

Reforzamiento de contenido :  
"LENGUAJE DE CONSULTAS A UNA  
BASE DE DATOS"

# Unidad 1 Gestión de bases de datos

## Competencias

- Comprender la importancia del uso y potencialidades de la base de datos.
- Conocer qué es un modelo de datos y sus lenguajes.

## Bases de Datos.

En informática se conoce como base de datos a una colección organizada de información estructurada. A esta información se le llama datos. Regularmente los datos son almacenados electrónicamente y este es el concepto que vamos a estudiar.

Una base de datos es usualmente controlada por un DBMS (sistema de gestión de base de datos / **Data base manager system**).

Al conjunto de los datos o información, **DBMS** y otras herramientas que estén asociados a la manipulación de información, se les conoce como base de datos o por su abreviación **BD**.



imagen 1 "<https://learnsql.com/blog/what-is-dbms/>"

Fuente: <https://www.oracle.com/cl/database/what-is-database/#link1>

## Información y modelo de datos.

Dentro del contexto que estamos estudiando, información se conoce como el conjunto de datos que tengan un significado, de modo que reduzcan la incertidumbre y aumenten el conocimiento de quien necesite utilizarla. Estos datos deben estar siempre disponibles y servirán para clarificar incertidumbres sobre determinados temas.

## Lenguajes que utilizan los DBMS.

Una base de datos debe permitir representar información del mundo real, de modo que esta pueda ser interpretada y ordenada. Además, deben ser entendibles y tener relaciones claras y restricciones. Para proporcionar estas características los **DBMS** utilizan lenguajes internos, como **DDL**, **DML** o **DCL**.

Para especificar estructura y restricciones a los datos se utiliza un lenguaje de definición de datos (Data definition language - DDL por sus sigla en inglés).

Para especificar la manipulación de estos datos, se utiliza el lenguaje de manipulación de datos (Data manipulation language -DML por su sigla en inglés).

Para especificar control de acceso a estos datos, se utiliza el lenguaje de control DCL (Data control language -DCL por su sigla en inglés)

A pesar de estas definiciones de diferentes lenguajes, en este curso estudiaremos como **SQL** (lenguaje de consultas estructurado / Structured Query Language) abarca y posee comandos para efectuar estas tareas.

## SQL (lenguaje de consultas estructurado)

SQL es un lenguaje de consulta estructurado (en inglés, Structured Query Language). Nos permite realizar diferentes operaciones en los sistemas de gestión de bases de datos relacionales. Consiste en un lenguaje de definición de datos (DDL), un lenguaje de manipulación de datos (DML) y un lenguaje de control de datos (DCL).

SQL es un lenguaje de programación usado por casi todas las **bases de datos relacionales**.



## Características de los DBMS (Sistemas de gestión de base de datos)

Mediante el uso de sentencias DDL o DML, los usuarios de los sistemas de gestión de base de datos pueden definir estructuras, y restricciones, también recuperar, actualizar, insertar o borrar datos. Regularmente los usuarios no necesitan conocer el detalle de almacenamiento de una base de datos, solo requieren tener una vista abstracta de los datos.

Por este motivo, la arquitectura de un DBMS generalmente se basa en una arquitectura de tres niveles ( externo, conceptual e interno), con esto se trata de separar la forma en que los usuarios ven los datos, de los detalles de almacenamiento físico de los mismos.

Este principio de independencia de datos hace posible que el administrador de la base de datos, pueda cambiar la estructura física de la base de datos ( nivel interno), sin que la manera en la cual los diferentes usuarios ven los datos (nivel externo) se vea afectado.

## Funciones de un DBMS

### **Seguridad.**

Los datos almacenados en una base de datos pueden llegar a tener un gran valor. Un DBMS debe garantizar que los datos se encuentren seguros frente a usuarios malintencionados, que intenten leer datos privilegiados, o frente a ataques que deseen manipular o destruirlos. O simplemente ante errores de algún usuario no autorizado. Para asegurar esta función los sistemas **DBMS** tienen complejos sistemas de permisos a usuarios y grupos de usuarios.

