

PASO DEL MODELO E/R AL MODELO RELACIONAL

ENTIDAD →

Siempre
Generan tabla

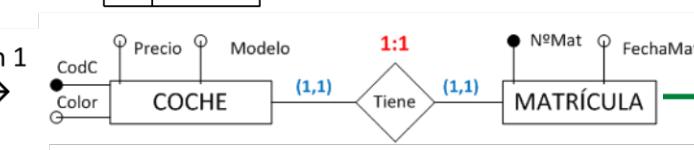


CLIENTES		
PK	CodCli	
Nombre Dirección		

Elena Fernández Chirino (IES TRASSIERA)



- Participación min 1 en ambos lados → **NO genera tabla**



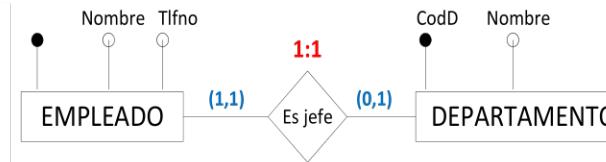
COCHES		
PK	CodC	
Color Precio Modelo		

O

COCHES		
PK	CodC	
Color Precio Modelo		

MATRÍCULAS		
PK	NumMat	
FechaMat		

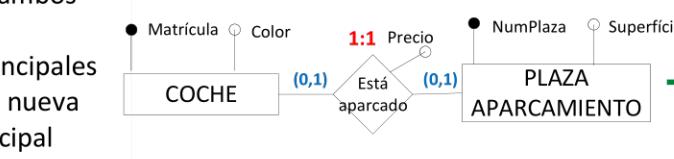
- Participación min 0 en un lado → **NO genera tabla**. La foránea pasa al lado 0



EMPLEADOS		
PK	CodE	
Nombre Tlfno		

DEPARTAMENTOS		
PK	CodD	
Nombre CodE		

- Participación min 0 en ambos lados → **SÍ genera tabla**. Las dos principales pasan como foráneas a la nueva tabla y solo una es principal

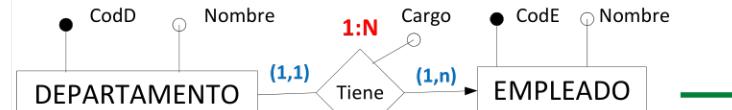


COCHES		
PK	Matrícula	
Color Modelo		

COCHES-APMTO		
PK,FK1	Matrícula	
FK2 NumPlaza Precio		

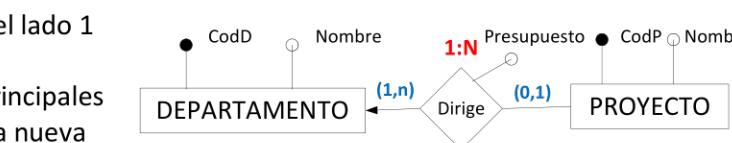
APARCAMIENTOS		
PK	NumPlaza	
Superficie		

- Participación min 1 en el lado 1 → **NO genera tabla**. La principal del lado 1 pasa al lado n como foránea



DEPARTAMENTOS		
PK	CodD	
Nombre CodD Cargo		

- Participación min 0 en el lado 1 lados → **SÍ genera tabla**. Las dos principales pasan como foráneas a la nueva tabla y solo es principal la del lado n

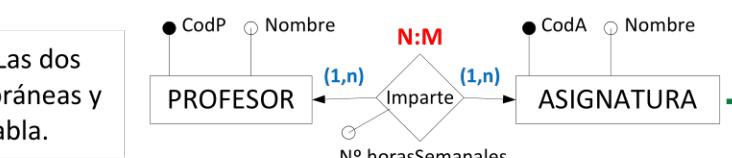


DEPARTAMENTOS		
PK	CodD	
Nombre FK1 CodD Cargo		

DIRIGIDOS		
PK,FK1	CodD	
FK2 CodP Presupuesto		

PROYECTOS		
PK	CodP	
Nombre		

- SIEMPRE genera tabla**. Las dos principales pasan como foráneas y principal a la nueva tabla.



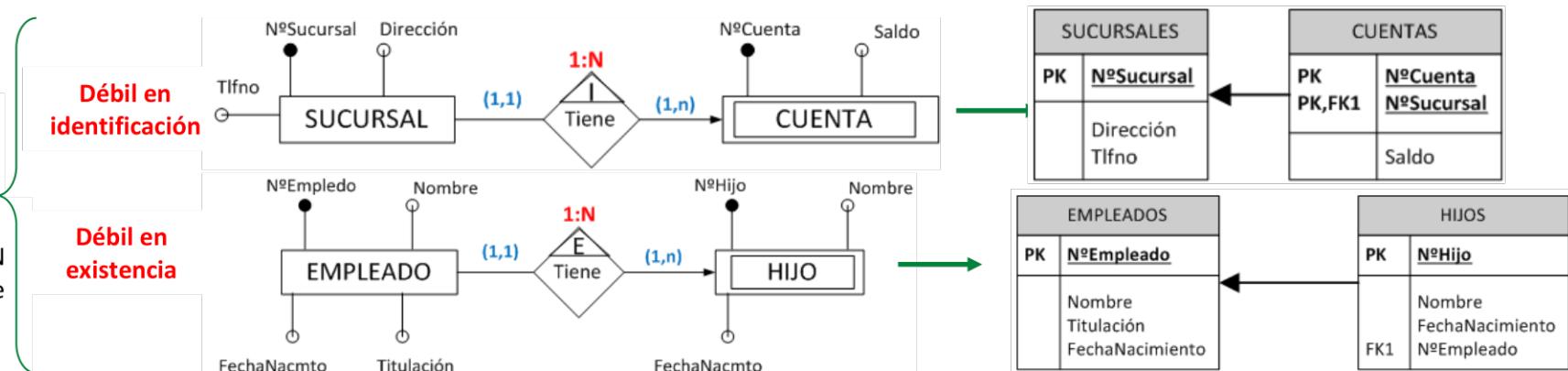
PROFESORES		
PK	CodP	
Nombre		

IMPARTE		
PK,FK1	CodP	
CodA NºhorasSemanales		

ASIGNATURAS		
PK	CodA	
Nombre		

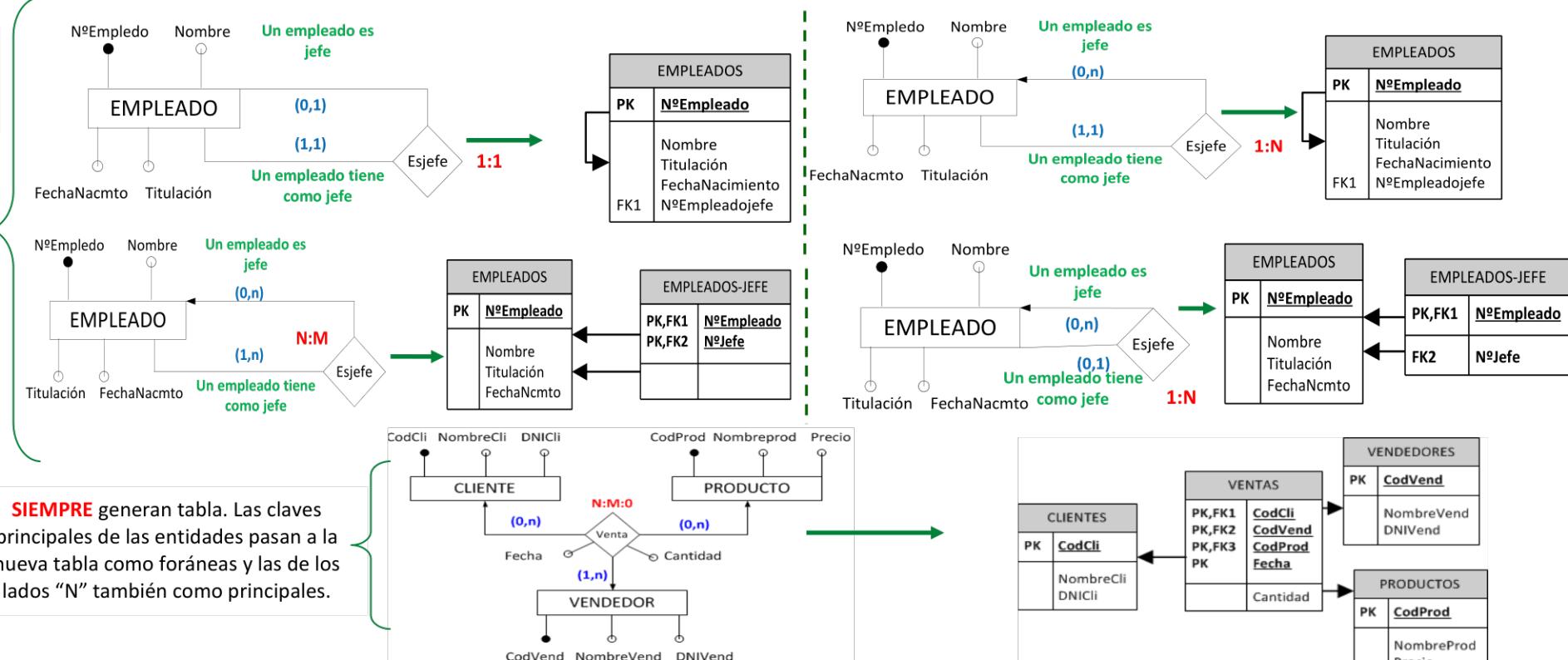
RELACIONES → DE DEPENDENCIA

NO generan tabla (La clave ppal del lado 1 pasa al lado N como foránea y en el caso de la débil en identificación, también como ppal.



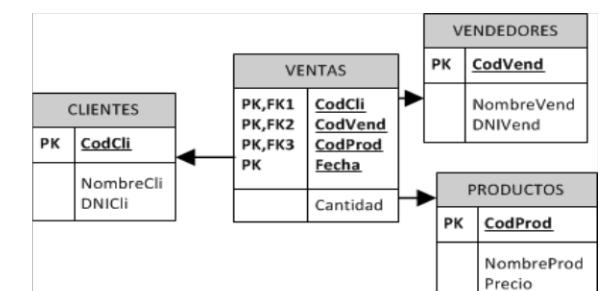
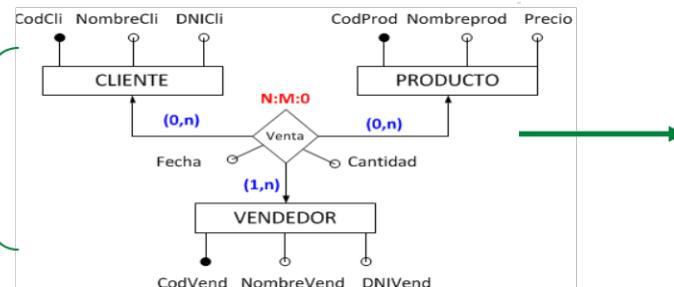
RELACIONES → REFLEXIVAS

SÍ generan tabla o **NO**, dependiendo de las cardinalidades y las participaciones. Igual que las relaciones binarias normal. Solo hay que tener en cuenta que no puede haber dos campos que se llamen igual en la misma tabla (a la hora de propagar la clave foránea)



RELACIONES → N-ARIAS

SIEMPRE generan tabla. Las claves principales de las entidades pasan a la nueva tabla como foráneas y las de los lados "N" también como principales.



JERARQUÍAS

- Se creará una tabla para la entidad supertipo. A no ser que tenga tan pocos atributos que dejarla sea una complicación.
- Si la entidad subtipo no tiene atributos y no está relacionada con ninguna otra entidad, desaparece.
- Si la entidad subtipo tiene algún atributo o interviene en alguna relación, se crea una tabla. Hereda la clave de la entidad supertipo. Para relacionarla con la entidad padre, la clave principal de ésta pasa a la entidad hijo como clave principal y foránea.
- Si la relación es exclusiva, y alguna de las entidades subtipo no genera tabla, el atributo que genera la jerarquía se incorpora en la tabla de la entidad supertipo.

ELEMENTOS DEL LENGUAJE SQL

El lenguaje SQL está compuesto por comandos, cláusulas, operadores, funciones y literales . Todos estos elementos se combinan en las instrucciones y se utilizan para crear, actualizar y manipular bases de datos.

- **COMANDOS:**

Comandos DDL:

Comando:	Descripción:
CREATE	Se utiliza para crear nuevas tablas, campos e índices.
DROP	Se utiliza para eliminar tablas e índices.
ALTER	Se utiliza para modificar tablas.

Comandos DML:

Comando:	Descripción:
SELECT	Se utiliza para consultar filas que satisfagan un criterio determinado.
INSERT	Se utiliza para cargar datos en una única operación.
UPDATE	Se utiliza para modificar valores de campos y filas específicos.
DELETE	Se utiliza para eliminar filas de una tabla.

Comandos DCL:

Comando:	Descripción:
GRANT	Permite dar permisos a uno o varios usuarios o roles para realizar tareas determinadas.
REVOKE	Permite eliminar permisos que previamente se han concedido con GRANT.

- **CLÁUSULAS:**

Llamadas también condiciones o criterios, son palabras especiales que permiten modificar el funcionamiento de un comando.

Cláusulas:	Descripción:
FROM	Se utiliza para especificar la tabla de la que se van a seleccionar las filas.
WHERE	Se utiliza para especificar las condiciones que deben reunir las filas que se van a seleccionar.
GROUP BY	Se utiliza para separar las filas seleccionadas en grupos específicos.
HAVING	Se utiliza para expresar la condición que debe satisfacer cada grupo.
ORDER BY	Se utiliza para ordenar las filas seleccionadas de acuerdo a un orden específico.

- **OPERADORES:**

Permiten crear expresiones complejas. Pueden ser aritméticos (+, -, *, /, ...) o lógicos (<, >, , <>, And, Or, ...).

Operadores lógicos

Operadores:	Descripción:
AND	Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
NOT	Devuelve el valor contrario de la expresión.

Operadores de comparación

Operadores:	Descripción:
<	Menor que.
>	Mayor que.
< >	Distinto de.
< =	Menor o igual.
> =	Mayor o igual.
=	Igual.
BETWEEN	Se utiliza para especificar un intervalo de valores.
LIKE	Se utiliza para comparar.
IN	Se utiliza para especificar filas de una base de datos.

- **FUNCIONES:**

Para conseguir valores complejos. Por ejemplo, la función promedio para obtener la media de un salario. Existen muchas funciones, aquí tienes la descripción de algunas.

Funciones de agregado:

Función:	Descripción:
AVG	Calcula el promedio de los valores de un campo determinado.
COUNT	Devuelve el número de filas de la selección.
SUM	Devuelve la suma de todos los valores de un campo determinado.
MAX	Devuelve el valor más alto de un campo determinado.
MIN	Devuelve el valor mínimo de un campo determinado.

- **LITERALES:**

Les podemos llamar también constantes y serán valores concretos, como por ejemplo un número, una fecha, un conjunto de caracteres, etc.

Literales:	Descripción:
23/03/97	Literal fecha.
María	Literal caracteres.
5	Literal número.

UD 01 - ALMACENAMIENTO DE LA INFORMACIÓN

1. Los ficheros de información. Concepto de fichero.

En los setenta las necesidades empresariales de contabilidad y facturación se solventaban utilizando un número reducido de archivos en papel agrupados y ordenados (ficheros).

Con la primera informatización se posibilita el acceso de forma más rápida a través del ordenador. Por eso se sigue hablando de ficheros, formularios, carpetas, directorios...

Los ficheros permitieron llevar a cabo el almacenamiento de datos de forma permanente en dispositivos de memoria masiva.

Fichero o archivo: Información relacionada tratada como un todo y organizada de forma estructurada. Es una secuencia de dígitos binarios que organiza información relacionada con un mismo aspecto. Están formados por registros lógicos divididos en campos que contienen informaciones elementales que forman un registro.

Los datos pueden ser añadidos, suprimidos, consultados, actualizados en cualquier momento.

Los ficheros son muy voluminosos, por lo que solo pueden ser llevados a la memoria principal partes de ellos.

Registro físico o bloque: Cantidad de información transferida entre el soporte en el que se almacena el fichero y la memoria principal del ordenador, en una operación de lectura/grabación.

En cada operación de lectura/grabación se transfieren varios registros del fichero.

Factor de bloqueo: Número de registros que entran en un bloque.

Bloqueo de registros: Operaciones de agrupar varios registros en un bloque.

Parámetros de uso

El uso de un fichero puede definirse por una serie de parámetros:

a. **Capacidad o volumen:** Espacio, en caracteres, que ocupa el fichero. Se calcula multiplicando el número previsto de registros por su longitud media.

b. **Actividad:** Cantidad de consultas y modificaciones en el fichero.

Debe tenerse en cuenta:

- Tasa de consulta o modificación: Porcentaje de registros consultados o modificados en cada tratamiento del fichero respecto a los registros totales.

- Frecuencia de consulta o modificación: Veces que se accede al fichero para consultar o modificar en un periodo de tiempo fijo.

c. **Volatilidad:** Cantidad de inserciones y borrados en el fichero.

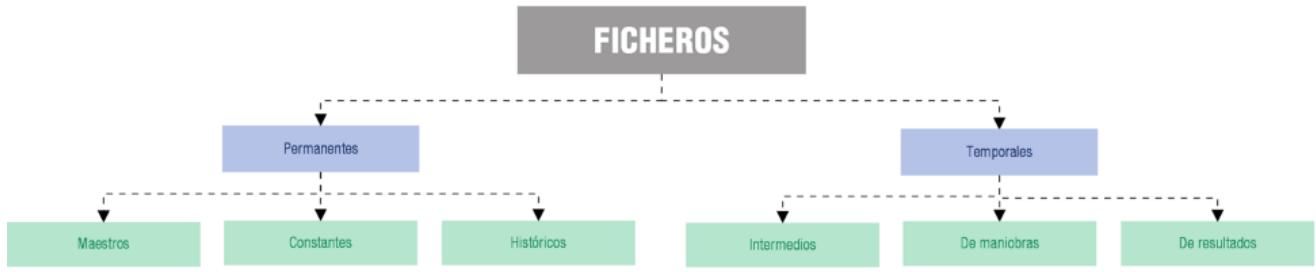
Debe conocerse:

- **Tasa de renovación:** Porcentaje de registros renovados en cada tratamiento respecto al total de registros contenidos.

- **Frecuencia de renovación:** Veces que se accede al fichero para renovarlo en un periodo de tiempo fijo.

d. **Crecimiento:** Variación de la capacidad del fichero que se mide con el porcentaje de registros en que aumenta el fichero en cada tratamiento (tasa de crecimiento)

1.1. Tipos de ficheros



1. **Ficheros permanentes:** Amplio periodo de permanencia en el sistema. Información relevante para el funcionamiento de una aplicación.
 1. **Ficheros maestros:** Parte central de la aplicación, núcleo. Estado actual de los datos que pueden modificarse desde la aplicación. Ej.: Archivo con los datos de los usuarios
 2. **Ficheros constantes:** No suelen ser modificados, se accede a ellos para realizar consultas. Datos fijos de la aplicación. Ej.: Archivo con códigos postales
 3. **Ficheros históricos:** Reconstrucción de situaciones. Fueron considerados ficheros actuales en un periodo o situación anterior. Ej.: Usuarios dados de baja.
2. **Ficheros temporales:**
 1. **Ficheros intermedios:** Resultados de una aplicación que serán usados por otra
 2. **Ficheros de maniobras:** Datos de una aplicación que no pueden mantenerse en memoria por falta de espacio
 3. **Ficheros de resultados:** Datos que van a transferirse a un dispositivo de salida

1.2. Soportes de información

Inicialmente, eran tambores de cinta magnética. Similar a los casetes pero de mayor dimensión y capacidad de almacenamiento (formato digital, ceros y unos, en orden secuencial)
 Posteriormente, avances en el hardware: disquete, disco duro. Dispositivos de acceso aleatorio no es necesario pasar por todos los datos desde el inicio hasta la zona donde se encuentra la información.

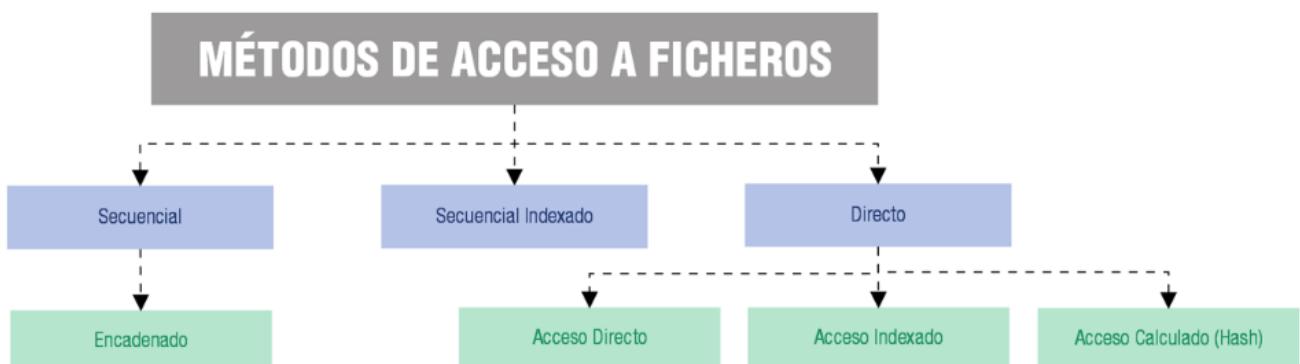
Distinguimos:

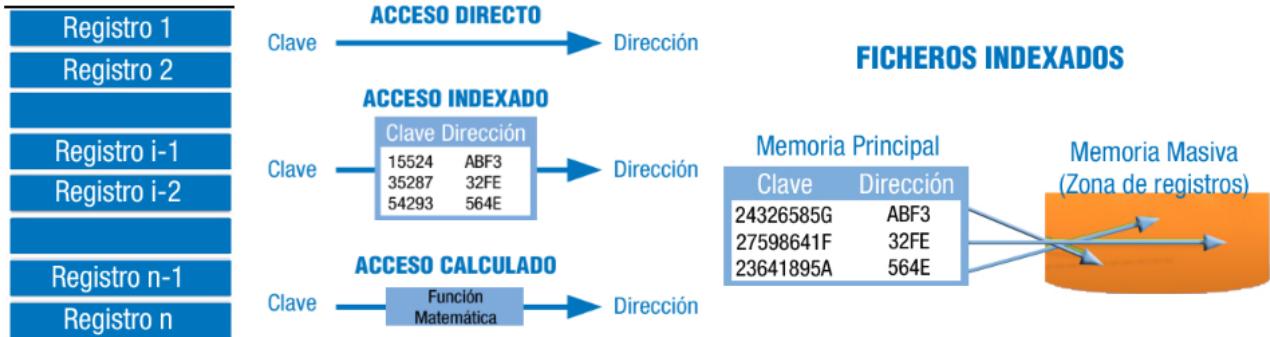
- **Soportes de acceso secuencial o no direccionables:** Para leer dato que está en mitad debe leerse todo hasta llegar. Copias de seguridad.
- **Soportes de acceso directo o direccionables:** Acceso de forma directa ubicándose en la posición deseada.

1.3. Métodos de acceso

Las modificaciones del acceso buscaban:

- Proporcionar acceso rápido a los ficheros
- Economizar el almacenamiento
- Facilitar la actualización de registros
- Permitir que la estructura refleje la organización real de la información





1.4. Ficheros secuenciales

- Los registros están almacenados de forma continua. La única forma de acceder a él es leyendo un registro tras otro de principio a fin. En ellos suele haber una marca indicativa de fin de fichero (EOF).
- Pueden usar dispositivos o soportes no direccionables o de acceso secuencial como las cintas magnéticas de almacenamiento de datos, cd, dvd (espiral continua)
- Los registros se identifican por la información en su campo **clave** o **llave**. Ordenando un archivo secuencial por su clave se puede hacer más rápido la lectura o escritura.
- La lectura siempre es hacia adelante
- No permite multiusuario
- Tiene estructura rígida de campos. Los registros deben aparecer en orden
- La apertura del fichero condiciona la lectura o escritura
- Aprovechan al máximo el soporte (no hay huecos vacíos)
- Se pueden grabar en soporte secuencial y en soporte direccionable
- Los lenguajes de programación pueden trabajar con ellos
- No se pueden insertar registros entre los ya grabados.

1.5. Ficheros de acceso directo o aleatorio

- Se puede acceder indicando la posición relativa dentro del archivo o, usualmente, a través de una clave que forma parte del registro.
- En dispositivos de acceso directo (discos magnéticos)
- No se encuentran en posiciones consecutivas sino en posiciones lógicas
- La clave es transformada y se obtiene la dirección física a la que corresponde el registro. Puede ser **acceso directo**, **indexado** o **calculado**. En el acceso directo la clave es la propia dirección debiendo ser numérica y comprendida en su rango de valores (más rápido)
- La medida básica de posicionamiento del puntero es el byte. Según la codificación sea Unicode o ANSI se utilizarán 1 o 2 bytes por carácter, respectivamente.

Características fundamentales:

- Posicionamiento inmediato**
- Registros de **longitud fija**
- El registro **puede abrirse en modo mixto**, para lectura y escritura
- Pueden usarse **múltiples usuarios**
- Se borran colocando un cero** en la posición que ocupan
- Permiten el uso de **algoritmos de compactación de huecos**
- Los archivos se crean con un **tamaño definido** (hay un máximo de registros establecidos durante la creación)
- Solo en soportes de acceso directo o direccionables**
- Usados cuando el acceso a los datos de un registro se hace siempre empleando la misma clave y la **velocidad de acceso a un registros es importante**
- Permiten la **actualización de los registros en el mismo fichero, sin necesidad de copiarlo**
- Permiten realizar actualización en tiempo real.

1.6. Ficheros indexados

Utilizan **índices** que permiten el acceso a un registro del fichero de forma directa, sin leer los anteriores. Existe:

- **Zona de registros:** Se encuentran en ella los datos del archivo
- **Zona de índices:** Contiene una tabla con las claves de los registros y posiciones que se encuentran en los mismos. Está ordenada por el campo clave.

La tabla de índices se carga en memoria principal para las búsquedas de la fila con la clave del registro a encontrar. Después se accede a esa fila en la zona de registro y se posiciona en la dirección indicada.

La zona de registros debe incluir todas las direcciones posibles del archivo (**problema determina su tamaño y mantenerla ordenada**)

El registro tiene que tener campo o combinación de campos que permita identificarlo de **forma única (campo clave)**

Puede usar tanto **acceso secuencial** (se leen ordenados por el contenido del campo clave, independientemente del orden en el que fueran grabados, accediendo a través del índice) como **acceso directo** (se accede al registro deseado por el índice).

1.7 Ficheros secuenciales indexados o parcialmente indexados

Tienen también una zona de índices y otra de registros pero esta está **dividida en segmentos** ordenados (bloques de registros).

En la tabla de índices, cada fila hace referencia a los segmentos. **La clave corresponde al último registro y el índice apunta al registro inicial**. Al acceder al primer registro del segmento, dentro de él se localiza **secuencialmente** el registro buscado.

- Es **muy usado para cuando hay pocos registros o para aquellos en los que se maneja el fichero completo** (para todo, vamos).
- Se **permite acceso secuencial**, interesante cuando la tasa de actividad alta. Además leyendo por el campo clave.
- Se **permite el acceso directo**, usando para ello las tablas de índices.
- Se pueden **actualizar los registros** sin necesidad de crear fichero de copia
- **Ocupa más espacio en disco** que los ficheros secuenciales por el uso de índices
- Solo admite **sopores direccionales** (direccionalables)
- Es más **caro**: Necesita hardware y software más sofisticado.

1.8 Ficheros de acceso calculado o hash

Con el acceso calculado o hash los accesos en ficheros indexados pueden ser más rápidos porque en lugar de consultar una tabla **se usa una transformación o función matemática conocida (hash) que a partir de la clave genera la dirección de cada registro del archivo**. Si la clave es alfanumérica debe previamente ser transformada en un número.

Este tipo de ficheros presenta como problema que a partir de diferentes claves se obtenga la misma dirección al aplicar la función matemática o transformación (**colisión**). Las claves que generan la misma dirección (**sinónimos**). Para evitarlo se aplican métodos como tener un bloque de excedentes o zona de sinónimos, crear archivo de sinónimos...

Algunos de los métodos de transformación son:

- **Módulo**. Dirección igual al resto de división entera entre clave y número de registros
- **Extracción**. Dirección igual a una parte de las cifras que extraen la clave.

Los buenos hash producen el menor número de colisiones. A ser posible, biunívoca (correspondencia uno a uno). Obtiene un número entre 1 y n, siendo n el número de direcciones del fichero.

2. Bases de datos

Los ficheros de almacenamiento provocan que las aplicaciones pierdan independencia y surjan inconvenientes como información duplicada, incoherencia de datos, fallos de seguridad... Se dio paso a los sistemas basados en ficheros a los sistemas gestores de bases de datos.

Base de datos: Colección de datos relacionados lógicamente entre sí, con una definición y descripción comunes y estructurados de una determinada manera. Representa entidades y sus interrelaciones, almacenados con la mínima redundancia y **posibilitando acceso eficiente** por aplicaciones y usuarios.

Es el conjunto de datos de distinto tipo relacionado entre sí, junto con un programa de gestión de dichos datos

La base de datos consta de:

- **Entidades:** Objeto real o abstracto con características que lo diferencian de otros datos, del cual se almacena información. (Doctor, consulta..)
- **Atributos:** Datos que se almacenan en la entidad. Cualquier propiedad o característica de esta. (Nombre, apellido, hora...)
- **Registros:** Es donde se almacena la información de cada entidad. Conjunto de datos que contienen atributos de una repetición de entidad. (Jesús López Navas 23/03/2010 ...)
- **Campos:** Dónde se almacenan los atributos de cada registro. "Jesús"

2.1. Ventajas

- **Acceso múltiple:** Acceso simultáneo de usuarios y aplicaciones
- **Utilización múltiple:** Cada usuario o aplicación dispone de una visión de la estructura, solo accede a lo que le corresponde.
- **Flexibilidad:** El acceso puede establecerse de diferentes formas. Tiempo de respuesta reducido.
- **Confidencialidad y seguridad:** Control de acceso. Evita acceso de usuarios no autorizados.
- **Protección contra fallos:** Hay mecanismos definidos para recuperar datos de forma fiable.
- **Independencia física:** Cambio de soporte físico no afecta a bases de datos o aplicaciones.
- **Independencia lógica:** Cambios de la base de datos no afecta a las aplicaciones
- **Redundancia:** Datos almacenados una sola vez (salvo casos de necesidad)
- **Interfaz de alto nivel:** Manejo con lenguajes de alto nivel de forma cómoda
- **Consulta directa:** Herramientas interactivas para acceder a los datos

2.2. Usos

Veamos los cuatro roles posibles en los usuarios de bases de datos:

Rol	Funciones
Administrador	Persona encargada de crear (implementar físicamente) la base de datos . Escoge tipos de ficheros, índices, ubicación... Toma las decisiones relacionadas con el funcionamiento físico , considerando el sistema que lo va a usar. Establece política de seguridad y acceso .
Diseñador	Identifican los datos, relaciones entre ellos, restricciones... Deben conocer a fondo los datos y procesos a representar en la base de datos, conociendo las reglas de negocio . El diseñador debe implicar en el proceso a todos los usuarios de la base de datos.
Programador de aplicaciones	Implementan los programas de aplicación que servirán a los usuarios finales. Posibilitan consultas, inserción, actualización o eliminación (CRUD). Usan lenguajes de tercera o cuarta generación (C, Fortran, Smalltalk, Ada, C++, C#, Cobol, Delphi, Java...)
Usuarios finales	Clientes finales. Al implementar, diseñar, mantener se busca cumplir los requisitos establecidos por el usuario final para gestionar su información.

Las bases de datos son usadas en Banca (clientes, cuentas, transacciones); Líneas aéreas (clientes, horarios, vuelos); Universidades (estudiantes, carreras); Telecomunicaciones (llamadas, saldo);

Medicina; Justicia; Legislación; Organismos; Posicionamiento geográfico; Hostelería y turismo; Ocio; Cultura...

2.3. Ubicación de la información

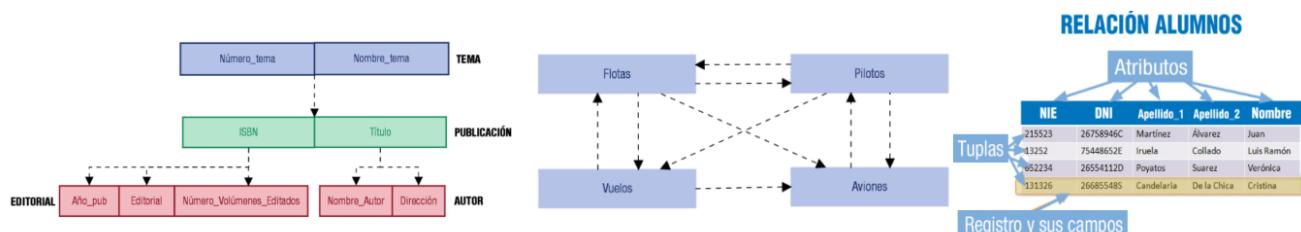
El tamaño de las bases de datos puede ser muy diferente. Pero todas se suelen almacenar en discos duros y otros dispositivos de almacenamiento.

A veces una gran base de datos podría necesitar servidores en lugares diferentes.

Los más usados:

- **Discos SATA:** (Serial Advanced Technology Attachment) Interfaz de transferencia entre la placa base y los dispositivos de almacenamiento (disco duro, cd, dvd, unidades de estado sólido). Proporciona mayor velocidad y aprovechamiento con varias unidades, mayor longitud del cable de transmisión, capacidad de conectar las unidades al instante sin tener que apagar el ordenador. Transferencias de 150 Megabytes por segundo (SATA 150 MB/s, Serial ATA-150). Hay dispositivos SATA II a 300 MB/s (Serial ATA-300) y SATA III hasta 600 MB/s
- **Discos SCSI:** (Small Computer System Interface): Para discos duros con gran capacidad de almacenamiento y velocidad de rotación. SCSI Estándar, SCSI Rápido, SCSI Ancho-Rápido (FastWide). Tiempo medio de acceso 7 milisegundos, velocidad de transmisión secuencial 5MB/s en estándar; 10 MB/s en Rápidos, 20 MBps en FastWide. El controlador SCSI puede manejar hasta 7 discos SCSI.
- **RAID:** (Redundant Array of Independent Disk) Matriz de discos independientes. Basada en almacenamiento redundante. Se montan dos o más discos duros formando un bloque de trabajo para obtener ampliación de capacidad, mejor velocidad, seguridad de almacenamiento. Hay varios sistemas de RAID:
- **Sistemas NAS:** (Network Attached Storage): Almacenamiento masivo en red. Permiten compartir la capacidad del servidor con ordenadores personales o clientes a través de la red. Hay sistema operativo optimizado para dar acceso a los datos por protocolos específicos. Dispositivos con capacidades muy altas y conectan por red.
- **Sistemas SAN:** (Storage Area Network): Red de área de almacenamiento. Red para conectar servidores, arrays de discos y librerías de soporte. Los recursos están disponibles para varios servidores en red de área local amplia. La información no reside en ninguno de los servidores. Se optimiza el poder de procesamiento para aplicaciones comerciales y se le puede proporcionar capacidad al servidor que más lo necesite.

3. Modelos de bases de datos



Principales tipos de Bases de Datos

Los principales tipos de Bases de Datos son:

- Relacionales: la información que almacena la Base de Datos está relacionada entre sí. Los datos relacionados (registros o filas) son almacenados en tablas que constan de varios campos (columnas).
- No relacionales: los datos no tienen porque estar relacionados entre sí y por lo tanto no tienen que almacenarse en estructuras fijas como las tablas del modelo de base de datos relacional.

Las Bases de Datos NoSQL pertenecen al modelo no relacional. Las principales características y ventajas de este tipo son:

- SQL no es el lenguaje de consulta/modificación de datos principal, aunque sí lo soportan, de ahí el nombre No Sólo SQL.
- Los datos no tienen que almacenarse en tablas.

- Generalmente, su arquitectura es distribuida almacenándose la información en más de una máquina del sistema. Por lo tanto, los sistemas que las soportan tienen una mayor escalabilidad horizontal (a mayor número de nodos mayor rendimiento) y también mayor tolerancia ante fallos en los distintos nodos.
- Son más eficientes en el procesamiento de los datos que las Bases de Datos relacionales
- Son más eficientes en el procesamiento de los datos que las Bases de Datos relacionales, por eso son la elección para aplicaciones que hacen un uso intensivo de estos ("streaming", etc.).
- Utilizan lo que se conoce como consistencia eventual que consiste en que los cambios realizados en los datos serán replicados a todos los nodos del sistema, lo cual aumenta el rendimiento de estos sistemas en contraposición a las propiedades ACID de las Bases de Datos relacionales.

¿Por qué JSON en bases de datos relacionales?

Las bases de datos Not only SQL, se basan en un esquema flexible de datos, en los cuales no necesitas declarar o crear primero dicho esquema para comenzar a almacenar información como tampoco es estrictamente necesario el proceso de normalización. Dentro de la industria desde hace años, llegaron a irrumpir las bases de datos de tipo documento tales como MongoDB, las cuales mostraron que al no estar amarradas al esquema tradicional de SQL, podían ofrecer una velocidad de escritura y lectura aún muy superior a lo manejado por las bases de datos relacionales; sin embargo esa realidad se ha vuelto a modificar gracias a los últimos esfuerzos de MySQL gracias a su implementación nativa para guardar, modificar y eliminar datos en formato JSON.

3.1. Modelo jerárquico o en árbol

La información se organiza en una jerarquía en la que la relación de entidades siempre es de padre/hijo. Sigue estructura de modelo de árbol invertido. Hay nodos con atributos o campos. Un nodo puede tener más de un hijo pero solo un padre.

Los datos se almacenan en **estructuras lógicas (segmentos)** que se relacionan entre sí usando **arcos**.

IBM creó un sistema administrador de información (IMS) con las bases del modelo que la mayoría de SGI de los setenta adoptaron. Este modelo están en desuso actualmente.

3.2. Modelo en red (Bases de datos de primera generación)

La información se organiza en **registros (nodos)** y **enlaces**. En los registros se almacenan los datos. Los enlaces permiten relacionar estos datos.

Parecido con las jerárquicas pero **pueden tener más de un parente**.

Aparece en los sesenta como respuesta a los límites del modelo jerárquico. IDS de Bachman es el primer modelo de base de datos en red. Se trató de crear un estándar por parte de CODASYL, con gran aceptación.

3.3. Modelo relacional (Bases de datos de segunda generación)

- Desarrollado por Codd en 1970. Las más utilizadas actualmente.
- El usuario lo percibe como un conjunto de tablas (nivel lógico). A nivel físico, en cambio puede tener diferentes estructuras de almacenamiento.
- Usa **tablas bidimensionales** (relaciones) para representación lógica de datos y relaciones. Cada tabla (entidad) posee nombre único y contiene un conjunto de columnas
- Se llama registro, entidad, ocurrencia de entidad o tupla a cada registro de la tabla y campo o atributo a cada columna
- Conjunto de valores que puede tomar un atributo es el dominio
- La clave es un conjunto de atributos que identifica de forma única a una tupla.

Las tablas deben cumplir:

- Todos los registros son del mismo tipo
- Solo puede tener un tipo de registro

- No hay campos o atributos repetidos
- No hay registros duplicados
- No hay orden en el almacenamiento de registros
- Cada registro o tupla se identifica por clave compuesta por uno o varios atributos

Las consultas se construyen en SQL.

En su diseño tiene gran relevancia la **normalización**. Consiste en definir las reglas que determinan las dependencias entre los datos de una base de datos relacional. Si definimos esta relación o dependencia entre los elementos de una determinada base de datos de la manera más sencilla posible, conseguiremos que la cantidad de espacio necesario para guardar los datos sea el menor posible y la facilidad para actualizar la relación sea la mayor posible. Es decir, optimizaremos su funcionamiento.

3.4. Modelo orientado a objetos (Bases de datos de tercera generación)

Define la base de datos en términos de objetos: propiedades y operaciones. Objetos con misma estructura y comportamiento pertenecen a una clase y las clases se organizan en jerarquías. Las operaciones de cada clase son métodos.

Hay sistemas basados en el modelo relacional que han evolucionado para incorporar objetos (sistemas objeto-relacionales).

Se basa como toda la POO en:

- **Encapsulación.** Se oculta información al resto de objetos
- **Herencia.** Los objetos heredan comportamientos en la jerarquía de clases
- **Polimorfismo.** Una operación puede ser aplicada a distintos tipos de objetos.

3.5. Modelo NoSQL

Como ventajas tienen:

- Pueden ejecutarse en **máquinas con pocos recursos**
- Tienen **escalabilidad horizontal** (simplemente se añaden más nodos)
- Pueden manejar **gran cantidad de datos** (estructura distribuida, tablas hash)
- **No generan cuellos de botella.** No necesitan transcribir sentencias SQL para ejecutarlas en el punto de entrada.

