia a éste en muchas ubicaciones de la aplicación. Es decir, permiten la reutilización de los componentes software.

- De globalización. Definidos más arriba.

- Archivos.
    - Archivos Estáticos. Los archivos estáticos los podemos definir como los archivos que no deben ser modificados por el usuario
    - Hojas de Estilo en Cascada. Las hojas de estilo en cascada (Cascading Style Sheets, CSS) son un lenguaje formal usado para definir la presentación de un documento estructurado escrito en HTML o XML
    - Imágenes. Las imágenes de Application Express se dividen en dos categorías:
        - Imágenes de Espacio de Trabajo que están disponibles para todas las aplicaciones de un espacio de trabajo determinado.
        - Imágenes de Aplicación que están disponibles sólo para una aplicación.