### **Integridad.**

Adoptar las medidas necesarias para garantizar la validez de los datos almacenados, es decir proteger los datos ante fallos de hardware, datos mal ingresados o cualquier instancia capaz de corromper los datos que ahí estén almacenados. Los DBMS proveen mecanismos para garantizar la recuperación de la base de datos a un estado consistente conocido de forma automática.

### **Respaldo.**

Los DBMS deben proporcionar una forma eficiente para realizar copias de respaldo de los datos almacenados.

### **Control de concurrencia.**

En la mayoría de los entornos lo más habitual es que sean muchas personas las que accedan a una base de datos simultáneamente, un DBMS debe ser capaz de manejar ese acceso concurrente para evitar inconsistencias.

### **Manejo de transacciones.**

Una transacción es un pequeño programa que se ejecutará como una sola operación. Los DBMS proveen mecanismos para programar estas operaciones, de forma que si una de ellas falle, sea posible volver al estado anterior a su ejecución.

### **Tiempo de respuesta.**

Siempre es deseable que un DBMS tarde lo menos posible en darnos la información solicitada o almacenar cambios en nuestros datos.

## **Evolución de la base de datos**

Las bases de datos han evolucionado enormemente desde su inicio a principios de los años sesenta. Las bases de datos de navegación, como la base de datos jerárquica (que se basaba en un modelo similar a un árbol y solo permitía una relación de uno a muchos), y la base de datos de red (un modelo más flexible que permitía múltiples relaciones), eran los sistemas originales utilizados para almacenar y manipular los datos. Aunque simples, estos primeros sistemas eran inflexibles.

En la década de 1980, las bases de **datos relacionales** se hicieron populares, seguido de bases de datos orientadas a objetos en los años noventa. Más recientemente, surgieron las bases de **datos NoSQL** como respuesta al crecimiento de internet y la necesidad de una mayor velocidad y procesamiento de datos no estructurados.

Hoy, las bases de datos en la nube y las bases de datos independientes están abriendo nuevos caminos en cuanto a cómo se recopilan, almacenan, administran y utilizan los datos.

Fuente: <https://www.oracle.com/cl/database/what-is-database/#link3>

## Tipos de bases de datos

Hay muchos tipos de bases de datos. La mejor base de datos para una organización específica depende de cómo la organización pretende utilizar los datos. En este curso solo veremos bases de datos del tipo relacional, pero es bueno conocer los otros tipos existentes.

### **Bases de datos relacionales.**

Las bases de datos relacionales se popularizaron en los años ochenta. Los elementos de una base de datos relacional se organizan como un conjunto de **tablas** con columnas y filas. La tecnología de base de datos relacional proporciona la manera más eficiente y flexible de acceder a información estructurada.

### **Bases de datos orientadas a objetos.**

La información en una base de datos orientada a objetos se representa en forma de objetos, como en la programación orientada a objetos.

### **Bases de datos distribuidas.**

Una base de datos distribuida consta de dos o más archivos ubicados en diferentes sitios. La base de datos puede almacenarse en múltiples computadoras, ubicadas en la misma ubicación física o dispersas en diferentes redes.

### **Almacenes de datos.**

Un almacén de datos es un tipo de base de datos diseñada específicamente para consultas y análisis rápidos, y funciona como un depósito central de datos.

### **Bases de datos NoSQL.**

Una base de datos NoSQL, o una base de datos no relacional, permite que los datos no estructurados y semiestructurados se almacenen y manipulen, a diferencia de una base de datos relacional, que define cómo deben componerse todos los datos insertados en la base de datos. Las bases de datos NoSQL se hicieron populares a medida que las aplicaciones web se hacían más comunes y más complejas.