Diferencias:

- **No usan SQL** o solo lo usan como pequeño apoyo
- **No usan estructuras fijas de tablas.** Usan otros tipos de almacenamiento como clave-valor, objetos, grafos.
- **No permiten operaciones JOIN.** Deben desnormalizarse los datos o realizar JOIN por software
- **Arquitectura distribuida.** Puede estar compartida en varias máquinas con mecanismos hash

Tipos:

- **Bases de datos clave-valor:** El modelo más popular y más sencillo. Cada elemento está identificado por llave única que permite la recuperación de la información de forma rápida (almacenada en objeto binario BLOB). Son eficientes en lectura y escritura. Ej.: Cassandra, Big Table, HBase.
- **Bases de datos documentales:** Información almacenada como documento JSON o XML. Se usa clave única para cada registro. Se pueden hacer búsquedas por clave-valor y también por contenido. Son más versátiles. Pueden emplearse en proyectos que tradicionalmente funcionaban con BBDD relacionales. Ej.: MongoDB, CouchDB.
- **Bases de datos en grafo:** Información representada como nodos de un grafo y relaciones son aristas del mismo. Se puede usar la teoría de grafos para recorrerla. Ofrecen negación eficiente. Ej.: Neo4j, InfoGrid, Virtuoso.

Las más usadas actualmente son:

Cassandra: Apache. Tipo clave-valor. Lenguaje propio (CQL). En Java.

Redis: Tipo clave-valor. Como array gigante para almacenar datos (cadenas, hash, conjuntos, listas)

MongoDB: Orientada a documentos. De esquema libre. Rápido ya que está en C++.

CouchDB: Apache. En GNU/Linux. Javascript como lenguaje de interacción. Permite creación de vistas para retornar valores de varios documentos. Permite uso de JOIN.

3.6. Otros modelos

Bases de datos objeto-relacionales: Híbrido entre relacionales y orientadas a objetos. Así se evita el coste de conversión de relacionales a orientadas a objetos. Se usa lo bueno del modelo relacional incorporando los objetos. Se almacenan tuplas aunque la estructura no está restringida sino que pueden definirse a partir de otras (herencia directa). Se basa en SQL99 (añadir procedimientos almacenados de usuario, triggers, tipos definidos, consultas recursivas, bases de datos OLAP , tipos LOB...). Permite incorporar funciones de lenguajes como SQL, Java, C. Las bases relacionadas clásicas de gran tamaño Oracle, SQL Server son objeto-relacionales.

Bases de datos deductivas o lógicas: Almacenan información y realizan deducciones a través de inferencias (derivan nuevas informaciones a partir de las existentes en base de datos introducidas por el usuario). Contrarrestan limitaciones del modelo relacional en respuesta a consultas recursivas y deducción de relaciones indirectas entre datos.

Bases de datos multidimensionales: Tienen varias dimensiones. En vez de un valor se encuentran varios dependiendo de los ejes definidos o con estructura orientada a consultas complejas y alto rendimiento. Matrices multidimensionales, cuadros de múltiples entradas, funciones de varias variables sobre conjuntos finitos (cubo). Facilita manejo de grandes cantidades de datos en una empresa.

Bases de datos transaccionales: Velocidad para gestionar el intercambio de información. Muy fiables. Sistemas bancarios, análisis de calidad y producción industrial.

4. Tipos de bases de datos

1. Segundo su contenido

1. Con información actual: Información muy concreta y actualizada, de tipo numérico (estadísticas, series históricas...)
2. Directorios: Datos sobre personas o instituciones (profesionales, investigadores, empresas, editoriales)
3. Bases de datos documentales: Cada registro es un documento (publicación impresa, documento audiovisual)
 1. Bases de datos de texto completo. Documentos en formato electrónico, volcado de su texto
 2. Archivos electrónicos de imágenes. Enlace directo con la imagen del documento original
 3. Bases de datos referenciales: No contienen el texto general sino solo información para describir y permitir la localización de los documentos impresos, sonoros, audiovisuales, electrónicos. Luego habrá que localizarlos por otro servicio (archivio, biblioteca, fonoteca..)

2. Segundo su uso

1. Individual: Mismo usuario.
2. Compartida
3. Acceso público
4. Propietarias o bancos de datos

3. Segundo la variabilidad de la información

1. Estáticas: Solo lectura. Datos históricos que pueden ser analizados
2. Dinámicas: Se modifica con el tiempo. Actualización, adición..

4. Segundo la localización de la información

1. Centralizadas: Datos se almacenan en un único punto
 1. Basada en anfitrión: Cliente y máquina servidor son la misma. Se conectan directamente a la máquina donde están los datos
 2. Basada en Cliente/Servidor. BBDD está en máquina servidor y acceden desde el cliente a través de red
2. Distribuidas Se almacenan en lugares diferentes

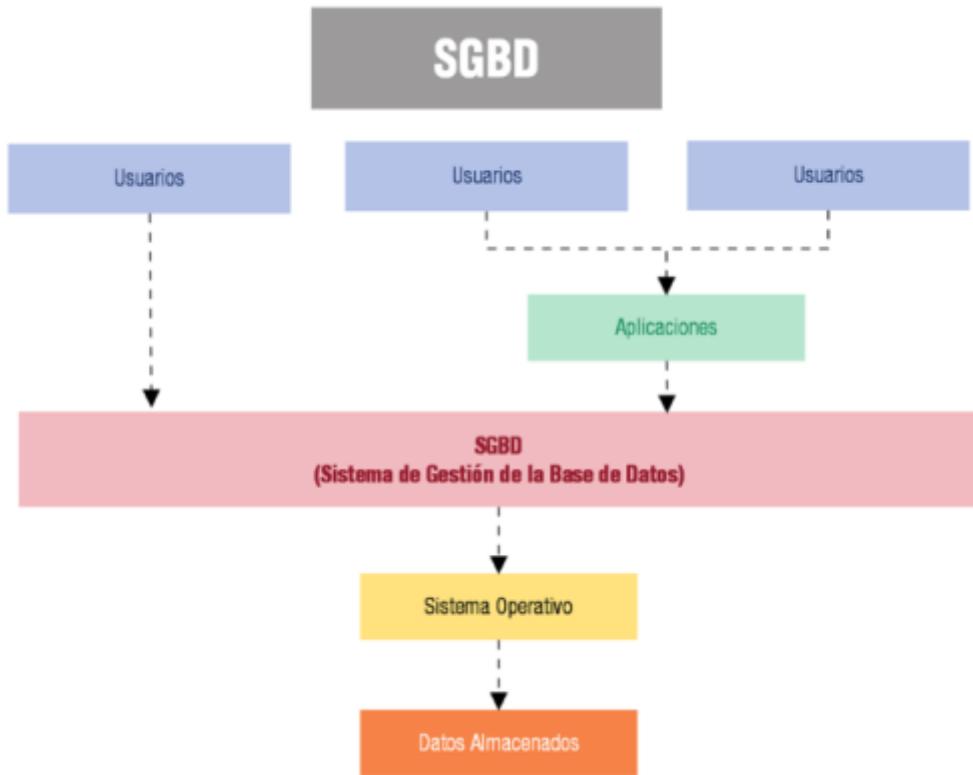
5. Según el organismo producto
 1. Organismos públicos y administración
 1. De acceso público
 2. De acceso interno
 2. Instituciones sin ánimo de lucro
 3. Entidades privadas o comerciales
 1. Uso interno para circulación de información dentro de la empresa
 2. Uso interno ocasionalmente ofreciendo servicios al exterior
 3. Comerciales (uso externo)
 4. Cooperación en red: Elaboración compartida por instituciones...
6. Modo de acceso
 1. Acceso local
 2. CD-ROM
 3. En línea
 1. Vía telnet o internet
 2. Vía web
7. Según cobertura temática
 1. Científico-tecnológicas
 1. Multidisciplinares
 2. Especializadas
 2. Económico-empresariales
 3. Medios de comunicación
 4. Político-administrativo y jurídico
 5. Sanitario
 6. Gran público

5. Sistemas gestores de bases de datos

Sistema Gestor de Bases de Datos: Conjunto coordinado de programas, procedimientos, lenguajes.. que suministra a cualquiera de los roles que usan la base de datos, los medios para describir y manipular los datos contenidos en ella manteniendo su integridad, confidencialidad y seguridad.

Ventajas:

- **Independencia física:** La visión del usuario y la manipulación es independiente de cómo está físicamente
- **Visión abstracta** de los datos, oculta esa complejidad.
- **Integridad**
- **Facilidad de acceso**, rapidez, evita pérdidas
- **Seguridad** y privacidad
- **Eficiencia**
- Accesos **concurrentes** y posibilidad de compartir datos
- Facilidad de **intercambio** entre sistemas
- **Copias de seguridad y recuperación** en caso de fallo
- El **SGBD interacciona con el sistema operativo**. Los datos almacenados serán usados por otras aplicaciones, el SGBD ofrecerá facilidad a estas para el acceso y manipulación de la información basándose en los métodos del sistema operativo.



5.1. Funciones

Tres funciones fundamentales:

Función de descripción o definición: El diseñador de la BBDD puede crear estructuras apropiadas para integrar los datos (estructura, relaciones, restricciones). Permite definir la estructura interna, conceptual y externa. Se hace mediante el lenguaje de definición de datos o DDL.

A nivel interno: Espacio de disco reservado para la BBDD, longitud de campos, modo de representación (lenguaje para definición de estructura externa)

A nivel conceptual: Definición de entidades, atributos, relaciones, restricciones...

A nivel externo: Vistas de usuarios

Función de manipulación: Permite buscar, añadir, suprimir, modificar datos de la misma. Mediante el lenguaje de manipulación de datos o DML. También se define la vista externa de los usuarios de la base de datos o vistas parciales que cada usuario tiene de los datos.

Se entiende por manipulación el CRUD.

Función de control: El administrador establece mecanismos de protección. Creación y modificación se usuarios, sistemas para crear copias de seguridad, cargas de ficheros, auditoría, protección de ataques, configuración. Se hace mediante el lenguaje de control de datos o DCL.

DDL, DML, DCL mediante el SQL.

5.2. Componentes

El SGBD es un paquete software complejo que proporciona servicios para almacenamiento y explotación de datos de forma eficiente. Se compone de:

1. **Lenguajes de la base de datos:** Uso de lenguajes e interfaces para los diferentes tipos de usuarios. Especificar datos, estructura, relaciones, reglas, control de acceso, características. DDL, DML, DCL.
2. **Diccionario de datos:** Descripción de los datos almacenados. Lugar donde se deposita la información sobre los datos que forman. Interesante para programadores de aplicaciones. Características lógicas de las estructuras que almacenan los datos, nombre, descripción, contenido, organización. En BBDD relacionales: Definición de tablas, vistas, índices, disparadores, procedimientos, funciones. Espacio asignado y usado. Restricciones. Permisos. Auditoría.

3. **Gestor de la base de datos:** Garantizar acceso correcto, seguro, íntegro y eficiente. Proporciona interfaz entre los datos y los programas que los manejan. Interactúa con el sistema operativo. Garantiza los accesos concurrentes...
4. **Usuarios de la base de datos:** Con diferentes permisos. Administrador de base de datos (DBA) tiene el control centralizado y es responsable del buen funcionamiento, de autorizar, de vigilar su uso, de adquirir software y hardware necesario. Usuarios con diferentes necesidades, accesos y privilegios (diseñadores, operadores y mantenimiento, analistas y programadores, usuarios finales ocasionales simples avanzados autónomos)
5. **Herramientas de la base de datos:** Conjunto de aplicaciones que permiten a los administradores la gestión de bbdd, usuarios, permisos, generar formularios, informes, interfaces gráficas, aplicaciones.

5.3. Arquitectura

Los estándares que han cobrado más importancia son ANSI/SPARC/X3, CODASYL, ODMG (este para las de objetos). ANSI e ISO son el referente de estandarización conformando un único modelo.

Nivel interno o físico: Se describe estructura física por un esquema que describe el sistema de almacenamiento y sus métodos de acceso. Es el más cercano al almacenamiento físico. Archivos que contienen la información, organización, métodos de acceso, tipos de registro, longitud, campos.

Nivel lógico o conceptual: Estructura completa de la BBDD. Esquema entidades, atributos, relaciones, operaciones de usuarios, restricciones.

Nivel externo o de visión de usuario: Vistas que los usuarios perciben. Solo la parte que les interesa, ocultando el resto.

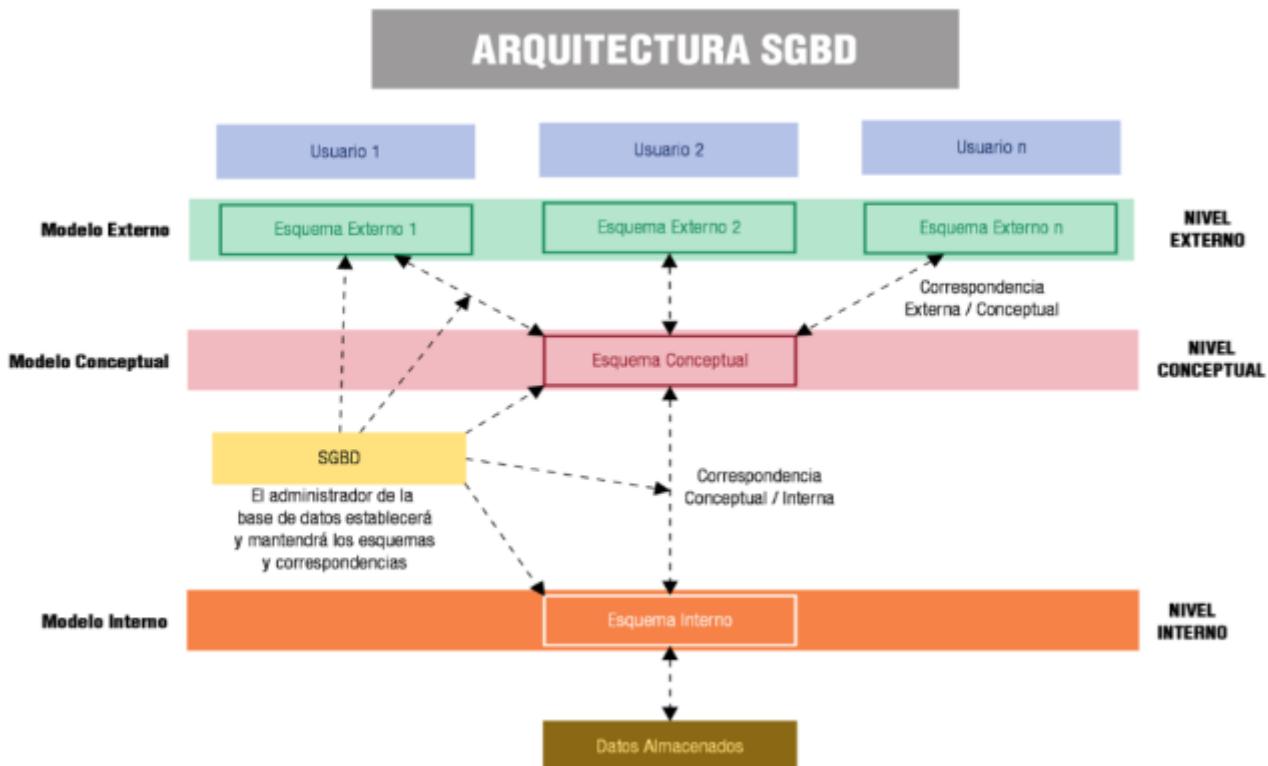
Existirá un único esquema interno, un único esquema conceptual y varios esquemas externos para uno o varios usuarios.

Esto garantiza:

Independencia lógica: Modificar esquema conceptual sin alterar esquemas externos, ni programas.

Independencia física: Modificar esquema interno sin modificar el conceptual o el externo.

(Reorganizar sistema, ficheros...)



5.4. Tipos

Según el modelo lógico en que se basan: Relacional, orientado a objetos, jerárquico, en red

Según el número de usuarios: monousuario, multiusuario

Según el número de sitios en los que se distribuye la base de datos: centralizado, distribuidos

(homogéneos que usan el mismo SGBD o heterogéneos con acceso a varias BBDD autónomas preexistentes dando lugar a SGBD federados o sistemas multibase de datos en los que los SGBD participantes tienen cierta autonomía local)

Coste: Entre 10000 y 100000 euros. Los más económicos mono usuarios entre 0 y 3000. Los más completos más de 100000 euros.

Propósito: General (cualquier tipo de BBDD y aplicación) o específico cuando el rendimiento es fundamental y se necesita para una aplicación específica (ejemplo sistemas de procesamiento de transacciones en líneas en reservas o cosas así)

5.5. SGBD comerciales

SGBD	Descripción
ORACLE	Reconocido como uno de los mejores a nivel mundial. Es multiplataforma, confiable y seguro. Es Cliente/Servidor. Basado en el modelo de datos Relacional. De gran potencia, aunque con un precio elevado hace que sólo se vea en empresas muy grandes y multinacionales. Ofrece una versión gratuita Oracle Database Express Edition 11g Release 2.
MYSQL	Sistema muy extendido que se ofrece bajo dos tipos de licencia, comercial o libre. Para aquellas empresas que deseen incorporarlo en productos privativos, deben comprar una licencia específica. Es Relacional, Multihilo, Multiusuario y Multiplataforma. Su gran velocidad lo hace ideal para consulta de bases de datos y plataformas web.
DB2	Multiplataforma, el motor de base de datos relacional integra XML de manera nativa, lo que IBM ha llamado pureXML, que permite almacenar documentos completos para realizar operaciones y búsquedas de manera jerárquica dentro de éste, e integrarlo con búsquedas relacionales.
Microsoft SQL SERVER	Sistema Gestor de Base de Datos producido por Microsoft. Es relacional, sólo funciona bajo Microsoft Windows, utiliza arquitectura Cliente/Servidor. Constituye la alternativa a otros potentes SGBD como son Oracle, PostgreSQL o MySQL.
SYBASE	Un DBMS con bastantes años en el mercado, tiene 3 versiones para ajustarse a las necesidades reales de cada empresa. Es un sistema relacional, altamente escalable, de alto rendimiento, con soporte a grandes volúmenes de datos, transacciones y usuarios, y de bajo costo.
Otros SGBD comerciales importantes	DBASE, ACCESS, INTERBASE y FOXPRO

5.6. SGBD libres

SGBD	Descripción
MySQL	Es un sistema de gestión de base de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones. Distribuido bajo dos tipos de licencias, comercial y libre. Multiplataforma, posee varios motores de almacenamiento, accesible a través de múltiples lenguajes de programación y muy ligado a aplicaciones web.
PostgreSQL	Sistema Relacional Orientado a Objetos. Considerado como la base de datos de código abierto más avanzada del mundo. Desarrollado por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre y/o apoyados por organizaciones comerciales. Es multiplataforma y accesible desde múltiples lenguajes de programación.
Firebird	Sistema Gestor de Base de Datos relacional, multiplataforma, con bajo consumo de recursos, excelente gestión de la concurrencia, alto rendimiento y potente soporte para diferentes lenguajes.
Apache Derby	Sistema Gestor escrito en Java, de reducido tamaño, con soporte multilenguaje, multiplataforma, altamente portable, puede funcionar embebido o en modo cliente/servidor.

SGBD	Descripción
SQLite	Sistema relacional, basado en una biblioteca escrita en C que interactúa directamente con los programas, reduce los tiempos de acceso siendo más rápido que MySQL o PostGreSQL, es multiplataforma y con soporte para varios lenguajes de programación.

Otro SGBD libre importante es MariaDB,

7. Bases de datos centralizadas

SGBD centralizado: SGBD implantando en una sola plataforma (mainframe) desde la que se gestiona directamente la totalidad de los recursos. Los centros de procesos de datos tradicionales funcionan con esta arquitectura.

- Son tecnologías sencillas, experimentadas, robustas.
- No hay múltiples elementos de procesamiento, ni comunicaciones entre BBDD
- Se componen de: Los datos, el software de gestión de bases de datos y los dispositivos de almacenamiento asociados
- Su seguridad podría verse comprometida más fácilmente.

Ventajas	Inconvenientes
Evita redundancia (No inconsistencias, no desperdicio de espacio)	Menor poder de cómputo que una distribuida
No inconsistencia (Una sola entrada representa el hecho)	Si falla, se pierde el procesamiento y la información de todo el sistema
Seguridad centralizada	En caso de un desastre o catástrofe, la recuperación es difícil de sincronizar.
Integridad	Las cargas de trabajo no se pueden difundir entre varias computadoras, ya que los trabajos siempre se ejecutarán en la misma máquina.
Mejor rendimiento en procesamiento	Los departamentos de sistemas retienen el control de toda la organización.
Más barato	Necesario mantenimiento central de datos

8. Bases de datos distribuidas

Base de datos distribuida (BDD): Conjunto de múltiples bases de datos lógicamente relacionadas que se encuentran distribuidas entre diferentes nodos interconectados por una red.

Sistema de bases de datos distribuida (SBDD): Sistema en el que múltiples sitios de bases de datos están ligados por un sistema de comunicaciones. El usuario en cualquier sitio puede acceder a los datos de cualquier parte de la red.

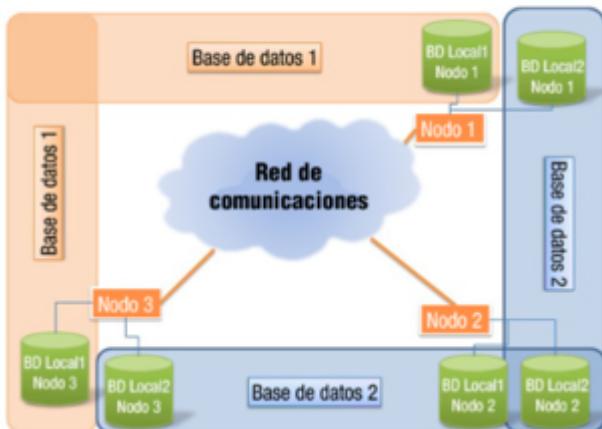
Sistema gestor de bases de datos distribuida (SGBDD): Se encarga del manejo de la BDD dando un mecanismo de acceso que hace que la distribución sea transparente (inapreciable) a los usuarios.

El SGBDD trabaja a través de un conjunto de sitios o nodos con sistema de procesamiento de datos completo con base de datos local, sistema gestor de bases de datos e interconectados entre sí. Si están muy dispersos (Red WAN, Wide Area Network, Red de área amplia), si están cerca (Red LAN, Local Area Network, Red de área local)

Ventajas	Inconvenientes
Acceso y procesamiento de los datos es más rápido (varios nodos comparten carga de trabajo)	La probabilidad de violaciones de seguridad es creciente si no se toman las precauciones debidas.
Desde un lugar puede accederse a información alojada en diferentes lugares.	Complejidad añadida que es necesaria para garantizar la coordinación apropiada entre los nodos.

Ventajas	Inconvenientes
Más barato que centralizadas	La inversión inicial es menor, pero el mantenimiento y control pueden resultar costosos.
Tolerancia a fallos. Mediante la replicación, si un nodo deja de funcionar el sistema completo no deja de funcionar.	Control de concurrencia y los mecanismos de recuperación son mucho más complejos (los datos pueden estar replicados)
Se adapta más naturalmente a la estructura de las organizaciones. Permiten la incorporación de nodos de forma flexible y fácil.	El intercambio de mensajes y el cómputo adicional necesario para conseguir la coordinación entre los distintos nodos constituyen una forma de sobrecarga.
Aunque los nodos están interconectados, tienen independencia local.	Dada la complejidad del procesamiento entre nodos es difícil asegurar la corrección de los algoritmos, el funcionamiento correcto durante un fallo o la recuperación.

BASES DE DATOS RELACIONALES DISTRIBUIDAS, HOMOGÉNEAS Y ALTAMENTE INTEGRADAS



8.1. Fragmentación

Extraer los datos en SGBDD se hace mediante fragmentación de distintas tablas que están en diferentes lugares.

Se llama **problema de fragmentación** al particionamiento de la información para distribuir cada parte a los diferentes sitios de la red. Encontrar el nivel de particionamiento adecuado.

Debe considerarse el **grado de fragmentación** a aplicar porque afectará a las consultas. Sin fragmentación las relaciones o tablas se toman como unidad de fragmentación. También puede fragmentarse a nivel de tupla (fila, registro) o a nivel de atributo (columna, campo) de una tabla. La fragmentación no debe ser ni nula, ni demasiado alta. Debe estar equilibrado y contemplar la casuística de las aplicaciones.

Reglas de la fragmentación

Si la relación R se descompone en R1, R2,...,Rn

- **Completitud:** cada elemento de R debe estar en algún fragmento de Ri.
- **Reconstrucción:** la reconstrucción de la relación a partir de los fragmentos debe asegurar que se preservan las restricciones
- **Disyunción:** si la relación se descompone verticalmente, sus atributos primarios se repiten en todos los fragmentos.

Tipos de fragmentación:

- **Fragmentación horizontal:** Sobre las tuplas (registros). La relación tiene subrelaciones que contienen un subconjunto de las tuplas de la primera. Existen la primaria y la derivada.
- **Fragmentación vertical:** Sobre los atributos. Tiene subconjuntos de atributos de R y la clave primaria de R. Así se partitiona en conjunto de relaciones más pequeñas. La fragmentación

óptima tiene un esquema que minimiza el tiempo de consultas del usuario. Es más complicada que la horizontal porque hay gran cantidad de alternativas posibles.

- **Fragmentación híbrida o mixta:** Ambas combinadas. Primero una vertical y luego horizontal (HV). Al contrario (VH). Para representarlo se usan árboles.

9. Primeros pasos en MySQL Server

MySQL es un sistema gestor de base de datos simple y de buen rendimiento. Es software libre (GNU GPL). Tiene estabilidad y alto grado de desarrollo.

Está realizado en C/C++. Hay ejecutables para diecinueve plataformas. Hay API para C,C++,Eiffel, Java, Perl, PHP, Python, Ruby, TCL. Optimizado para múltiples procesadores. Rápido. Puede usarse como cliente servidor. Tiene una administración de usuarios y privilegios. Puede conectarse por TCP/IP, Sockets UNIX, sockets NT, soporta ODBC.

Se descarga y se instala.

Se puede instalar después MySQL Workbench, herramientas de diseño de bases de datos que integra desarrollo de software, administración de BBDD, diseño de BBDD, creación y mantenimiento del SGBD MySQL.

Se descarga y se instala y listo.

10. Primeros pasos en Oracle Database

Oracle Database Express Edition 11g Release 2 es un sistema de bases de datos libre para desarrollo, implementación y distribución basado en Oracle.

Buen sistema para desarrolladores, para administradores de bases de datos, para proveedores independientes, estudiantes que necesiten practicar, adiestramiento, base de datos inicial...

UD 02 – BASES DE DATOS RELACIONALES

1. Fases de análisis y de diseño de bases de datos relacionales. El modelo E/R en el esquema conceptual ubicado en la fase de análisis.

Modelo de datos: Conjunto de herramientas conceptuales que permiten describir los datos, sus relaciones, su semántica (significado) y las restricciones aplicables.

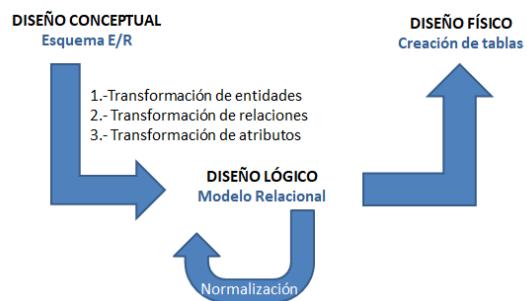
Los SGBD proponen una **arquitectura** que simplifica el modelo de datos a sus usuarios. Se proponen tres niveles de abstracción:

- Interno o físico
- **Lógico o conceptual**
- Externo o de visión de usuario

El **nivel lógico o conceptual** describe la estructura completa de la base de datos a través del **esquema conceptual**: Representa la información de forma independiente al SGBD.

Fase de análisis	Fase de diseño
Análisis de entidades: Localizar y definir entidades y atributos	Diseño de tablas
Análisis de relaciones entre entidades	Normalización
Esquema conceptual a través del modelo E-R	Retrodiseño si se necesita (modificación del esquema conceptual para mantener la equivalencia entre entidades y tablas)
Fusión de vistas: Reunir en un único esquema todos los existentes según las vistas de cada perfil de usuario	Diseño de transacciones: Localizar transacciones que operan en el esquema conceptual
Aplicación de enfoque de datos relacional	Diseño de sendas de acceso: Métodos de acceso a la estructura de datos

Suprimir el esquema conceptual puede conllevar pérdida de información respecto al problema real a solucionar! En él deben estar todos los aspectos del mundo real que se va a modelar.



Modelo E/R en el diseño del esquema conceptual

Peter Chen describe el **modelo Entidad/Relación (ER)** (1976-1977). Con el tiempo este modelo ha sido mejorado al **modelo de Entidad/Relación extendido (EER)**.

El **modelo de Entidad/Relación**:

- Modelo semántico, Representa el significado de los datos

- No está orientado a ningún sistema físico concreto ni tiene ámbito informático puro de aplicación (puede describir procesos de producción, estructuras...)
- Permitiría representar cualquier tipo de sistema y a cualquier nivel de abstracción

2. Entidades

Entidad: Es un objeto real o abstracto con características que lo diferencia de otros.

(Simil: Instancia)

Conjunto de entidades: Entidades con las mismas características o propiedades (Simil: Clase)

Por lo general se usa “entidad” para identificar “conjuntos de entidades”.

A cada elemento del conjunto de entidades (“entidad”) se le llama “ocurrencia de entidad”, “instancia de entidad”

Las entidades se nombran con sustantivos en singular.

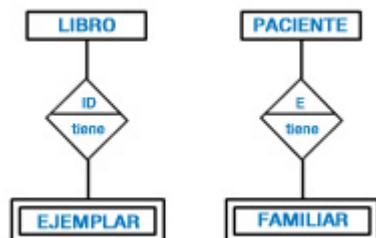
2.1. Tipos de entidades: Fuertes y débiles

- **Entidades fuertes o regulares:** Existen por sí mismas, no dependen de otras entidades. Ej.: DOCTOR no depende de la existencia de PACIENTE. Se presenta con el nombre de la entidad (en mayúscula) encerrado dentro de un rectángulo.
- **Entidades débiles:** Su existencia depende de una entidad fuerte con la que se relaciona. Ej.: EDIFICIO Y AULA. El código del aula no la identifica completamente. Necesitamos saber en qué edificio está localizada. Se representa como las entidades fuertes pero rodeadas de un rectángulo doble.



Dependencias de las entidades débiles:

- Dependencia en existencia: Si desaparece una entidad fuerte, desaparecen las entidades débiles que dependan de la primera. Se incluye “E” en el interior de la relación.
- Dependencia en identificación. Se da dependencia en existencia y además la entidad débil no puede identificarse por sí misma, debe hacerse mediante la clave de la entidad fuerte asociada. Se incluye “ID” en el interior de la relación.



3. Atributos

Atributo: Cada propiedad o característica que tiene una entidad o a una relación. Toman valores de uno o varios dominios.

Se representa con el nombre del atributo (en minúscula) rodeado por una elipse y conectándose con la entidad mediante una línea recta.

Dominio: Conjunto de valores permitidos para un atributo. Varios atributos pueden definirse dentro del mismo dominio (Ej.: Nombre, apellido primero, apellido segundo definidos en el dominio de cadenas de caracteres de determinada longitud). Deben establecerse límites adecuados para que el SGBD pueda realizar buenas verificaciones y garantice la integridad de los datos.



3.1. Clasificaciones de tipos de atributos

SEGÚN A QUIEN PERTENEZCAN:

- **Atributos de entidad:** Pertenecen a una entidad.
- **Atributos de relación:** Pertenecen a una relación.
Las entidades ALUMNO y ASIGNATURA pueden estar unidas por la relación CURSA. Esta puede tener un atributo nota que indique la nota que un alumno ha obtenido en determinada asignatura.

También sirven para **históricos**. Son relaciones que tienen datos como fecha y hora. Ej.: Al emitir una factura se registra el momento en el que se realiza esa acción. FACTURA EMITIDA CLIENTE. La relación EMITIDA posee el atributo fecha_emision.

OBLIGATORIEDAD

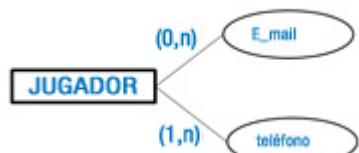
- **Atributo obligatorio:** Siempre debe estar definido. Las claves son atributos obligatorios. Ej.: dni
- **Atributo opcional:** Puede estar definido o no. Ej.: color_pelo. El opcional tiene la línea con círculo discontinuo.

COMPOSICIÓN

- **Atributo simple o atómico:** No puede dividirse en otras partes o atributos. Es un único elemento. No pueden extraerse partes más pequeñas... Ej.: dni
- **Atributo compuesto:** Pueden dividirse en subpartes que serían otros atributos con significado propio. Ej.: dirección que se divide en calle, numero, localidad

VALORES

- **Atributo monovaluado:** Solo un valor para cada entidad
- **Atributo multivaluado:** Diferentes valores para cada entidad. Ej.: Un EMPLEADO puede tener varios email. En ese caso debe considerarse la **cardinalidad** (número mínimo y máximo de valores que puede tomar para cada entidad o relación a la que pertenece). Cardinalidad mínima: Valores que deben existir para que la entidad sea válida (Suele ser 0 -puede no tener ningún valor- o 1 -al menos un valor-). Cardinalidad máxima: Cantidad máxima de valores (Suele ser 1 -debe tener como mucho 1- o n – puede tener múltiples valores-).
La cardinalidad se indica entre paréntesis (mínima,máxima) sobre la flecha que une el atributo a la entidad.



FORMA DE OBTENCIÓN

- **Atributo derivado:** Se puede obtener del valor de otros atributos almacenados. El derivado tiene una D en la flechas de unión o está con círculo discontinuo. Ej.: Edad
- **Atributo almacenado:** Debe almacenarse.

En cualquier caso, los valores de los atributos deben ser tales que permitan identificar únicamente a la entidad.

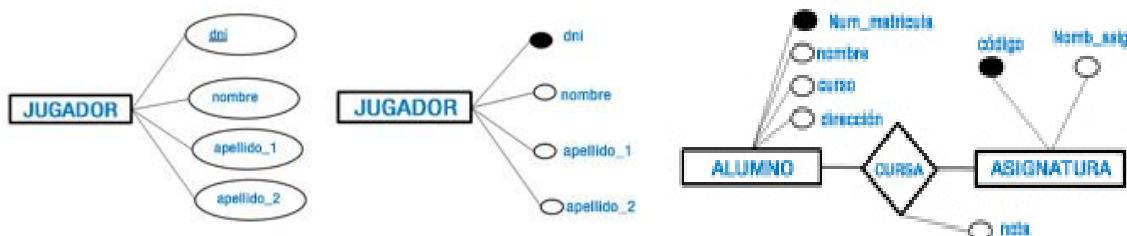
3.2. Claves

- **Superclave:** Conjunto de atributos que permite identificar de forma única a una entidad. Puede tener atributos no obligatorios (que por sí solos no identifican una entidad).
- **Clave candidata:** Superclave más pequeña que puede obtenerse en una entidad. (Cuando de una superclave no puede obtenerse otro subconjunto que sea superclave).
- **Clave primaria o principal (Primary Key):** Clave escogida de entre todas las claves candidatas. (Atributo o conjunto de atributos que toman valores únicos y distintos para cada entidad, identificándola únicamente. No puede tener valores nulos). La clave puede ser simple o compuesta de varios atributos.
- **Claves alternativas:** El resto de claves candidatas.

Representación

Si se usan elipses los atributos que forman la clave primaria aparecen subrayados. (Y si hay claves alternativas se podrían distinguir con IP -identificador principal-)

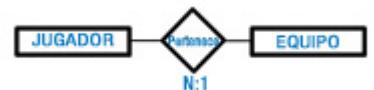
Si se usan círculos para representar los atributos, los atributos que forman la clave primaria tienen un círculo negro. Las claves alternativas tiene un hemicírculo negro.



4. Relaciones

La **relación** es una asociación entre diferentes entidades. No pueden aparecer dos veces relacionada en las mismas entidades.

Se representa con un rombo en cuyo interior está inscrito el nombre de la relación. Está conectado por líneas rectas y puede tener o no punta de flecha según el tipo de relación.



A veces habrá que indicar el papel o rol en las líneas que conectan las entidades con la relación. (Útil en relaciones con entidades reflexivas).

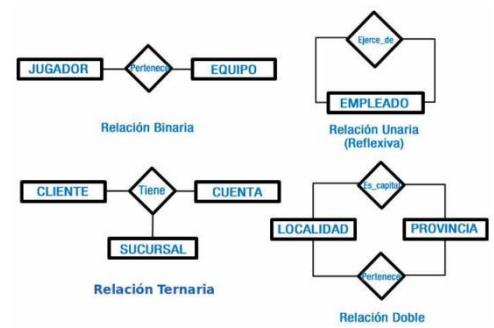
Debemos hablar del:

- Grado de una relación
- Cardinalidad de entidades
- Cardinalidad de relaciones

4.1. Grado de una relación

Número de entidades que participan en una relación. Puede ser:

- **Relación unaria** (Grado 1) (**Reflexivas o recursivas**): 1 entidad
- **Relación binaria** (Grado 2): 2 entidades. En general se busca tener esto en el esquema conceptual.
- **Relación ternaria** (Grado 3): 3 entidades.
- **Relación N-aria** (grado n): n entidades. No son usuales. Deben ser simplificadas a menor grado
- **Relación doble**: Dos entidades están relacionadas a través de dos relaciones. Son complejas de manejar.



4.2. Cardinalidad de relaciones

Cardinalidad de relación: Número de entidades que intervienen en una relación.

(Número máximo de ocurrencias de cada entidad que pueden intervenir en una ocurrencia de relación.)

- **Uno a uno (1:1):** Un A se relaciona solo con un B y viceversa. Ejemplo: ALUMNO posee EXPEDIENTE
- **Uno a muchos (1:N):** Un A se relaciona con muchos B. Pero un B solo se relaciona con un A. Ejemplo: DOCENTE imparte muchas ASIGNATURA. ASIGNATURA solo tiene un DOCENTE.
- **Muchos a uno (N:1):** Un A se relaciona con un B. Un B se relaciona con muchos A. Ejemplo: JUGADOR pertenece a un EQUIPO. Pero EQUIPO puede tener muchos JUGADOR.
- **Muchos a muchos (M:N):** Un A tiene muchos B y viceversa. Ejemplo: ALUMNO matriculado en muchas ASIGNATURA. ASIGNATURA tiene muchos ALUMNO.

La representación:

Nota: Cuando se representa con número, este se pone en el lado de la entidad a la que se está mencionando. (JUGADOR N:1 EQUIPO) . Corresponde a las cardinalidades máximas de las entidades (dadas la vuelta). Para representarla en el gráfico se pone, bajo el rombo



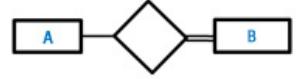
“cardinalidad_maxima_izquierda”: “cardinalidad_maxima_derecha”. Hay muchas más en verdad:

4.3. Cardinalidad de entidades

Número de relaciones en las que una entidad puede aparecer.

La participación de una entidad puede ser:

- Obligatoria (total): Cada ocurrencia necesita como mínimo de una ocurrencia de B. Se representaría con doble línea.
- Opcional (parcial): No es necesaria una ocurrencia de B.



Se representa con el número mínimo y máximo de correspondencias en las que puede tomar parte cada ejemplar de la entidad, entre paréntesis.
(cardinalidad_minima:cardinalidad_maxima).

Cardinalidad mínima: Número mínimo de asociaciones en las que aparece.
(Cero(opcional) o uno. Si tiene cardinalidad mínima de más de uno, se indica solo un uno.)

Cardinalidad máxima: Número máximo de asociaciones en las que aparece. Puede ser uno, otro valor concreto mayor que uno (3,4,5,...) o muchos (n).

Ejemplos

Entidades fuertes JUGADOR y EQUIPO.

JUGADOR pertenece como mínimo a ningún equipo y como máximo a un equipo. (0,1).

EQUIPO como mínimo tiene un jugador y como máximo muchos (1, n).

¡Se colocan al revés!

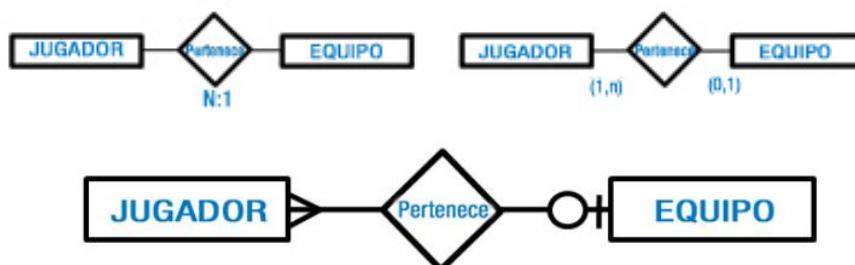
Para la cardinalidad de EQUIPO se coloca la cardinalidad junto a la entidad JUGADOR para expresar que el equipo debe tener mínimo un jugador y máximo varios.

Para la cardinalidad de JUGADOR se coloca la cardinalidad junto a EQUIPO para mostrar que el jugador puede pertenecer a ningún equipo o a uno.

Notación alternativa para representar cardinalidad de entidades

→	Muchos
+	Uno
○	De cero a muchos
←	De uno a muchos
○ +	De cero a uno

La cardinalidad máxima de las entidades equivale a la cardinalidad de la relación.

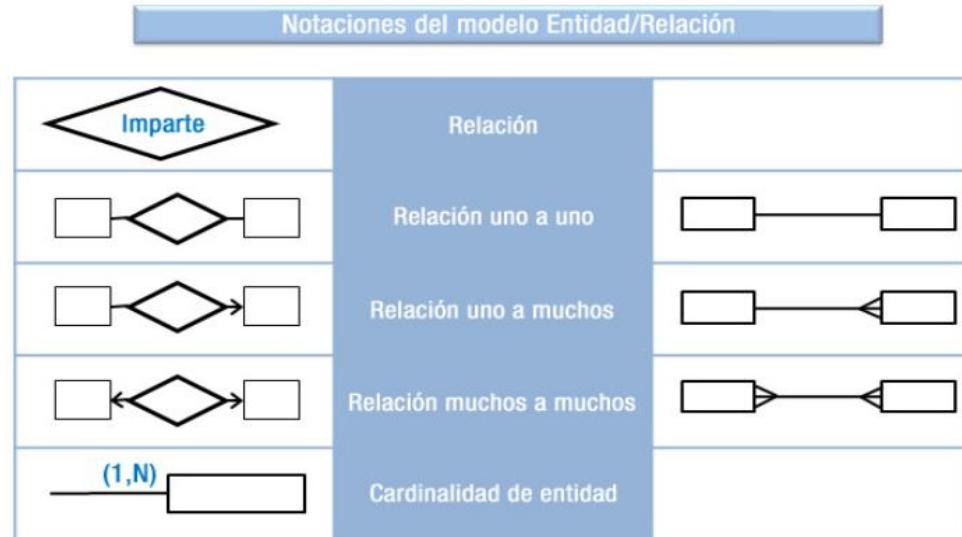
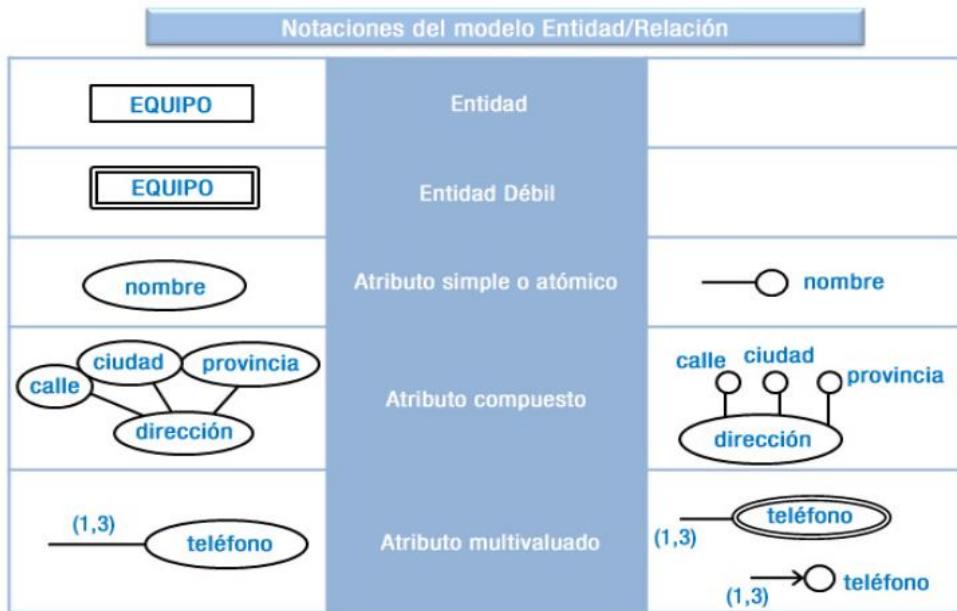


Supongamos que seguimos diseñando una base de datos para un sitio de juegos online. En un punto del proceso de diseño se ha de modelar el siguiente requisito: cada usuario registrado podrá crear las partidas que desee (a las que otros usuarios pueden unirse), pero una partida solo podrá estar creada por un único usuario. Un usuario podrá o no crear partidas. ¿Cuáles serían las etiquetas del tipo (cardinalidad mínima, cardinalidad máxima) que deberían ponerse junto a las entidades USUARIO

y PARTIDA respectivamente, si éstas están asociadas por la relación CREAR (partida)?

(1,N) y (0,N) (1,1) y (1,N) (1,1) y (0,N)

5. Simbología del modelo E/R



Notaciones del modelo Entidad/Relación



6. El modelo E/R extendido

Algunos requisitos tienen dificultad para ser representados con la simbología tradicional del modelo E/R.

En el modelo E/R extendido se incorporan nuevas extensiones para circunstancias especiales en elementos de difícil representación como relaciones con cardinalidad N:M o no identificación clara de entidades.

6.1 Restricciones en las relaciones

Restricción de exclusividad: O se produce una relación u otra pero nunca ambas, ni a la vez, ni por separado. Una entidad participa en dos o más relaciones y cada ocurrencia de entidad solo puede pertenecer a una de las relaciones únicamente.

Se usa un arco que engloba a todas aquellas relaciones que son exclusivas.

Ej.: EMPLEADOS o son diseñadores (DISEÑAN) o son operarios (FABRICAN) PRODUCTOS. Pero no pueden diseñar y fabricar, ni a la vez, ni por separado.

Restricción de exclusión: Las ocurrencias de las entidades solo pueden asociarse usando una única relación. O se produce una u otra pero nunca ambas simultáneamente. La entidad puede actuar en todas las relaciones pero siempre por separado.

Se representa con línea discontinua entre las relaciones

Ej: Un PROFESOR no puede RECIBIR e IMPARTIR el mismo curso. Pero si imparte un curso, puede estar recibiendo uno.

Restricción de inclusividad: Para que dos ocurrencias de entidad se asocien a través de una relación, deben haberlo estado antes a través de otra relación.

Se indica la cardinalidad mínima y máxima de la restricción.

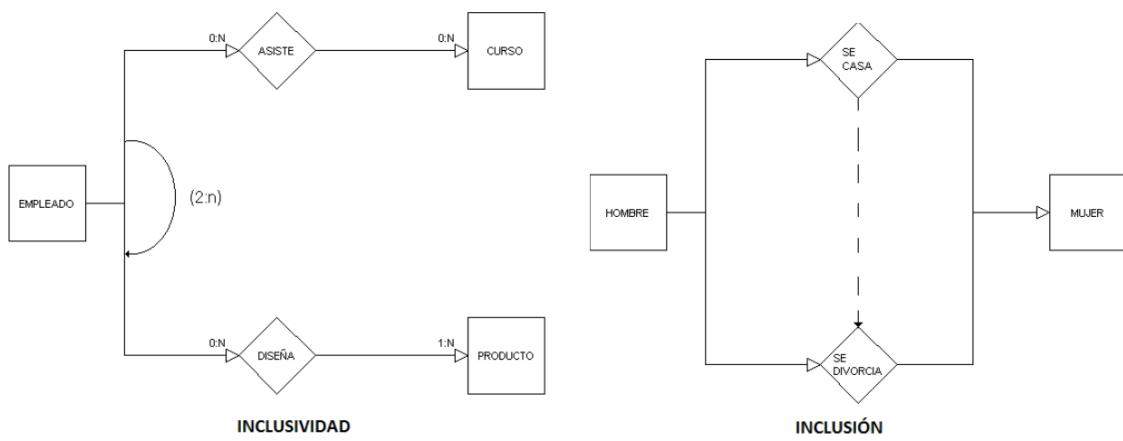
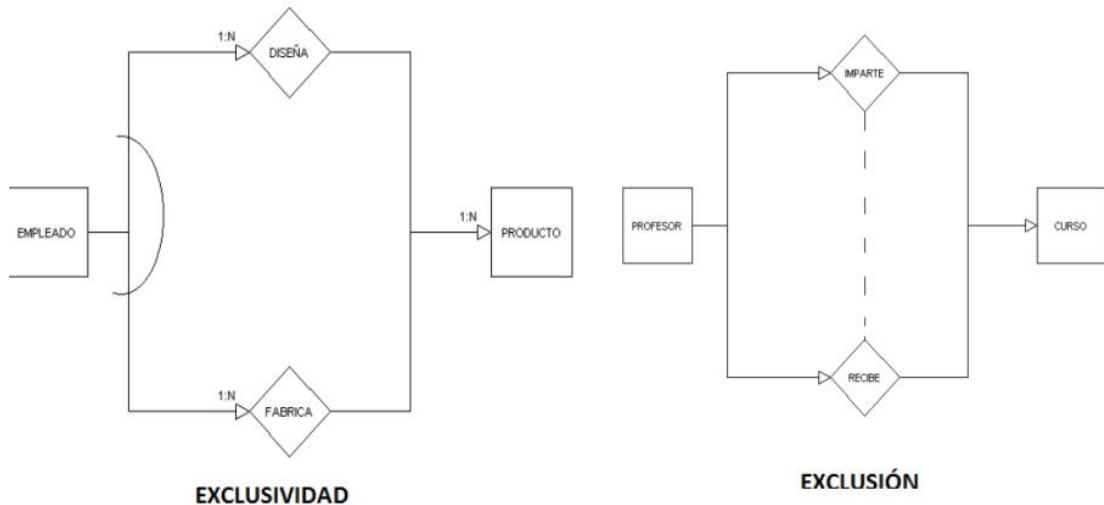
Se representa con un **arco acabado en flecha que parte de la relación que debe cumplirse primero hacia la otra relación**.

Ejemplo.: Para que EMPLEADO pueda DISEÑAR debe haber ASISTIDO a dos cursos. (2,n)

Restricción de inclusión: Para participar en la asociación con otro elemento de entidad mediante interrelación, ambos deben estar asociados por una interrelación por naturaleza.

Se usa una flecha discontinua entre las dos relaciones.

Ejemplo.: Para que un PERSONA se DIVORCIE de PERSONA debe haber una interrelación previa, haberse casado con ella.



6.2. Generalización y especialización

Se basan en el uso de **JERARQUÍAS**. Al modelar la base de datos podemos encontrarnos con entidades que tienen características comunes. Puede crearse un tipo de entidad de nivel más alto que englobe esas características y dividirse la entidad en diferentes subgrupos de entidades por sus características diferenciadoras.

Entidades de nivel superior → Subclase, supertipo

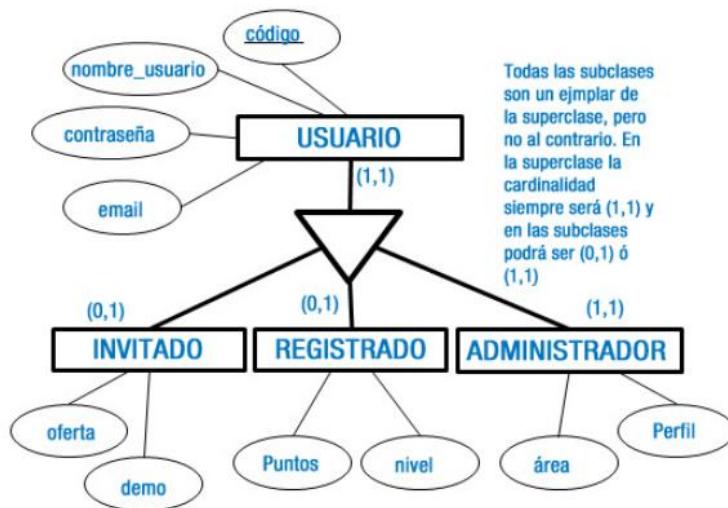
Entidades de nivel inferior → Subclase, subtipo

Se puede hacer **especialización de superclase en subclases y generalización de subclases en superclases**.

Las jerarquías se caracterizan por la **herencia**. Los atributos de la superclase son heredados por las subclases. Si la superclase intervino en una relación, las subclases también lo harán.

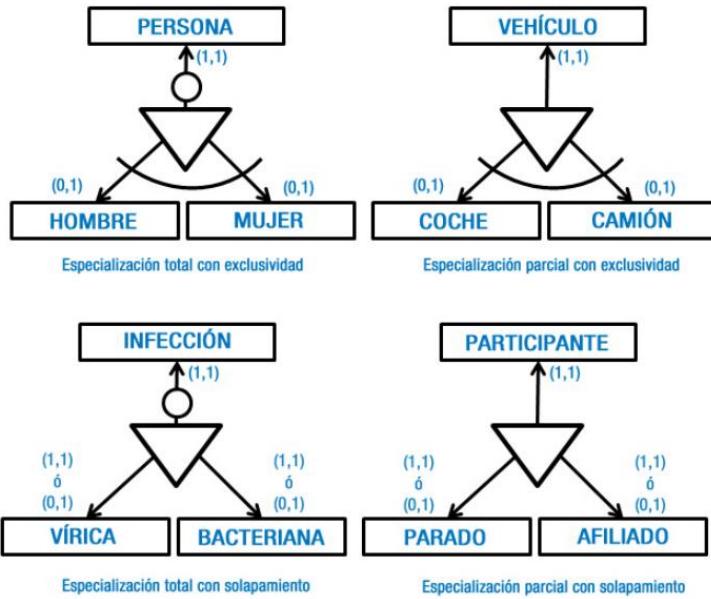
Se representan las relaciones entre superclase y subtipos con “ES UN” “IS A” (un tipo de). La jerarquía se representa mediante un triángulo invertido sobre el que queda la superclase y, conectadas a él a través de líneas rectas, las subclases.

Ejemplo: Superclase USUARIO. Subclases INVITADO, REGISTRADO, ADMINISTRADOR. Con sus propias características y heredan las de la superclase.



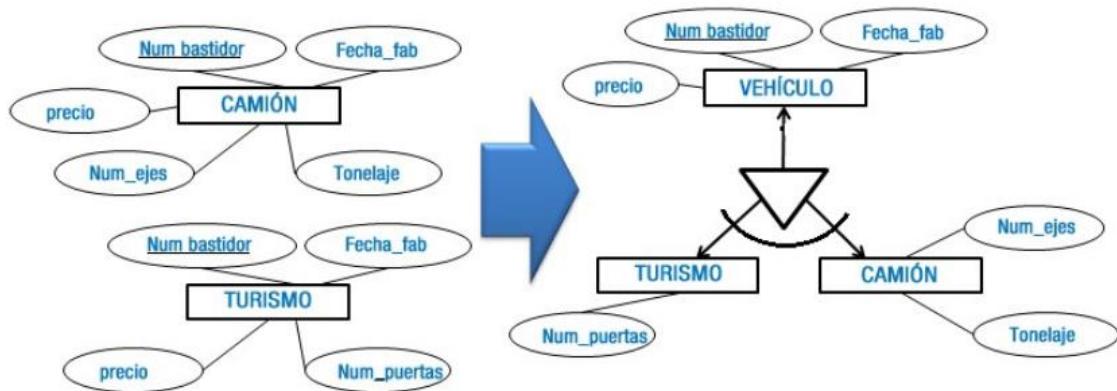
La generalización/especialización puede tener las siguientes restricciones semánticas:

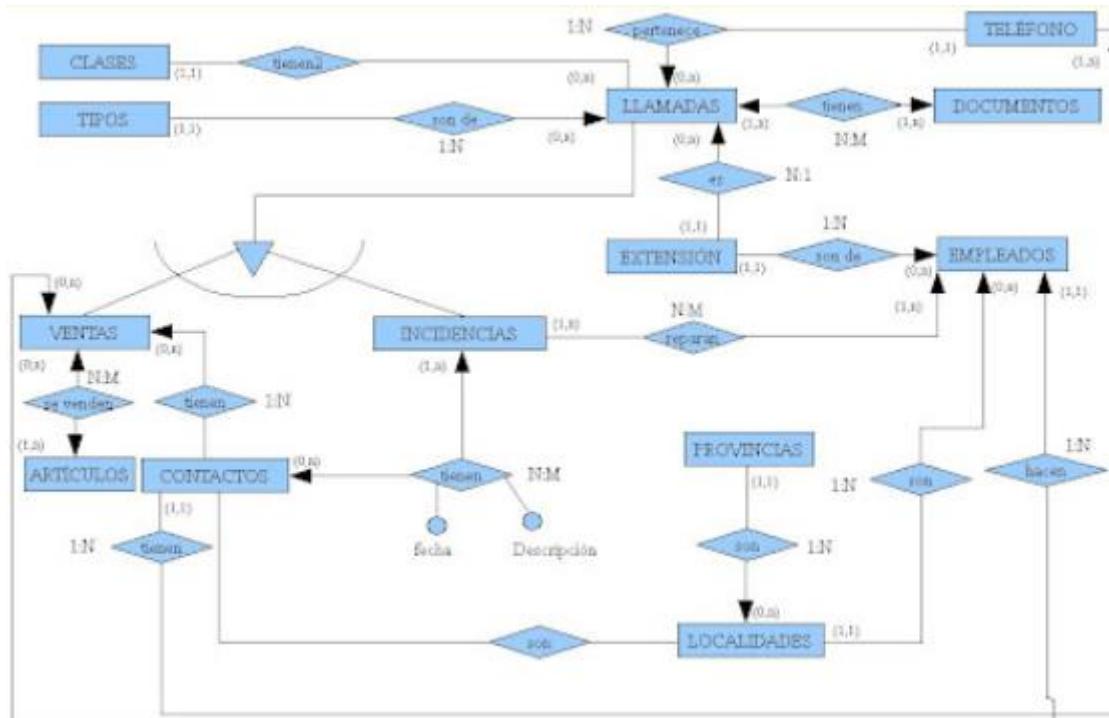
- **TOTALIDAD:** Todo ejemplar de superclase pertenece a alguna de las subclases
- **PARCIALIDAD:** No todos los ejemplares de la superclase pertenecen a alguna de las subclases.
- **SOLAPAMIENTO:** Un ejemplar de la superclase puede pertenecer a más de una subclase.
- **EXCLUSIVIDAD:** Un ejemplar de superclase pertenece a solo una subclase.



	SOLAPAMIENTO	EXCLUSIVIDAD
TOTAL	INFECCIÓN (VIRICA / BACTERIANA) (debe ser de uno de ellos y podría ser de los dos)	PERSONA (HOMBRE / MUJER) (debe ser uno de ellos y solo uno)
PARCIAL	PARTICIPANTE (PARADO / AFILIADO) (puede ser otro y podría ser los dos a la vez)	VEHICULO (COCHE / CAMION) (puede ser otro, pero no los dos a la vez)

Veamos como las entidades CAMION y TURISMO pasan a ser VEHICULO (CAMION / TURISMO) en una jerarquía parcial con exclusividad.





6.3. Agregación

Permite representar las relaciones entre relaciones. Se engloban como un rectángulo las entidades y la relación a abstraer, creándose una nueva entidad agregada que puede participar en otras relaciones con otras entidades. La cardinalidad máxima y mínima de la entidad agregada siempre es (1,1) no indicándose por ello en el esquema.

Es una abstracción en la que las relaciones se tratan como entidades de niveles más alto, usándose para expresar relaciones entre relaciones o entre entidades y relaciones.

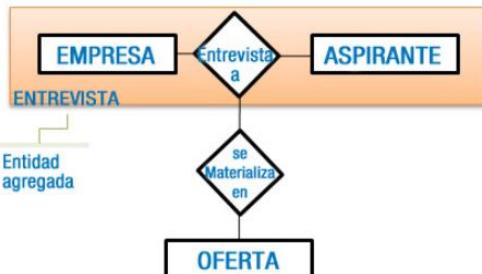
Supongamos un ejemplo en el que hemos de modelar la siguiente situación: una empresa de selección de personal realiza entrevistas a diferentes aspirantes. Puede ser que, de algunas de estas entrevistas a aspirantes, se derive una oferta de empleo, o no. En el siguiente gráfico se representan tres soluciones, las dos primeras erróneas y una tercera correcta, utilizando una agregación.



Solución 1: Errónea, ya que estaríamos representando que, por cada entrevista realizada por una empresa a un aspirante, se genera una oferta de empleo



Solución 2: Errónea, porque en el modelo E/R no pueden establecerse relaciones entre varias relaciones



Solución 3: En el modelo E/R Extendido, puede crearse una entidad agregada llamada ENTREVISTA, compuesta por la relación "Entrevista a" que existe entre EMPRESA y ASPIRANTE. Entre esta nueva entidad y OFERTA si puede establecerse una relación "se materializa en"

Existen dos clases de agregaciones:

- **Compuesto/Componente:** Un todo por la unión de las partes, que pueden ser objetos distintos y desempeñar papeles distintos.
- **Miembro/Colección:** Un todo se obtiene por unión de las partes y desempeñan el mismo papel o rol. Puede incluir una restricción de orden (indicando el atributo de ordenación) entre los miembros.



7. Diseño conceptual: Elaboración de diagramas E/R

Primero se parte del documento de especificación de requerimientos (conjunto de requerimientos, requisitos o condiciones que la base de datos debe cumplir), enunciado del problema a modelar que, cuanto más completo sea, mejor.

1. **Identificación de entidades.** Localizar elementos que serán entidades. Buscar nombres o sustantivos sin fijarnos en características o propiedades. También identificar objetos que existen por sí mismos. El número de entidades debe ser manejable y se les otorgarán nombres representativos en mayúsculas y singular. Recomiendan que para que sea entidad debe cumplir:
 - Existencia propia
 - Cada ejemplar debe diferenciarse del resto de ejemplares

- Todos los ejemplares deben tener las mismas propiedades
- 2. **Identificación de relaciones.** Buscar verbos o expresiones que conecten unas con otras. Mayormente habrá relaciones binarias aunque puede haber recursivas o unarias también. Se dará nombre representativo en minúscula.
- 3. **Representación de la cardinalidad de entidades y relaciones**
- 4. **Identificación de atributos de entidad y relaciones.** Buscamos nombres de características, propiedades... ¿qué debe tenerse en cuenta en esa entidad o relación?. No todos estarán directamente en el documento de especificación de requerimientos, el diseñador podrá establecerlos aplicando el sentido común o indagando en el problema. Debe tenerse en cuenta si los atributos son simples/compuestos, derivados/calculados o si el atributo se repite en varias entidades. En este caso debe plantearse la posibilidad de una jerarquía de especialización. Debería recopilarse del atributo: Nombre y descripción; Atributos simples que lo componen si es compuesto; Método de cálculo si es derivado; En atributos con cardinalidad uno a muchos, valorar asignar ese atributo o atributos a la entidad con mayor cardinalidad participante en la relación.
- 5. **Identificación de claves.** Establecer una o varias claves candidatas y escoger una de ellas como clave primaria de la entidad que la describan de manera única. La identificación de claves permite determinar la fortaleza (al menos una clave candidata) o la debilidad (ninguna clave candidata) de las entidades encontradas. Se representa la existencia de la clave primaria y las entidades fuertes y débiles.
- 6. **Determinación de jerarquías.** Si hay entidades con características comunes que puedan ser generalizadas o si es necesario expresar en el esquema las particularidades de un tipo de entidad por lo que deben ser especializadas, debe analizarse con detenimiento el documento de especificación y si se identifica alguna jerarquía, representarla adecuadamente.

7.1. Metodologías

Partiendo de una versión preliminar del esquema conceptual o diagrama E/R que, tras sucesivos refinamientos, será modificado para llegar al E/R definitivo se podrían tener las siguientes metodologías:

- **Metodología descendente (top-down):** Partir del esquema generar e ir descomponiendo este en niveles, cada uno con mayor número de detalles. Objetos abstractos se refinan paso a paso hasta llegar al esquema final.
- **Metodología ascendente (bottom-up):** Se parte de los atributos. Se agrupan en entidades para crear relaciones entre estas y las posibles jerarquías hasta obtener un diagrama completo. Se parte de objetos atómicos y se obtienen abstracciones de mayor nivel a partir de ahí.
- **Metodologías dentro-fuera (inside-out):** El esquema se desarrolla en una parte del papel y a medida que se analiza la especificación de requerimientos se va completando con entidades y relaciones hasta ocupar todo el documento.
- **Metodología mixta:** En problemas complejos. Requerimientos divididos en subconjuntos que serán analizados independientemente. Se crea un esquema que servirá como estructura en la que se irán interconectando los conceptos importantes con el resultado del análisis de los subconjuntos creados. Se aplica

técnica descendente para dividir los requerimientos y en cada subconjunto se emplea técnica ascendente.

7.2. Redundancia en diagramas de E/R

Redundancia es la reproducción, reiteración, insistencia.... Almacenamiento de los mismos datos varias veces en varios lugares.

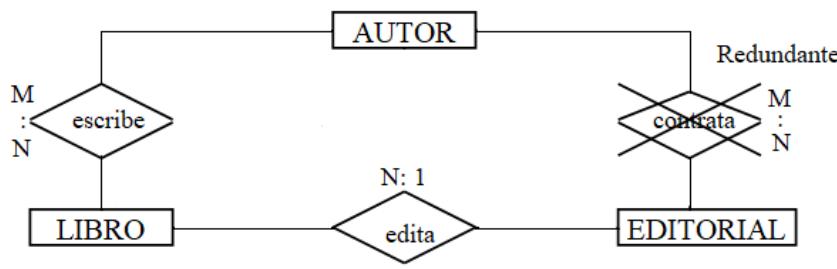
Esto provoca problemas como:

- Aumento de la carga de trabajo. Operaciones de grabación o actualización necesitan hacerse en varias ocasiones
- Gasto extra de espacio de almacenamiento
- Inconsistencia: Los datos repetidos podrían no tener los mismo valores (valor actualizado en un lugar y en otro no).

Es imprescindible el control de la redundancia mediante un buen diseño, analizando el esquema y valorando qué elementos pueden incorporar redundancia.

- Atributos redundantes pueden ser elementos cuyo contenido se calcula en función de otro.
- Existencia de ciclos. Varias entidades unidas de forma circular o cíclica no tienen por qué implicar una redundancia. Para confirmar que puede eliminarse debemos asegurarnos de que
 - o El significado de las relaciones que componen el ciclo es el mismo
 - o Al eliminar la relación redundante el significado del resto de relaciones en el mismo
 - o Si la relación eliminada tiene atributos asociados, estos pueden ser asignados a alguna entidad participante en el esquema sin que se pierda su significado.
- No toda redundancia es perjudicial. Si por ejemplo el método de cálculo es complejo puede ser conveniente definirlo desde el principio en lugar de calcularlo a cada vez. Depende de la elección que haga el diseñador.

Si lo analizamos vemos que si se conocen los libros de un autor y las editoriales que los han editado, se puede deducir qué editoriales ha contratado dicho autor; de forma similar, si sabemos qué libros ha editado una editorial concreta, podemos deducir qué autores han escrito para ella (la han contratado), por lo que la interrelación contrata entre las entidades AUTOR y EDITORIAL es redundante. Ahora bien, si la relación considerada como redundante presentara cardinalidad M:N y tuviese atributos propios (por ejemplo fecha de contratación, en nuestro caso) no podría ser eliminada.



Redundancia en diagramas E/R (CC0)

7.3. Propiedades deseables en diagrama E/R

Las propiedades deseables serían:

- **Completitud:** Cada representación del diagrama tiene un equivalente en los requerimientos y viceversa.
- **Corrección:** Emplea todos los elementos del modelo E-R.
 - o **Corrección sintáctica:** No hay representaciones erróneas
 - o **Corrección semántica (SIGNIFICAR):** La representación significa exactamente lo que se estipula en los requerimientos. Posibles errores: Usos de atributo en lugar de entidad, uso de entidad en lugar de relación, uso de mismo identificador para dos entidades o relaciones, indicar erróneamente alguna cardinalidad u omitirla...
- **Minimalidad:** Evitar que sea redundante. ¿Al eliminar cualquier elemento, se pierde información? -> Es mínimo.
- **Legibilidad:** Fácil de interpretar. Depende de cómo se dispongan los elementos y conexiones (aspectos estéticos).
- **Escalabilidad:** Es capaz de incorporar cambios dados por nuevos requerimientos.

8. Comenzando el diseño lógico: Transformación de diagramas E/R al modelo relacional

Cuando el esquema E/R cumple los requerimientos del problema a modelar se transforma al esquema lógico basado en el modelo de datos (modelo de datos relacional) en el que se basa el SGDB con el que se implementarán las tablas.

La transformación tendrá tres fases:

- Transformación de las entidades
- Transformación de las relaciones
- Transformación de los atributos

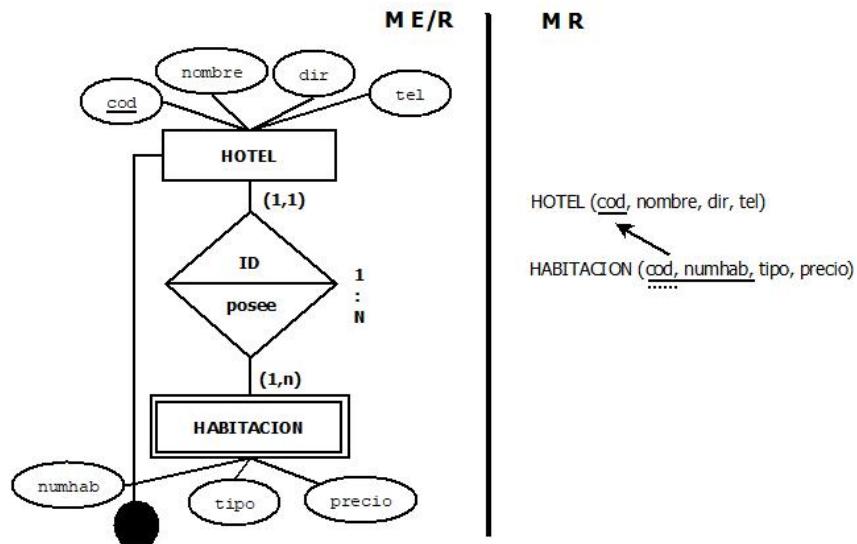
Cada tabla se representa con su nombre y, a continuación, y entre paréntesis, su lista de atributos. La clave principal de cada tabla se representa con subrayado continuo. La clave o claves ajenas con subrayado discontinuo. Las clases alternativas no se representan.

8.1. Transformación de entidades

Norma general: Cada entidad se transforma en una tabla nombrándose igual que la entidad de la que proviene.

Casos especiales

Dependencia en identificación: Si una entidad débil depende en identificación de una entidad fuerte, la clave primaria de la entidad fuerte se exporta a la tabla de la entidad débil donde formará parte de su clave primaria siendo a la vez una clave ajena que referenciará a la tabla de la entidad fuerte.



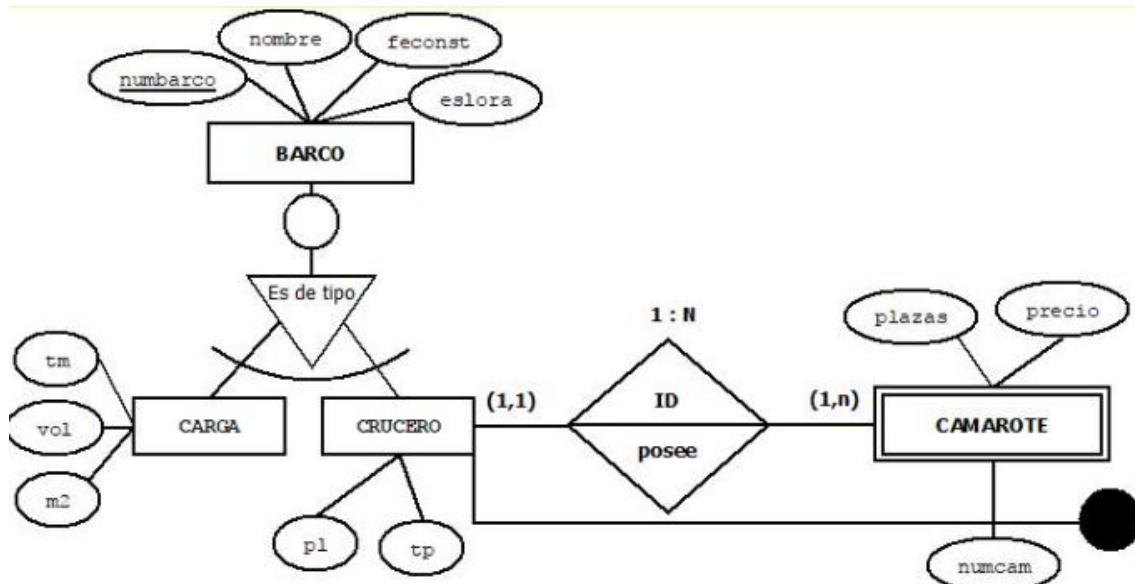
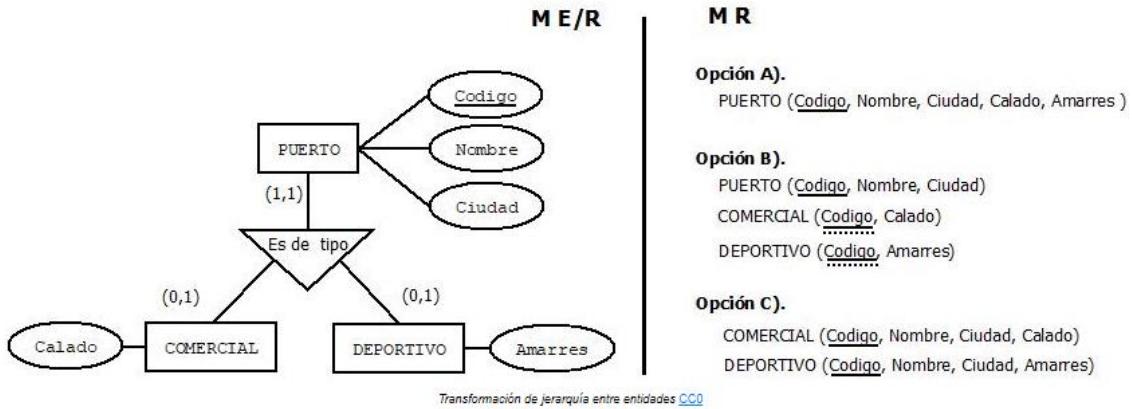
Generalización y especialización: Hay varias opciones.

Opción A) **Crear una única tabla para el supertipo y los subtipos que recoja todos los atributos.** Si los subtipos no tienen atributos diferenciadores, añadir un atributo indicando el subtipo al que se refiere. Es la solución más simple pero puede provocar valores nulos en los atributos propios de cada subtipo (Solo usar en casos en los que se diferencien en muy pocos atributos y las relaciones sean las mismas para los subtipos o no existan).

Opción B) **Crear una tabla para el supertipo con los atributos generales y una para cada uno de los subtipos con sus correspondientes atributos específicos.**

(Aconsejable cuando los subtipos tienen atributos distintos y/o sus relaciones con otras entidades sean diferentes. Permite mantener agrupados en una tabla los atributos comunes).

Opción C) **Crear una tabla para cada subtipo que además de los atributos específicos contenga los atributos comunes del supertipo.** Suele aplicarse a jerarquías totales con exclusividad en las que se dan subtipos con atributos dispares y relaciones diferentes con otras entidades.



CARGA (numbarco, nombre, feconst, eslora, tm, vol, m2)

BARCO (numbarco, nombre, feconst, eslora)

CRUCERO (numbarco, nombre, feconst, eslora, pl, tp)

CARGA (numbarco, tm, vol, m2)

CAMAROTE (numbarco, numcam, plazas, precio)

PASAJEROS (numbarco, pl, tp)

CAMAROTE (numbarco, numcam, plazas, precio)

8.2. Transformación de relaciones

Cardinalidad Muchos a muchos N:M

Siempre se transforma en una tabla nueva.

Clave primaria compuesta la concatenación de las claves primarias de las entidades que relaciona. Cada una de esas claves será parte de la clave primaria de la nueva tabla y clave foránea que referenciará a la tabla de procedencia donde es clave primaria.

Si la relación contiene atributos, pasan a formar parte de la nueva tabla. Debe comprobarse que la clave primaria compuesta de la nueva tabla es suficiente (quizás deba completarse con algún atributo de la relación, especialmente si es de tipo fecha).

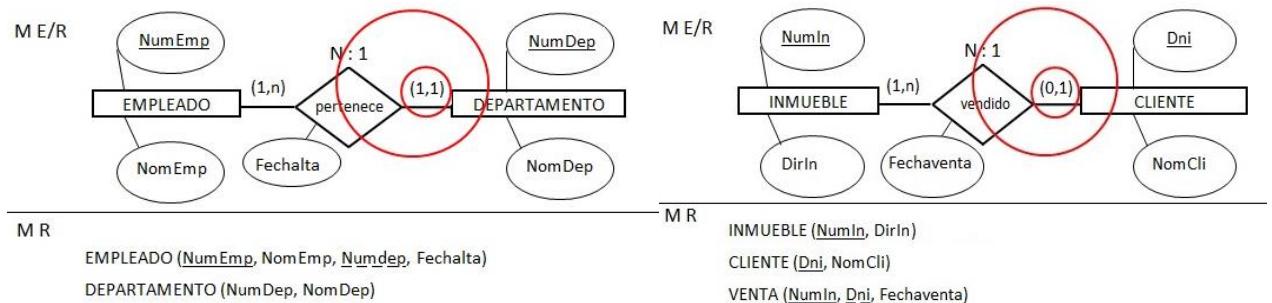
Cardinalidad 1:N

Dos opciones:

Cardinalidad de la tabla de cardinalidad 1 es (1,1): La clave primaria de la tabla de cardinalidad 1, ponerla en la tabla de cardinalidad N como clave foránea. Los atributos de la relación pasan a la entidad con tipo N.

Cardinalidad de la tabla de cardinalidad 1 es (0,1): Transformar la relación en una nueva tabla. Esta tendrá como clave primaria la clave primaria de la entidad con cardinalidad N y como clave ajena la clave primaria de la entidad con cardinalidad 1. La nueva tabla tendrá los atributos de la relación.

Si en este caso se propagase la clave, habrá algunos NULL en los campos foráneos. Esto TAMBIÉN se hace muchos casos.



Cardinalidad 1:1

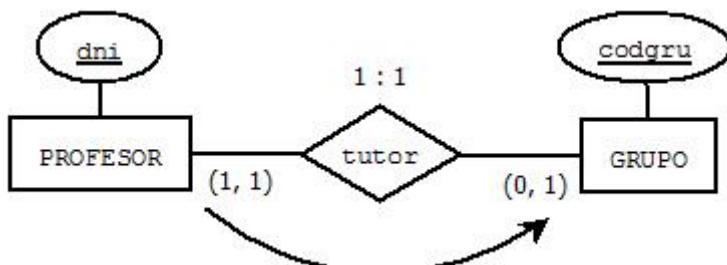
Dos opciones:

Las dos tablas tienen cardinalidad (0,1). Crear una nueva tabla. En ambas tablas habrá ocurrencias no relacionadas con la otra entidad.

Al menos una tiene cardinalidad (1,1). Propagar la clave primaria de una de las entidades como clave ajena de la otra entidad

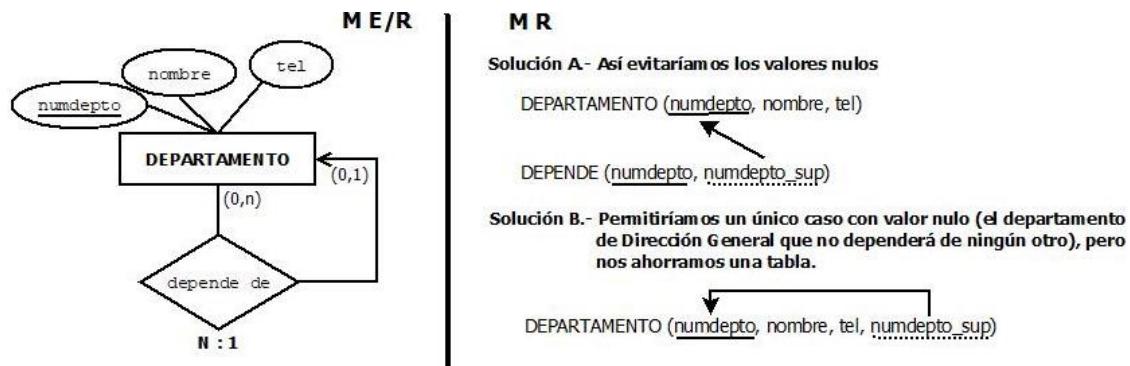
Si es (0,1) y (1,1) se hará de la tabla con (1,1) a la que tiene (0,1).

Si las dos tienen (1,1) entonces da igual.



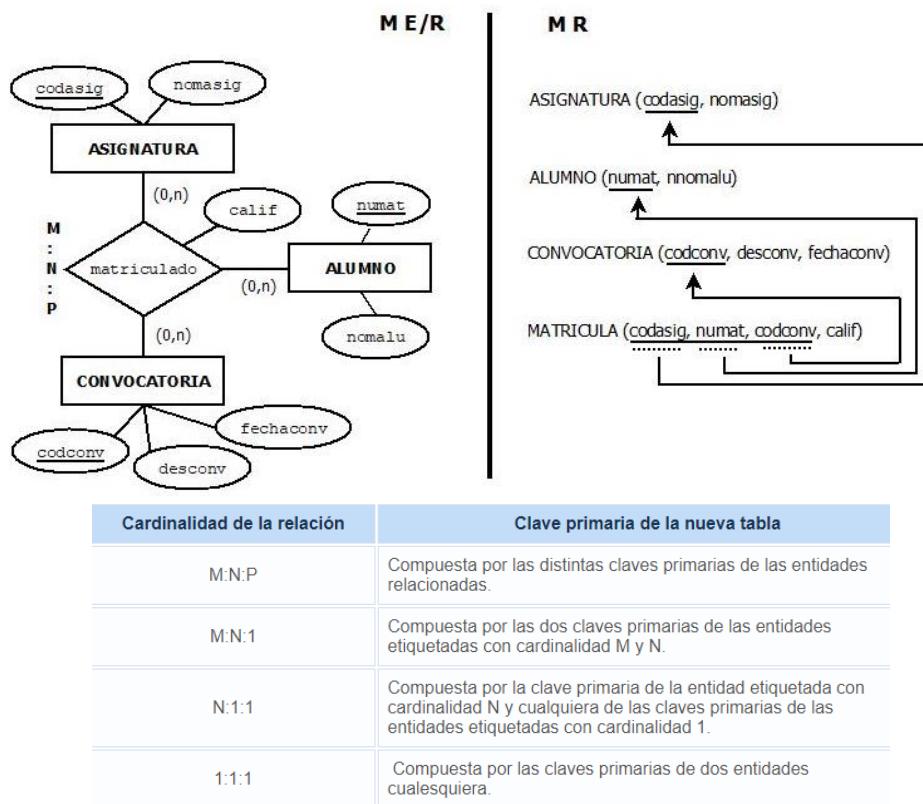
Relaciones de grado 1 o reflexivas

Se siguen las mismas pautas dadas, solo que con una sola entidad.