## Unidad 2 Instalación y configuración de Oracle SQL

### Competencias:

- Conocer las ventajas de utilizar Oracle SQL
- Instalar y configurar Oracle SQL en sistema operativo windows
- Instalar SQL Developer

### Introducción

En este plan formativo se utilizara el DBMS de la empresa ORACLE. Sin embargo en el mercado existen un sin número de gestores. Por ejemplo:

- [Db2](#) de IBM.
- [SQL Server](#) de Microsoft.
- [MySQL](#) de Oracle.
- [PostgreSQL](#).

Podrás encontrar sus características en los enlaces proporcionados.

[Oracle Database](#) es un sistema de gestión de base de datos de tipo relacional, desarrollado por Oracle Corporation. Esta herramienta permite analizar datos y efectuar recomendaciones para mejorar el rendimiento y la eficiencia en el manejo de aquellos datos que se encuentran almacenados.

Algunas de las características más relevantes de Oracle SQL son: Multiplataforma: Puede ejecutarse en varios sistemas operativos, incluidos Windows Server, Unix y varias distribuciones de GNU / Linux.

Es el motor de base de datos relacional más usado a nivel mundial. Permite el uso de particiones para hacer consultas, informes, análisis de datos, etc.

Algunas de las características más relevantes de Oracle SQL son:

- Multiplataforma: Puede ejecutarse en varios sistemas operativos, incluidos Windows Server, Unix y varias distribuciones de GNU / Linux.
- Es el motor de base de datos objeto-relacional más usado a nivel mundial.
- Permite el uso de particiones para hacer consultas, informes, análisis de datos, etc. Esto favorece su eficiencia.

Dentro de las desventajas podemos mencionar principalmente:

- El costo de la licencia en las versiones empresariales (EE, SE).
- Sus limitaciones en la edición express (XE).

## Instalar Oracle SQL



*Esta instalación considera que no tienes instalado Oracle SQL en tu sistema operativo.*

Especificaremos la instalación del gestor de la base de datos (**Oracle Database XE**) y la herramienta para interactuar con ella (**SQL developer**).

### Oracle Database XE

Ingresa al siguiente enlace y descarga Oracle Database para el sistema operativo que utilizas.

<https://www.oracle.com/database/technologies/xe-downloads.html>

Oracle Database XE Downloads	
Oracle Database Express Edition (XE) Release 18.4.0.0.0 (18c)	
Download	Description
 <a href="#">Oracle Database 18c Express Edition for Linux x64</a>	(2,521,766,408 bytes - February 20, 2020) [Sha256sum: 4df0318d72a0b97f5468b36919a23ec07533f5897b324843108e0376566d50c8]
 <a href="#">Oracle Database 18c Express Edition for Windows x64</a>	(2,054,264,100 bytes - February 20, 2019) [Sha256sum: 7C1A85D05F6FCC5BE70E7BA4017BEEC70C8D095E685031695B8486C40C114F1A]

Aceptar los términos para descargar el software.

×


You must accept the [Oracle License Agreement](#) to download this software.

☒

I reviewed and accept the Oracle License Agreement

Required

You will be redirected to the login screen in order to download the file.

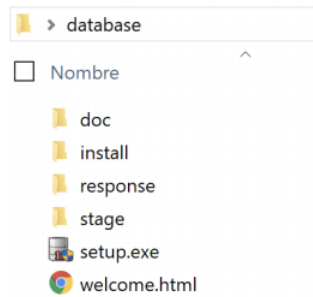
Download OracleXE184\_Win64.zip 

En algunas ocasiones la página web de oracle nos solicitara registrarnos para poder descargar el software, el registro es gratuito.

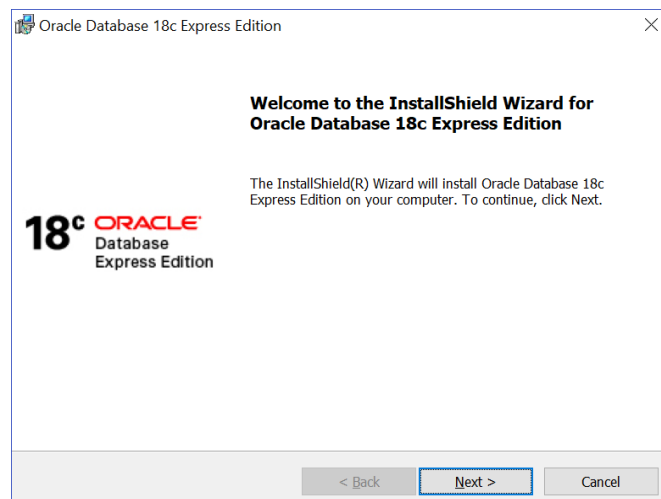
Este es el [enlace](#) hacia la guía oficial de instalación.

## Instalación en windows.

1. Extraer el contenido del archivo descargado en una carpeta de tu sistema.

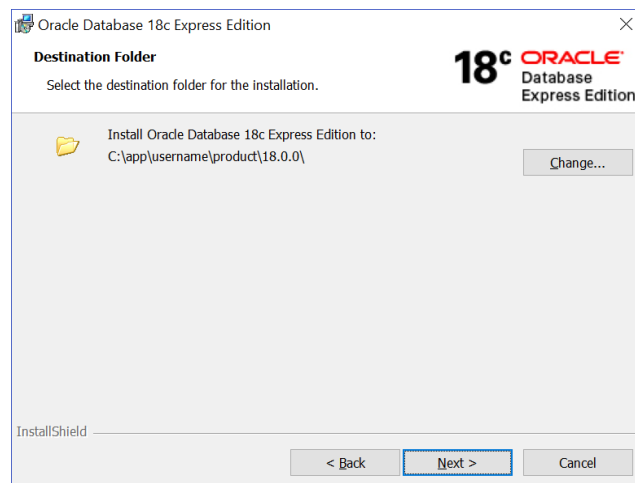


2. Iniciar el proceso haciendo doble clic en el archivo setup.exe.

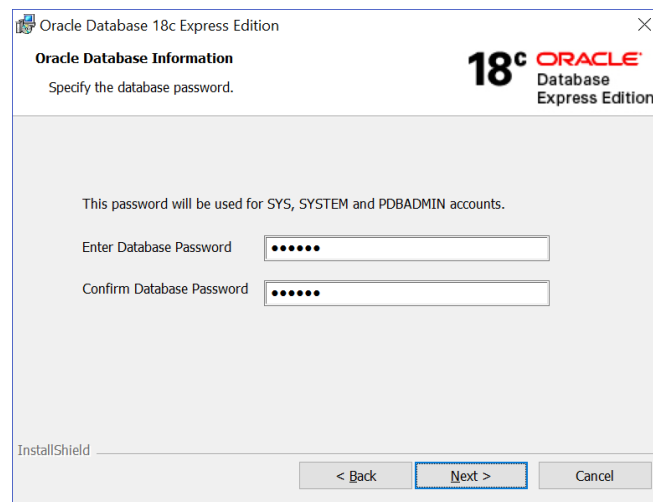




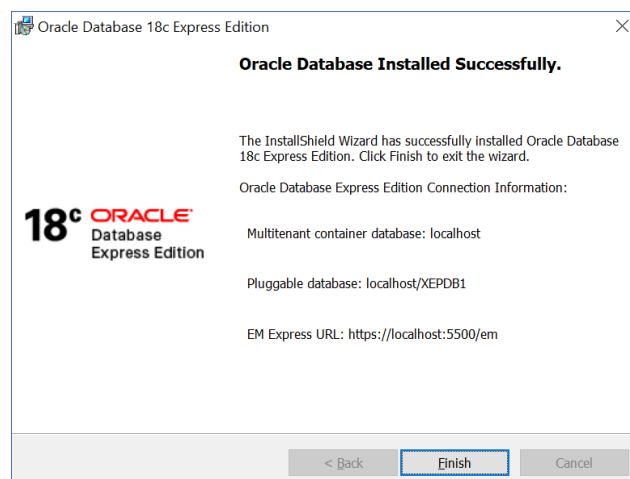
3. Seleccionar la carpeta de destino.