Relaciones de grado superior a 2

Recomendable descomponerlas en varias de grado 2. En este caso se suelen transformar en una nueva tabla que contendrá como claves ajenas las tres claves primarias de las entidades que relaciona. La elección de la clave primaria dependerá de la cardinalidad.



Relaciones con restricciones de exclusividad, exclusión, inclusividad, inclusión

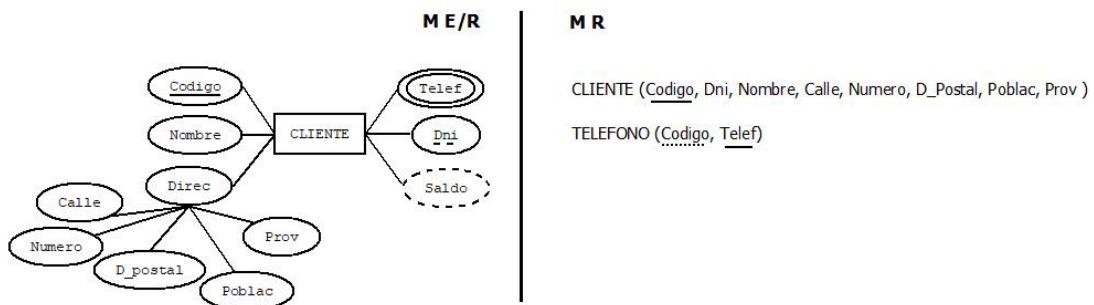
No se consideran en el modelo relacional. Se harán con disdparadores.

8.3. Transformación de atributos

Atributos de entidades

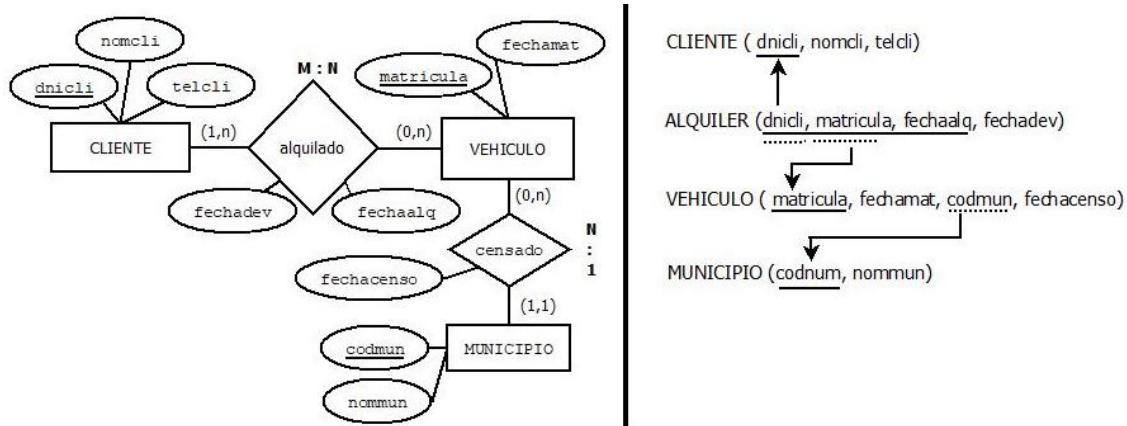
Como norma general cada atributo de entidad se transforma en una columna de la tabla a la que ha dado lugar la entidad. Se distinguen según el atributo sea:

- Clave primaria: Pasa a ser clave primaria de la tabla
- Claves alternativas: Pasan a ser columnas de la tabla
- Atributos no identificadores: Pasan a ser columnas de la tabla
- Atributos compuestos: Sus atributos elementales pasan a ser columnas de la tabla y desaparece el atributo compuesto
- Atributos calculados: Deben eliminarse de la tabla o razonarse por qué se mantienen
- Atributos multivaluados: Origina nueva tabla con atributo multivaluado y la clave primaria de la entidad que también será clave ajena. La clave de la nueva tabla estará compuesta por todos sus atributos.



Atributos de relaciones

Si la relación se transforma en tabla todos los atributos pasan a ser columnas de la tabla. Si se transforma mediante propagación de clave, los atributos migran junto a la clave a la tabla que corresponda.



9. Ejemplillos

Para la **representación del diagrama E/R** previamente deberás:

1. Identificar las entidades (fuertes y débiles), las relaciones entre las entidades y los atributos (tanto de entidades como posiblemente en algún caso de relaciones).
2. Identificar las cardinalidades de las entidades y después las cardinalidades de las relaciones.

3. Identificar las claves (candidatas y primarias) y los atributos calculados, compuestos y multivaluados.
4. Estudiar si existen otras características del modelo E/R que no se puedan representar con el modelo ER.

Para la representación del Modelo Relacional:

1. Transformar las entidades y sus atributos, con especial atención a los casos de dependencia en identificación y de jerarquías.
 2. Pasar a tablas las relaciones y sus posibles atributos.
 3. Identificar las claves primarias de las tablas (subrayado continuo) y las claves ajena (subrayado discontinuo). El uso de flechas puede ayudarte para representar más gráficamente las relaciones entre tablas.
- Las cardinalidades de entidad se componen de una cardinalidad mínima y otra cardinalidad máxima y se representa así:
(mínima , máxima)
 - La cardinalidad **mínima** nos da la información de obligar o no a que un registro de una entidad participe en la relación:

Tomará el valor **0** cuando **NO** sea obligatorio.
 Tomará el valor **1** cuando **SI** sea obligatorio.
 - La cardinalidad **máxima** nos da la información de la cantidad de registros de la otra relación con las que se puede relacionar un mismo registro.

Tomará el valor **1** cuando sólo se pueda relacionar **con un único registro**.
 Tomará el valor **n** cuando se pueda relacionar **con varios registros**.
 - Por tanto los posibles valores de las cardinalidades de entidad serán:
(0,1) , (1,1) , (0,n) , (1,n)

Una empresa desea guardar la información referente a la comercialización de sus productos. Para ello nos pide que le gestionemos una base de datos donde guardemos entre otros los datos de sus comerciales como el DNI, nombre, apellidos, salario, número de hijos que tiene, y fecha de nacimiento.

Por otro lado también interesa saber de los vehículos que conducen los comerciales la matrícula, la marca, el modelo, la última fecha de revisión y los kilómetros que tiene. Cada comercial sólo puede conducir un único vehículo que tiene asignado aunque podemos tener vehículos sin ningún comercial asignado.

La empresa tiene una serie de oficinas repartidas en varias ciudades y de ellas interesa conocer el código, la dirección, la localidad y la provincia. En cada oficina pueden trabajar varios comerciales pero un comercial sólo podrá pertenecer a una única oficina.

Los productos que comercializa la empresa son muy variados y de ellos se guarda un número de referencia único, un nombre, una descripción, el precio por unidad y el descuento que puede tener. Cuando un comercial vende los productos se desea almacenar la fecha de venta y la cantidad vendida de cada producto

ENTIDADES	Atributos de entidad
COMERCIAL	<u>DNI</u> , nombre, apellidos, salario, número_hijos y fecha_nacimiento
VEHICULO	<u>Matrícula</u> , marca, modelo, fecha de revisión y kilómetros
OFICINA	<u>Código</u> , dirección, localidad y provincia
PRODUCTO	<u>Núm_referencia</u> , nombre, descripción, precio y descuento

RELACIONES	Atributos de relación
conducir	---
trabajar	---
vender	fecha_venta y cantidad

La clave. Lo que esté fuera de la pregunta en singular y lo de dentro en plurar.

RELACIÓN: Conducir

«Cada comercial sólo puede conducir un único vehículo que tiene asignado aunque podemos tener vehículos sin ningún comercial asignado.»



- Un **comercial**, ¿cuántos **vehículos** puede conducir? **(0,1)**
- Un **vehículo**, ¿por cuántos **comerciales** puede ser conducido? **(0,1)**

RELACIÓN: Trabajar

«En cada oficina pueden trabajar varios comerciales pero un comercial sólo podrá pertenecer a una única oficina.»



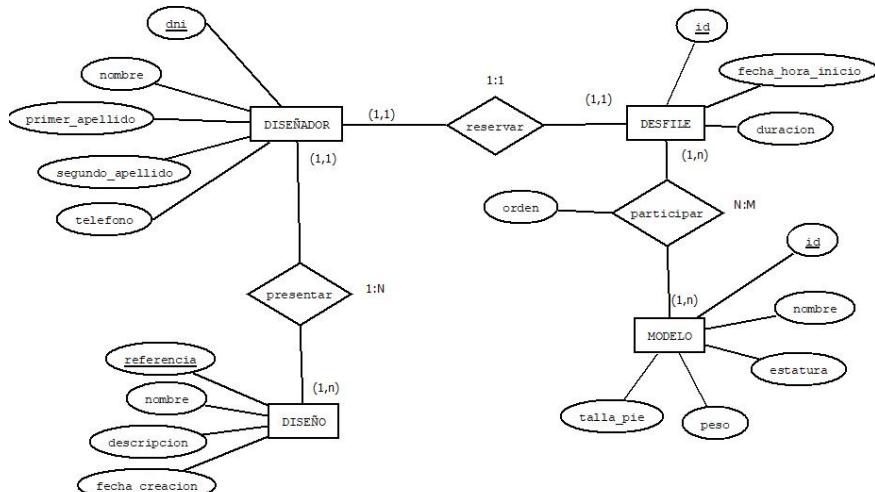
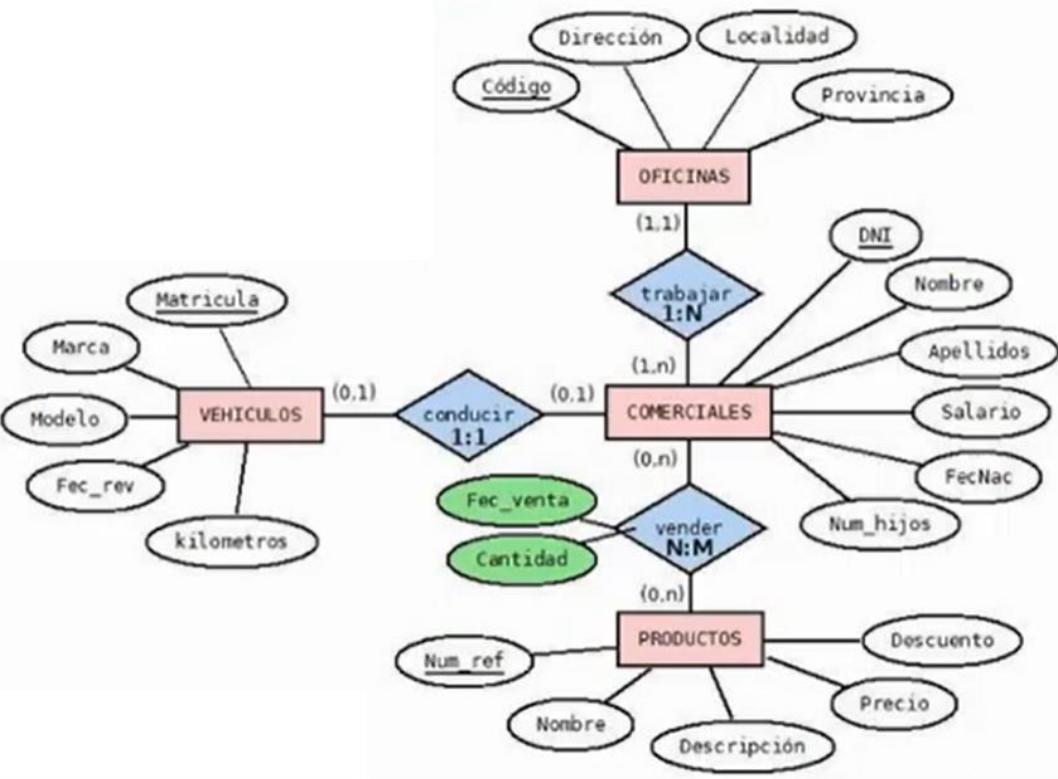
- En una **oficina**, ¿cuántos **comerciales** pueden trabajar? **(1,n)**
- Un **comercial**, ¿a cuántas **oficinas** puede pertenecer? **(1,1)**

RELACIÓN: Vender

«Cuando un comercial vende los productos se desea almacenar la fecha de venta y la cantidad vendida de cada producto.»



- Un **comercial**, ¿cuántos **productos** puede vender? **(0,n)**
- Un **producto**, ¿por cuántos **comerciales** puede ser vendido? **(0,n)**



DISEÑO (referencia, dni, nombre, descripcion, fecha_creacion)

DISEÑADOR (dni, nombre, primer_apellido, segundo_apellido, telefono)

DESFILE (id, dni, fecha hora inicio, duracion)

PARTICIPAR (id_desfile, id_modelo, orden)

MODELO (id, nombre, estatura, peso, talla_pie)

10. Normalización de modelos relacionales

Normalización: Es imponer a las tablas del modelo relacional una serie de restricciones a través de un conjunto de transformaciones consecutivas. Este proceso garantizará que las tablas contienen los atributos necesarios y suficientes para describir la realidad de la entidad que representan, separando los atributos que por su contenido podrían generar la creación de otra tabla.

Codd establece una técnica para llevar a cabo el diseño de la estructura lógica de los datos representados en el modelo relacional. Esta técnica debe usarse como refinamiento que conseguirá:

- Suprimir dependencias erróneas entre atributos
- Optimizar procesos inserción, modificación y borrado de la BBDD.

La normalización:

- Se basa en el análisis de dependencias entre atributos.
- Se realiza en varias etapas consecutivas asociadas a una **forma normal** que establece unos requisitos a cumplir por la tabla sobre la que se aplica. Si no se satisface una forma normal no puede pasarse al análisis de la siguiente.
- A cada avance los criterios son más restrictivos y por tanto el modelo relacional más robusto.

10.1 Tipos de dependencias

El análisis de dependencias entre atributos debe tener en cuenta los siguientes conceptos:

- **Dependencia funcional (dependencia funcional simple):** Dados A y B (siendo cada uno de ellos un atributo o un conjunto de ellos), B depende funcionalmente de A, sí y solo sí, para cada valor de A solo puede existir un valor de B. Se da siempre entre atributos de una misma tabla. El atributo A se llama “determinante” porque A determina el valor de B. Se representa como $A \rightarrow B$
- **Dependencia funcional completa (dependencia funcional plena):** Dados los atributos A_1, A_2, \dots, A_k y B se dice que B depende funcionalmente de forma completa de A_1, A_2, \dots, A_k si y solo si B depende del conjunto de atributos A_1, A_2, \dots, A_k pero no de ninguno de sus subconjuntos.
- **Dependencia funcional transitiva:** Dados A, B y C (siendo cada uno de ellos un atributo o un conjunto de ellos) se dice que hay una dependencia transitiva entre A y C, si B depende funcionalmente de A y C depende funcionalmente de B.

Ejemplito:

EMPLEADO(dni, nombre, dirección, localidad, cod_localidad, nombre_hijo, edad_hijo)

LIBRO(titulo_libro, num_ejemplar, autor, editorial, precio)

¿Qué atributos presentan una **dependencia transitiva** en la tabla **EMPLEADO**?

Cod_localidad y localidad dependen funcionalmente de DNI pero cod_localidad y localidad tienen otra dependencia funcional.

DNI -> Cod_localidad -> Localidad. Hay dependencia transitiva entre localidad y DNI.

¿Qué atributos presentan **dependencia funcional de la clave primaria** de la tabla EMPLEADO?

Asumiendo que todos los empleados tienen nombres distintos, vemos una dependencia de nombre y dirección. DNI -> Nombre -> Dirección

¿Qué atributos presentan una **dependencia funcional completa** en la tabla LIBRO?

Editorial y Precio depende del conjunto de atributos de la clave primaria de la tabla, pero no de los atributos por separado.

Autor **depende funcionalmente** solo de titulo_libro por lo que no es dependencia funcional completa.

10.2. Formas normales

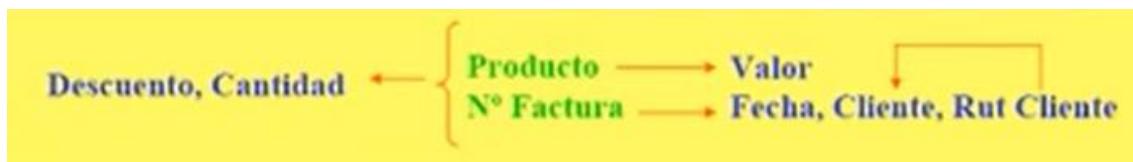
Requisitos a cumplir por las tablas según la forma normal que se aplique.

- **Primera forma normal (FN1):**
- **Segunda forma normal (FN2):**
- **Tercera forma normal (FN3):**
- **Forma normal de Boyce Codd:**
- **Cuarta forma normal (FN4):** Basada en dependencias multivaluadas.
- **Quinta forma normal (FN5):** Basada en dependencias de Join o de reunión
- **Forma normal de dominio clave (DKNF):** Basada en restricciones impuestas sobre dominios y claves

Es recomendable para garantizar que no hay problemas en la actualización de datos aplicar el proceso de normalización hasta Tercera Forma Normal o incluso hasta Forma Normal de Boyce-Codd.

Sea

FACTURA(NºFactura, Producto, Fecha, RUT Cliente, Cliente, Valor, Cantidad, Descuento)



Descuento y Cantidad dependen funcionalmente de la clave primaria Producto, Nº Factura. Valor del producto depende funcionalmente del producto.

Fecha, Cliente y RutClient dependen funcionalmente de Nº Factura.

Existe una dependencia funcional de cliente y RutCliente.

Aficion_Persona (codPersona, codAficion, nombrePersona, apellidosPersona, teléfonos, localidad, provincia, nombreAvision)

<u>codPersona</u>	<u>codAficion</u>	nombrePersona	apellidosPersona	teléfonos	localidad	provincia	nombreAficion
A1	BOX	Antonio	López Sánchez	666123456 9555555555	Dos Hermanas	Sevilla	Boxeo
A1	FUT	Antonio	López Sánchez	666123456 9555555555	Dos Hermanas	Sevilla	Fútbol
B2	BOX	José	Ruiz García	666654321 955000000 600111111	Alcalá de Guadaira.	Sevilla	Boxeo
B2	LEC	José	Ruiz García	666654321 955000000 600111111	Alcalá de Guadaira.	Sevilla	Lectura

Primera Forma Normal

- **Cualquier atributo que no pertenece a la clave toma valores atómicos,** es decir, los valores de los atributos son indivisibles. (No multivaluados)
- **Cualquier atributo que no pertenece a la clave tiene una dependencia funcional de la clave.**

Podríamos separar por apellidos (si lo necesitamos). Pero claramente está mal porque una persona tiene varios teléfonos.

Aficion_Persona (codPersona, codAficion, nombrePersona, apellidosPersona, localidad, provincia, nombreAficion).

Teléfono (codPersona, teléfono).

1 FN

teléfonos

Afición-persona

<u>CodPersona</u>	<u>teléfono</u>
A1	666123456
A1	9555555555
B2	666654321
B2	955000000
B2	600111111

<u>codPersona</u>	<u>codAficion</u>	nombrePersona	apellidosPersona	localidad	provincia	nombreAficion
A1	BOX	Antonio	López Sánchez	Dos Hermanas	Sevilla	Boxeo
A1	FUT	Antonio	López Sánchez	Dos Hermanas	Sevilla	Fútbol
B2	BOX	José	Ruiz García	Alcalá de Guadaira.	Sevilla	Boxeo
B2	LEC	José	Ruiz García	Alcalá de Guadaira.	Sevilla	Lectura

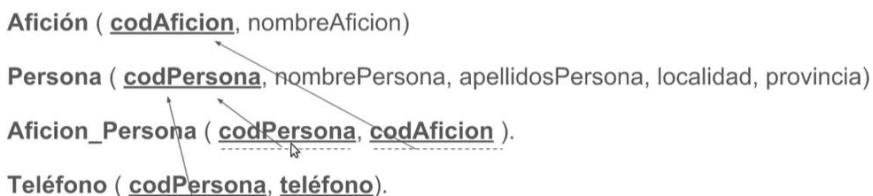
(Se crea a partir de la tabla inicial una nueva tabla con atributos los que presentan dependencia funcional de la clave primaria (atómicos). La clave de esta es la

misma clave primaria de la tabla. Después con los atributos no atómicos se crea otra tabla y se elige entre ellos uno que será la clave primaria de la tabla junto a la clave primaria de la tabla inicial. Se comprueba que la segunda tabla esté en 1FN. Si no está en 1FN se toma la segunda tabla como tabla inicial y se repite el proceso).

Segunda Forma Normal

Está en primera forma normal y...

- **Todo atributo que no pertenece a la clave tiene una dependencia funcional completa de la clave (se obtiene a partir de toda la clave y no de una parte de la clave).**



2 FN		afición		Afición-persona		telefonos	
persona	CodAficion	nombreAficion		codPersona codAficion		CodPersona teléfono	
	BOX	Boxeo		A1	BOX	A1	666123456
	FUT	Fútbol		A1	FUT	A1	955555555
	LEC	Lectura		B2	BOX	B2	666654321
				B2	LEC	B2	955000000
		CodPersona	nombre	apellidos	localidad	provincia	
		A1	Antonio	López Sánchez	Dos Hermanas	Sevilla	
		B2	José	Ruiz García	Alcalá de Guadaíra	Sevilla	

(Se crea una nueva tabla con los atributos que dependen funcionalmente de forma completa de la clave. La clave será la misma clave primaria que la tabla inicial. Con los restantes, se crea otra tabla que tendrá por clave el subconjunto de atributos de la clave inicial de los que dependen de forma completa. Si la tabla está en 2FN el proceso termina y, si no, se toma la segunda tabla como inicial y se repite el proceso).

Tercera Forma Normal

Está en segunda forma normal y...

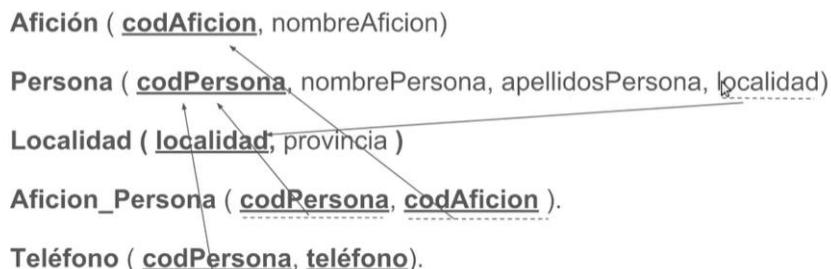
- Todo atributo que no pertenece a la clave, no depende transitivamente de la clave.

La provincia se podría determinar a partir de la localidad que a su vez se determina de persona. (Dependencia funcional transitiva de provincia respecto a la clave).

Si hay un único atributo que no pertenece a la clave ya podemos asegurar que está en tercera forma normal.

CodPersona -> Localidad -> Provincia

Generamos una nueva relación:



afición		Afición-persona		teléfonos	
<u>CodAficion</u>	nombreAfición	<u>codPersona</u>	<u>codAficion</u>	<u>CodPersona</u>	<u>teléfono</u>
BOX	Boxeo	A1	BOX	A1	666123456
FUT	Fútbol	A1	FUT	A1	955555555
LEC	Lectura	B2	BOX	B2	666654321
		B2	LEC	B2	955000000
				B2	600111111

localidad

<u>localidad</u>	provincia
Dos Hermanas	Sevilla
Alcalá de Guadaira	Sevilla

(Se crea una nueva tabla con los atributos que no dependen dependen transitivamente de forma completa de la clave. La clave será la misma clave primaria que la tabla inicial. Con los restantes, se crea otra tabla con los atributos no clave que intervienen en la dependencia transitiva y se elige uno de ellos como clave primaria, si cumple los requisitos para ellos. Si la tabla está en 3FN el proceso termina y, si no, se toma la segunda tabla como inicial y se repite el proceso).

Forma Normal de Boyce-Codd

Está en forma normal de Boyce-Codd si está en 3FN y...

- Todo determinante es una clave candidata. (Un determinante es todo atributo simple o compuesto del que depende funcionalmente de forma completa algún otro atributo de la tabla.

Aquellas tablas en las que todos sus atributos forman parte de la clave primaria están en FNBC.

Si se encuentra un determinante que no es clave primaria no está en FNBC. Esta redundancia suele ocurrir por una mala elección de la clave. Para normalizarlo debe descomponerse la tabla inicial en dos, siendo cuidadosos para evitar la pérdida de información en la descomposición.

COMPRAS (cod_compra, cod_prod, nomb_prod, fecha, cantidad, precio, fecha_rec, cod_prov, nomb_prov, tfno).

Se pide normalizarla hasta FNBC.

Comprobamos 1FN:

La tabla COMPRAS está en 1FN ya que todos sus atributos son atómicos y todos los atributos no clave dependen funcionalmente de la clave.

Comprobamos 2FN:

Nos preguntaremos ¿Todo atributo depende de todo el conjunto de atributos que forman la clave primaria, o sólo de parte?. Como vemos, existen atributos que dependen sólo de una parte de la clave, por lo que esta tabla no está en 2FN.

Veamos las dependencias:

$\text{cod_prod} \rightarrow \text{nomb_prod}$, y cod_prod es parte de la clave primaria.

Al no estar en 2FN, hemos de descomponer la tabla COMPRAS en:

COMPRA1 (cod_compra, cod_prod, fecha, cantidad, precio, fecha_rec, cod_prov, nomb_prov, tfno).
PRODUCTO (cod_prod, nomb_prod).

Una vez hecha esta descomposición, ambas tablas están en 2FN. Todos los atributos no clave dependen de toda la clave primaria.

Comprobamos 3FN:

PRODUCTO está en 3FN, ya que por el número de atributos que tiene no puede tener dependencias transitivas. ¿COMPRA1 está en 3FN? Hemos de preguntarnos si existen dependencias transitivas entre atributos no clave.

Veamos las dependencias:

$\text{cod_prov} \rightarrow \text{nomb_prov}$ $\text{cod_prov} \rightarrow \text{tfno}$ (siendo cod_prov el código del proveedor y nomb_prov el nombre del proveedor)

COMPRA1 no está en 3FN porque existen dependencias transitivas entre atributos no clave, por tanto hemos de descomponer:

COMPRA2 (cod_compra, cod_prod, fecha, cantidad, precio, fecha_rec, cod_prov)

PROVEEDOR (cod_prov, nomb_prov, tfno)

Comprobamos FNBC:

PRODUCTO está en FNBC, ya que está en 3FN y todo determinante es clave candidata. COMPRA2 está en FNBC, ya que está en 3FN y todo determinante es clave candidata. PROVEEDOR está en FNBC, ya que está en 3FN y todo determinante es clave candidata.

La tabla inicial COMPRAS queda normalizada hasta FNBC del siguiente modo:

PRODUCTO (cod_prod, nomb_prod)

COMPRA2 (cod_compra, cod_prod, fecha, cantidad, precio, fecha_rec, cod_prov)

PROVEEDOR (cod_prov, nomb_prov, tfno)

De haber hecho el modelo de entidad-relación correctamente a partir del documento de requerimientos (universo del discurso) no habríamos llegado a esos casos iniciales...

UD 03 - IMPLEMENTACIÓN DE BASES DE DATOS RELACIONALES

1. Modelo de datos

Modelo de datos: Lenguaje utilizado para la descripción de una base de datos. Permite describir las **estructuras** (tipos de datos y relaciones entre ellos), las **restricciones de integridad** (condiciones que deben cumplir según necesidades del modelo) y las **operaciones de manipulación de los datos** (insertado, modificación, borrado de datos).

Otra definición: Conjunto de herramientas conceptuales para describir la representación de la información en términos de datos. Comprenden: Estructuras y tipos de datos, operaciones y restricciones.

Los modelos se clasifican según su nivel de abstracción en:

- **Modelos de datos conceptuales:** Describen estructuras de datos y restricciones de integridad. Usados en la etapa de análisis del problema dado. Ej.: Modelo E-R
 - **Modelos de datos lógicos:** Se centran en operaciones y se implementan en algún SGBD. Ej.: Modelo Relacional
 - **Modelos de datos físicos:** Estructuras de datos a bajo nivel implementadas dentro del propio SGBD.
-

El modelo de datos es un lenguaje que presenta dos sublenguajes:
DDL para describir, abstractamente, las estructuras de datos y las relaciones de integridad
DML para describir operaciones de manipulación de datos.

2. Terminología del modelo relacional

Modelo relacional fue propuesto por Codd en los laboratorios de IBM. Es un modelo lógico que establece estructura sobre los datos independientemente de la forma en la que luego se almacenen. El modelo relacional viene de la relación entre este modelo y el concepto matemático de relación: Dados A y B, una relación entre ellos es un subconjunto de $A \times B$

2.1. Relación o tabla. Tuplas. Dominios

- Una **relación** es una **tabla con filas y columnas**.
- Las **tuplas** son **filas**.
- Los **atributos** son **columnas**.

El orden de las filas y columnas es irrelevante.

Atributos: Cada dato que se almacena en la relación (tabla)

- El atributo no puede tomar cualquier valor en la tupla, sino que tiene asociado un dominio de valores.

A menudo el dominio se define a través de la declaración de un tipo para el atributo. Una característica principal de los dominios es que sean atómicos (no puedan ser separados en dominios más simples).

El dominio debe tener:

- Nombre: Sueldo
- Definición lógica: Sueldo neto del empleado
- Tipo de datos: Número entero
- Formato: 9.999€

Tuplas: Cada elemento de la relación. Si la tabla guarda datos de un cliente como DNI o nombre, una tupla o registro es ese DNI y nombre concreto del cliente.

Cada tupla debe cumplir que:

- Se corresponde con un elemento del mundo real
- No hay dos tuplas iguales.

2.2. Grado. Cardinalidad

Grado: Tamaño de una tabla en base a su número de atributos (columnas). A mayor grado, mayor complejidad para trabajar con ella.

Cardinalidad: Número de tuplas o filas de una relación o tabla. Resultan de realizar el producto cartesiano de los dominios (Todas las combinaciones posibles que podría haber)

2.3. Sinónimos

Nomenclatura relacional	=	Nomenclatura tabla	=	Nomenclatura ficheros
relación	=	tabla	=	fichero
tupla	=	fila	=	registros
atributo	=	columna	=	campos
grado	=	nº columnas	=	nº campos
cardinalidad	=	nº filas	=	nº registros

3. Relaciones. Características de una relación (tabla)

- Cada tabla tiene un nombre distinto
- Cada atributo de la tabla toma un solo valor en cada tupla (fila)
- Cada atributo (columna) tiene un nombre distinto en cada tabla, pudiendo coincidir en tablas distintas
- No puede haber dos tuplas (filas) completamente iguales
- El orden de las tuplas (filas) no importa
- El orden de los atributos (columnas) no importa
- Los datos de un atributo (columna) deben ser del mismo dominio.

3.1. Tipos de relaciones (tablas)

Las clasificamos en:

- **Persistentes:** Solo pueden ser borradas por los usuarios
- **Base:** Independientes. Se crean indicando su estructura y sus ejemplares (conjunto de tuplas o filas)

- **Vistas:** Solo almacenan una definición de consulta, de la que se produce una tabla cuyos datos proceden de bases o de otras vistas. Si los datos de la base cambian, los de la vista también.
- **Instantáneas:** Son vistas que almacenan los datos que muestran además de la consulta. Solo modifican el resultado cuando el sistema se refresca. Es como una fotografía de la relación, válida durante un periodo de tiempo concreto.
- **Temporales:** Tablas eliminadas automáticamente por el sistema.

4. Tipos de datos

Con la asignación de tipos de datos se selecciona un dominio para el atributo.

Cada campo o atributo debe poseer un nombre y un tipo de dato (texto, numérico, fecha/hora, Sí/no, autonumérico, memo (texto largo), moneda (representa dinero), objeto OLE (Object Linking and Embedding, enlazado e incrustación de objetos), almacena gráficos, imágenes o textos creados por otras aplicaciones).

5. Claves

Superclave: Atributo o conjunto de atributos que identifican de forma única las tuplas (filas) de una relación (tabla).

Toda la fila como conjunto es una superclave, al no poder haber dos tuplas completamente iguales.

Clave candidata: Atributo o conjunto de atributos que identifican única y mínimamente a cada una de las tuplas de la relación. Siempre hay al menos una clave candidata formada por el conjunto de todos los atributos (superclave), aunque puede haber más de una.

Si la clave candidata está formada por más de un atributo, entonces es una clave **compuesta**, en caso contrario es **simple**.

La clave candidata debe cumplir la **UNICIDAD** (no hay dos tuplas con los mismos valores) y la **IRREDUCIBILIDAD** (si se elimina algún atributo ya no es única)

Para ver las claves candidatas no nos fijamos en un momento concreto sino que conocemos el significado real de los atributos, mirando los posibles valores que podemos llegar a tener.

Clave principal: Clave candidata elegida para identificar cada una de las tuplas. La tabla solo puede tener una clave principal.

Clave alternativa: Clave candidata no elegida como principal

Clave externa, ajena o secundaria: Clave candidata exportada a otra tabla que permite relacionar una o varias tuplas de la tabla receptora con una tupla de la tabla exportadora.

Otra definición: Atributo o conjunto de atributos de una relación cuyos valores coinciden con los valores de la clave primaria de alguna otra relación (o de la misma).

Representan relaciones entre datos.

Son los datos de atributos de una tabla cuyos valores están relacionados con atributos de otra tabla. Las claves ajenas sí pueden repetirse en la tabla.

6. Índices. Características

Índice es una estructura de datos que permite acceder a diferentes filas a través de un campo o campos, permitiendo un acceso mucho más rápido a los datos. (Similar a, en la vida real, distribuir por hojas y crear un índice para consultar las palabras que comienzan por la letra que queremos).

- Son útiles para consultas frecuentes a un rango de filas o fila de una tabla. (Ej.: Usuarios con fecha de ingreso anterior a fecha concreta).
- Los índices son independientes física y lógicamente de los datos, pueden ser creados y eliminados sin afectar ni a las tablas, ni entre ellos.
- No hay límite de columnas a indexar aunque lo que tiene sentido es crearlo para ciertas columnas al agilizar las operaciones de búsqueda en bases de datos grandes.
- Las operaciones de agregar, actualizar o borrar se incorporan automáticamente a los índices aunque las operaciones se ralentizan ya que es necesario actualizar tanto la tabla como el índice.

- Si se elimina un índice, puede observarse una ralentización en el acceso a datos.

7. Valor NULL. Operaciones con este valor

Cuando el valor del campo se desconoce, se le asigna el valor especial `NULL`.

Al trabajar con claves secundarias el valor nulo indica que la tupla o fila no está relacionada con ninguna otra tupla o fila.

No es lo mismo `NULL` que espacio en blanco o cero.

El valor nulo no es ni verdadero ni falso. (Indefinido)

- `TRUE AND NULL -> NULL` (Porque aunque la izquierda es verdadero, la derecha es null)
- `FALSE AND NULL -> FALSE` (Porque la izquierda es falso)
- `TRUE OR NULL -> TRUE` (Porque la izquierda es verdadero)
- `FALSE OR NULL -> NULL` (Porque la izquierda es falso y se evalua el derecho)
- `NOT NULL -> NULL`
- `IS NULL` devuelve verdadero si el valor que se compara es `NULL`

Información adicional: https://es.wikipedia.org/wiki/Null_%28SQL%29

8. Vistas

Una **vista** es una tabla "virtual" cuya filas o columnas se obtienen a partir de una o varias tablas del modelo. Es un filtro de las tablas a las que hace referencia. (O de otras vistas, o de otras tablas o de otras vistas de otras bases de datos).

No hay restricción a la hora de consultar vistas y muy pocas restricciones al modificar datos de ellas.

Al actualizar una vista, realmente actualizamos la tabla (y viceversa). Oracle no permite actualizar vistas. SQL Server y MySQL Server sí aunque con muchas restricciones.

Ventajas:

- Seguridad: No permitir acceso a toda la tabla
- Comodidad: Evita sentencias muy complejas

9. Usuarios. Roles. Privilegios

Usuarios

Usuario: Conjunto de permisos que se aplican a una conexión de base de datos. Tiene funciones como:

- Ser el propietario de ciertos objetos (tablas, vistas)
- Realizar copias de seguridad
- Definir cuota de almacenamiento
- Definir el tablespace (unidad lógica de almacenamiento dentro de BBDD Oracle) por defecto

Privilegios

Permiso dado a un usuario para realizar ciertas operaciones que pueden ser:

- Del sistema.
- Del objeto

Roles

Agrupación de permisos del sistema y de objeto.

10. SQL

Structured Query Language (SQL) es el lenguaje fundamental de los Sistemas de Gestión de Bases de Datos relacionales. Es un lenguaje declarativo que define qué se desea hacer en lugar de cómo hacerlo (eso lo hace el SGBD).

Es necesario consultar la documentación del SGBD para conocer la sintaxis concreta porque algunos comandos, tipos... pueden no seguir el estandar.

SQL se caracteriza por su potencia, versatilidad, facilidad de aprendizaje.

Permite:

- Consultar datos
- Definir la estructura de los datos
- Manipulación
- Especificación de conexiones seguras

10.1. Elementos del lenguaje. Normas de escritura

Diferenciamos entre:

- Comandos: Instrucciones que pueden crearse en SQL. Distinguimos entre:
 - Definición de datos: DDL (Data Definition Language) crear y definir nuevas BBDD, tablas, campos
 - Manipulación de datos: DML (Data Manipulation Language) generar consultas para ordenar, filtrar y extraer datos
 - Control y seguridad de datos: DCL (Data Control Language) administran derechos y restricciones de los usuarios
- Cláusulas: Palabras especiales que permiten modificar el funcionamiento de un comando
- Operadores: Permiten crear expresiones complejas y pueden ser aritméticos o lógicos
- Funciones: Consiguen valores complejos
- Literales: Son constantes o valores completos.

Las instrucciones:

- Siempre terminan con punto y coma
- No distinguen entre mayúsculas o minúsculas
- Los comandos pueden separarse con saltos de línea o espacios. Se pueden tabular líneas.
- Los comentarios son con `/* */` (casi siempre, salvo algún SGBD)

11. Lenguaje de descripción de datos (DDL)

Las instrucciones DDL generan acciones que no se pueden deshacer. Deben usarse con precaución y realizar copias de seguridad.

Con MySQL Workbench se puede:

- Realizar el diseño de la BBDD con Data Modeling e implantar con ingeniería directa la base de datos en el servidor o generar el script SQL con las sentencias DDL para cargarlo posteriormente
- Trabajar en modo gráfico con esta herramientas

En Oracle cada usuario de una base de datos tiene un esquema que tendrá el mismo nombre que el usuario con el que se ha accedido y sirve para almacenar los objetos que posea ese usuario.

Los objetos podrán ser tablas, vistas, índices u otros objetos relacionados con la definición de la BBDD. Podrá crear y manipularlos el usuario y los administradores de la BBDD (en principio, salvo modificaciones de los permisos)

11.1. Creación y borrado de bases de datos. Objetos de la base de datos

Según el SGBD quizás haya que definir un espacio de nombres separado para cada conjunto de tablas (esquemas o usuarios en Oracle).

Crear base de datos implica indicar archivos y ubicaciones y otras indicaciones técnicas y administrativas (lo hace el Administrador).

Crear

```
CREATE DATABASE nombreDeLaBaseDeDatos;
-- MySQL
CREATE SCHEMA nombreDeLaBaseDeDatos;
CREATE DATABASE nombreDeLaBaseDeDatos;
CREATE SCHEMA IF NOT EXISTS nombreDeLaBaseDeDatos;
CREATE DATABASE IF NOT EXISTS nombreDeLaBaseDeDatos;
```

Borrar

```
DROP DATABASE nombreDeLaBaseDeDatos;
```

11.2. Creación de tablas

Los objetos básicos de SQL son las tablas. Es bueno tener planificado:

- El nombre de la tabla
- El nombre de sus columnas
- El tipo y tamaño de datos en cada columna
- Las restricciones sobre los datos
- La información adicional
- Las reglas para los nombres de tablas (No puede estar duplicada la tabla en el mismo esquema, debe comenzar por carácter alfabético, longitud máxima 30 caracteres, solo se permiten letras del alfabeto inglés-dígitos-guion bajo, no puede usarse palabras reservadas de SQL, no distingue entre mayúsculas y minúsculas, si tiene espacios en blanco hay que entrecerrarlo con comillas dobles y en el estándar SQL99 (respetado por Oracle) si se crea entre comillas dobles se hará sensible a mayúsculas.

La sintaxis básica es

```
CREATE TABLE [esquema.]NOMBRE_TABLA(
    columna1 tipo_dato,
    columna2 tipo_dato,
    ...
    columnaN tipo_dato);
```

En MySQL

```
CREATE TABLE [IF NOT EXISTS] [esquema.]NOMBRE_TABLE(
    columna1 tipo_dato [restricciones de columna]
    columna2 tipo_dato [restricciones de columna]
    ...
    [reestricciones de tabla])
    [{ENGINE|TYPE} = Tipo_tabla];
)
```

ENGINE indica el tipo de almacenamiento para la tabla mediante Tipo Tabla. Ej.: Las tablas MyISAM y las tablas InnoDB (Tipos de almacenamiento pueden verse con `SHOW ENGINES;`)

Restricciones de columna: Afectan solo a esa columna (clave primaria, no nulo, etc.)

Restricciones de tabla: se indican después de especificar todas las columnas, se les puede asignar un nombre y pueden afectar a varias columnas.

SOLO PUEDEN CREARSE TABLAS SI SE POSEEN LOS PERMISOS NECESARIOS

11.2.1. Tipos de datos

MySQL

Numéricos

BIT o **BOOL** (0 o 1)

TINYINT o **UNSIGNED TINYINT** (-128 a 127. Con unsigned 0 a 255)

SMALLINT o **UNSIGNED SMALLINT**

MEDIUMINT o **UNSIGNED MEDIUMINT**

INT

BIGINT

FLOAT(m,d) m ancho de pantalla, d parte decimal

DOUBLE

DECIMAL

Caracteres

CHAR(n). De 1 a 255 caracteres. De longitud fija.

VARCHAR. De 1 a 255 caracteres. De longitud variable.

TINYTEXT, **TINYBLOB**. Text es para texto plano sin formato y sin distinguir mayúsculas y minúsculas.

BLOB es para objetos binario (cualquier info desde texto hasta imágenes o archivos de sonido y video)

TEXT y **BLOB**. Rango de 255 - 65535 caracteres. TEXT compara contenido sin distinguir mayus y minus, BLOB sí distingue

MEDIUMTEXT, **MEDIUMBLOB**

LONGTEXT, **LONGBLOB**

Varios

DATE. Fecha. Formato por defecto YYYY MM DD

DATETIME. Fecha y hora.

TIMESTAMP. Desde 1-01-1970 hasta 2037

TIME. Almacena una hora.

YEAR. Almacena un año

SET. Ninguno uno o varios valores de una lista

ENUM. Solo puede almacenar un valor de la lista.

Oracle

Numéricos

NUMBER Números fijos y en punto flotante. Admiten hasta 28 dígitos de precisión. Puede indicarse la precisión y la escala.

FLOAT Datos numéricos en punto flotante.

Caracteres

CHAR(n). De longitud fija. Aunque se introduzca un valor más corto, se rellana al tamaño indicado.

VARCHAR2(n). De longitud variable.

VARCHAR(n). En desuso. Se usar VARCHAR2(n)

NCHAR(n). De longitud fija. Solo caracteres Unicode.

NVARCHAR2. De longitud variable. Solo caracteres Unicode.

LONG o **LONG RAW** o **RAW** Caracteres de longitud variable / Cadenas binarias de ancho variable.

Como máximo hasta 2GB. Deberán ser convertidos al moverse entre sistemas. Es un tipo de datos obsoleto. Se usan los datos de tipo LOB y Oracle recomienda que se convierta si aun se está usando. No puede usarse en WHERE, GROUP BY.... La tabla solo puede contener una columna de tipo LONG y solo soporta acceso secuencial.

LOB (BLOG CLOB NCLOB BFILE). Permite almacenar y manipular bloques grandes de datos no estructurados en formato binario o de carácter. Admite hasta 8 TB. Puede contener varias columnas de ese tipo. Soportan acceso aleatorio. Las tablas con columnas de tipo LOB no pueden ser replicadas.

BLOB Datos binarios no estructurados. Hasta 8TB.

NCLOB. Datos de tipo carácter. Hasta 8TB. Segundo Unicode nacional.

BFILE. Datos binarios no estructurados en archivos del sistema operativo, fuera de la base de datos.

La columna BFILE almacena localizador del archivo a uno externo que contiene los datos. Debe asegurarse de que exista el archivo y de que los procesos de Oracle tengan permisos de lectura.

ROWID. Dirección única de cada fila de la tabla de la BBDD. Este campo no aparece en las operaciones del CRUD, es un tipo de dato usado exclusivamente por oracle.

OOOOOOFFFBBBBBRRR OOOOOO (Segmento de la base de datos) FFF (Número de fichero del tablespace que contiene la fila) BBBB (Bloque de datos que contiene la fila) RRR (Número de fila en el bloque)

UROWID es ROWID universal en tablas que no sean Oracle, externas.

Otros

DATE Fecha y hora

TIMESTAMP Datos de tipo hora, fraccionando los segundos

TIMESTAMP WITH TIMEZONE Con zona horaria

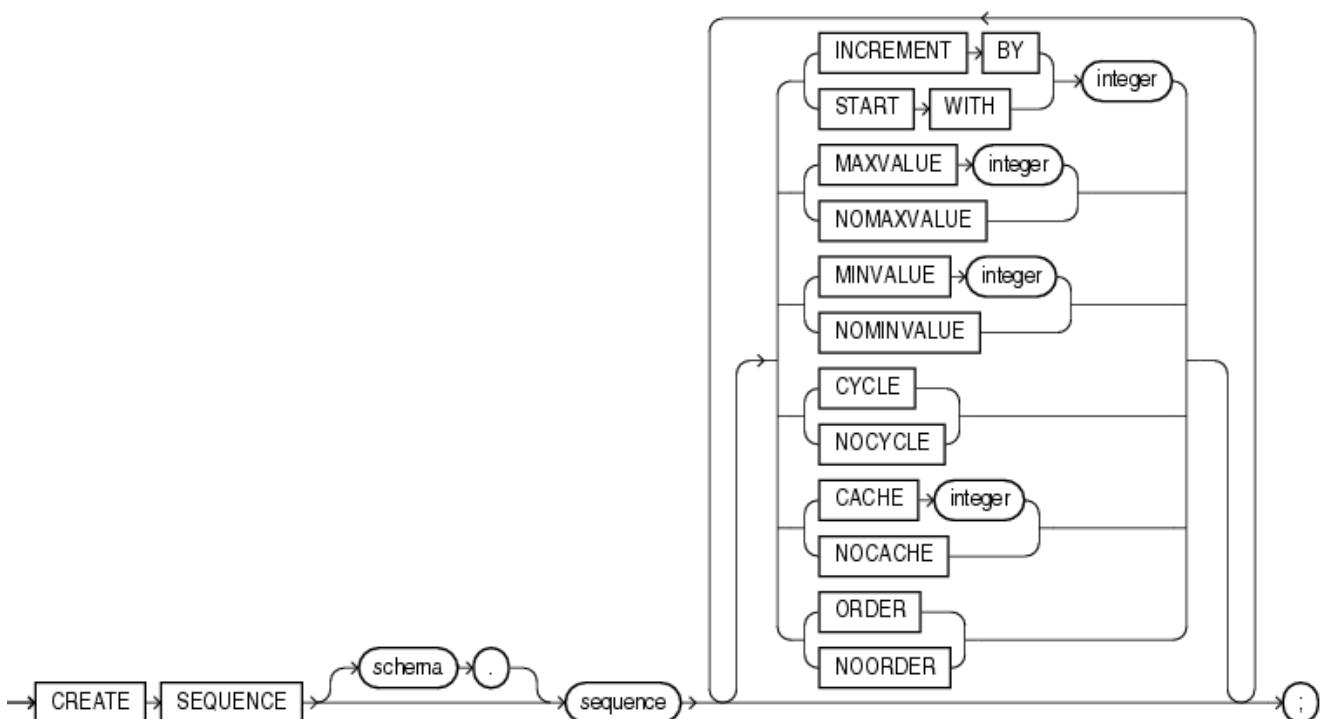
TIMESTAMP WITH LOCAL TIME ZONE. Se ajustará la hora en el SELECT a la zona horaria de la sesión actual

XMLType. Tipo de datos abstracto. Se trata de un clob. Se asocia a un esquema XML para la definición de su estructura.

11.2.2. Creación de secuenciales en Oracle

Los *sequence* son objetos de la base de datos con los que los usuarios pueden generar números únicos.

- Pueden usarse para generar claves primarias automáticamente.
- Se incrementan independientemente del commit o el rollback de la transacción.
- Un usuario nunca puede adquirir el sequence generado por otro usuario.
- Estos son independientes de las tablas pudiéndose usar para una o varias tablas.
- Los posibles saltos de secuenciales pueden deberse a transacciones revertidas.



```
CREATE SEQUENCE nombre_secuencial
START WITH 1000
INCREMENT BY 1
MINVALUE 1 -- Menor o igual al START WITH y menor que MAXVALUE. Hay NOMINVALUE.
MAXVALUE 1000000000000 -- Por defecto es NOMAXVALUE (10^27)
/*PREALMACENAR*/
NOCACHE -- Los valores de la secuencia no están preasignados (Si omite CACHE y NOCACHE,
almacena en cache 20 sequence de forma predeterminada. Con CACHE se puede definir)
/*REINICIAR*/
NOCYCLE; -- No continuar generando valores al alcanzar los límites. Es lo predeterminado.
Con CYCLE volvería a reiniciarse.
/*ORDENADOS*/
-- ORDER garantiza que los secuenciales se generen en orden de peticiones. Util si se estan
usando numeros de secuencia como timestamps. NOORDER no se generan en orden de peticon. Es
el valor por defecto.
```

11.3. Restricciones

Condición que una o varias columnas deben cumplir obligatoriamente.

Cada restricción lleva un nombre (importante que sea un nombre que ayude a identificarla y que sea único para cada esquema -usuario-). Si no se le pone, el SGBD lo pondrá automáticamente.

Como consejo se puede usar:

- Tres letras para el nombre de tabla
- _
- Tres letras para la columna afectada
- _
- Dos letras para el tipo de restricción (PK Primary Key, FK Foreign Key, NN Not Null, UK Unique, CK Check (Validación))

Para indicar las restricciones a nivel de columna, la sentencia en SQL estándar es:

```
CREATE TABLE NOMBRETABLA (
    Columna1 Tipo_Dato
        [CONSTRAINT nombredelarestricción]
        [NOT NULL]
        [UNIQUE]
        [DEFAULT valor]
        [CHECK condición]
        [PRIMARY KEY]
        [FOREIGN KEY]
        [REFERENCES nombreTabla [(columna [, columna ])]]
        [ON DELETE CASCADE]],
    Columna2 Tipo_Dato
        [CONSTRAINT nombredelarestricción]
        [NOT NULL]
        [UNIQUE]
        [DEFAULT valor]
        [CHECK condición]
        [PRIMARY KEY]
        [FOREIGN KEY]
        [REFERENCES nombreTabla [(columna [, columna ])]]
        [ON DELETE CASCADE]]
        [CHECK condición],...);
```

Ejemplito en Oracle:

```
CREATE TABLE USUARIOS(
    login VARCHAR(15) CONSTRAINT usu_log_PK PRIMARY KEY,
    password VARCHAR(8) NOT NULL,
    fecha_ingreso DATE DEFAULT SYSDATE;
);
```

Ejemplito en MySQL:

```
CREATE TABLE USUARIOS(
    login VARCHAR(15) PRIMARY KEY,
    password VARCHAR(8) NOT NULL,
    fecha_infreso TIMESTAMP DEFAULT CURRENT_TIMESTAMP;
)
```

Otra restricción puede ser **[AUTO_INCREMENT]** en MySQL con la que se asigna un identificador único a cada fila generando secuenciaa de números de forma automática comenzando por el 1. (En Oracle están los sequence).

Igualmente, también se pueden definir las columnas de la tabla y después especificar las restricciones pudiendo referir varias columnas en una única restricción.

```
-- Crear la tabla sin restricciones
CREATE TABLE USUARIOS (
```

```

        Login VARCHAR(15),
        Password VARCHAR(8) NOT NULL,
        Fecha_Ingreso DATE DEFAULT SYSDATE
    );

-- Añadir la restricción PRIMARY KEY
ALTER TABLE USUARIOS
ADD CONSTRAINT usu_log_PK PRIMARY KEY (Login);

-- Añadir la restricción NOT NULL
ALTER TABLE USUARIOS
MODIFY Password VARCHAR(8) NOT NULL;

```

NOT NULL

Se obliga a que la columna tenga un valor.

```
CREATE TABLE USUARIOS(
    f_nacimiento DATE NOT NULL);
```

UNIQUE

Para que no se repitan valores en la columna. Oracle crea un índice automáticamente cuando se habilita esta restricción y lo borra al deshabilitarla.

Ejemplo Oracle:

```
CREATE TABLE USUARIOS (
    Login VARCHAR(25)
    CONSTRAINT Usu_Log_UK UNIQUE);
```

Ejemplo MySQL y Oracle

```
CREATE TABLE USUARIOS (
    Login VARCHAR(25) UNIQUE);
);
```

y a la vez

```
CREATE TABLE USUARIOS (
    Login VARCHAR (25),
    Correo VARCHAR (25),
    CONSTRAINT Usuario_UK UNIQUE (Login, Correo));
```

La restricción fija que es independiente a "Correo" y común a "Login" y a "Correo". Se está especificando que la combinación de AMBOS VALORES ha de ser única.

PRIMARY KEY

Oracle y MySQL

```
CREATE TABLE USUARIOS (
    Login VARCHAR (25) PRIMARY KEY);
```

Oracle poniendo nombre

```
CREATE TABLE USUARIOS (
    Login VARCHAR (25)
    CONSTRAINT Usu_log_PK PRIMARY KEY);
```

Claves compuestas obligatoriamente deben ser restricciones de tabla tras haber declarado las columnas:

```
CREATE TABLE USUARIOS (
    Nombre VARCHAR (25),
    Apellidos VARCHAR (30),
    F_Nacimiento DATE,
    CONSTRAINT Usu_PK PRIMARY KEY(Nombre, Apellidos, F_Nacimiento));
```

REFERENCES. FOREIGN KEY

Al indicar la clave ajena debe hacerse referencia a la tabla (CONSTRAINT) y a los campos de donde procede (REFERENCES).

En Oracle puede ponerse la restricción junto al campo que va a ser la clave ajena.

```
CREATE TABLE USUARIOS (
    Login VARCHAR (25) PRIMARY KEY,
    Cod_Partida INTEGER(8) CONSTRAINT Cod_Part_FK REFERENCES PARTIDAS(Cod_Partida));
```

Si el campo al que hace referencia es la clave principal en su tabla no es necesario indicar el REFERENCES

```
CREATE TABLE USUARIOS (
    Login VARCHAR (25) PRIMARY KEY,<br />Cod_Partida INTEGER(8)
    CONSTRAINT Cod_Part_FK
    REFERENCES PARTIDAS);
```

En MySQL la clave ajena se pone al final y debe indicarse FOREIGN_KEY para indicar a qué campo se refiere

```
CREATE TABLE USUARIOS (
    Login VARCHAR (25) PRIMARY KEY,
    Cod_Partida INTEGER(8),
    F_Partida DATE,
    CONSTRAINT Partida_Cod_F_FK FOREIGN KEY (Cod_Partida)
    REFERENCES PARTIDAS(Cod_Partida));
```

```
CREATE TABLE USUARIOS (
    Login VARCHAR (25) PRIMARY KEY,
    Cod_Partida INTEGER(8),
    F_Partida DATE,
    CONSTRAINT Partida_Cod_F_FK FOREIGN KEY (Cod_Partida, F_Partida)
    REFERENCES PARTIDAS(Cod_Partida, F_Partida));
```

Al relacionar campos se necesita que el dato del campo que es clave ajena haya sido incluido en la tabla de procedencia. --> Integridad referencial.

Si se hace referencia una tabla no creada, dará fallo. Deben crearse primero las que no tengan claves ajenas.

Si se quieren borrar las tablas tienen que borrarse las de claves ajenas antes.

Otras soluciones es añadir tras la cláusula REFERENCE:

- ON DELETE CASCADE: Borra todos los registros cuya clave ajena sea igual al registro borrado
- ON DELETE SET NULL: Coloca el valor NULL en todas las claves ajenas relacionadas con la borrada
- ON DELETE RESTRICT: (Igual que NOACTON en MySQL) no permite borrar el registro principal si hay registros asociados.
- ON UPDATE CASCADE: Modificar el valor de la clave ajena de todos los registros cuya clave ajena sea igual a la del registro modificado.
- ON UPDATE SET NULL: Coloca el valor NULL en todas las ajenas relacionadas con la modificada.
- ON UPDATE RESTRICT: (Igual que NOACTON en MySQL) no permite modificar la clave primaria del registro principal si hay registros asociados con ese valor en su clave ajena.

```
CREATE TABLE USUARIOS (
    Login VARCHAR (25) PRIMARY KEY,<br />      Cod_Partida INTEGER(8),
    F_Partida DATE,
    CONSTRAINT Partida_Cod_F_FK FOREIGN KEY (Cod_Partida)
    REFERENCES PARTIDAS(Cod_Partida) ON DELETE CASCADE);
```

Por defecto en MySQL si no se indica nada son ON DELETE RESTRICT y ON UPDATE RESTRICT

DEFAULT Y VALIDACION

Para asignarle un valor por defecto **DEFAULT**.

En Oracle pueden añadirse expresiones, constantes, funciones SQL y variables. Como en DATE la función SYSDATE.

En MySQL no pueden añadirse expresiones o SQL, solo en el caso del TIMESTAMP para CURRENT_TIMESTAMP.

Para comprobar que los valores son adecuados se usa **CHECK** comprobando una condición determinada:

```
CREATE TABLE USUARIOS (Login VARCHAR (25) PRIMARY KEY,
    Credito FLOAT(4) CHECK (Crédito BETWEEN 0 AND 2000));
```

Para poner varios CHECK a la misma columna se ponen varios CONSTRAINT seguidos y separados por comas.

En MySQL existe el tipo de datos ENUM para crear enumeraciones (campos que admiten solo valores fijos):

```
CREATE TABLE USUARIOS (
    Login VARCHAR (25) PRIMARY KEY,
    Pais ENUM ('España', 'Francia', 'Bélgica'));
```

11.4. Eliminación de tablas

Las tablas se eliminan con el comando **DROP TABLE**

```
DROP TABLE tabla1, tabla2...
```

En Oracle no se permite el borrado de varias tablas y además puede usarse **CASCADE CONSTRAINT** para que se borren las restricciones donde es clave ajena y después la tabla en cuestión.

```
DROP TABLE tabla [CASCADE CONSTRAINT];
```

Al borrar la tabla desaparecen sus datos, las vistas siguen existiendo pero no funcionan.
Se dispone de **TRUNCATE TABLE** que permite eliminar los datos pero sin eliminar su estructura.

11.5. Modificación de tablas

Cambiar el nombre de una tabla:

```
RENAME TABLE NombreViejo TO NombreNuevo
```

Añadir columnas a la tabla:

```
ALTER TABLE NombreTabla ADD
(ColumnaNueva1 TipoDatos [Propiedades]
ColumnaNueva1 TipoDatos [Propiedades]);
```

Eliminar columnas de la tabla:

```
ALTER TABLE NombreTabla DROP COLUMN  
(Columna1, Columna2...);
```

Modificar columnas de una tabla:

```
ALTER TABLE NombreTabla MODIFY  
(Columna1 TipoDatos [prop],  
Columna2 TipoDatos [prop]);
```

Renombrar columnas de una tabla

```
-- En Oracle  
ALTER TABLE NombreTabla RENAME COLUMN NombreAntiguo TO NombreNuevo;  
-- En MySQL  
ALTER TABLE NombreTabla CHANGE COLUMN NombreAntiguo NombreNuevo definicionColumna;
```

Ejemplo:

```
-- En Oracle  
ALTER TABLE USUARIOS RENAME COLUMN User TO Login;  
-- En MySQL  
ALTER TABLE USUARIOS CHANGE COLUMN User Login VARCHAR(10);
```

Borrar restricciones

```
--Oracle  
ALTER TABLE NombreTabla DROP CONSTRAINT NombreRestriccion;
```

Modificar el nombre de las restricciones

```
--Oracle  
ALTER TABLE NombreTabla RENAME CONSTRAINT NombreViejo To NombreNuevo;
```

Activar o desactivar restricciones

```
--Oracle  
ALTER TABLE NombreTabla DISABLE CONSTRAINT NombreRestriccion [CASCADE];  
ALTER TABLE NombreTabla ENABLE CONSTRAINT NombreRestriccion [CASCADE];
```

Con CASCADE se desactivan/activan también las que dependan de esta.

En MySQL

```
-- Si queremos añadir restricciones, por ejemplo la restricción UNIQUE:  
ALTER TABLE NombreTabla ADD [CONSTRAINT [nombrecons]] UNIQUE [INDEX|KEY]  
[nombre_indice];  
-- Si queremos eliminar una clave primaria:  
ALTER TABLE NombreTabla DROP PRIMARY KEY;  
--Si queremos añadir una restricción de clave ajena:  
ALTER TABLE NombreTabla ADD [CONSTRAINT [nombrecons]] FOREIGN KEY [nombre_indice]  
(nombre_columna_indice,...);
```

Ejemplo: Queremos poner una restricción a sueldo para que tome valores entre 1000 y 1200,

```
ALTER TABLE EMPLEADOS ADD CONSTRAINT emp_sue_CK CHECK (Sueldo BETWEEN 1000 AND 1200);
```

11.6. Creación y eliminación de índices

Puede crearse con `CREATE INDEX NombreIndice ON NombreTabla (Columna1, Columna2...)`

También cuando se crea la tabla con la opción INDEX:

```
CREATE TABLE NombreTabla
campo1 tipoDatos,
campo2 tipoDatos,...,
INDEX [nombreIndice] (campos);
```

O con un `ALTER TABLE`:

```
ALTER TABLE NombreTabla
ADD INDEX [nombreIndice] (campo);
```

El índice puede eliminarse con:

```
DROP INDEX NombreIndice ON NombreTabla;
```

Los que hay pueden verse con

```
SHOW INDEX FROM NombreBaseDeDatos.NombreTabla;
```

La mayoría de los índices se crean de forma implícita al poner las restricciones PRIMARY KEY, FOREIGN KEY o UNIQUE

11.7. Creación y eliminación de vistas

```
-- CREACION
CREATE VIEW nombre_vista AS sentencia_select;
-- Ejemplo
CREATE VIEW vista_empleado AS SELECT nombre, oficina FROM empleado;

-- MODIFICACION
CREATE OR REPLACE VIEW nombre_vista AS sentencia_select;
ALTER VIEW nombre_vista AS sentencia_select;
-- Alter solo puede usarse cuando la vista ya esta creada

-- ELIMINACION
DROP VIEW nombre_vista;
```

12. Lenguaje de control de datos (DCL)

12.1 Creación de usuarios

Cuando se crea el usuario se establecen las claves de acceso que podrá modificar el Administrador o el propietario. La BBDD almacena encriptadas las claves en la tabla del diccionario `DBA_USERS`. En MySQL están encriptadas en la BBDD `mysql` en la tabla `user`.

Pueden crear el que tenga cuenta con privilegios de administrador. Los usuarios creados pueden verse con las vistas `ALL_USERS` y `DBA_USERS`. Para ver en mi sesión los usuarios que existen se pone:
`DESC SYS.ALL_USERS;`

Crear usuarios en Oracle

```
CREATE USER quinidio
IDENTIFIED BY prueba123
[DEFAULT TABLESPACE tablespace]
```

```
[TEMPORARY TABLESPACE tablespace]
[QUOTA int {K | M} ON tablespace]
[QUOTA UNLIMITED ON tablespace]
[PROFILE perfil];
```

TABLESPACE: Se almacenan los objetos que el usuario cree. Si no se especifica TABLESPACE será SYSTEM.

QUOTA: Espacio en Megabytes o Kilobytes en el Tablespace asignado. Si no se especifica el usuario no tendrá espacio y no puede crear objetos.

PROFILE: Perfil al usuario. Si no, se le asigna el por defecto.

Pueden modificarse los usuarios con `ALTER USER`:

```
ALTER USER quinidio
IDENTIFIED BY clave_acceso
[DEFAULT TABLESPACE tablespace ]
[TEMPORARY TABLESPACE tablespace]
[QUOTA int {K | M} ON tablespace]
[QUOTA UNLIMITED ON tablespace]
[PROFILE perfil];
```

El usuario sin privilegios de admin solo puede cambiar su clave de acceso.

Para eliminar usuario se usa el comando `DROP USER`:

```
DROP USER quinidio [CASCADE]
```

Con CASCADE se borran todos los objetos del usuario antes de borrarlo. Sin esa opción no dejará borrar al usuario si tiene tablas creadas.

Crear usuarios en MySQL

```
CREATE USER quinidio@`equipo`
IDENTIFIED BY `miclave`
```

En equipo se pone el nombre del equipo desde el que el usuario puede conectarse (localhost, una ip, % cualquier máquina).

Los usuarios creados pueden verse con `select * from mysql.user;`

12.2 Permisos

Aunque se tenga el usuario, si intentamos crear una tabla no habrá permisos para ello. Los permisos pueden agruparse formando **roles**.

Dar permisos

Los permisos se otorgan con el comando `GRANT`

En Oracle

Si pongo `GRANT ALL` doy todos los privilegios
Si pongo `TO PUBLIC` estará disponible para todos los usuarios
Puedo poner `TO 'miRol'` para otorgar al rol 'miRol'

Privilegios de objetos

```
GRANT SELECT, INSERT, UPDATE, DELETE ON mytable TO quinidio;
```

añadir `WITH GRANT OPTION` permite que el receptor del privilegio se lo asigne a otros

Privilegios de sistema

Asigno un rol, doy permisos para ejecutar un tipo de comando SQL o realizar acciones sobre objetos (CREATE USER, ALTER, CREATE, CREATE VIEW...)

Voy a crear un rol:

```
CREATE ROLE CONNECT;
GRANT SELECT ON SELECT_MYAPP TO CONNECT;
```

Ahora se lo asigno a un usuario concreto:

```
GRANT CONNECT TO quinidio;
```

Añadir **WITH ADMIN OPTION** permite que el receptor de los privilegios los conceda a otros usuarios o roles

En MySQL

```
GRANT SELECT, INSERT, UPDATE, DELETE ON mytable TO quinidio;
```

Nota. En versiones de MySQL anteriores a la versión 8.x, el comando GRANT también permitía crear a un usuario en el momento de concederle permisos.

```
GRANT {permiso[(listacolumnas)] [, permiso[(listacolumnas)]...|ALL]}
ON [tipoobjeto]{nombretabla | *.* | basedatos.*}
TO {usuario1 [IDENTIFIED BY [PASSWORD] contraseña} [,<code>usuario1 [IDENTIFIED BY
[PASSWORD] contraseña] ...
[WITH GRANT OPTION];
```

Revocar permisos

Los permisos se revocan con el comando **REVOKE**

En Oracle

Sobre objetos

```
REVOKE INSERT, UPDATE ON mytable TO 'quinidio'@'%';
```

Sobre sistema o roles

```
REVOKE CONNECT FROM quinidio; -- Le quito el rol connect
REVOKE DROP USER FROM alejandro; -- Le quito el permiso de eliminar usuarios
```

En MySQL

```
REVOKE INSERT, UPDATE ON mytable TO quinidio;
REVOKE DROP USER FROM usuario1@'%';
```

UD 04 - REALIZACIÓN DE CONSULTAS

1. Introducción

SQL nació a partir de la publicación "A relational model of data for large shared data banks" de Edgar Frank.

IBM aprovechó el modelo que planteó Codd para desarrollar un lenguaje acorde con el modelo relacional (SEQUEL). Este lenguaje con el tiempo se convirtió en SQL. En 1979 la empresa Relational Software sacó al mercado la primera implementación comercial de SQL. Esa empresa es hoy día Oracle.

En 1992 ANSI e ISO completaron la estandarización de SQL y se definieron las sentencias básicas que se debía contemplar para que SQL fuera estándar. (ANSI-SQL o SQL92)

Hoy día todas las bases de datos cumplen este estándar, aunque cada fabricante puede añadir sus propias mejoras.

Después de la DDL (Data Definition Language) viene la DML (Data Manipulation Language): El conjunto de sentencias orientadas a consultas y al manejo de los datos creados.

En Oracle: Las sentencias pueden ser ejecutadas desde el Application Express de oracle usando SQL Workshops y eligiendo SQL Commands. También se puede desde el entorno de SQL *Plus. Run SQL Command Line. El primer paso es conectarse usando un nombre de usuario que tenga permisos necesarios para la tabla deseada (Orden CONNECT seguida del nombre de usuario)

En MySQL: Las sentencias pueden ser ejecutadas desde cualquier cliente gráfico (Workbench de MySQL por ejemplo) o desde la línea de comandos (Command Line Client), pedirá la contraseña del usuario root.

En ambos casos debe iniciarse o arrancarse el servidor de BBDD y conectar o abrir conexión con el servidor con un usuario con permisos necesarios

```
mysql -u root -p // Usuario root y pide la contraseña
```

2. La sentencia SELECT

La sentencia SELECT consta de 4 cláusulas:

- Cláusula **SELECT** (obligatoria) seguida de los nombres de las columnas separados por comillas simples
- Cláusula **FROM** (obligatoria) seguida del nombre de las tablas del que proceden las columnas de arriba
- Cláusula **WHERE** (opcional) seguida de criterio de selección o condición
- Cláusula **ORDER BY** (opcional) seguida de criterio de ordenación

(Después de SELECT se puede poner **ALL** (por defecto) o **DISTINCT** para que suprima filas con resultados iguales)

2.1. Cláusula SELECT

- Puede anteponerse el nombre de las tablas de la que proceden
- Pueden incluirse todas las columnas (con un *)
- Pueden ponerse alias a los nombres de las columnas (recomendado ya que estas son las cabeceras de presentación de los datos obtenidos). Debe ponerse entre comillas dobles a continuación del nombre de la columna
- Puede sustituirse el nombre de las columnas por constantes, expresiones o funciones SQL.

```
SELECT nombre "Nombre de la sala" FROM SALA;  
SELECT 4*3/100 "MiExpresion", capacidad FROM SALA;
```

2.2. Cláusula FROM

Se definen los nombres de la tabla.

- Si es más de una deben aparecer separadas por comas (se le llama consulta combinada o join: habrá que aplicar una condición de combinación a través del WHERE).
- Puede asociarse un alias a las tablas para abbreviar
- En Oracle se puede añadir el nombre del usuario que es propietario de las tablas **USUARIO.TABLA** así se distinguirá entre las tablas de un usuario y de otro (pueden tener el mismo nombre)

2.3. Cláusula WHERE

Se pondrá la condición que han de cumplir las filas.

El criterio de búsqueda o condición puede tener complejidad variable conjugándose operadores de diversos tipos, funciones, expresiones...

2.4. Cláusula HAVING

Si existen grupos de registros o si se quieren seleccionar datos de la tabla restringiéndolos a cumplir una condición dada por operadores aritméticos de agrupación (SUM, MAX, COUNT, MIN, AVG...) ya no puede usarse WHERE sino que debe usarse HAVING.

```
SELECT nombre, apellido1, apellido2
FROM ASISTENTE
GROUP BY NOMBRE;

SELECT nombre
FROM empleado
GROUP BY nombre
HAVING AVG(salario) < 200;
```

Normalmente **HAVING** va acompañado de **GROUP BY** porque opera sobre los grupos que esta devuelve.

2.5. Ordenación de registros. Cláusula ORDER BY.

Para especificar criterio de ordenación. Puede ordenarse por tipo carácter, número o fecha.
Es posible ordenar por más de una columna o a través de una expresión creada por columnas, una constante (aunque no tenga sentido) o funciones SQL.

Se sigue el primer criterio y en caso de empate el segundo, el tercero...

```
SELECT nombre, apellido1, apellido2
FROM ASISTENTE
ORDER BY apellido1, apellido2, nombre;
```

Se pueden referenciar los campos por su posición en la lista de selección.

Si se pone un número mayor a la cantidad de campos aparece un mensaje de error y la sentencia no se ejecuta.

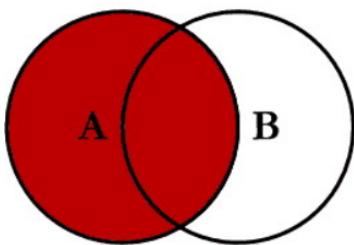
```
FROM ASISTENTE
ORDER BY 4; -- Ordena por la empresa
```

Se puede añadir:

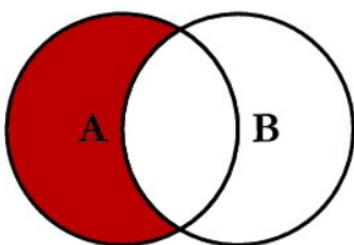
- **DESC** De mayor a menor
- **ASC** De menor a mayor

2.6. Tipos de JOIN

SQL JOINS

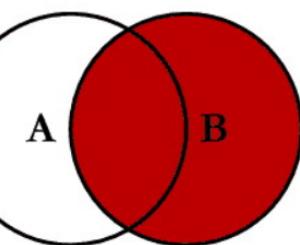
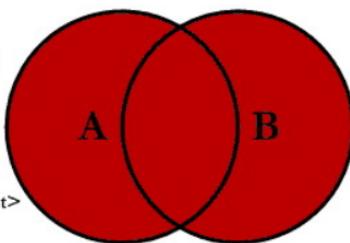


```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

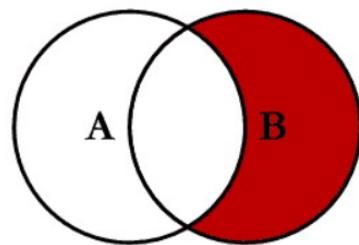


```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

© C.L. Moffatt, 2008

INNER JOIN: Devuelve las filas que tengan columnas cuya condición coincide en ambas tablas.

```
SELECT d.nombre, e.nombre
FROM departamento d
JOIN empleado e
ON d.id = e.departamento;
```

LEFT JOIN: Devuelve todas las filas de la tabla de la izquierda y las coincidentes de la tabla derecha. En los campos no coincidentes de la izquierda aparece NULL.

Todos los proyectos, apareciendo los empleados que están asignados a algún proyecto.

```
SELECT p.nombre, e.nombre
FROM proyecto p
LEFT JOIN empleado e
ON p.id = e.proyecto;
```

Proyectos sin empleado

```
SELECT p.nombre
FROM proyecto p
LEFT JOIN empleado e
ON p.id = e.proyecto
AND e.proyecto IS NULL;
```

RIGHT JOIN: Devuelve todas las filas de la tabla de la derecha más las coincidentes con la de la tabla izquierda. En los campos no coincidentes de la derecha aparece NULL.

Todos los empleados, apareciendo los proyectos a los que está asignado algún empleado.

```
SELECT p.nombre, e.nombre
FROM proyecto p
RIGHT JOIN empleado e
ON p.id = e.proyecto;
```

Empleados sin proyecto.

```
SELECT p.nombre
FROM proyecto p
RIGHT JOIN empleado e
ON p.id = e.proyecto
AND p.id IS NULL;
```

OUTER JOIN o FULL OUTER JOIN: Todas las filas de la tabla de la derecha y todas las filas de la tabla de la izquierda. En los campos no coincidentes aparece NYLL.

El FULL OUTER JOIN no existe en MySQL, ahí debe hacerse UNION entre LEFT JOIN y RIGTH JOIN.

Listas todos los proyectos y todos los empleados

```
SELECT p.nombre, e.nombre
FROM proyecto p
OUTER JOIN empleado e
ON p.id = e.proyecto;
```

```
-- En MySQL
SELECT nombre, e.nombre
FROM proyecto p
LEFT JOIN empleado e
ON p.id = e.proyecto
UNION
SELECT p.nombre, e.nombre
FROM proyecto p
RIGHT JOIN empleado e
ON p.id = e.proyecto.
```

3. Operadores

3.1. Operadores de comparación o relacionales

Para comparar dos variables.

```
= != < > <= >= IN, NOT IN, BETWEEN, NOT BETWEEN, LIKE '%ddfs_'
(% sustituye a un conjunto de caracteres _ a un caracter), IS NULL, IS NOT NULL.
```

Cuando se hace un ORDER BY los valores NULL aparecen en primer lugar con ASC y al final con DESC.

Se pueden usar expresiones regulares con REGEXP

3.2. Operadores aritméticos y de concatenación

Para realizar cálculos con valores numéricos + - * /

La concatenación se hace con CONCAT en MySQL o con || en Oracle

```
-- MySQL
SELECT CONCAT(cdemp, " ", nombre)
FROM empleado;
-- Oracle
SELECT nombre, apellido1 || ' ' || apellido2
FROM ponente;
```

3.3. Operadores lógicos

Para evaluar más de una expresión lógica. **AND OR NOT**

3.4. Precedencia

El orden de precedencia es:

- Multiplicación y división
- Suma y resta
- Concatenación (En el caso de Oracle)
- Todas las comparaciones
- Operadores (IS NULL, IS NOT NULL, LIKE, BETWEEN...)
- NOT
- AND
- OR

4. Consultas calculadas

Los campos calculados se obtienen a través de la sentencia **SELECT** poniéndose la expresión deseada.

No modifica los valores originales de las columnas, ni de la tabla de resultados, solo muestra una columna nueva con los valores calculados.

```
SELECT tema, precio, precio + 10 AS nuevoPrecio  
FROM CONFERENCIA;  
  
SELECT nombre, salario, salario + 25  
FROM empleado;
```

(También pueden ir en WHERE)

5. Funciones

Hay funciones ya creadas que facilitan la creación de consultas más complejas.

Estas varían según el SGBD. Realmente son operaciones que se realizan sobre los datos y realizan un cálculo determinado.

Oracle proporciona la tabla DUAL con un único campo llamado DUMMY y una única FILA.

MySQL proporciona para pruebas las bases de datos de nombre Test. Esta base de datos no contiene ninguna tabla.

Veamos, sobre todo, las de Oracle.

5.1. Funciones numéricas

- Valor absoluto - **ABS(n)**:
Oracle: `SELECT ABS(-17) FROM DUAL;` -- Resultado 17
MySQL: `SELECT ABS(-17);` -- Resultado 17 (No se necesita consultar ninguna tabla)
- e^n - **EXP(n)**:
`SELECT EXP(2) FROM DUAL;` -- Resultado 7,38
- **CEIL(n)** Entero inmediatamente superior o igual a n
- **FLOOR(n)** Entero inmediatamente inferior o igual a n
- **MOD(m,n)** Resto dividir m y n
- **POWER(valor,exponente)** Potencia valor ^ exponente
- **ROUND(n, decimales)** Redondea n al siguiente con los decimales indicados
- **SQRT(n)** Raíz cuadrada
- **TRUNC(m,n)** Trunca número a la cantidad especificada. Si no se pone segundo argumento se truncan todos los decimales.
- **SIGN(n)** Si es positivo retorna 1. Si es negativo -1. Si es 0 retorna 0.

5.2. Funciones de cadena de caracteres

Oracle

- CHR(n) Carácter cuyo valor codificado es n. `SELECT CHR(81) FROM DUAL;` -- Resultado: Q
- ASCII(n) Valor ascii. `SELECT ASCII('Q') FROM DUAL;` -- Resultado 81
- CONCAT(cad1, cad2) Concatena
- LOWER(cad) Pasa a minúscula
- UPPER(cad) Pasa a mayúscula
- INITCAP(cad) Devuelve con primer carácter en mayúscula
- LPAD(cad1, n, cad2) Devuelve cad1 con longitud n, ajustada a la derecha, rellenando por la izquierda con cad2. `*****M`
- RPAD(cad1, n, cad2) Devuelve cad1 con longitud n, ajustada a la derecha, rellenando por la derecha con cad1. `M****`
- REPLACE(cad, ant, nue) Reemplaza en la cadena, la ant por la nue
- SUBSTR(cad, m, n) Divide la cadena cad compuesta por n caracteres a partir de la posición m
- LENGTH(cad): Longitud de la cadena. Ojo. Van entre comillas simples!!!!!!
- TRIM(cad): Suprime espacios en blanco a izquierda y a derecha y dobles del interior
- LTRIM(cad): Solo a izquierda
- RTRIM(cad): Solo a derecha
- INSTR(cad, cadBuscada, [posInicial, nAparicion]): Posición en la que se encuentra la cadena buscada en la cadena inicial cad. Se puede empezar a buscar en una posición concreta e indicar un número de aparición. Si no encuentra nada devuelve 0.

MySQL

- ASCII(n)
- CONCAT(cad1, cad2)
- LOWER(cad)
- UPPER(cad)
- INSTR(cadena, subcadena): Posición de la primera ocurrencia
- RPAD(cad1, n, cad2)
- REPLACE(cad, ant, nue)
- SUBSTR(cad, m, n)
- LENGTH(cad)
- TRIM(cad)
- LTRIM(cad)
- RTRIM(cad)

5.3. Funciones de manejo de fechas

En Oracle los dos datos para manejar fechas son `DATE` y `TIMESTAMP`

Pueden realizarse operaciones como suma o diferencia.