4. Indica la contraseña general para la administración de la base de datos. Recuerda tomar nota de ella.







5. Finalizar la instalación.



## Oracle SQL Developer.

Descargar desde este [enlace](#) el instalador dependiendo de tu sistema operativo.

Platform	Download	Notes
Windows 64-bit with JDK 8 included	 <a href="#">Download</a> (426 MB)	<ul style="list-style-type: none"><li>• MD5: a3e084788d67a7138d1bc2890e7131ec</li><li>• SHA1: 877640b0414e40f2e084c1a86e747e2d23d6ecd</li><li>• <a href="#">Installation Notes</a></li></ul>
Windows 32-bit/64-bit	 <a href="#">Download</a> (437 MB)	<ul style="list-style-type: none"><li>• MD5: 270ba91438750d85faa0ab0a1c8481b9</li><li>• SHA1: b18cfb3ad26fbce1e6e6e454a83d741cf996c41</li><li>• <a href="#">Installation Notes</a></li><li>• JDK 8 or 11 required</li></ul>
Mac OSX	 <a href="#">Download</a> (351 MB)	<ul style="list-style-type: none"><li>• MD5: 81b571468b82cb0fa4084af8388defc8</li><li>• SHA1: f02c55d4d69f73f8b52e7771a7feb298b9b15412</li><li>• <a href="#">Installation Notes</a></li><li>• JDK 8 or 11 required</li></ul>
Linux RPM	 <a href="#">Download</a> (338 MB)	<ul style="list-style-type: none"><li>• MD5: a998ae16b818d066180f2a48720f75c0</li><li>• SHA1: deaae056714266356de53e10fc7441f2bfe73d64</li><li>• <a href="#">Installation Notes</a></li><li>• JDK 8 or 11 required</li></ul>

Se recomienda utilizar la versión con JDK 8 incluido.

***JDK es una suite de desarrollo que incluye librerías y archivos necesarios del gestor de base de datos.***

### Instalación en windows.

1. En windows solo debes descomprimir el contenido del archivo .zip descargado.

[Enlace a la guía oficial.](#)

## Unidad 3 Estructuras de una Base de datos relacional.

### Competencias

- Conocer el concepto de tabla en una base de datos.
- Conocer cómo se relacionan las tablas mediante claves primarias y foráneas.
- Conocer los tipos de datos que se pueden utilizar.

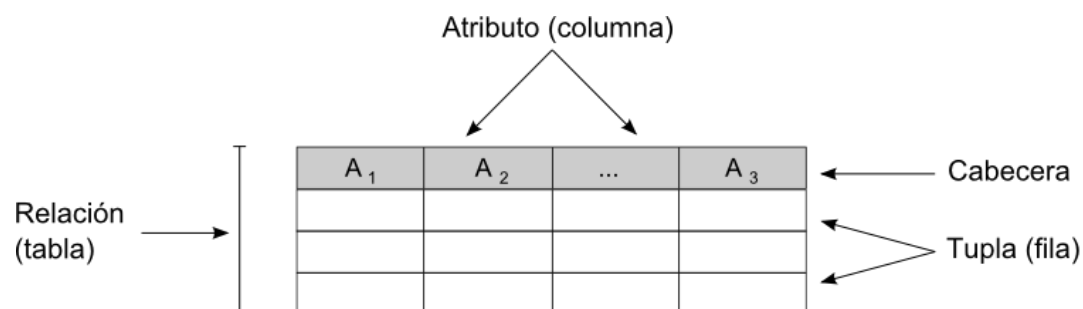
### Introducción

Las bases de datos Relacionales son aquellas compuestas por varias tablas donde se almacena la información y posteriormente se relacionan entre sí.

Un aspecto relevante a tomar en cuenta es que para implementar bases de datos relacionales, es necesario tener conocimiento previo sobre qué es lo que vamos a almacenar.

### Tablas

Las bases de datos relacionales, se componen de tablas con información, podemos hacer una analogía de estas tablas, como una especie de planilla de cálculo.



Vamos a simular que necesitamos guardar información en una tabla de estas características. Para ello simularemos las columnas que tendría una tabla de una agenda telefónica.

En esta agenda vamos a guardar el nombre, apellido, número telefónico, dirección y edad de una serie de individuos.

Resulta que el registro en sí será la tabla **directorio\_telefónico**, donde tendremos los campos **nombre**, **apellido**, **número\_telefónico**, **dirección** y **edad**.

Para empezar a crear nuestra base de datos utilizaremos tablas, donde alojaremos esta información.

Una base de datos se compone de múltiples tablas. Cada una de éstas presentarán dos dimensiones: filas, que representan a los registros en la tabla columnas, que van a representar los atributos ingresados en cada registro, definiendo el tipo de dato a ingresar.

directorio_telefonico
nombre
apellido
numero_telefonico
dirección
edad

## Claves primarias y foráneas

De manera adicional a las filas y columnas, las tablas también cuentan con claves primarias y foráneas. Estas buscan generar identificadores para cada registro de estas tablas mediante algún valor específico de una columna o atributo.

**Clave primaria (primary key)** Cuando hacemos referencia a esta columna dentro de su tabla de origen, hablaremos de una clave primaria. Esta clave siempre será de carácter único.

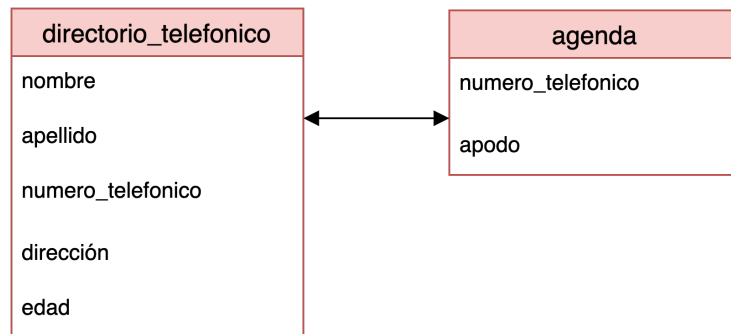
**Clave foránea (foreign key)** Cuando hacemos referencia a una columna identificadora en otra tabla a la cual hacemos referencia, hablamos de una clave foránea.

## Ejemplo claves primarias y foráneas

Supongamos que en base a nuestra tabla **directorio\_telefónico** , deseamos incorporar información de otra tabla llamada **agenda** que tiene las columnas **apodo** y **número\_telefónico**.

Ante esta situación, vamos a identificar que la columna **número\_telefónico** en la tabla **directorio\_telefónico** será la clave primaria y la columna **número\_telefónico** en la tabla **agenda** corresponderá a la clave foránea.

Este comportamiento es posible dado que el número registrado será congruente en ambas tablas.



## Tipos de Datos

Una de las principales características del trabajo con bases de datos, es la necesidad de declarar los distintos tipos de datos existentes en cada campo a completar. Estos aplican restricciones sobre lo que puede ingresar a los registros. No podemos ingresar caracteres cuando piden un número, ni sobrepasar el límite de caracteres posibles.

Los tipos de datos más comunes son:

**INT** : Números enteros de 4 bytes que pueden tomar valor desde -2147483648 hasta +2147483647.

**NUMBER (p, s)** : Números decimales de 22 bytes que pueden tener una precisión (p) entre 1 y 38 y una escala (s) entre -84 y 127.

**CHAR(s)** : Cadena de hasta 2000 bytes de longitud fija.

**VARCHAR(s)** : Cadena de hasta 4000 bytes de longitud variable. A diferencia de CHAR, si no se ocupa toda la memoria, esta queda libre.