- SYSDATE. Fecha y hora actual
- SYSTIMESTAMP. Fecha y hora actual en formato TIMESTAMP
- ADD_MONTHS(fecha, n)
- MONTHS_BETWEEN(fecha1, fecha2)
- LAST_DAY(fecha) Último día del mes al que pertenece la fecha (DATE)
- NEXT_DAY(fecha, d) Día que corresponde al añadir a la fecha el día d.
`NEXT_DAY('29/04/2024','MARTES')`
- EXTRACT(valor FROM fecha) Extrae valor concreto: Ejemplo `SELECT EXTRACT(MONTH FROM SYSDATE) FROM DUAL`
Con + y - se pueden sumar y restar días, dando la fecha correspondiente.

En MySQL también hay DATE y TIMESTAMP (mismo formato que DATETIME)

Entre sus funciones más comunes:

- CURDATE() o NOW(). Fecha actual y fecha y hora actual, respectivamente
 - CURRENT_TIMESTAMP() Fecha y hora actual en formato DATETIME
 - ADDDATE('2024-11-01',4) añade a la fecha los días indicados
 - DATETIME('fecha1','fecha2') Días de diferencia entre fechas
 - DATE_FORMAT (fecha,formato) Fecha formateada según formato especificado '%d/%m/%Y' (19/04/2024)
 - DAY(fecha) Día del mes de una fecha
 - MONTH(fecha) Mes de una fecha
 - YEAR(fecha) Año de una fecha
 - DAYNAME(fecha) Nombre del día de una fecha
 - MONTHNAME(fecha) Nombre del mes de una fecha
- Las funciones pueden emplearse enviando como argumento el nombre de un campo o columna de tipo fecha.

5.4. Funciones de conversión

Oracle y MySQL convierte automáticamente datos de forma que el resultado de la expresión tenga sentido (texto -> número y al revés; texto -> fecha y viceversa), pero hay ocasiones en las que se querrá realizar esas conversiones de modo explícito.

Oracle

- **TO_NUMBER(cad, formato)**. Convierte textos en números. Se usa para dar un formato concreto.

Formatos para números y su significado.

Símbolo	Significado
9	Posiciones numéricas. Si el número que se quiere visualizar contiene menos dígitos de los que se especifican en el formato, se rellena con blancos.
0	Visualiza ceros por la izquierda hasta completar la longitud del formato especificado.
\$	Antepone el signo de dólar al número.
L	Coloca en la posición donde se incluya, el símbolo de la moneda local (se puede configurar en la base de datos mediante el parámetro NSL_CURRENCY)
S	Aparecerá el símbolo del signo.
D	Posición del símbolo decimal, que en español es la coma.
G	Posición del separador de grupo, que en español es el punto.

- **TO_CHAR(d, formato)**. Convierte número o fecha d a cadena de caracteres. Se usa para fechas ya que de número a texto se hace de forma implícita.
- **TO_DATE(cad, formato)**. Convierte textos a fechas, indicando el formato.

Formatos para fechas y su significado.

Símbolo	Significado
YY	Año en formato de dos cifras
YYYY	Año en formato de cuatro cifras
MM	Mes en formato de dos cifras
MON	Las tres primeras letras del mes
MONTH	Nombre completo del mes
DY	Día de la semana en tres letras
DAY	Día completo de la semana
DD	Día en formato de dos cifras
D	Día de la semana del 1 al 7
Q	Semestre
WW	Semana del año
AM	Indicador a.m.
PM	Indicador p.m.
HH12	Hora de 1 a 12
HH24	Hora de 0 a 23
MI	Minutos de 0 a 59
SS	Segundos dentro del minuto
SSSS	Segundos dentro desde las 0 horas

MySQL

`CONVERT(expresión, tipo_dato)` Devuelve la expresión convertida al tipo de dato suministrado. Ej.:
`SELECT CONVERT(21, DECIM(6,2));` --Devuelve 21.00
`CAST(expresión AS tipo_dato)`. Análoga a la anterior. `SELECT CAST(21 AS DECIMAL(6,2));` --Devuelve 21.00.

5.5. Otras funciones: NVL y DECODE - IFNULL

Las **funciones con nulos** permiten manejar los campos NULL

Oracle

`NVL(valor, expr1)`: Si el valor es nulo, devuelve expr1 (Debe ser del mismo tipo de valor)

`DECODE(expr1, cond1, valor1, [expr2, valor2, cond2], default)` Se evalúa la expr1, si se cumple devuelve el valor1. Si no, evalúa la siguiente y así hasta que una de las condiciones se cumpla. Si ninguna se cumple se devuelve el default.

MySQL

IFNULL(expr1, expr2): Devuelve expr1 si es distinta de null, en caso contrario devuelve expr2.
IF(expr1, expr2, exp3) Devuelve expr2 o expr3 en función de si expr1 es verdadera o falsa.

6. Consultas de resumen

Las funciones de agrupamiento o agregado toman un grupo de datos y producen un único dato que resume el grupo. Todas tienen una estructura **FUNCION [ALL|DISTINCT] EXPRESIÓN** y debe tenerse en cuenta que:

- ALL indica que deben tomarse todos los valores de la columna (valor por defecto)
- DISTINCT indica que se considerarán todas las repeticiones del mismo valor como uno solo (solo considera valores distintos).
- En la expresión nunca puede aparecer una función de agregado ni una subconsulta
- Todas las funciones se aplican a las filas del origen de datos una vez ejecutada la cláusula WHERE (si la tuviéramos)
- Todas las funciones (salvo COUNT) ignoran los valores NULL
- Podemos encontrar una función de agrupamiento en cualquier lugar en el que pueda aparecer el nombre de una columna. Puede formar parte de una expresión pero no se pueden anidar funciones.
- No se pueden mezclar funciones de columnas con nombres de columna ordinarios, salvo ciertas excepciones.

¡Recuerda que se puede poner DISTINCT si quieres que solo coja valores diferentes!

6.1. Funciones de agregado: SUM y COUNT

Función SUM(expresión): Suma de los valores. Solo para números.

```
SELECT SUM(precio) FROM CONFERENCIA;
SELECT SUM(DISTINCT precio) FROM CONFERENCIA;
```

Función COUNT(expresión): Cuenta elementos de un campo. Cuenta el total de filas pero no cuenta los registros con NULL, **salvo que se tenga en la expresión el comodín (*)**. Puede contar cualquier tipo de dato incluido texto.

```
SELECT COUNT(nombre) FROM PONENTE;
SELECT COUNT(*) FROM PONENTE;
SELECT (DISTINCT nombre) FROM PONENTE;
```

6.2. Funciones de agregado: MIN y MAX

Función MIN(expresión): Valor mínimo de la expresión sin considerar nulos. Se puede incluir nombre de un campo, constante, función (no otras funciones agregadas)

Función MAX(expresión): Valor máximo de la expresión sin considerar NULL. Idem a MIN

6.3. Funciones de agregado: AVG, VAR, STDEV, STDEVP

Función AVG: Devuelve el promedio, omitiendo los NULL. Se aplica a campos de número, el tipo de dato del resultado varía según necesidades del sistema para representar el valor.

```
SELECT AVG(gratificacion) AS MEDIA_GRATIFICACIONES FROM PARTICIPAR;
SELECT AVG(nhoras) FROM trabaja;
```

Función VAR: Devuelve la varianza estadística

Función STDEV: Devuelve la desviación típica de una muestra

Función STDEVP: Devuelve la desviación típica de una población

Resumidamente las funciones de agregación: Toman un grupo de datos de una columna, producen un único dato que resume el grupo, usan una función de agregado en una consulta que la convierte en consulta de resumen.

7. Agrupamiento de registros **GROUP BY**

Para agrupaciones según un determinado campo se usa **GROUP BY**

- En **GROUP BY** se colocan las columnas por las que se va agrupar. En **HAVING** la condición que han de cumplir los grupos para que se realice la consulta.
- Es importante el orden: WHERE; GROUP BY; HAVING; ORDER BY
- Las columnas que aparecen en el SELECT y que no aparecen en GROUP BY deben tener una función de agrupamiento o producirá error.
- Otra opción es poner en la cláusula GROUP BY las mismas columnas que aparecen en el SELECT

```
SELECT cddep, count(cddep) FROM empleado GROUP BY cddep;  
SELECT cddep, COUNT(cddep) FROM empleado WHERE salario > 1500 GROUP BY cddep HAVING  
COUNT(cddep) >2;  
SELECT empresa, COUNT(codigo) AS NUM_ASISTENTES FROM ASISTENTE WHERE empresa IS NOT NULL  
GROUP BY empresa;  
SELECT <code>empresa, COUNT(codigo) AS NUM_ASISTENTES FROM ASISTENTE WHERE empresa IS NOT  
NULL GROUP BY empresa HAVING empresa = 'BigSoft' OR empresa= 'ProgConsulting';
```

8. Consultas multitabla

A la hora de coger datos de varias tablas en un único SELECT se podrán hacer operaciones de composición interna y de composición externa.

MySQL

En SQL 1999 se especifica sintaxis para consultar tablas que MySQL incorpora para separar las condiciones de asociación respecto a la selección de registros:

```
SELECT tabla1.columna1, tabla1.columna2, ..., tabla2.columna1, tabla2.columna2, ...  
FROM tabla1  
    [JOIN tabla2 USING (columna) |  
     [JOIN tabla2 ON (tabla1.columna=tabla2.columna)] |  
     [LEFT | RIGTH OUTER JOIN tabla2 ON (tabla1.columna=tabla2.columna)] ]
```

Oracle

```
SELECT tabla1.columna1, tabla1.columna2, ..., tabla2.columna1, tabla2.columna2, ...  
FROM tabla1  
    [CROSS JOIN tabla2] |  
    [NATURAL JOIN tabla2] |  
    [JOIN tabla2 USING (columna) |  
     [JOIN tabla2 ON (tabla1.columna=tabla2.columna)] |  
     [LEFT | RIGTH | FULL OUTER JOIN tabla2 ON (tabla1.columna=tabla2.columna)] ]
```

8.1. Composiciones internas

Combinar dos o más tablas sin ninguna restricción da lugar al producto cartesiano (todas las combinaciones de todas las filas de esas dos tablas).

Se indica poniendo en la cláusula FROM las tablas que se quieren poner separadas por comas. Se obtienen todas las posibles combinaciones de filas. A más tablas, mayor es el resultado final (operación costosa).

Esta operación no es la más utilizada porque lo normal es que se quiera discriminar de alguna forma (asociar tablas - join)

Para hacer composición interna se parte del producto cartesiano y se eliminan las filas que no cumplen la condición de composición. Lo importante es emparejar campos que tengan valores iguales.

- Pueden combinarse tantas tablas como se quiera
- El criterio de combinación puede estar formado por más de una pareja de columnas
- En la SELECT pueden citarse columnas de ambas tablas condicionen o no la combinación
- Si las columnas se llaman igual, debe identificarse especificando la tabla de procedencia o un alias

Las columnas que aparecen en el WHERE se llaman **columnas de emparejamiento** porque emparejan las filas de las dos tablas. No tienen por qué estar en la lista de selección. Emparejamos tablas relacionadas entre sí y una de las columnas de emparejamiento será la clave principal en su tabla.

También puede combinarse una tabla consigo misma pero poniendo de forma obligatoria un alias a uno de los valores a repetir.

```
SELECT e.cdempl, e.nombre, e.fecha_ingreso, d.nombre, d.ciudad
FROM empleado e, departamento d
WHERE e.cddesp= d.cddesp;

SELECT e.cdempl, e.nombre, je.nombre
FROM empleado e, empleado je
WHERE e.cdjefe = je.cdempl;

SELECT referencia, tema, nombre, capacidad FROM CONFERENCIA, SALA
WHERE CONFERENCIA.sala= SALA.Nombre;

SELECT DISTINCT A.codigo, A.nombre FROM ASISTENTE A,CONFERENCIA C, ASISTIR AT, WHERE
A.codigo= AT.codasistente AND C.referencia=AT.refconferencia AND C.Tema LIKE '%DATOS%';
```

8.2. Composiciones externas

A veces interesará seleccionar algunas filas de una tabla aunque estas no tengan correspondencia con las filas de otra tabla.

En MySQL

Composición externa LEFT OUTER JOIN o RIGHT OUTER JOIN

```
SELECT tabla1.columna1, tabla1.columna2, ..., tabla2.columna1, tabla2.columna2, ...
FROM tabla1
[LEFT | RIGTH OUTER JOIN tabla2 ON (tabla1.columna=tabla2.columna)]
```

Obtener un listado con los códigos y nombres de los distintos departamentos y el código, nombre y salario de sus empleados. Ten en cuenta que deben aparecer todos los departamentos aunque no tengan asignados empleados.

```
SELECT d.cddesp, d.nombre, e.cdempl, e.nombre, e.salario, e.cddesp
FROM departamento d LEFT OUTER JOIN empleado e ON e.cddesp=d.cddesp;
```

En Oracle

Se añade un signo entre paréntesis (+) en la igualdad entre campos.

El carácter (+) va detrás del nombre de la tabla en la que se desean aceptar valores nulos

Obtener un listado con los nombres de los distintos departamentos y sus jefes con sus datos personales. Ten en cuenta que deben aparecer todos los departamentos aunque no tengan asignado ningún jefe.

```
SELECT D.NOMBRE_DPTO, D.JEFE, E.NOMBRE, E.APELLIDO1, E.APELLIDO2
FROM DEPARTAMENTOS D, EMPLEADOS E
WHERE D.JEFE = E.DNI(+);
```

8.3. Composiciones en la versión SQL99

SQL99 tiene mejoras en la sintaxis para composiciones en consulta.

- **CROSS JOIN:** Producto cartesiano de las filas de ambas tablas. No es necesario WHERE
- **NATURAL JOIN:** Detecta las claves de unión automáticamente basándose en el nombre. Funciona incluso si no están definidas claves primarias o ajena
- **JOIN USING:** Establecer relaciones indicando qué campos se quieren relacionar
- **JOIN ON:** Unir tablas en la que los nombres de columna no coinciden en ambas o se necesitan asociaciones más complicadas.
- **OUTER JOIN:** Eliminar el uso del signo (+) en composiciones externas usando OUTER JOIN (Oracle)
- **LEFT OUTER JOIN:** Composición externa izquierda. Todas las filas de la izquierda se devuelven aunque no haya columna en tablas combinadas.
- **RIGTH OUTER JOIN:** Composición externa derecha. Todas las filas de la derecha se devuelven aunque no haya columna en tablas combinadas.
- FULL OUTER JOIN:** Composición externa se devuelven todas las filas de los campos no relacionadas en ambas tablas (Oracle)

```
-- MySQL
SELECT e.cdempl, e.nombre, e.fecha_ingreso, d.nombre, d.ciudad
FROM empleado e
INNER JOIN departamento d ON e.cddesp= d.cddesp;

SELECT e.cdempl, e.nombre, je.nombre
FROM empleado e
INNER JOIN empleado je ON e.cdjefe = je.cdempl;

-- Oracle
SELECT referencia, tema, nombre, capacidad FROM CONFERENCIA JOIN SALA ON <
CONFERENCIA.sala= SALA.nombre;

SELECT D.NOMBRE_DPTO, D.JEFE, E.NOMBRE, E.APELLIDO1, E.APELLIDO2
FROM DEPARTAMENTOS D LEFT OUTER JOIN EMPLEADOS E ON ( D.JEFE = E.DNI);
```

Recomendación: Usar sintaxis de SQL99 usando JOIN tanto en composiciones internas como externas.

9. Otras consultas multitableas: Unión, intersección y diferencia de consultas

- **UNION:** Combina las filas del prime SELECT y del segundo SELECT desapareciendo duplicados.

```
SELECT NombreCia, Ciudad FROM PROVEEDORES WHERE Pais = 'Alemania'
UNION
SELECT NombreCia, Ciudad FROM CLIENTES WHERE Pais = 'Alemania';
```

- **INTERSECT:** Devuelve las que aparecen en ambos SELECT desapareciendo duplicados

```
SELECT nombre, domicilio FROM ingles INTERSECT
SELECT nombre, domicilio FROM frances INTERSECT
SELECT nombre, domicilio FROM portugues;
```

- **MINUS:** Devuelve filas que están en el prime SELECT pero no en el segundo SELECT. Las filas duplicadas del primer SELECT desaparecen.

```
SELECT nombre, domicilio FROM INGLES
MINUS
SELECT nombre,domicilio FROM PORTUGUES;
```

Es importante que se use en los dos SELECT el mismo número y tipo de columnas y en el mismo orden

10. Subconsultas

La subconsulta puede ir dentro de las cláusulas WHERE, HAVING o FROM.

El tipo de datos que devuelve la subconsulta y la columna con la que se compara.

Si se quiere que la subconsulta devuelva más de un valor y comparar el campo que se tiene condichos valores deben usarse instrucciones especiales entre el operador y la consulta:

- ANY: Compara con cualquier fila de la columna. Instrucción válida si hay un registro que permite que la comparación sea cierta.
- ALL: Compara con todas. Instrucción válida si es cierta en todas
- IN. Comprueba si el valor se encuentra en el resultado de la subconsulta.
- NOT IN: Compara si no se encuentra.

Obtener el asistente más joven.

```
SELECT nombre  
FROM asistente  
WHERE fechanac <= ALL (SELECT fechanac FROM asistente);
```

Obtener nombre, descripción y precio que superen o igualen el precio medio de las rutas

```
SELECT nombre, descripción, precio  
FROM ruta  
WHERE precio>= (SELECT AVG(precio) FROM ruta);
```

dni y nombre de empleados que no han ido a un restaurante

```
SELECT DNI, nombre  
FROM empleado  
WHERE DNI NOT IN (SELECT empleado FROM restaurante);
```

11. Preparación de Oracle y MySQL

Oracle

Se crea un espacio de trabajo (workspace) con el usuario y la contraseña. Este usuario será el administrador del espacio de trabajo y tendrá permisos para crear tablas.

Con la línea de comandos de SQL se puede ejecutar el archivo descargado
Inicio -> Todos los programas -> Oracle Database 11g Express Edition -> Run SQL Command Line.

```
connect TAREA/admin  
@Ruta_del_archivo/archivo.sql
```

Se crean las tablas y se insertan los datos.

Después nos podemos desconectar con `disconnect`

MySQL

Con el cliente gráfico de Workbench podemos iniciar sesión o conectarnos al servidor de MySQL. Puede cargarse el script. Pulsar en File / Run SQL o sobre el rayo amarillo y se ejecutará el script. Se actualiza el cliente pulsando la flecha de recargar y se verá la nueva base de datos. O escribir `SHOW DATABASES`

UD 05 - TRATAMIENTO DE DATOS

1. Introducción

Para el tratamiento de datos pueden usarse instrucciones SQL (flexibilidad, rapidez, mayor detalle) o herramientas gráficas que permiten hacerlo de forma más visual pero con menos detalle.

2. Edición de información mediante herramientas gráficas

Hay herramientas gráficas y herramientas en modo texto (terminal, consola, línea de comandos: para esto usaremos SQL).

Oracle

Oracle Database Express con Application Express (Base de datos Oracle 11g Express Edition > Get started)

MySQL

MySQL Community Server dispone de Workbench para trabajar en modo gráfico. También está PhpMyAdmin y Navicat.

2.1. Inserción de registros

Oracle

Basicamente se pulsa en Datos e Insertar fila. Se escriben los datos. Se pulsa en crear. En caso de que se produzca un error podrá haber un error con un campo de tipo numérico.

MySQL

Lo mismo.. Símbolo +, Apply...

2.2. Modificación de registros

Seleccionar la fila, modificarla, Apply... Si se han modificado correctamente mostrará mensaje de éxito y de error en caso contrario.

2.3. Borrado de registros

Oracle: Elegir Datos y hacer click en Editar. Darle a suprimir. Darle a Aceptar para confirmar supresión. Puede ser que no deje eliminar por una restricción de integridad. Debe eliminarse el registro que hace referencia a él o modificarlo para que haga referencia a otro registro.

MySQL: En el menú contextual se pulsa "Select Rows", aparecen las filas o registros con botones de insertar, modificar, eliminar. Se sitúa el cursor en la fila que se quiere eliminar. Se selecciona el botón con (-) que elimina una fila. Se hace click en Apply para aplicar los cambios. Sale mensaje de éxito o de error.

Antes de aplicar o confirmar la operación MySQL muestra el SQL correspondiente a la sentencia que se va a ejecutar.

3. Edición de la información mediante sentencias SQL

Oracle: Apliquémoslas desde Application Express de Oracle (entorno web) usando el botón SQL Worksheet y pulsando SQL commands.

MySQL: Desde la ventana SQL de workbench o desde la línea de comandos o cliente en modo texto.

MySQL Workbench tiene activa la opción SQL_SAFE_UPDATES que no permite ejecutar sentencias de actualización y borrado. Para ejecutar las sentencias deberá desactivarse esta opción en EDIT, PREFERENCES y en SQL Editor desmarcar Safe Updates.

3.1. Inserción de registros

La sentencia INSERT se puede hacer en dos formas básicas:

- Añadir una fila indicando los valores que deben tomar las columnas
- Extraer las filas de una tabla ya existente y añadirlas a otra tabla (Consultas SELECT con INSERT)

```
INSERT INTO nombre_tabla (lista_campos) VALUES (lista_valores);
```

- Puede hacerse el INSERT con la lista de campos y la lista de valores
- Si se hace un INSERT en el que no se especifiquen los valores de todos los campos se obtiene NULL en los campos que no se han indicado
- Es posible omitir la lista de campos si se indican todos los valores de cada campo y en el orden en el que se encuentra definidos en la tabla.

```
INSERT INTO ASISTENTE (codigo, nombre, apellido1, apellido2, sexo, fechaNac, empresa) VALUES ('AS0015', 'Julia', 'Hernández', 'Sáez', 'M', '11/10/1992', 'BK Programación');
```

```
INSERT INTO departamento VALUES ('10', 'I+D', 'Sevilla');
```

```
INSERT INTO departamento (cddep, nombre) VALUES ('11', 'Ventas');
```

- Si la lista de campos no se corresponde con la lista de valores se obtiene un error en la ejecución. (Por ej.: ORA-00913: demasiados valores)

Insert extendido

Consiste en insertar varias filas con una única sentencia INSERT.

```
INSERT INTO departamento VALUES ('20', 'DEPAR20', 'ALMERÍA'), ('21', 'DEPART21', 'ALMERIA'), ('22', 'DEPART22', 'GRANADA');
```

Para los casos en los que sean valores AUTONUMÉRICOS se puede:

- Omitir el nombre de la columna autonomérica
- Poner un 0 y el sistema calculará automáticamente el valor

```
INSERT INTO alumnos (nombre, apellidos, idcurso)
VALUES('Daniel', 'Ro', 2);
```

```
INSERT INTO alumnos (idalumnos, nombre, apellidos, idcurso) VALUES (0, 'Manuel', 'Bellido', 2);
```

Insert con replace

Si el valor de la clave primaria a insertar coincide con el existente lo borra para insertar el nuevo

```
REPLACE INTO departamento VALUES ('09', 'InformáticaReplace', 'Almería')
```

3.2. Modificación de registros

La estructura de UPDATE es:

```
UPDATE nombre_tabla SET nombre_campo = valor [, nombre_campo = valor]...
[ WHERE condición ];
```

Se puede especificar los nombres de los campos que se deseen de la tabla indicada. A cada campo se le debe asociar el nuevo valor usando el signo =. Cada emparejamiento de campo=valor debe separarse usando comas.

La cláusula WHERE seguida de la condición es opcional para indicar que solo afecte a los registros que cumplan esa condición.

```
UPDATE CONFERENCIA SET precio = 20;
UPDATE CONFERENCIA SET precio = 20, tema = NULL;
UPDATE CONFERENCIA SET Credito = 30 WHERE turno = 'T';
```

Al terminar la sentencia UPDATE se muestra la cantidad de registros actualizados o el error si lo hubiese.

3.3. Borrado de registros

La estructura es la siguiente:

```
DELETE FROM nombre_tabla [ WHERE condición ];
```

Ejemplos:

```
DELETE FROM PONENTE WHERE especialidad = 'Programación';
DELETE FROM empleado WHERE cddep='22';
```

Igualmente devuelve el mensaje de error o el número de filas actualizadas.

5. Subconsultas y composiciones en órdenes de edición

5.1. Inserción de registros a partir de una consulta

```
INSERT [INTO] <tabla> [(<columna>,...)] SELECT ...
```

La inserción de los registros se puede hacer también usando una sentencia SELECT:

```
-- Este INSERT
INSERT INTO ASISTENTE (codigo, nombre, apellido1, apellido2, sexo, fechaNac, empresa) VALUES
('AS0015','Julia', 'Hernández', 'Sáez','M', '11/10/1992','BK Programación');

-- Puede ser equivalente a este
INSERT INTO (SELECT codigo, nombre, apellido1, apellido2, sexo, fechaNac, empresa FROM
ASISTENTE VALUES ('AS0015','Julia', 'Hernández', 'Sáez','M', '11/10/1992','BK Programación');
```

Es posible insertar en la tabla valores que se obtienen directamente del resultado de una consulta si tienen la misma estructura que la tabla asistente. **No es necesario usar la palabra VALUES**

```
INSERT INTO ASISTENTES_SUPLENTES SELECT * FROM ASISTENTE WHERE empresa = null;
```

Se puede hacer esto, por ejemplo. Fíjate que hacemos una consulta y estamos insertando el cdemp y el 'ECS' por defecto:

```
INSERT INTO trabaja (cdemp,cdpro) SELECT cdemp,'ECS' FROM empleado WHERE cddep = '03';
INSERT INTO trabaja (cdemp,cdpro) SELECT cdemp,'ECS' FROM empleado WHERE cdemp NOT IN
(SELECT DISTINCT cdemp FROM trabaja);
```

5.2. Modificación de registros a partir de una consulta

La actualización también puede hacerse con consultas para realizar modificaciones más complejas de los datos.

Modifica el precio de las conferencias que se celebren en salas cuya capacidad sea mayor que 200. El precio que se les asignará será el de la conferencia que tenga mayor precio.

```
UPDATE CONFERENCIA SET precio = (SELECT MAX(precio) FROM CONFERENCIA) WHERE sala IN <br />
(SELECT nombre FROM SALA WHERE capacidad >200);<br />
```

Si queremos asignar a los empleados sin departamento, al departamento o uno de los departamentos en cuyos proyectos se ha trabajado cero horas, la sentencia sería:

```
UPDATE empleado
SET cddep = (SELECT p.cddep FROM proyecto p
INNER JOIN trabaja t ON t.cdpro=p.cdpro
GROUP BY t.cdpro
HAVING SUM(t.nhoras)=0
ORDER By cddep<br />
LIMIT 1)
WHERE cddep IS NULL;
```

En caso de que para poder realizar la actualización de una tabla necesitemos consultar la misma tabla que se está actualizando, MySQL daría un error ya que la tabla que se está actualizando estaría bloqueada y no permitiría realizar consultas.

Se puede creando una **tabla temporal** de la tabla que se quiere usar.

```
-- Esta falla
UPDATE EMPLEADO SET SALARIO=SALARIO+100 WHERE SALARIO <= (SELECT AVG(SALARIO) FROM
EMPLEADO);

--- Así si
UPDATE EMPLEADO SET SALARIO=SALARIO+10 WHERE SALARIO <= (SELECT AVG(SALARIO) FROM (SELECT *
FROM EMPLEADO) AS EMP);
```

5.3. Supresión de registros a partir de una consulta

```
DELETE FROM (SELECT * FROM ASISTENTE WHERE Empresa='Bigsoft' AND sexo='H');

DELETE FROM (SELECT * FROM ASISTENTE WHERE Empresa='Bigsoft') WHERE sexo='H';

DELETE FROM proyecto WHERE cdpro NOT IN (SELECT cdpro FROM trabaja);

DELETE FROM empleado
WHERE cdemp IN (SELECT t.cdempl FROM trabaja t
INNER JOIN proyecto p ON p.cdpro=t.cdpro
WHERE p.cddep='02');
```

Igualmente, no puedes hacer subconsulta sobre la tabla en la que se van a borrar registros. Debe usarse **tabla temporal**.

```
DELETE FROM EMPLEADO WHERE SALARIO >
(SELECT AVG(SALARIO) FROM (SELECT * FROM EMPLEADO) AS EMP1);
```

5. Integridad referencial

Dos tablas están relacionadas si tienen en común uno o más campos llamados **clave ajena** o **clave foránea**

La restricción de integridad referencial requiere que haya coincidencia en todos los valores que tengan en común ambas tablas. Cada valor del campo que forma parte de la integridad referencial definida debe corresponder en la otra tabla con otro registro que contenga el mismo valor en el campo referenciado.

Ejemplo: PARTIDAS con FK que apunta a JUEGOS. No puede existir ninguna partida cuyo código de juego no se corresponda con los juegos de la tabla JUEGOS.

Hablaremos de:

- Clave ajena: Campo o conjunto de campos incluidos en la definición de la restricción que deben hacer referencia a una clave de referencia
- Clave de referencia: Clave única o primaria de la tabla a la que se hace referencia desde clave ajena.
- Tabla hija o dependiente: Tabla que incluye la clave ajena y depende de los valores de la clave de referencia
- Tabla padre o de referencia: Tabla que es referenciada por la clave ajena en la tabla hija. Determina las inserciones o actualizaciones que son permitidas en la tabla hija.

Motores de almacenamiento MySQL

- **Motor MyISAM:** No gestiona integridad referencial, ni transacciones. Los datos se almacenan en un formato independiente que permite pasar tablas entre distintas plataformas. Útiles para datos estables que solo tienen operaciones de lectura.
- **Motor InnoDB:** Gestiona integridad referencial con claves foráneas, soporta control de transacciones y bloqueo por registro. Para lograr esto se sacrifica un poco la velocidad de MySQL. Las tablas MyISAM se bloquean al realizar inserción o actualización en un registro. En las tablas InnoDB el bloqueo solo se hace en el registro que se esté modificando para que más usuarios usen de forma simultánea la base de datos.

5.1. Integridad en actuación y supresión de registros

- Si se modifica el valor de la clave ajena en la tabla hija debe ser un valor que haga referencia a la clave principal de uno de los registros de la tabla padre.
- No se puede modificar el valor de la clave principal de un registro si una clave ajena hace referencia a dicho registro.
- No pueden suprimirse registros que son referenciados con clave ajena desde otra tabla.

Cuando se hace borrado de registros en tabla de referencia se puede configurar la clave ajena para que se conserve la integridad referencial:

- **RESTRICT No permitir supresión:** Opción por defecto. No permite borrar en la tabla de referencia un registro que está siendo referenciado desde otra tabla. Produce error en la operación de borrado.
- **CASCADE Supresión en cascada:** Los registros de la tabla hija que hacían referencia se borran también.
- **SET NULL Definir nulo en suprimir.** Los valores de la clave ajena que hacen referencia a los registros que hayan sido borrados de la tabla de referencia son cambiados por NULL.
- **NO ACTION** No se hace nada en las otras

Modificaciones Actuará según lo indicando en `ON UPDATE (RESTRICT, NO ACTION, CASCADE, SET NULL)`

Eliminaciones Actuará según lo indicado en `ON DELETE (RESTRICT, NO ACTION, CASCADE, SET NULL)`

5.2. Supresión en cascada

Las opciones de cascada / nulo se pueden establecer en la creación de tablas. En la pestaña de restricciones (crear, borrar, activar, desactivar restricciones)

En SQL:

```
CONSTRAINT JUEGOS_CON FOREIGN KEY (Cod_Juego) REFERENCES JUEGO (Codigo) ON DELETE CASCADE
```

6. Transacciones

Una transacción es una unidad atómica (indivisible) de trabajo que contiene una o más sentencias SQL. A todas ellas se les aplica una operación COMMIT o ROLLBACK.

Mientras que no se haga COMMIT los resultados pueden deshacerse. **El efecto de una sentencia DML no es permanente hasta no hacer COMMIT sobre la transacción en BBDD**

Propiedades básicas de las transacciones (ACID):

- **Atomicidad** (Atomicity): O se hacen todas o no se hace ninguna
- **Consistencia** (Consistency): Parte un estado consistente hasta otro estado consistente
- **Aislamiento** (Isolation): No es visible por otras transacciones hasta que no finaliza
- **Durabilidad** (Durability): Los cambios por transacciones que vuelvan sus modificaciones son permanentes.

Las tablas que soportan transacciones son más seguras y fáciles de recuperar si se produce algún fallo. Aunque pueden aumentar el tiempo de proceso de instrucciones.

Las sentencias de control de transacciones gestionan las sentencias DML y las agrupan en transacciones. Se puede:

- Hacer permanentes los cambios `COMMIT`
- Deshacerlos `ROLLBACK` desde que fue iniciada o desde un punto de restauración (marcador que se puede establecer en el contexto de la transacción `ROLLBACK TO SAVEPOINT`). Ojo ROLLBACK finaliza la transacción, ROLLBACK TO SAVEPOINT, no.
- Establecer punto de restauración `SAVEPOINT`
- Indicar propiedades de la transacción `SET TRANSACTION`
- Especificar si una restricción se comprueba tras cada sentencia DML o cuando se realice el commit `SET CONSTRAINT`

6.1. Hacer cambios permanentes

La transacción comienza con la primera SQL ejecutable. Se puede

- Usar COMMIT
- Usar una sentencia DDL como CREATE, DROP, RENAME, ALTER (ejecuta un COMMIT antes y después de cada DDL)
- Cerrar adecuadamente las aplicaciones de gestión (vuelca la transacción)

Existe opción de "confirmación automática" (autocommit) en las aplicaciones gráficas.

Iniciar transacción con `START TRANSACTION` o `BEGIN`

Confirmar con `COMMIT` o deshacer con `ROLLBACK` (ante cancelación, excepción, fallo de sistema).

InnoDB siempre tiene actividad de usuario entre transacción. MySQL siempre comienza nueva conexión con la ejecución automática habilitada (`AUTOCOMMIT = 1`) Cada instrucción SQL se trata como transacción en sí misma y se confirma en cuanto la ejecución termina.

Puede verse el valor: `show variables like '%autocommit%';`

Se puede cambiar con `SET autocommit = OFF;`

Si la conexión tiene ejecución automática, igualmente el usuario puede llevar a cabo una transacción si lo hace con los comandos habituales:

```
START TRANSACTION;
UPDATE TRABAJA SET nhoras = nhoras + 50 WHERE cfpro='DAG';
COMMIT;
```

Instrucciones que implican COMMIT automático

DDL que definen o modifican objetos: CREATE, ALTER, DROP, RENAME, TRUNCATE

Tablas de la base de datos administrativa MySQL: GRANT, REVOKE

Control de transacciones y bloqueo de tablas: START TRANSACTION, BEGIN, LOCK, UNLOCK

Carga de datos: LOAD DATA

Administración de tablas: ANALIZE, CHECK, OPTIMIZE, REPAIR

6.2. Deshacer cambios

Siempre se recomienda finalizar explícitamente las transacciones.

Veamos un ejemplito en Oracle:

```
CREATE TABLE emp_name AS SELECT employee_id, last_name FROM employees;
CREATE UNIQUE INDEX empname_ix ON emp_name (employee_id);
CREATE TABLE emp_sal AS SELECT employee_id, salary FROM employees;
CREATE UNIQUE INDEX empsal_ix ON emp_sal (employee_id);
CREATE TABLE emp_job AS SELECT employee_id, job_id FROM employees;
CREATE UNIQUE INDEX empjobid_ix ON emp_job (employee_id);

DECLARE
    emp_id      NUMBER(6);
    emp_lastname VARCHAR2(25);
    emp_salary   NUMBER(8,2);
    emp_jobid    VARCHAR2(10);
BEGIN
    SELECT employee_id, last_name, salary, job_id INTO emp_id, emp_lastname,
        emp_salary, emp_jobid FROM employees WHERE employee_id = 120;
    INSERT INTO emp_name VALUES (emp_id, emp_lastname);
    INSERT INTO emp_sal VALUES (emp_id, emp_salary);
    INSERT INTO emp_job VALUES (emp_id, emp_jobid);
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Inserts have been rolled back');
END;
/
```

6.3. Deshacer cambios parcialmente

- SAVEPOINT id: Crea un SAVEPOINT.
- ROLLBACK TO SAVEPOINT id: ROLLBACK al SAVEPOINT
- RELEASE SAVEPOINT id: Elimina el SAVEPOINT creado

Cualquier operación COMMIT o ROLLBACK sin argumentos elimina todos los save points creados.

SAVEPOINT con ROLLBACK

```
CREATE TABLE emp_name AS SELECT employee_id, last_name, salary FROM employees;
CREATE UNIQUE INDEX empname_ix ON emp_name (employee_id);

DECLARE
    emp_id      employees.employee_id%TYPE;
    emp_lastname employees.last_name%TYPE;
    emp_salary   employees.salary%TYPE;
BEGIN
    SELECT employee_id, last_name, salary INTO emp_id, emp_lastname,
        emp_salary FROM employees WHERE employee_id = 120;
    UPDATE emp_name SET salary = salary * 1.1 WHERE employee_id = emp_id;
    DELETE FROM emp_name WHERE employee_id = 130;
    SAVEPOINT do_insert;
    INSERT INTO emp_name VALUES (emp_id, emp_lastname, emp_salary);
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        ROLLBACK TO do_insert;
        DBMS_OUTPUT.PUT_LINE('Insert has been rolled back');
END;
/
```

```
START TRANSACTION;
INSERT INTO proyectosx.departamento VALUES('88','depTrans1', 'ciudad1');
SAVEPOINT sp1;
INSERT INTO proyectosx.departamento VALUES ('90', 'depTrans2', 'ciudad2'), ('91',
'depTrans3', 'ciudad3');
ROLLBACK TO SAVEPOINT sp1;
```

7. Problemas asociados al acceso simultáneo a los datos

En las bases de datos multiusuario las sentencias contenidas en varias transacciones simultáneas pueden actualizar los datos simultáneamente. Estas deben generar resultados consistentes. Por tanto se debe asegurar:

- Conurrencia de datos: Los usuarios pueden acceder a los datos al mismo tiempo
- Consistencia de datos: Asegura que cada usuario tiene una vista consistente de los datos. Incluye los cambios visibles de las transacciones del mismo usuario y de otros.

Debe haber mecanismos (bloqueos) para prevenir modificación concurrente del mismo dato. Así se cumplen los requerimientos de:

- Consistencia: Los datos consultados por un usuario no son cambiados por otros hasta no finalizar la operación completa.
- Integridad: Los datos y sus estructuras deben reflejar todos los campos efectuados en orden correcto.

Los bloqueos se realizan de forma automática. No requieren la actuación del usuario en MySQL y Oracle.

7.1. Políticas de bloqueo

Bloquear una tabla o vista permite que nadie más pueda hacer uso de la misma de modo que tenemos la exclusividad de lectura y/o escritura sobre ella.

Los bloqueos afectan a la interacción de lectores (consultas) y escritores (modificaciones) del recurso.

- Un registro es bloqueado solo cuando es modificado por un escritor: Al actualizar la sentencia un registro, la transacción obtiene bloqueo solo para ese registro.
- Un escritor de un registro bloquea a otro escritor concurrente del mismo registro: Una transacción que modifica una fila impide que otra transacción modifique el mismo registro
- Un lector nunca bloquea a un escritor: La única excepción es `SELECT .. FOR UPDATE`
- Un escritor nunca bloquea a un lector. Mientras se modifica la BBDD proporciona al lector una vista del registro sin los cambios que se están realizando.

Existe:

- **Bloqueo pesimista:** Lo hace inmediatamente en cuanto el bloqueo se solicita. Garantiza que el registro es actualizado.
- **Bloqueo optimista:** El acceso al registro o tabla se cierra en el momento en el que los cambios a ese registro se actualizan en disco. Esto solo es apropiado cuando hay poca posibilidad de que alguien necesite acceder al registro mientras está bloqueado o no podemos estar seguro de que la actualización tenga éxito porque el intento de actualizar el registro puede dar error si otro usuario lo actualiza mientras.

Supongamos que un usuario está en proceso de modificación de un registro, y otro en ese mismo momento quiere leer ese mismo registro. ¿Qué tipo de bloqueo debes establecer para que el segundo usuario obtenga los datos con los cambios que está efectuando el primero? (Pesimista)

7.2. Bloqueos compartidos y exclusivos

Un registro solo puede obtener un bloqueo exclusivo pero puede tener varios bloqueos compartidos.

Hay dos tipos de bloqueos:

- **Bloqueo exclusivo:** Previene que sea compartido el recurso asociado. La transacción obtiene bloqueo exclusivo al modificar los datos. La primera transacción que bloquea un recurso es la que puede modificar hasta que el bloqueo exclusivo es liberado.
- **Bloqueo compartido:** Permite que sea compartido el recurso asociado, según la operación. Varios usuarios que lean datos pueden compartir los datos, realizando bloqueos compartidos

para evitar el acceso concurrente de un escritor que desea un bloqueo exclusivo. Varias transacciones obtiene bloqueos compartidos del mismo recurso.

Ejemplo: Una transacción que usa `SELECT ... FOR UPDATE` obtiene bloqueo exclusivo del registro (permite a otras sesiones modificar registros que no sean el bloqueado) y bloqueo compartido de la tabla (evita que otras sesiones modifiquen la estructura de la tabla).

En MySQL la política de bloqueos:

1. Ordena internamente las tablas a bloquear
2. Si una tabla debe bloquearse en lectura y escritura, hace la solicitud de bloqueo de escritura en primer lugar
3. Se bloquea cada tabla hasta que la sesión obtiene todos sus bloqueos (evita el deadlock o bloqueo mutuo) (ningún proceso podría obtener sus bloqueos)

En InnoDB los bloqueos se obtienen a nivel de fila. Así varios usuarios pueden bloquear varias filas simultáneamente.