**CHAR** ocupará toda la memoria solicitada.

**DATE** : Almacena fecha según formato definido en los parámetros de la base de datos. Permite un rango entre el 01-01-4172 AC hasta 31-12-9999 DC.

**TIMESTAMP** : Almacena fecha y hora juntos: yyyy-mm-dd hh:mm:ss

Para consultar en detalle los tipos de datos, podemos ir a la documentación oficial de Oracle.

Fuente:

<https://docs.oracle.com/en/database/oracle/oracle-database/21/sqlrf/Data-Types.html#GUID-A3C0D836-BADB-44E5-A5D4-265BA5968483>

# Unidad 4 Utilizando una base de datos

## Competencias

- Construir una base de datos.
- Crear tablas con sus atributos y tipos de datos correspondientes.
- Crear claves primarias y foráneas.
- Crear relaciones entre tablas.

## Introducción

Hasta el momento hemos instalado, configurado y aprendido ciertas características de Oracle SQL, ya es momento que empecemos a crear nuestra primera base de datos con las primeras tablas.

En esta ocasión vamos a realizar procesos como crear, insertar, actualizar y eliminar. Estos responden a las operaciones elementales que podemos realizar en una tabla.

Realizaremos estos procesos a través del entorno SQL Developer, que nos facilitará de forma visual la creación y administración de nuestros datos.

## Creación de tablas

La primera tarea que debemos realizar es la creación de la tabla que permitirá almacenar la información en filas y columnas. Para crear una tabla dentro de nuestro motor, debemos utilizar el comando SQL **CREATE TABLE** acompañado del nombre de la tabla y declarar los atributos con su respectivo tipo de dato, ingresándolos entre paréntesis. La forma canónica de creación es la siguiente:

```
CREATE TABLE nombre_tabla( columna1 tipo_de_dato1, columna2 tipo_de_dato2, columna3 tipo_de_dato3 );
```

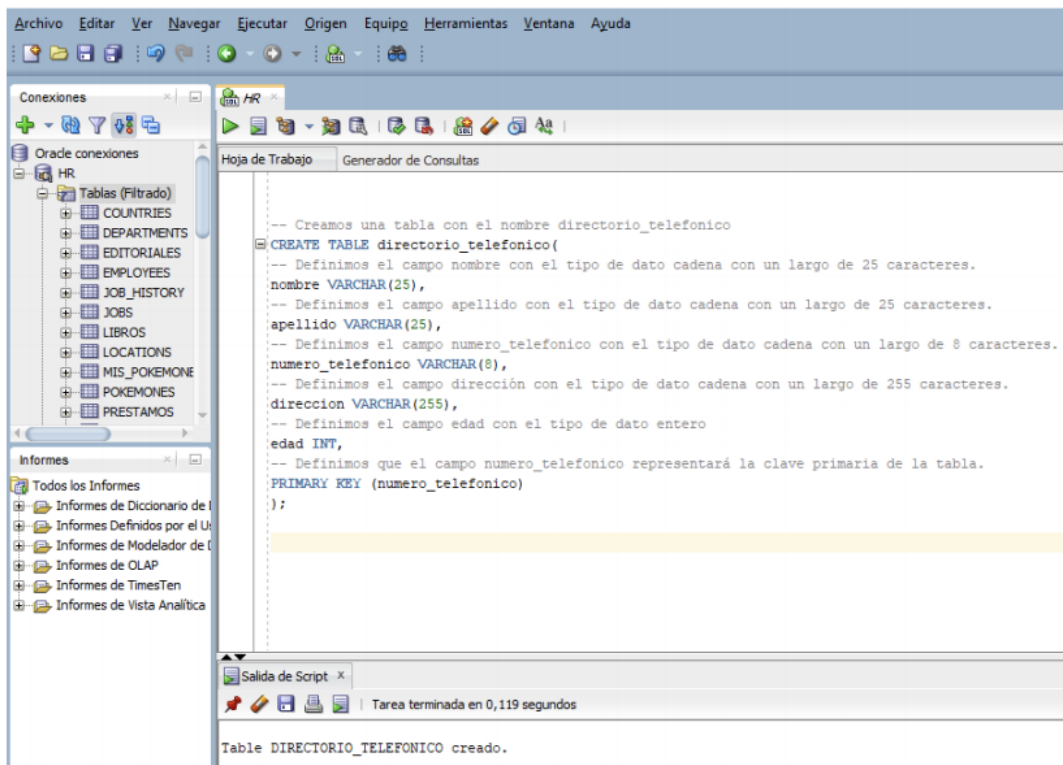
Volviendo a nuestro ejemplo, vamos a crear la tabla directorio\_telefónico con los campos nombre , apellido , número\_telefónico , dirección y edad.