Lo dicho también aplica para InnoDB:

- Bloqueo compartido: Permite lectura de filas. Varias transacciones adquieren bloqueos sobre las filas pero ninguna transacción puede modificarlas hasta que no se liberan
- Bloqueo exclusivo: Permite a la transacción bloquear filas para actualización o borrado. Las transacciones deben esperar a que se libere el bloqueo de las filas afectadas.

También hay bloqueo de múltiple granularidad: Una transacción indica que bloquea algunas filas de una tabla para lectura o escritura (intención de bloqueo) así MySQL puede gestionar conflictos y evitar deadlocks porque varias transacciones compartirán la reserva de la tabla ya que el bloqueo se da fila a fila en el momento de la modificación. Las dos filas reservan la misma tabla y solo cuando una de ellas modifique la tabla esta quedará bloqueada.

7.3. Bloqueos automáticos

Oracle y MySQL bloquean automáticamente recursos usados por transacción para prevenir que otras transacciones requieran acciones que impliquen usos exclusivos sobre el recurso. Se divide en:

- **Bloqueos DML:** Protegen los datos, garantizando la integridad. (sobre un registro o sobre la tabla completa) Ej.: Evitar que dos clientes compren en mismo artículo.
- **Bloqueos DDL:** Protegen la definición del esquema mientras operación DDL actúa sobre él. Los usuarios no pueden solicitar este bloqueo
- **Bloqueo del sistema:** Para proteger BBDD interna y estructuras de memoria

7.4. Bloqueos manuales

Pueden hacerse bloqueos manuales útiles para aplicaciones que requieren consistencia a nivel de transacciones o lecturas repetitivas o para aplicaciones que requieren que las transacciones tengan acceso exclusivo a un recurso sin esperar a que otras transacciones finalicen.

Oracle

La cancelación de bloqueos automáticos se puede hacer a nivel de sesión o de transacción.

Se puede a nivel de sesión establecer el nivel requerido de aislamiento de la transacción con `ALTER SESSION`

A nivel de transacción se omite el bloqueo por defecto si las sentencias incluyen:

- `SET TRANSACTION ISOLATION LEVEL`
- `LOCK TABLE`
- `SELECT ... FOR UPDATE`

Los bloqueos finalizan una vez que la transacción finaliza.

MySQL

Los clientes pueden bloquear tablas para evitar que puedan modificar datos que se necesiten en la sesión con `LOCK`:

```
LOCK TABLES >nombre_tabla [[AS] alias] tipo_bloqueo [, nombre_tabla [[AS] alias]
tipo_bloqueo READ [LOCAL] | [LOW_PRIORITY] WRITE]
```

Para desbloquear: **UNLOCK TABLES**

Tipos de bloqueo

READ

La sesión o cliente puede leer pero ni él ni nadie puede escribir. Lo pueden adquirir varios a la vez. Permite que cualquiera pueda leer. LOCAL permite que haya inserciones concurrentes mientras dura el bloqueo.

WRITE

La sesión o cliente puede leer y escribir pero nadie más puede acceder a ella ni bloquear.

Los bloqueos de escritura tienen prioridad sobre los de lectura. El modificador LOW PRIORITY permite que los bloqueos de lectura se adquieran antes que los de escritura (De hecho escritura se adquirirá cuando no queden bloqueos lectura pendientes):

```
USE proyectosx;
LOCK TABLES departamento READ
SELECT * FROM departamento; //realiza la lectura y muestra los departamentos.
UPDATE departamento //da error, pues la tabla está bloqueada para lectura
SET ciudad="ciudad" WHERE nombre='depar20';
UNLOCK TABLES; //desbloquea la tabla
```

UD 06 - PROGRAMACIÓN DE BASES DE DATOS

1. Introducción

PL/SQL es un lenguaje procedimental estructurado en bloques que permite utilizar sentencias SQL para manipular datos y sentencias de control de flujo para procesar los datos. Combina la potencia de SQL para manipular datos, con la potencia de los lenguajes procedimentales para procesarlos.

Fue ideado por Oracle (podemos probarlo en la aplicación gráfica Oracle 11g o con SQL Developer), aunque todos los gestores de bases de datos utilizan un lenguaje procedimental parecido hoy día. (Procedimientos almacenados en MySQL o PL/pgSQL en PostgreSQL)

El motor PL/SQL acepta como entrada bloques PL/SQL o subprogramas -> ejecuta sentencias procedimentales -> envía sentencias SQL al servidor de bases de datos.

PL/SQL permite mejor rendimiento en entornos red cliente-servidor porque permite mandar bloques PL/SQL desde el cliente al servidor a través de la red, sin tener que mandar una a una las sentencias correspondientes.

<https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-SEQUENCE.html>

2. Conceptos básicos

2.1. Variables y constantes

Declaración de variable: Nombre de la variable, tipo de dato y opcionalmente una asignación (con el operador `:=` o con `DEFAULT`).

Declaración de constante: Igual que una variable, anteponiendo la palabra `constant` al tipo de dato. Puede forzarse que no sea nula indicando `NOT NULL` y, si no se le asigna valor lanzará error en tiempo de ejecución `VALUE_ERROR`

```
id SMALLINT;
hoy DATE := sysdate;
hoy DATE DEFAULT sysdate;
pi CONSTANT REAL := 3.1415;
id SMALLINT NOT NULL := 9999; -- ¿Ves? La tengo que inicializar
```

Conversión de tipos

- Conversión implícita. En tipos parecidos se puede hacer.
- Conversión explícita. Es la recomendada para evitar errores inesperados. Con funciones de conversión como `TO_CHAR`, `TO_DATE`, `TO_NUMBER`

Precedencia de operadores: La habitual del lenguaje matemático (Exponenciación, negación lógica; Identidad, negación; Multiplicación, división; Suma, resta, concatenación; Comparaciones; Conjunción lógica; Disyunción lógica)

2.2. Unidades léxicas

- Elementos más pequeños con sentido propio y que al combinarlos siguiendo las reglas de sintaxis dan lugar a sentencias válidas** sintácticamente.
- Es no sensible a mayúsculas (es equivalente, salvo en literales cadena o carácter)
- Cada unidad léxica puede separarse por espacios, saltos de líneas o tabuladores.

Las unidades léxicas pueden ser:

- **Delimitadores**

Delimitadores Simples.		Delimitadores Compuestos.	
Símbolo.	Significado.	Símbolo.	Significado.
+	Suma.	**	Exponenciación.
%	Indicador de atributo.	<>	Distinto.
.	Selector.	=	Distinto.
/	División.	<=	Menor o igual.
(Delimitador de lista.	>=	Mayor o igual.
)	Delimitador de lista.	..	Rango.
:	Variable host.		Concatenación.
,	Separador de elementos.	<<	Delimitador de etiquetas.
*	Producto.	>>	Delimitador de etiquetas.
"	Delimitador de identificador acotado.	--	Comentario de una línea.
=	Igual relacional.	/*	Comentario de varias líneas.
<	Menor.	*/	Comentario de varias líneas.
>	Mayor.	:=	Asignación.
@	Indicador de acceso remoto.	=>	Selector de nombre de parámetro.
;	Terminador de sentencias.		
-	Resta/negación.		

- **Identificadores**

Usados para nombres elementos. Es una letra seguida opcionalmente de letras, números \$ _ #

No pueden usarse palabras reservadas.

Nos permite definir identificadores acotados en los que se puede usar cualquier carácter (longitud máxima 30 caracteres).

Hay identificadores predefinidos que son palabras reservadas y no se pueden usar
Algunas palabras reservadas en PL/SQL no lo son para SQL (ej. type lo está. Habría que usarlo en PL/SQL como "type" si una columna se llama así)

- **Literales**

Notación decimal o exponencial en literales numéricos

Tipos carácter y cadena delimitados por comillas

Literales lógicos TRUE y FALSE

Literal NULL expresa una variable sin valor

- **Comentarios**

Igual que en SQL Una línea '--' o Varias /* */

2.3. Tipos de datos simples y subtipos

Numéricos

- **BINARY_INTEGER:** Subtipos NATURAL, NATURALN, POSITIVE, POSITIVEN, SIGNTYPE
- **NUMBER:** Almacena números racionales. Puede indicarse escala y precisión. Subtipos DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INTEGER, INT, NUMERIC, REAL, SMALLINT
- **PLS_INTEGER:** Mismo rango que BINARY_INTEGER pero con representación distinta por lo que sus operaciones aritméticas serán más eficientes.

Alfanuméricos

- **CHAR(n)**: Array de caracteres. Máximo 2000 bytes.
- **LONG**: Array de caracteres. Máximo 32760 bytes.
- **RAW**: Array de bytes. Máximo de 2000.
- **LONG RAW**: Array de bytes. Máximo 32760.
- **VARCHAR2**: Cadenas de longitud variable. Máximo 32760.

Grandes objetos

- **BFILE**: Puntero a fichero del sistema operativo.
- **BLOB**: Objeto binario 4 GB
- **CLOB**: Objeto carácter 2 GB.

Otros

- **BOOLEAN**: TRUE/FALSE
- **DATE**: Del 1 de enero de 4712ac al 31 de diciembre de 4712dc

2.4. Subtipos

Se pueden definir **subtipos de tipos de datos** para darles un nombre diferente que aumente la legibilidad de los programas. Se les podrán aplicar las mismas operaciones que a los tipos originales.

Con **%TYPE** puede indicarse el tipo de dato de una variable (columna) de la base de datos.
Con **%ROWTYPE** puede indicarse el tipo de un cursor o tabla de una base de datos.

```
SUBTYPE id_familia IS familias.identificador%TYPE;
SUBTYPE agente IS agente%ROWTYPE;

-- Si queremos restringir un subtipo se puede usar una variable para que nos lo permita
SUBTYPE apodo IS varchar2(20); -- No se puede D:

aux VARCHAR2(20);
SUBTYPE apodo IS aux%TYPE; -- Sí se puede :)
```

Los subtipos pueden intercambiarse con su tipo base o entre sí siempre que tengan el mismo tipo base o pertenezcan a la misma familia (si están relacionados de alguna manera y tienen similitudes en su comportamiento o estructura: Ej.: **literal** y **sentencia** como subtipos de **CHAR** y **VARCHAR2** que son tipos de datos de caracteres)

Ejemplo absurdo de los apuntes:

```
DECLARE
    SUBTYPE numero IS NUMBER;
    numero_tres_digitos NUMBER(3);
    mi_numero_de_la_suerte numero;
    SUBTYPE encontrado IS BOOLEAN;
    SUBTYPE resultado IS BOOLEAN;
    lo_he_encontrado encontrado;
    resultado_busqueda resultado;
    SUBTYPE literal IS CHAR;
    SUBTYPE sentencia IS VARCHAR2;
    literal_nulo literal;
    sentencia_vacia sentencia;
BEGIN
    ...
    numero_tres_digitos := mi_numero_de_la_suerte;
    ...
    lo_he_encontrado := resultado_busqueda;
    ..
    sentencia_vacia := literal_nulo;
```

3. Bloque PL/SQL: Estructuras de control y excepciones

El bloque PL/SQL tiene tres zonas:

- **Declaraciones:** `DECLARE` Definición de variables, constantes, cursores y excepciones
- **Proceso:** `BEGIN` Contiene sentencias ejecutables
- **Excepciones:** Manejo de errores en tiempo de ejecución

`BEGIN` y `END` son los obligatorios

```
[DECLARE]
    [declaraciones]
BEGIN
    [sentencias]
[EXCEPTION]
    [manejadores]
END;
```

```
DECLARE
    aux number := 10;
BEGIN
    DECLARE
        aux number := 5;
    BEGIN
        ...
        IF aux = 10 THEN      --evalúa a FALSE, no entraría
        ...
    END;
END;
```

3.1. Estructuras de control condicionales

Van entre `IF` y `END IF`

Sentencia IF-THEN

```
IF (b<>0) THEN
    c := a/b;
END IF;
```

Sentencia IF-THEN-ELSE

```
IF (b<>0) THEN
    c := a/b;
ELSE
    c := a+b;
END IF;
```

Sentencia IF-THEN-ELSIF

```
IF (b=0) THEN
    c := a/b;
ELSIF (b=1)
    c := a+b;
ELSE
    c := a+b+2;
END IF;
```

3.2. Estructuras de control iterativas

Van entre `LOOP` y `END LOOP`

Bucle infinito

```
LOOP
    sentencias
END LOOP;
```

While

```
WHILE condicion LOOP
    sentencias
END LOOP;
```

For

```
FOR contador IN [REVERSE] limiteinferior..limitesuperior LOOP
    sentencias
END LOOP;
```

Exit

```
LOOP
    sentencias
    IF condicion THEN
        EXIT
    END IF
END LOOP;
```

Exit when

```
LOOP
    sentencias
    EXIT WHEN encontrado;
END LOOP;
```

(MariaDB) REPEAT

```
REPEAT
    secuencia
UNTIL condicion
END REPEAT
```

3.3. Excepciones

Hay excepciones predefinidas internamente o excepciones definidas por el usuario (en la parte de las declaraciones de cualquier bloque) con: `EXCEPTION`;

Y luego deben ser arrojadas con: `RAISE`

Y capturadas en el bloque de `EXCEPTION`

```
DECLARE
    mi_excepcion EXCEPTION;
BEGIN
    ...
    IF loquesea THEN
        RAISE mi_exception
    ENDIF
    ...
EXCEPTION
    WHEN mi_exception THEN
        haz cosas
END;
```

Si se redefine una excepción que ya es global prevalece la definición local frente a la global (y la global no podrá ser capturada a menos que el bloque en el que estaba definida fuese un bloque nombrado y pueda definirse con `nombre_bloque.nombre_excepcion`). Las excepciones predefinidas están definidas globalmente.

Para poder continuar por la sentencia que la lanzó... puede encerrarse la sentencia dentro de un bloque y ahí capturar la excepción para seguir con la sentencia:

```
DECLARE
    declaraciones
BEGIN
    ...
    BEGIN
        cosas
    EXCEPTION
        WHEN laexcepcionquesea THEN
            cosas
    END
    ...
END;
```

Imagina que quieras intentar una transacción hasta que no te dé ningún error. En un bucle, se encapsula la transacción en un bloque y se capturan en él las posibles excepciones.

```
DECLARE
    id_fam NUMBER;
    nombre VARCHAR2(40);
    oficina NUMBER;
BEGIN
    ...
    LOOP
        BEGIN
            SAVEPOINT inicio;
            INSERT INTO familias VALUES
(id_fam, nombre, NULL, oficina);
            ...
            COMMIT;
            EXIT;
        EXCEPTION
            WHEN DUP_VAL_ON_INDEX THEN
                ROLLBACK TO inicio;
                id_fam := id_fam + 1;
        END;
    END LOOP;
    ...
END;
```

Excepciones predefinidas en Oracle.

Excepción.	SQLCODE	Lanzada cuando ...
ACCES_INTO_NULL	-6530	Intentamos asignar valor a atributos de objetos no inicializados.
COLECTION_IS_NULL	-6531	Intentamos asignar valor a elementos de colecciones no inicializadas, o acceder a métodos distintos de EXISTS.
CURSOR_ALREADY_OPEN	-6511	Intentamos abrir un cursor ya abierto.
DUP_VAL_ON_INDEX	-1	Índice único violado.
INVALID_CURSOR	-1001	Intentamos hacer una operación con un cursor que no está abierto.
INVALID_NUMBER	-1722	Conversión de cadena a número falla.
LOGIN_DENIED	-1403	El usuario y/o contraseña para conectarnos a Oracle no es válido.
NO_DATA_FOUND	+100	Una sentencia SELECT no devuelve valores, o intentamos acceder a un elemento borrado de una tabla anidada.
NOT_LOGGED_ON	-1012	No estamos conectados a Oracle.
PROGRAM_ERROR	-6501	Ha ocurrido un error interno en PL/SQL.
ROWTYPE_MISMATCH	-6504	Diferentes tipos en la asignación de 2 cursos.
STORAGE_ERROR	-6500	Memoria corrupta.
SUBSCRIPT_BEYOND_COUNT	-6533	El índice al que intentamos acceder en una colección sobrepasa su límite superior.
SUBSCRIPT_OUTSIDE_LIMIT	-6532	Intentamos acceder a un rango no válido dentro de una colección (-1 por ejemplo).
TIMEOUT_ON_RESOURCE	-51	Un timeout ocurre mientras Oracle espera por un recurso.
TOO_MANY_ROWS	-1422	Una sentencia SELECT...INTO... devuelve más de una fila.
VALUE_ERROR	-6502	Ocurre un error de conversión, aritmético, de truncado o de restricción de tamaño.
ZERO_DIVIDE	-1476	Intentamos dividir un número por 0.

Propagación de excepciones

```
DECLARE  
  b EXCEPTION;  
BEGIN
```

```
DECLARE  
  a EXCEPTION;  
BEGIN  
  IF x=0 THEN RAISE a;  
  ELSIF x=1 THEN RAISE b;  
  ELSE RAISE c;  
  END IF;  
EXCEPTION  
  WHEN a THEN ...  
END;
```

```
...  
EXCEPTION  
  WHEN b THEN ...  
END;
```

Propagación de las excepciones

a es capturada aquí ya que existe un manejador para ella

```

DECLARE
  b EXCEPTION;
BEGIN

```

Propagación de las excepciones

```

DECLARE
  a EXCEPTION;
BEGIN
  IF x=0 THEN RAISE a;
  ELSIF x=1 THEN RAISE b;
  ELSE RAISE c;
  END IF;
EXCEPTION
  WHEN a THEN ...
END;

```

b es capturada aquí ya que en el bloque anterior no había manejador para ella y en este sí.

```

...
EXCEPTION
  WHEN b THEN ...
END;

```

podemos utilizar una variable localizadora para saber qué sentencia ha sido la que ha lanzado la excepción (aunque de esta manera no podremos continuar por la siguiente sentencia).

```

DECLARE
  sentencia NUMBER := 0;
BEGIN
  ...
  SELECT * FROM agentes ...
  sentencia := 1;
  SELECT * FROM familias ...
  sentencia := 2;
  SELECT * FROM oficinas ...
  ...
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    IF sentencia = 0 THEN
      RAISE agente_no_encontrado;
    ELSIF sentencia = 1 THEN
      RAISE familia_no_encontrada;
    ELSIF sentencia = 2 THEN
      RAISE oficina_no_encontrada;
    END IF;
END;

```

4. Tipos de datos compuestos

4.1. Registros

Registro: Grupo de elementos relacionados almacenados en campos, cada uno con su propio nombre y tipo de dato.

Se declara con `TYPE nombre_registro IS RECORD (declaracion_campo,
declaracion_campo,...);`

Cualquier tipo de dato es válido para el registro salvo `REF CURSOR`. Para acceder a los campos se usa notación de punto.

```

TYPE direccion IS RECORD
(
calle          VARCHAR2(50),
numero        INTEGER(4) NOT NULL := 1
provincia     VARCHAR2(20),
pais          VARCHAR2(20) := 'España'
);
mi_direccion direccion;

mi_direccion.calle := 'Ramirez Arellano';
mi_direccion.numero := 15;

```

Un registro solo puede asignarse a otro cuando sean del mismo tipo (no basta que tengan mismo número de campos y emparejar).

No pueden compararse registros aunque sea del mismo tipo.

No puede comprobarse si el registro es nulo.

Se puede hacer SELECT de registros, pero no INSERT.

4.2. Colecciones: Arrays de longitud variable **VARRAY**

Colecciones de elementos con límite superior fijo. Al crearlos **indicamos su tamaño máximo y el array crecerá dinámicamente hasta alcanzar ese tamaño.** Su límite mínimo es 1 y su límite máximo es el tamaño máximo.

Se declara como: **TYPE nombre IS {VARRAY | VARYING} (tamaño_maximo) OF tipo_elementos [NOT_NULL]**

- Si el tipo de elemento es un registro, todos los campos deben ser de tipo escalar.
- Al definir el **VARRAY** este es nulo.
- Debe inicializarse con un constructor

```

TYPE familias_hijas IS VARRAY(100) OF familia;
familias_hijas1 familias_hijas := familias_hijas(familia(100, `Fam100`, 10, null)), ... ,
familia(105, 'Fam105', 10, null));

```

- Pueden usarse constructores vacíos **familias_hijas2 familias_hijas := familias_hijas();**
- Un array puede ser asignado a otro si ambos son del mismo tipo
- El VARRAY puede extenderse usando **EXTEND**. Sin parámetros se extiende en un elemento nulo el VARRAY. VARRAY(n) Con parámetros añade n elementos al VARRAY. VARRAY(n,i) añade n copias del i-ésimo elemento.
- **COUNT** dice el número de elementos del VARRAY
- **LIMIT** el número máximo del VARRAY
- **FIRST** siempre es 1
- **LAST** es COUNT
- **PRIOR** y **NEXT** devuelven antecesores y sucesores del elemento.
- Pueden saltar excepciones: **COLLECTION_IS_NULL**, **SUBSCRIPT_BEYOND_COUNT**, **SUBSCRIPT_OUTSIDE_LIMIT**, **VALUE_ERROR**

```

DECLARE
    TYPE tab_num IS VARRAY(10) OF NUMBER;
    mi_tab tab_num;
BEGIN
    mi_tab := tab_num();
    FOR i IN 1..10 LOOP
        mi_tab.EXTEND;
        mi_tab(i) := calcular_elemento(i);
    END LOOP;
    ...
END;

```

```

DECLARE
    TYPE numeros IS VARRAY(20) OF NUMBER;
    tabla_numeros numeros := numeros();
    num NUMBER;
BEGIN
    num := tabla_numeros.COUNT;      --num := 0
    FOR i IN 1..10 LOOP
        tabla_numeros.EXTEND;
        tabla_numeros(i) := i;
    END LOOP;
    num := tabla_numeros.COUNT; --num := 10
    num := tabla_numeros.LIMIT; --num := 20
    num := tabla_numeros.FIRST; --num := 1;
    num := tabla_numeros.LAST; --num := 10;
    ...
END;

```

```

DECLARE
    TYPE numeros IS VARRAY(20) OF INTEGER;
    v_numeros numeros := numeros( 10, 20, 30, 40 );
    v_enteros numeros;
BEGIN
    v_enteros(1) := 15; --lanzaría COLECTION_IS_NULL
    v_numeros(5) := 20; --lanzaría SUBSCRIPT_BEYOND_COUNT
    v_numeros(-1) := 5; --lanzaría SUBSCRIPT_OUTSIDE_LIMIT
    v_numeros('A') := 25; --lanzaría VALUE_ERROR
    ...
END;

```

4.3. Colecciones: Tablas anidadas

Son **colecciones de elementos sin límite superior fijo**. Aumentan dinámicamente su tamaño. Pueden borrarse elementos individuales.

Se declaran con `TYPE nombre IS TABLE OF tipo_elementos [NOT NULL]`

- Tienen las mismas restricciones que los VARRAY
- Automáticamente son nulas, deben inicializarse antes de usarse
- Puede usarse constructor nulo
- Para referenciar elementos se usa la misma sintaxis que los VARRAY
- Se usa `EXTEND` para extender la tabla.
- `LIMIT` no tiene sentido, devuelve `NULL`
- `EXISTS(n)` devuelve si existe o si no (si fue borrado...)
- `TRIM` borra elemento del final de la tabla `TRIM(n)` borra n elementos del final. Opera internamente. Si se encuentra con un elemento borrado con `DELETE` lo incluye para ser eliminado de la colección
- `DELETE(n)` borra el n-esimo elemento. `DELENTE(n,m)` borra del n al m. (Devolvería FALSE al consultar con `EXITS(n)`)

```
DECLARE
  TYPE numeros IS TABLE OF NUMBER;
  tabla_numeros numeros := numeros();
  num NUMBER;
BEGIN
  num := tabla_numeros.COUNT;  --num := 0
  FOR i IN 1..10 LOOP
    tabla_numeros.EXTEND;
    tabla_numeros(i) := i;
  END LOOP;
  num := tabla_numeros.COUNT;  --num := 10
  tabla_numeros.DELETE(10);
  num := tabla_numeros.LAST;  --num := 9
  num := tabla_numeros.FIRST;  --num := 1
  tabla_numeros.DELETE(1);
  num := tabla_numeros.FIRST;  --num := 2
  FOR i IN 1..4 LOOP
    tabla_numeros.DELETE(2*i);
  END LOOP;
  num := tabla_numeros.COUNT;  --num := 4
  num := tabla_numeros.LAST;  --num := 9
  ...
END;
```

```

DECLARE
  TYPE numeros IS TABLE OF NUMBER;
  tabla_num numeros := numeros();
  tabla1 numeros;
BEGIN
  tabla1(5) := 0;    --lanzaría COLECTION_IS_NULL
  tabla_num.EXTEND(5);
  tabla_num.DELETE(4);
  tabla_num(4) := 3;  --lanzaría NO_DATA_FOUND
  tabla_num(6) := 10; --lanzaría SUBSCRIPT_BEYOND_COUNT
  tabla_num(-1) := 0; --lanzaría SUBSCRIPT_OUTSIDE_LIMIT
  tabla_num('y') := 5;--lanzaría VALUE_ERROR
END;

```

5. Cursos

El **cursor** es una estructura que almacena el conjunto de filas devuelto por una consulta a la base de datos.

Oracle usa áreas de trabajo para ejecutar sentencias SQL y almacenar la información procesada.

Distinguimos entre implícitos y explícitos. PL/SQL declara implícitamente un cursor para todas las sentencias SQL de manipulación de datos, incluyendo consultas que devuelven una fila de resultados. Para consultas que devuelven más de una fila se debe declarar explícitamente un cursor para procesarlas individualmente.

Atributos del cursor

Permiten obtener información sobre las sentencias SQL recientemente ejecutadas. Pueden usarse en PL/SQL pero no en SQL. Pueden usarse en implícitos, aunque su sentido está en los explícitos.

- **%FOUND**: Cuando está abierto y, antes del primer **FETCH**, **%FOUND** devuelve NULL. Después del primer **FETCH** devuelve TRUE si el último **FETCH** devolvió una fila y FALSE en caso contrario. En implícitos, devuelve TRUE si un **INSERT**, **UPDATE**, **SELECT**, **DELETE** afectan o devuelven una o más filas. En caso contrario, devuelve FALSE.
- **%NOTFOUND**: Lo contrario a **%FOUND**
- **%ISOPEN**: Saber si está abierto. Cursos implícitos son cerrados por Oracle automáticamente.
- **%ROWCOUNT**: Cuando está abierto y, antes del primer **FETCH**, **%ROWCOUNT** es 0. Después de cada **FETCH**, **%ROWCOUNT** se incrementa e indica el número de filas procesadas. En implícitos, número de filas afectadas o devueltas por **INSERT**, **UPDATE**, **SELECT**, **DELETE**.

5.1. Cursos implícitos

Abierto implícitamente por Oracle para procesar sentencia SQL no asociada a un cursor declarado. No pueden usarse los comandos del cursor explícito. Sí pueden usarse los atributos del cursor.

```

DECLARE
  vNombre VARCHAR2(30);
  vFecha DATE;
  vEmpresa VARCHAR2(30);
BEGIN
  SELECT nombre, fechanac, empresa INTO vNombre, vFecha, vEmpresa
  FROM ASISTENTE WHERE CODIGO = 'AS0001';
  DBMS_OUTPUT.PUT_LINE(vNombre || ' - ' || vFecha || ' - ' || vEmpresa);
END;

-- Output: "Mario - 16/11/70 - BK Programación"

```

5.2. Cursores explícitos

Para recorrer una sentencia con varios resultados debemos introducirla en un cursor explícito que hayamos declarado.

El cursor se declara dándole un nombre y asociándolo con una consulta: `CURSOR nombre_cursor [(parametro...) RETURN tipo_devuelto] IS consulta_sql`

Ejemplos:

```
CURSOR cAgentes IS SELECT * FROM agentes;
CURSOR cFamilias RETURN familias%ROWTYPE IS SELECT * FROM familias WHERE...
```

- El cursor no puede tener restricción NOT NULL
- El cursor puede tomar parámetros, que pueden aparecer en la consulta como si fuesen constantes.
`CURSOR c1 (cat INTEGER DEFAULT 0) IS SELECT * FROM agentes WHERE categoria = cat;`
- `OPEN`: Al abrir el cursor se ejecuta consulta asociada y se identifica el conjunto resultado que son todas las filas que emparejan con ese criterio de búsqueda. `OPEN nombre_cursor`
- `FETCH`: Devuelve una fila del conjunto resultado. Tras cada `FETCH` avanza a la próxima fila.
`FETCH cFamilias INTO mi_id, mi_nom, mi_fam, mi_ofi;`
- Por cada valor de columna devuelto debe haber una variable que se corresponda después del `INTO`
- El cursor se procesa por medio de un bucle
- `CLOSE`: Al final del todo, el cursor debe cerrarse. Después podrá reabrirse. Si se intentan hacer operaciones con el cursor cerrado se lanza `INVALID_CURSOR`.

```
DECLARE

CURSOR cAsistentes IS
SELECT nombre, fechanac, empresa FROM ASISTENTE;

vNombre VARCHAR2(30);
vFecha DATE;
vEmpresa VARCHAR2(30);

BEGIN

OPEN cAsistentes; -- abro cursor
FETCH cAsistentes INTO vNombre, vFecha, vEmpresa; -- Cojo 1 y meto en variables

WHILE cAsistentes%FOUND LOOP -- Mientras haya registros
    DBMS_OUTPUT.PUT_LINE(vNombre || ' - ' || vFecha || ' - ' || vEmpresa);
    FETCH cAsistentes INTO vNombre, vFecha, vEmpresa; -- Cojo siguiente
END LOOP;

END;

/* Output:
Mario - 16/11/70 - BK Programación
María - 12/01/94 -
*/
```

Puedo crear si quiero una única variable que sea de tipo `cAsistentes` (así no tengo que definir una a una). (Bucles para cursores)

```
DECLARE

CURSOR cAsistentes IS
SELECT nombre, fechanac, empresa FROM ASISTENTE;

vMiReg cAsistentes%ROWTYPE; -- Registro que incluye las variables del cursor

BEGIN
```

```

OPEN cAsistentes; -- abro cursor
FETCH cAsistentes INTO vMiReg; -- Cojo 1 y meto en variables

WHILE cAsistentes%FOUND LOOP -- Mientras haya registros
    DBMS_OUTPUT.PUT_LINE(vMiReg.nombre || ' - ' || vMiReg.fechanac || ' - ' ||
vMiReg.empresa);
    FETCH cAsistentes INTO vMiReg; -- Cojo siguiente
END LOOP;

END;

/* Output:
Mario - 16/11/70 - BK Programación
María - 12/01/94 -
*/

```

```

BEGIN
    ...
    FOR cFamilias_rec IN cFamilias LOOP
        --Procesamos las filas accediendo a
        --cFamilias_rec.identificador, cFamilias_rec.nombre,
        --cFamilias_rec.familia, ...
    END LOOP;
    ...
END;

```

Evaluación de los atributos de un cursor explícito según las operaciones realizadas con él.

Operación realizada.	%FOUND	%NOTFOUND	%ISOPEN	%ROWCOUNT
Antes del OPEN	Excepción.	Excepción.	FALSE	Excepción.
Después del OPEN	NULL	NULL	TRUE	0
Antes del primer FETCH	NULL	NULL	TRUE	0
Después del primer FETCH	TRUE	FALSE	TRUE	1
Antes de los siguientes FETCH	TRUE	FALSE	TRUE	1
Después de los siguientes FETCH	TRUE	FALSE	TRUE	Depende datos.
Antes del último FETCH	TRUE	FALSE	TRUE	Depende datos.
Después del último FETCH	FALSE	TRUE	TRUE	Depende datos.
Antes del CLOSE	FALSE	TRUE	TRUE	Depende datos.
Después del CLOSE	Excepción.	Excepción.	FALSE	Excepción.

5.3. Cursos variables

- Son punteros a cursos
 - Pueden usarse para referirse a cualquier tipo de consulta (dinámicos; los otros serían estáticos)
1. Se debe definir un tipo **REF CURSOR** y declarar una variable de ese tipo.

```
TYPE tipo_cursor IS REF CURSOR RETURN agentes%ROWTYPE;
```

```
cAgentes tipo_cursor;
```

2. Despues de definir el cursor variable debe asociarse a una consulta (dinámicamente en la parte de ejecución) con la sentencia **OPEN – FOR**

```
OPEN nombre_variable_cursor FOR sentencia_select;
OPEN cAgentes FOR SELECT * FROM agentes WHERE oficina = 1;
```

- El cursor variable no puede tomar parámetros
- Pueden usarse los atributos de los cursos
- Se pueden usar varios OPEN-FOR para diferentes consultas. No necesita cerrarse antes de reabrirse; al abrir cursor variable para consulta diferente, la consulta previa se pierde.
- Tras su apertura es como un cursor normal.

```
DECLARE
TYPE cursor_Agentes IS REF CURSOR RETURN agentes%ROWTYPE;
cAgentes cursor_Agentes;
agente agentes%ROWTYPE;
BEGIN
...
OPEN cAgentes FOR SELECT * FROM agentes WHERE oficina = 1;
LOOP
  FETCH cAgentes INTO agente;
  EXIT WHEN cAgentes%NOTFOUND;
  ...
END LOOP;
CLOSE cAgentes;
...
END;
```

6. Abstracción en PL/SQL

6.1. Subprogramas

Los subprogramas pueden definirse al final de la parte declarativa de cualquier bloque. (Cualquier identificador, como es el caso de los subprogramas, debe definirse antes de usarse).

Para subprogramas en orden debe usarse la definición hacia adelante, para evitar errores de compilación. (Los que se usen por el subprograma y luego el subprograma, etc.)

Dentro de los subprogramas tenemos FUNCIONES (devuelven un valor) y PROCEDIMIENTOS (no devuelven un valor).

Una función se declara como:

```
FUNCTION nombre [parametros]
RETURN tipo_dato IS
[declaraciones locales]
BEGIN
sentencias
[EXCEPTION
manejadores]
END [nombre];
```

Un procedimiento se declara como:

```
PROCEDURE nombre [parametros] IS
[declaraciones locales]
BEGIN
sentencias
[EXCEPTION
manejadores]
END [nombre];
```

Como es obvio:

- No puede imponerse NOT NULL a los parámetros
- No pueden especificarse RESTRICCIONES
- La función debe acabar con la sentencia RETURN
- Se pueden **sobrecargar** funciones o procedimientos (llamar con el mismo nombre a subprogramas con el mismo cometido y distinto número y/o tipo de parámetros). No puede sobrecargarse programas que acepten el mismo número y tipo de parámetros. Tampoco subprogramas con mismo número y tipos diferentes pero de la misma familia o subtipos basados en la misma familia.
- Se puede aplicar **recursividad** (método que se llama a sí mismo)

6.1.1 Almacenar subprogramas en base de datos

Los subprogramas se almacenan en base de datos usando la misma sintaxis con la que se crean pero anteponiendo **CREATE** o **CREATE OR REPLACE** a **PROCEDURE** o a **FUNCTION** y finalizando el subprograma con una línea que contenga el carácter "/" para indicar que termina ahí.

El subprograma se invoca mencionándolo.

Si se quiere invocar desde SQL*Plus se usa sintaxis

```
EXECUTE nombre_procedimiento [(parametros)]
EXECUTE :variable_sql := nombre_funcion [(parametros)]
```

Si hay algún error en el programa almacenado en base de datos se sabrá porque este es compilado antes. Se informará de los mismos y debe corregirse con la cláusula OR REPLACE antes de poder ser utilizado.

Hay algunas vistas del diccionario de datos que ayudan a ver su código y los errores. También hay algunos comandos:

Vistas y comandos asociados a los subprogramas.

Información almacenada.	Vista del diccionario.	Comando.
Código fuente.	USER_SOURCE	DESCRIBE
Errores de compilación.	USER_ERRORS	SHOW ERRORS
Ocupación de memoria.	USER_OBJECT_SIZE	

Con la vista **USER_OBJECTS** se puede obtener el nombre de todos los subprogramas almacenados.

6.1.2 Parámetros de subprogramas

Parámetros actuales: Son las variables pasadas como parámetros a un subprograma

Parámetros formales: Son las variables referenciadas en la especificación del subprograma como parámetros.

Al llamar a un subprograma los parámetros actuales pueden pasarse (asociación entre parámetros actuales y formales) con notación posicional o nombrada.

Notación posicional: Primer parámetro formal, segundo con el segundo y así el resto

Notación nombrada: Operador (=>) para asociar el parámetro actual al parámetro formal. También puede usarse notación mixta.

Parámetros pueden ser de entrada, salida o de entrada/salida. Si no se especifica el modo será de entrada.

Si el parámetro es de salida o entrada/salida, este debe ser una variable.

- **Parámetro de entrada:** Para pasarle valores al subprograma. No puede ser modificado en el cuerpo. Puede ser constante o variable. Pueden inicializarse a un valor por defecto (puede invocarse en ese caso el subprograma prescindiendo del parámetro o pasándoselo y sobreescribiendo dicho valor).
- **Parámetro de salida:** Para devolver valores. Dentro del subprograma es como una variable no inicializada. Siempre debe ser una variable.
- **Parámetro de entrada/salida.** Para pasarle y devolver valores. Siempre debe ser una variable.

Para prescindir de parámetro entre medias de otros, debemos usar notación nombrada o si los parámetros restantes tienen valor por defecto, omitirlos todos.

Notación mixta

```
DECLARE
    PROCEDURE prueba( formal1 NUMBER, formal2 VARCHAR2) IS
        BEGIN
            ...
        END;
        actual1 NUMBER;
        actual2 VARCHAR2;
    BEGIN
        ...
        prueba(actual1, actual2);           --posicional
        prueba(formal2=>actual2,formal1=>actual1);   --nombrada
        prueba(actual1, formal2=>actual2);          --mixta
    END;
```

Parámetros de entrada

```
FUNCTION categoria( id_agente IN NUMBER )
RETURN NUMBER IS
    cat NUMBER;
BEGIN
    ...
    SELECT categoria INTO cat FROM agentes
    WHERE identificador = id_agente;
    RETURN cat;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        id_agente := -1; --illegal, parámetro de entrada
END;
```

Parámetros de salida

```
PROCEDURE nombre( id_agente NUMBER, nombre OUT VARCHAR2) IS
BEGIN
    IF (nombre = 'LUIS') THEN      --error de sintaxis
        END IF;
    ...
END;
```

6.1.3. Ejemplillos de subprogramas

```
CREATE OR REPLACE PROCEDURE ListadoAsistentes(pEmpresa VARCHAR2)
AS
CURSOR cursorAsistentes IS select nombre, apellido1, apellido2 from asistente where empresa
= pEmpresa;
vRegistro cursorAsistentes%ROWTYPE;
BEGIN -- Con cursor explícito
OPEN cursorAsistentes;
FETCH cursorAsistentes INTO vRegistro;
WHILE cursorAsistentes%FOUND LOOP
```

```

dbms_output.put_line(vRegistro.nombre || ' - ' || vRegistro.apellido1 || ' - ' ||
vRegistro.apellido2);
      FETCH cursorAsistentes INTO vRegistro;
END LOOP;
CLOSE cursorAsistentes; -- No lo dejes abierto!!
END ListaAsistentes;

```

```

begin
listadoasistentes('BigSoft');
end;

```

Luego ya le damos al PLAY (introducimos el parámetro) y hace lo que le pedimos.

```

CREATE OR REPLACE FUNCTION CantAsistentesConfe(pRefConfe VARCHAR2)
RETURN NUMBER
AS
vCantidad NUMBER;

BEGIN -- Con cursor implícito
SELECT COUNT(A.CODASISTENTE) INTO vCantidad
FROM CONFERENCIA C, ASISTIR A
WHERE C.REFERENCIA = A.REFCONFERENCIA
AND C.REFERENCIA=pRefConfe;
RETURN vCantidad;
END CantAsistentesConfe;

```

```

begin
CantAsistentesconfe('SEG1314');
end;

```

6.2. Paquetes

Los **paquetes** agrupan tipos, elementos y subprogramas. Tienen dos partes:

- Especificación: Declaramos interfaz.
- Cuerpo (a veces este no es necesario): Se implementa.

El paquete se declara con la sintaxis

```

CREATE [OR REPLACE] PACKAGE nombre AS
[declaraciones públicas especificaciones de subprogramas]
END;
CREATE [OR REPLACE] PACKAGE BODY nombre AS
[declaraciones privadas y cuerpo]
[BEGIN
    sentencias de inicializacion]
END [nombre]

```

La inicialización solo se ejecuta la primera vez que se referencia el paquete.

Las partes visibles del paquete se referencian con notación de punto

(call_center.borra_agente(10))

Paquete DBMS_Output

Paquete público con el que pueden enviarse mensajes desde subprogramas almacenados, paquetes y disparadores.

SQL*Plus permite visualizar los mensajes que hay en el buffer por medio del comando SET SERVEROUTPUT ON. Puede usarse para depurar subprogramas.

El paquete ofrece:

- **ENABLE.** Habilitar llamadas a los otros subprogramas. No es necesario con SERVEROUTPUT activado. Se puede pasar parámetro del tamaño del buffer.

```
ENABLE
ENABLE(buffer_size IN INTEGER DEFAULT 2000);
```

- **DISABLE.** Deshabilita llamadas a subprogramas y purga el buffer. Tampoco es necesario si usamos la opción SERVEROUTPUT

```
DISABLE
DISABLE();
```

- **PUT.** Coloca elementos en el buffer, convirtiéndolos a VARCHAR2

```
PUT
PUT(item IN NUMBER);
PUT(item IN VARCHAR2);
PUT(item IN DATE);
```

- **PUT_LINE.** Coloca elementos en el buffer y los termina con salto de línea.

```
PUT_LINE
PUT_LINE(item IN NUMBER);
PUT_LINE(item IN VARCHAR2);
PUT_LINE(item IN DATE);
```

- **NEW_LINE.** Coloca salto de línea en el buffer.

```
NEW_LINE
NEW_LINE();
```

- **GET_LINE.** Lee línea del buffer colocandola en el parámetro line y obviando el salto de línea. Devuelve 0 en status si ha traído alguna línea y 1 en caso contrario.

```
GET_LINE
GET_LINE(line OUT VARCHAR2, status OUT VARCHAR2);
```

- **GET_LINES.** Lee las líneas contenidas en numlines. Una vez ejecutado, numlines contiene el número de líneas que se ha traído. Las coloca en CHARARR, definido el paquete DBMS_OUTPUT como una tabla VARCHAR2(255);

```
GET_LINES
GET_LINES(lines OUT CHARARR, numlines IN OUT INTEGER);
```

6.3. Objetos

Tipo de objeto: Tipo de dato compuesto que encapsula datos y funciones y procedimientos para manejar esos datos.

- Las variables son atributos; Los subprogramas métodos.
- Tipos de objetos tienen: Especificación (declara atributos y métodos en ese orden; interfaz) y cuerpo (implementa la especificación).
- Todos los atributos son públicos.
- No pueden declararse atributos en el cuerpo, aunque sí declarar subprogramas locales que serán visibles en el cuerpo.

Los atributos pueden ser de cualquier tipo de datos Oracle, excepto:

- LONG y LONG RAW.
- NCHAR, NCLOB y NVARCHAR2.
- MLSLABEL y ROWID.
- Tipos específicos de PL/SQL: BINARY_INTEGER, BOOLEAN, PLS_INTEGER, RECORD, REF CURSOR, %TYPE y %ROWTYPE.
- Tipos definidos dentro de un paquete PL/SQL.

- No puede inicializarse un atributo en la declaración. Tampoco imponer NOT NULL.

El **método** es subprograma declarado en la especificación del tipo de objeto por medio de **MEMBER**. No puede llamarse igual que el tipo de objeto o que un atributo. Para cada método en la especificación debe haber un método implementando el cuerpo con la misma cabecera.

Los métodos en un tipo de objeto aceptan como primer parámetro una instancia de su tipo (parámetro **SELF**) y siempre está accesible a un método.

Si lo declaramos explícitamente debe ser el primer parámetro con el nombre SELF y el tipo del tipo de objeto.

Si SELF no está declarado explícitamente por defecto será IN para funciones e IN OUT para procedimientos.

Los métodos pueden sobrecargarse. Pero no si solo difieren en el modo o pertenecen a la misma familia. Tampoco si solo difiere en el tipo devuelto.

El objeto sigue las reglas y visibilidad que cualquier otra variable.

Cuando no se declara es automáticamente NULL. Deja de ser nulo al inicializarse o al asignarle otro. Si se intenta acceder a los atributos de NULL sale la excepción **ACCESS_INTO_NULL**

Los objetos tienen constructores por defecto con el mismo nombre que el tipo de objeto y que aceptan tantos parámetros como atributos del tipo de objeto y con el mismo tipo.

Un tipo de objeto puede tener a otro entre sus atributos. Este tipo debe ser conocido por Oracle. Si son mutuamente dependientes puede usarse declaración hacia adelante.

Ejemplito

```
CREATE OBJECT Oficina;      --Definición hacia delante
CREATE OBJECT Familia AS OBJECT (
    identificador      NUMBER,
    nombre            VARCHAR2(20),
    familia_          Familia,
    oficina_          Oficina,
    ...
);
CREATE OBJECT Agente AS OBJECT (
    identificador      NUMBER,
    nombre            VARCHAR2(20),
    familia_          Familia,
    oficina_          Oficina,
    ...
);
CREATE OBJECT Oficina AS OBJECT (
    identificador      NUMBER,
    nombre            VARCHAR2(20),
    jefe              Agente,
    ...
);
```

6.3.1. Funciones mapa y de orden

Los tipos de objetos no tienen orden predefinido por lo que no pueden ser comparados ni ordenados. Puede definirse el orden que seguirá el tipo de objeto por las funciones mapa y de orden.

Función miembro mapa: Función sin parámetros que devuelve un tipo de dato: DATE, NUMBER, VARCHAR2 y es similar a función hash. Se define anteponiendo la palabra MAP y solo puede haber una para cada tipo de objeto.

Función miembro de orden: Función que acepta parámetro del mismo tipo del tipo de objeto y devuelve número negativo si el objeto pasado es menor, cero si son iguales y positivo si es mayor. Anteponiendo la palabra ORDEN.

```
CREATE TYPE Familia AS OBJECT (
    identificador      NUMBER,
    nombre            VARCHAR2(20),
    familia_          NUMBER,
    oficina_          NUMBER,
    MAP MEMBER FUNCTION orden RETURN NUMBER,
    ...
);

CREATE TYPE BODY Familia AS
    MAP MEMBER FUNCTION orden RETURN NUMBER IS
        BEGIN
            RETURN identificador;
        END;
    ...
END;

CREATE TYPE Oficina AS OBJECT (
    identificador      NUMBER,
    nombre            VARCHAR2(20),
    ...
    ORDER MEMBER FUNCTION igual ( ofi Oficina ) RETURN INTEGER,
    ...
);

CREATE TYPE BODY Oficina AS
    ORDER MEMBER FUNCTION igual ( ofi Oficina ) RETURN INTEGER IS
        BEGIN
            IF (identificador < ofi.identificador) THEN
                RETURN -1;
            ELSIF (identificador = ofi.identificador) THEN
                RETURN 0;
            ELSE
                RETURN 1;
            END IF;
        END;
    ...
END;
```

6.4. Disparadores (Triggers)

Un **disparador** es un procedimiento ejecutado cuando se realiza una sentencia de manipulación de datos sobre una tabla dada y bajo unas circunstancias establecidas a la hora de definirlo.

Usos: Auditorías, integridad, generación de valores...

No deberían: Duplicar funcionalidades de Oracle, deben limitarse y ser codificados por medio de subprogramas que se llamen desde el disparador, debe cuidarse los disparadores recursivos.

Clasificación

- Si va a hacerse antes o después (Disparadores BEFORE y disparadores AFTER)
- Si va a lanzarse una vez por sentencia o una vez por fila (Disparadores de sentencia y disparadores de fila)
- Si va a lanzarse al insertar, actualizar, borrar (Disparadores INSERT, UPDATE, DELETE o mezclados)

La sintaxis es:

```
CREATE [OR REPLACE] TRIGGER nombre
momento acontecimiento ON tabla
[[REFERENCING (old AS alias_old|new AS alias_new)
FOR EACH ROW
[WHEN condicion]]
bloque_PL/SQL;
```

nombre: Se indica el nombre del disparador.

momento: Se indica cuándo será lanzado el disparador (BEFORE o AFTER)

acontecimiento: Acción que provoca el lanzamiento del disparador (INSERT UPDATE DELETE)

REFERENCING Y WHEN solo con algunos disparadores

- REFERENCING: Asignar alias a valores NEW y/o OLD de las filas afectadas por la operación
- WHEN: Que solo sea disparado cuando sea verdadera una condición evaluada para cada fila afectada

En los disparadores de fila pueden accederse a valores antiguos y nuevos con :old y :new.

Si el disparador se lanza al insertar el valor antiguo no tiene sentido; el nuevo es la que se inserrta.

Si lanza al actualizar los dos tienen sentido

Si se lanza al borrar solo tiene sentido el valor antiguo.

En el cuerpo del disparador puede accederse a predicados que indican qué tipo de operación se lleva a cabo: INSERTING, UPDATING, DELETING.

El disparador de fila no puede acceder a tabla asociada. Esa tabla está mutando.

Si el disparador es lanzado en cascada por otro disparador ,no puede acceder a ninguna de las tablas asociadas y así recursivamente.

```
CREATE TRIGGER prueba BEFORE UPDATE ON agentes
FOR EACH ROW
BEGIN
  ...
  SELECT identificador FROM agentes WHERE ...
/*devolvería el error ORA-04091: table AGENTES is mutating, trigger/function may not see
it*/
  ...
END;
/
```

Si tenemos varios tipos de disparadores sobre una misma tabla, el orden de ejecución será:

- Triggers before de sentencia.
- Triggers before de fila.
- Triggers after de fila.
- Triggers after de sentencia.

Existe vista del diccionario de datos con información sobre los disparadores: `USER_TRIGGERS`

```
SQL>DESC USER_TRIGGERS;
Name          Null?    Type
-----
TRIGGER_NAME      NOT NULL VARCHAR2(30)
TRIGGER_TYPE        VARCHAR2(16)
TRIGGERING_EVENT      VARCHAR2(26)
TABLE_OWNER        NOT NULL VARCHAR2(30)
TABLE_NAME        NOT NULL VARCHAR2(30)
REFERENCING_NAMES      VARCHAR2(87)
WHEN_CLAUSE        VARCHAR2(4000)
STATUS            VARCHAR2(8)
DESCRIPTION        VARCHAR2(4000)
TRIGGER_BODY        LONG
```

Ejemplitos

```

/*Cuando asistente se inscriba en tabla ASISTIR, que se actualice el campo RECAUDACIÓN de la
tabla CONFERENCIAS con la suma*/
CREATE OR REPLACE TRIGGER actualizaRecaudacion
AFTER INSERT ON asistir
FOR EACH ROW
BEGIN
    UPDATE conferencia SET recaudacion = recaudacion + precio
    WHERE referencia = :NEW.refconferencia;
END actualizaRecaudacion;

```

```

/*Almacenar en auditoría los movimientos de otras tablas*/
CREATE OR REPLACE TRIGGER auditarGratificacion
AFTER UPDATE ON participar
FOR EACH ROW
DECLARE
    usuario VARCHAR2(20);
BEGIN
    SELECT SYS_CONTEXT('userenv', 'current_user') INTO usuario FROM dual;
    INSERT INTO autoria_gratificacion
    VALUES (usuario, sysdate, :OLD.codponente, :OLD.refconferencia, :NEW.geatificacion,
    :NEW.gratificacion);
END auditarGratificacion;

```

```

/*Como un agente debe pertenecer a una familia o una oficina pero no puede pertenecer a una
familia y a una oficina a la vez, deberemos implementar un disparador para llevar a cabo
esta restricción que Oracle no nos permite definir.*/
CREATE OR REPLACE TRIGGER integridad_agentes
BEFORE INSERT OR UPDATE ON agentes
FOR EACH ROW
BEGIN
    IF (:new.familia IS NULL and :new.oficina IS NULL) THEN
        RAISE_APPLICATION_ERROR(-20201, 'Un agente no puede ser huérfano');
    ELSIF (:new.familia IS NOT NULL and :new.oficina IS NOT NULL) THEN
        RAISE_APPLICATION_ERROR(-20202, 'Un agente no puede tener dos padres');
    END IF;
END;

```

```

/*Actualizar secuencial*/
create or replace TRIGGER "operacionTrigger" BEFORE INSERT ON OPERACION
FOR EACH ROW
BEGIN
    <<COLUMN_SEQUENCES>>
    BEGIN
        IF :NEW.ID IS NULL THEN
            SELECT OPERACION_SEQUENCE.NEXTVAL INTO :NEW.ID FROM DUAL;
        END IF;
    END COLUMN_SEQUENCES;
END;

```

<<COLUMN_SEQUENCES>> actúa como un marcador o etiqueta que indica dónde comienza y termina un bloque de código dentro del trigger relacionado con la generación secuencial de valores para la columna ID.

6.5. Temporizadores

Un temporizador es un objeto que inicia la ejecución de un programa en momentos determinados por el creador del temporizador. Los hay de dos tipos:

- De un solo disparo (oneshot). Se programan para un momento determinado en el futuro. Cuando llega ese instante se disparan y se eliminan.
- De disparo recurrente (periodical). Se programan para que se disparen a intervalos definidos. Despues de cada disparo el temporizador se rearma, espera el intervalo y se vuelve a disparar. Esto se realiza continuamente hasta que se eliminan manualmente.

(MariaDB)

```

CREATE EVENT nombre ON SCHEDULE tiempo [ON COMPLETION PRESERVE]
[activacion] DO sentencia;

```

```
CREATE EVENT limpia_cuentas_2018 ON SCHEDULE AT '2018-01-01
00:00:00' DO DELETE FROM usuarios WHERE ultimoAcceso < '2017-06-30
23:59:59';
DROP EVENT IF EXISTS limpia_cuentas_2018;
ALTER EVENT limpia_cuentas_mensual ENABLE;
```

6.6. Interfaces de programación de aplicaciones para lenguajes externos

Las primeras APIs para acceder a bases de datos Oracle fueron proporcionadas por Oracle: Pro*C (el más usado por la popularidad de C, C++), Pro*Fortran y Pro*Cobol. Permitían embeber llamadas a BBDD en el programa y al compilarlo debía pasarse el precompilador adecuado que trasladaba esas llamadas embebidas en llamadas a una librería usada en tiempo de ejecución.

Después... JDBC (Java Database Connectivity), Java DB, Java Data Objects (JDO). Open Database Connectivity (ODBC).

UD 07 - USO DE BASES DE DATOS OBJETO-RELACIONALES

1. Características de las bases de datos objeto-relacionales

Las **bases de datos objeto-relacionales** son aquellas que han evolucionado del modelo relacional tradicional a un modelo híbrido que utiliza tecnología orientadas a objetos.

Los esquemas de la base de datos soportan DDL, DML y, además, clases, objetos, herencia... Da soporte a una extensión del modelo de datos con creación personalizada de tipo de datos y métodos.

Oracle implementa el modelo orientado a objetos como **extensión del modelo relacional**, soportando la funcionalidad estandar de bases de datos relacionales.

- El modelo objeto-relacional ofrece ventajas de las técnicas orientadas a objetos (reutilización, uso intuitivo de objetos) a la par que mantiene alta capacidad de concurrencia y buen rendimiento de las bases de datos relacionales.
- Es un método de acceso a **alto nivel**. Por debajo de la capa de objetos, los datos seguirán estando almacenados en columnas y tablas.
- La reutilización de objetos permite desarrollar aplicaciones de bases de datos de forma más rápida y eficiente. Además, los desarrolladores puede acceder directamente a las mismas estructuras de datos desde sus aplicaciones orientadas a objetos.
- Los datos son solo datos. Pero los objetos pueden contener además **acciones sobre los datos**.
- En PL/SQL la POO está basada en tipos de objetos.
- En las bases de datos objeto-relacionales los dominios de la base de datos ya no son solo atómicos luego no se cumple la 1FN, debido a que las tuplas también pueden ser una relación, que llevará a relaciones de relaciones.

2. Tipos de datos objeto

Tipo de dato objeto es un tipo de dato compuesto definido por el usuario. Representa a una **estructura de datos** y a **funciones y procedimientos** para manipular datos.

Las **variables** de un tipo de dato escalar pueden almacenar **un único valor** y las **colecciones varios siendo todos del mismo tipo**.

Los **tipos de datos objetos** permiten **almacenar datos de distintos tipos, posibilitando asociar código a dichos datos**

Los tipos de datos completos se componen de variables (atributos, propiedades) y métodos (funciones y procedimientos, acciones)

Al definir el tipo de objeto, se crea una **plantilla abstracta**.

Aunque los atributos son públicos (visibles desde otros programas cliente), la manipulación de datos deberá hacerse solo a través de los métodos declarados en el tipo objeto (ellos pueden hacer un chequeo de los datos para que se mantenga un estado apropiado en los mismos)

En la ejecución, la aplicación crea **instancias** de un tipo objeto, referencias a objetos reales con valores asignados en sus atributos.

3. Definición de tipos de objeto

La definición o declaración del tipo de objeto está dividida en una especificación y un cuerpo.

Especificación:

- Define el interfaz de programación donde se declaran los atributos y las operaciones para manipular los datos.
- Toda la información que un programa necesita para usar los métodos lo encuentra dentro de la especificación.

- En ella los atributos deben ser declarados **antes que** los métodos.
- Como mínimo debe tener un atributo declarado y un máximo de 1000.
- Los métodos son opcionales. Puede crearse sin método
- En ella pueden declararse atributos y métodos pero no constantes (CONSTANTS), excepciones (EXCEPTIONS), cursos (CURSORS) o tipos (TYPES)

Cuerpo:

- Se implementa el código fuente de los métodos.
- Si la especificación solo declara atributos, no es necesario declarar el cuerpo
- No pueden declararse atributos en el cuerpo
- Todas las declaraciones realizadas en la especificación son públicas (visibles fuera del tipo del objeto)

Definición del tipo objeto

Para definir el tipo de objeto se usa la sentencia `CREATE TYPE.... AS OBJECT`

Eliminar el tipo de objeto

```
DROP TYPE nombre_tipo_objeto;
```

3.1. Declaración de atributos

La declaración de atributos es parecida a la declaración de variables. Como en otras ocasiones puede ser cualquiera de los tipos de Oracle excepto:

- LONG y LONG RAW
- ROWID y UROWID
- Tipos específicos de PL/SQL: BINARY_INTEGER (y subtipos), BOOLEAN, PLS_INTEGER, RECORD, REF CURSOR, %TYPE y %ROWTYPE
- Tipos definidos en un paquete PL/SQL

No se pueden inicializar con := , ni usar DEFAULT, ni NOT NULL

El tipo de dato declarado puede ser **otro tipo de objeto**.

Definición del tipo objeto con atributos

```
CREATE (OR REPLACE) TYPE Usuario AS OBJECT(
    login VARCHAR2(10),
    nombre VARCHAR2(30),
    f_ingreso DATE,
    credito NUMBER
);
```

Modificación del tipo objeto con atributos

```
ALTER TYPE Usuario DROP ATTRIBUTE f_ingreso;
ALTER TYPE Usuario ADD ATTRIBUTE (apellidos VARCHAR2(40), localidad VARCHAR2(50));
ALTER TYPE Usuario
    ADD ATTRIBUTE cp VARCHAR2(5),
    MDOIFY ATTRIBUTE nombre VARCHAR2(35);
```

3.2. Definición de métodos

Con `MEMBER` o `STATIC`

Definición de especificación

```

CREATE (OR REPLACE) TYPE Usuario AS OBJECT(
    login VARCHAR2(10),
    nombre VARCHAR2(30),
    f_ingreso DATE,
    credito NUMBER,
    MEMBER PROCEDURE incrementoCredito(inc NUMBER)
);

```

Definición de cuerpo

```

CREATE OR REPLACE TYPE BODY Usuario as
    MEMBER PROCEDURE incrementoCredito(inc NUMBER) IS
        BEGIN
            credito := credito + inc
        END incrementoCredito;
END;

```

Debe existir el correspondiente cuerpo por cada especificación o el método debe declararse como **NOT INSTANTIABLE**, para indicar que el cuerpo del método se encontrará en un subtipo de ese tipo de objeto.

El código fuente de los métodos no solo puede escribirse en PL/SQL. También en otros lenguajes como Java o C.

Con **ALTER TYPE** puede añadirse, modificar o eliminar métodos de un tipo de objeto existente.

3.3. Parámetro SELF

SELF se refiere a una instancia (objeto) del mismo tipo de objeto. Siempre se declara automáticamente aunque no esté declarado específicamente.

- En funciones MEMBER: Si no se declara su modo por defecto se toma como INT
- En procedimientos MEMBER: Si no se declara su modo por defecto se toma como INT OUT
- Los métodos STATIC no puede usar este parámetro especial.
- No puede especificar el modo OUT para el parámetro SELF
- Al hacer referencia a SELF dentro del cuerpo de un método se está haciendo referencia al objeto que invoca ese método. **SELF.nombre_atributo**, **SELF.nombre_metodo**....(Equivalente a **this**, vamos)

3.4. Sobrecarga

Los métodos pueden ser sobrecargados: **Utilizar el mismo nombre para métodos diferentes**, siempre que sus parámetros sean diferentes en cantidad o en tipo de dato.

Se ejecutará aquel en el que haya una coincidencia entre los parámetros actuales y los formales del método declarado.

```

MEMBER PROCEDURE setNombre(Nombre VARCHAR2)
MEMBER PROCEDURE setNombre(Nombre VARCHAR2, Apellidos VARCHAR2)

```

3.5. Métodos constructores

Cada tipo de objeto tiene un método constructor.
Oracle crea uno por defecto.

Pueden declararse métodos propios constructores reescribiendo ese método o definiendo un nuevo con otros parámetros.

Se puede hacer en el nuevo método constructor una verificación de que los datos que se van a asignar a los atributos son correctos (por ejemplo)

Para reemplazar el constructor por defecto debe usarse la sentencia **CONSTRUCTOR FUNCTION** seguida del nombre del tipo de objeto en el que se encuentra. Después se indican los parámetros necesarios. Por último el valor de retorno de la función que es el propio objeto usando la cláusula **RETURN SELF AS RESULT**

Declaración y cuerpo

```
CONSTRUCTOR FUNCTION Usuario(login VARCHAR2, credito NUMBER)
    RETURN SELF AS RESULT

CREATE OR REPLACE TYPE BODY Usuario AS
    CONSTRUCTOR FUNCTION Usuario(login VARCHAR2, credito NUMBER)
        RETURN SELF AS RESULT
    IS
        BEGIN
            IF (credito >= 0) THEN
                SELF.credito := credito;
            ELSE
                SELF.credito := 0;
            END IF;
            RETURN;
        END;
    END;
```

Métodos MAP / ORDER

Las instancias de tipo de objeto no tienen orden predefinido. Para establecerlo debe crearse método **MAP** o método **ORDER**

MAP

Para comparar. Debe empezar la declaración con la palabra MAP. Solo puede haber un método declarado.

```
MAP MEMBER FUNCTION ordenarUsuario RETURN VARCHAR2
```

El cuerpo del método debe retornar el valor que se usará para realizar las comparaciones.

```
CREATE OR REPLACE TYPE BODY Usuario AS
    MAP MEMBER FUNCTION ordenarUsuario RETURN VARCHAR2 IS
        BEGIN
            RETURN (apellidos || ' ' || nombre);
        END ordenarUsuario;
    END;
/
```

ORDER

- Permitirá establecer un orden. Debe empezar la declaración con la palabra ORDER
- Solo puede haber un método ORDER.
- Debe retornar un valor numérico que permita establecer el orden entre los objetos. (ejemplo -1, 0, 1)

```
CREATE OR REPLACE TYPE BODY Usuario AS
    ORDER MEMBER FUNCTION ordenUsuario(u Usuario) RETURN INTEGER IS
        BEGIN
            /* La función substr obtiene una subcadena desde la posición indicada hasta el final*/
            IF substr(SELF.login, 7) < substr(u.login, 7) THEN
                RETURN -1;
            ELSIF substr(SELF.login, 7) > substr(u.login, 7) THEN
                RETURN 1;
            ELSE
                RETURN 0;
            END IF;
```

```
END;  
END;
```

Se puede declarar un método MAP o un método ORDER pero no los dos. Cuando se vaya a ordenar o a mezclar un alto número de objetos es preferible usar el método MAP porque ORDER sería menos eficiente

4. Utilización de objetos

Veamos cómo se declaran variables para almacenar objetos, se dan valores iniciales a los atributos, se accede a su contenido y se llama a los métodos que ofrece el tipo de objeto.

4.1. Declaración de objetos

Cuando el tipo de objeto ha sido creado, puede usarse para declarar variables de ese tipo en cualquier bloque PL/SQL, subprograma o paquete. Puede usarse como tipo de dato para variable, atributo, elemento de tabla, parámetro formal o resultado de una función igual que se usaban los tipos habituales.

Si hemos creado un objeto usuario, podemos declarar una variable:

```
u1 Usuario;
```

Puede ponerse en la declaración de un procedimiento para que se pase un objeto por atributo:
`PROCEDURE setUsuario(u IN Usuario)` y pasarlo sin más `setUsuario(u1)`

La función puede retornar objetos: `FUNCTION getUsuario(codigo INTEGER) RETURN Usuario;`

Los objetos se crean durante la ejecución del código como instancias y cada uno puede contener valores diferentes en sus atributos.

Los **ámbitos** siguen las reglas habituales de PL/SQL. En el bloque o subprograma los objetos se crean cuando se entra en ese bloque o subprograma y se destruyen automáticamente cuando se sale de ellos. En el paquete, se instancian cuando se hace referencia al paquete y dejan de existir cuando finaliza la sesión en la BBDD.

4.2. Inicialización de objetos

Para instanciar el objeto se debe llamar al método constructor usando `NEW` seguido del nombre del objeto (lo de siempre, vamos).

El orden de los parámetros debe coincidir con cómo están declarados los atributos, así como sus tipos de datos (obvio).

```
Usuario := NEW Usuario('luisitom','LUIS','TOMAS UREÑA', '01/02/03', 100);
```

Lo habitual sería incluso inicializarlos en la declaración (y eso que nos quitamos)
En estos casos puede suprimirse el NEW

```
Usuario := NEW Usuario('luisitom','LUIS','TOMAS UREÑA', '01/02/03', 100);
```

El método constructor puede llamarse en cualquier lugar en dónde se llame a una función habitualmente. Podría por ejemplo usarse como parte de una expresión.

Hay posibilidad de usar los **nombres de los parámetros formales** al llamar al constructor, en lugar de usar el modelo posicional de los parámetros. Así no es necesario respetar el orden:

```
DECLARE  
u1 Usuario;  
BEGIN  
u1 := NEW Usuario('user1', -10);  
/* Se mostrará el crédito como cero, al intentar asignar un crédito negativo */  
dbms_output.put_line(u1.credito);
```

```
END;  
/
```

4.3. Acceso a los atributos de objetos

Se accede con la notación de punto. `nombre_objeto.nombre_atributo`

```
unNombre := usuario1.nombre;  
dbms_output.put_line(usuario1.nombre);
```

Se modifica:

```
usuario1.nombre:= 'Nuevo Nombre';
```

Los objetos pueden ser encadenados para acceder a tipos de objetos anidados.

Si se intenta **acceder a un objeto no inicializado** da excepción `ACCESS_INTO_NULL`

Puede comprobarse si un objeto es NULL con `IS NULL`

Si se intenta **llamar a un método de un objeto no inicializado** da excepción `NULL_SELF_DISPATCH`

Si se pasa como parámetro de IN un NULL, se evalúa como NULL

Si el parámetro es de tipo OUT o IN OUT, lanza excepción al intentar modificar sus atributos.

4.4. Llamada a los métodos de objetos

Utilizando un punto entre el nombre del objeto y el método.

```
credito := usuario1.getCredito();
```

Las llamadas a métodos pueden encadenarse.

```
sitio1.getUsuario.setNombreCompleto('Juan', 'García Fernández');
```

Los métodos `MEMBER` son invocados usando instancia del objeto

```
nombre_objeto.metodo()
```

Los métodos `STATIC` son invocados usando el tipo del objeto

```
nombre_tipo_objeto.metodo()
```

4.5. Herencia

PL/SQL admite herencia simple de objetos, mediante la cual se pueden definir subtipos de los tipos de objeto. Estos subtipos **contienen todos los atributos y métodos del tipo padre** y además atributos y métodos adicionales o incluso sobreescribir métodos del tipo padre.

Para que pueda heredarse un objeto debemos poner al final de su declaración `NOT FINAL` (porque por defecto se declaran como `FINAL`).

Para indicar que un tipo de objeto es heredado se usa la palabra reservada `UNDER`:

```
CREATE OR REPLACE TYPE Envio AS OBJECT(  
    referencia VARCHAR2(5),  
    fecha DATE  
);  
/  
CREATE OR REPLACE TYPE Paquete UNDER Envio (  
    peso NUMBER(3)  
);  
/  
CREATE OR REPLACE TYPE Sobre UNDER Envio (  
    peso NUMBER(3)  
);
```

```
DECLARE  
    vEnvioMaria Paquete;  
    vEnvioPepe Sobre;  
BEGIN  
    vEnvioMaria:= NEW Paquete('20001','P3','Normal');
```

```
vEnvioPep := NEW Sobre (('20002','P3','Urgente',5,10,5,5,250);
```

Si no se indica lo contrario, siempre se pueden crear instancias de los tipos declarados. Con la opción **NOT INSTANTIABLE** se puede declarar tipos de objetos de los que no se pueden crear objetos. Estos tipos tendrán como función ser padres de otros objetos.

En la creación de un objeto se puede observar que se asignan valores para todos los atributos incluyendo los heredados.

```
CREATE TYPE Persona AS OBJECT (
    nombre VARCHAR2(20),
    apellidos VARCHAR2(30)
) NOT FINAL;
/

CREATE TYPE UsuarioPersona UNDER Persona (
    login VARCHAR(30),
    f_ingreso DATE,
    credito NUMBER
);
/

DECLARE
    u1 UsuarioPersona;
BEGIN
    u1 := NEW UsuarioPersona('nombre1', 'apellidos1', 'user1', '01/01/2001', 100);
    dbms_output.put_line(u1.nombre);
END;
/
```

5. Tipos de datos colección

Los **tipos de datos colección** son ofrecidos por Oracle para almacenar en memoria un conjunto de datos de un tipo determinado. (Parecidos a lenguajes y matrices en otros lenguajes). Las colecciones solo pueden tener una dimensión y los elementos se indexan mediante valor numérico o cadena de caracteres.

Oracle ofrece:

- **VARRAY**. Se le establece dimensión máxima. Al tener longitud fija, la eliminación de elementos no ahorra espacio en memoria.
- **NESTED TABLE**. Tabla anidada. Puede almacenar cualquier número de elementos. Es de tamaño dinámico. No tienen que existir forzosamente valores para todas las posiciones de la colección.
- **Arrays asociativos**. Usan valores arbitrarios para sus índices. No tienen que ser necesariamente consecutivos.

Para almacenar número fijo de elementos, recorrerlos de forma ordenada u obtener y manipular toda la colección como un valor -> VARARRAY

Para ejecutar consultas de forma eficiente, manipular número arbitrario, realizar operaciones de inserción, actualización, borrado de forma masiva -> NESTED TABLE.

Las colecciones pueden ser declaradas como una instrucción SQL o en el bloque de declaraciones del programa PL/SQL. El tipo de dato de los elementos que puede contener una colección puede ser cualquiera excepto REF_CURSOR. No pueden ser de los tipos **BINARY_INTEGER**, **PLS_INTEGER**, **BOOLEAN**, **LONG**, **LONG RAW**, **NATURAL**, **NATURALN**, **POSITIVE**, **POSITIVEN**, **REF_CURSOR**, **SIGNTYPE**, **STRING**.

Cualquier tipo de objeto previamente declarado puede ser usado como tipo de elemento para una colección.

La tabla de la base de datos puede contener columnas que sean colecciones y sobre ellas hacer operaciones de consulta y manipulación de datos igual que se hace con tablas con los tipos de datos habituales.

5.1. Declaración y uso de colecciones

La declaración de las colecciones sigue el formato siguiente:

```
TYPE nombre_tipo IS VARRAY (tamaño_max) OF tipo_elemento;
TYPE nombre_tipo IS TABLE OF tipo_elemento;
TYPE nombre_tipo IS TABLE OF tipo_elemento INDEX BY tipo_índice;
```

indicando el nombre de la colección, el tamaño máximo en el caso del VARRAY y el tipo de elemento que la compone. El tipo_índice representa el tipo de dato que se usará para el índice. Puede ser PLS_INTEGER, BINARY_INTEGER, VARCHAR2.

Si se hace en SQL, fuera del subprograma se debe declarar como

```
CREATE (OR REPLACE) TYPE nombre_tipo IS...
```

Hasta que la colección no se inicialice, esta es NULL. Para inicializarla debe usarse el constructor pasando los valores iniciales de la colección. Ejemplo:

```
DECLARE
    TYPE Colores IS TABLE OF VARCHAR(10);
    misColores Colores;
BEGIN
    misColores := Colores('Rojo', 'Naranja', 'Amarillo', 'Verde', 'Azul');
END;
```

No solo en el bloque, sino en la zona de declaraciones también puede ser inicializada:

```
DECLARE
    TYPE Colores IS TABLE OF VARCHAR(10);
    misColores Colores := Colores('Rojo', 'Naranja', 'Amarillo', 'Verde', 'Azul');
```

Para obtener uno de los elementos se debe indicar el nombre de la colección y su indice

```
misColores(2)
```

El contenido de la posición puede modificarse

```
misColores(3) := 'Gris'
```

```
CREATE TYPE ListaColores AS TABLE OF VARCHAR2(20);
/
CREATE TABLE flores (nombre VARCHAR2(20), coloresFlor ListaColores)
    NESTED TABLE coloresFlor STORE AS colores_tab;

DECLARE
    colores ListaColores;
BEGIN
    INSERT INTO flores VALUES('Rosa', ListaColores('Rojo','Amarillo','Blanco'));
    colores := ListaColores('Rojo','Amarillo','Blanco','Rosa Claro');
    UPDATE flores SET coloresFlor = colores WHERE nombre = 'Rosa';
    SELECT coloresFlor INTO colores FROM flores WHERE nombre = 'Rosa';
END;/
```

6. Tablas de objetos

Los objetos pueden almacenarse en tablas de igual manera que los tipos de datos habituales. Los tipos de datos objetos pueden estar en una tabla exclusivamente formada por elementos de ese tipo o como un tipo de columna más entre otros tipos de datos.

Tabla exclusivamente formada por un tipo de dato objeto (Tabla de objetos) debe usarse:

```
CREATE TABLE NombreTabla OF TipoObjeto
```

Si una tabla hace uso de un tipo de objeto **no puede eliminarse ni modificar la estructura de dicho tipo de objeto**. (No puede volver a definirse).

Además previamente debe estar declarado ese tipo de objeto.

Los atributos del tipo de objeto se muestran como si fueran las columnas de la tabla.

6.1. Tablas con columnas tipo objeto

Cuando es una columna más en una tabla simplemente debe hacerse como si fuera una columna más:

```
CREATE TABLE Gente (
    dni VARCHAR2(10),
    unUsuario Usuario,
    partidasJugadas SMALLINT
);
```

Los datos del campo `unUsuario` se muestran como integrantes de cada objeto `Usuario`.

6.2. Uso de sentencia SELECT

Las sentencia SELECT puede usarse para obtener datos de las filas almacenadas en tablas de objetos o en tablas de columnas de tipos de objetos.

Si se trata de tablas con columnas de tipo objeto el acceso a los atributos debe hacerse indicando previamente el nombre asignado a la columna que contiene los objetos.

```
SELECT g.unUsuario.nombre, g.unUsuario.apellidos FROM Gente g;
```

6.3. Inserción de objetos

Para insertar el objeto, debe suministrarse a la sentencia insert un objeto instanciado de su tipo de objeto correspondiente.

```
DECLARE
    u1 Usuario;
    u2 Usuario;
BEGIN
    u1 := NEW Usuario('luitom64', 'LUIS', 'TOMAS BRUNA', '24/10/2007', 50);
    u2 := NEW Usuario('caragu72', 'CARLOS', 'AGUDO SEGURA', '06/07/2007', 100);
    INSERT INTO UsuariosObj VALUES (u1);
    INSERT INTO UsuariosObj VALUES (u2);
END;
```

También se podría crear el objeto dentro de la sentencia INSERT directamente

```
INSERT INTO UsuariosObj VALUES (Usuario('luitom64', 'LUIS', 'TOMAS BRUNA',
'24/10/2007', 50));
```

Igual en tablas con columnas de tipo objeto

```
INSERT INTO Gente VALUES ('22900970P', Usuario('luitom64', 'LUIS', 'TOMAS BRUNA',
'24/10/2007', 50), 54);
INSERT INTO Gente VALUES ('62603088D', u2, 21);
```

6.4. Modificación de objetos

Para modificar un objeto almacenado es igual que siempre también.

Si se trata de tabla de objeto se hace referencia a los atributos justo detrás del nombre de la tabla. Si es con columnas de tipo objeto se debe referenciar al nombre de la columna que contiene los objetos.

```
UPDATE UsuariosObj u
SET u.credito = 0
WHERE u.login = 'luitom64';
```

```
UPDATE Gente g
    SET g.unUsuario.credito = 0
    WHERE g.unUsuario.login = 'luitom64';
```

También puede cambiarse todo un objeto por otro.

```
UPDATE Gente g
    SET g.unUsuario = Usuario('juaesc82', 'JUAN', 'ESCUDERO LARRASA', '10/04/2011', 0)
    WHERE g.unUsuario.login = 'caragu72';
```

6.5. Borrado de objetos

Similar a como se hacía.

```
DELETE FROM UsuariosObj u WHERE u.credito = 0;
DELETE FROM Gente g WHERE g.unUsuario.credito = 0;
```

6.6. Consultas con función VALUE

Para hacer referencia a un objeto en lugar de a alguno de sus atributos puede usarse la función **VALUE** junto con el nombre de la tabla de objetos o su alias.

```
INSERT INTO Favoritos SELECT VALUE(u) FROM UsuariosObj u WHERE u.credito >= 100;
SELECT u.login FROM UsuariosObj u JOIN Favoritos f ON VALUE(u)=VALUE(f);
```

Cuando es una columna de tipo objeto la referencia que se hace a la columna permite obtener directamente un objeto sin necesidad de VALUE

```
SELECT g.dni FROM Gente g JOIN Favoritos f ON g.unUsuario=VALUE(f);
```

Con la cláusula **INTO** puede guardarse en variables el objeto obtenido con VALUE

```
DECLARE
    u1 Usuario;
    u2 Usuario;
BEGIN
    SELECT VALUE(u) INTO u1 FROM UsuariosObj u WHERE u.login = 'luitom64';
    dbms_output.put_line(u1.nombre);
    u2 := u1;
    dbms_output.put_line(u2.nombre);
END;
```

6.7. Referencias a objetos

El paso de objetos es ineficiente si son de gran tamaño. Es más conveniente pasar un puntero a dicho objeto. Eso se puede hacer como una referencia (REF).

Se crea usando el modificador REF delante del tipo de objeto y se puede usar con variables, parámetros, capos, atributos, variables de entrada o salida...

Solo pueden hacerse referencia a tipos de objetos que han sido declarados previamente.

```
CREATE OR REPLACE TYPE Partida AS OBJECT (
    codigo INTEGER,
    nombre VARCHAR2(20),
    usuarioCreador REF Usuario
);
/
```

```

DECLARE
    u_ref REF Usuario;
    p1 Partida;
BEGIN
    SELECT REF(u) INTO u_ref FROM UsuariosObj u WHERE u.login = 'luitom64';
    p1 := NEW Partida(1, 'partida1', u_ref);
END;
/

```

El orden de declaración no puede invertirse salvo en caso de hacer una **declaración de tipo anticipada**. Se indica el nombre del objeto que se detallará más adelante.

```

CREATE OR REPLACE TYPE tipo2;
/
CREATE OR REPLACE TYPE tipo1 AS OBJECT (
    tipo2_ref REF tipo2
    /*Declaración del resto de atributos del tipo1*/
);
/
CREATE OR REPLACE TYPE tipo2 AS OBJECT (
    tipo1_ref REF tipo1
    /*Declaración del resto de atributos del tipo2*/
);

```

6.8. Navegación a través de referencias

No se puede acceder directamente a los atributos de un objeto referenciado que se encuentre almacenado en una tabla.

Para eso debe usarse la función **DEREF** que toma referencia al objeto y retorna el valor del objeto.

Suponiendo que disponemos de las siguientes variable declaradas:

```

u_ref REF Usuario;
u1 Usuario;

```

Si `u_ref` hace referencia a un objeto de tipo `Usuario` que se encuentra en la tabla `UsuariosObj`, para obtener información sobre alguno de los atributos de dicho objeto referenciado, hay que utilizar la función `DEREF`.

Esta función se utiliza como parte de una consulta SELECT por lo que hay que utilizar una tabla tras la cláusula `FROM`. Esto puede resultar algo confuso, ya que las referencias a objetos apuntan directamente a un objeto concreto que se encuentra almacenado en una determinada tabla. Por tanto, no debería ser necesario indicar de nuevo en qué tabla se encuentra. Realmente es así. Podemos hacer referencia a cualquier tabla en la consulta, y la función `DEREF` nos devolverá el objeto referenciado que se encuentra en su tabla correspondiente.

La base de datos de Oracle ofrece la **tabla DUAL** para este tipo de operaciones. Esta tabla es creada de forma automática por la base de datos, es accesible por todos los usuarios, y **tiene un solo campo y un solo registro**. Por tanto, es como una tabla comodín que devuelve una única fila con una única columna.

```

SELECT DEREF(u_ref) INTO u1 FROM Dual;
dbms_output.put_line(u1.nombre);

```

Por tanto, para obtener el objeto referenciado por una variable `REF`, debes **utilizar una consulta sobre cualquier tabla**, independientemente de la tabla en la que se encuentre el objeto referenciado. Sólo existe la condición de que siempre se obtenga una sola fila como resultado. Lo más **cómodo es utilizar esa tabla DUAL**. Aunque se use esa tabla comodín, **el resultado será un objeto** almacenado en la tabla `UsuariosObj`.

PostgreSQL también es un sistema de gestión de bases de datos objeto-relacional.