## Comentarios

Las líneas precedidas por -- representan comentarios específicos sobre cada línea del código y no serán interpretados como una instrucción.

```
-- Creamos una tabla con el nombre directorio telefónico  
CREATE TABLE Directorio_telefonico(
```

-- Definimos el campo nombre con el tipo de dato cadena con un largo de 25 caracteres.  
nombre **VARCHAR(25)**,  
-- Definimos el campo apellido con el tipo de dato cadena con un largo de 25 caracteres.  
apellido **VARCHAR(25)**,  
-- Definimos el campo número telefónico con el tipo de dato cadena con un largo de 8 caracteres.  
numero\_telefonico **VARCHAR(8)**,  
-- Definimos el campo dirección con el tipo de dato cadena con un largo de 255 caracteres.  
direccion **VARCHAR(255)**,  
-- Definimos el campo edad con el tipo de dato entero  
edad **INT**,  
-- Definimos que el campo número telefónico representará la clave primaria de la tabla.  
**PRIMARY KEY** (numero\_telefonico)  
);



Para poder ver la estructura que tiene nuestra tabla, podemos usar el comando.

```
SELECT * FROM directorio_telefonico;
```

Deberíamos ver la siguiente estructura.

nombre	apellido	numero_telefonico	direccion	edad

## Creando una tabla con clave foráneas

Posterior a la creación de ésta, definiremos los componentes de la segunda tabla agenda con el campo numero\_telefonico y asignaremos una clave foránea proveniente de la tabla directorio\_telefonico.

Profundicemos sobre la última instrucción. La forma canónica de implementar una clave foránea responde a los siguientes componentes:

```
-- Creamos una tabla con el nombre agenda
CREATE TABLE Agenda(
-- Definimos el campo nick con el tipo de dato cadena con un largo de 25 caracteres
nick VARCHAR(25),
-- Definimos el campo numero_telefonico con el tipo de dato cadena con un largo de 8
caracteres.
numero_telefonico VARCHAR(8),
-- Vinculamos una clave foránea entre nuestra columna numero_telefonico y su similar en la
tabla directorio_telefonico FOREIGN KEY (numero_telefonico)
REFERENCES Directorio_telefonico(numero_telefonico)
);
```

De manera similar, nuestra tabla Agenda se ve de la siguiente manera, sin registros de momento.

nick	numero_telefonico



## Sentencias para la Manipulación de datos

Un lenguaje de manipulación de datos (Data Manipulation Language, o DML en inglés) es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado. El lenguaje de manipulación de datos más popular hoy día es SQL, usado para recuperar y manipular datos en una base de datos relacional.

Comando	Descripción
Select	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado
Insert	Utilizado para cargar lotes de datos en la base de datos en una única operación.
Update	Utilizado para modificar los valores de los campos y registros especificados Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.
Delete	Utilizado para eliminar registros de una tabla

## INSERT

Una sentencia INSERT de SQL agrega uno o más registros a una (y sólo una) tabla en una base de datos relacional.

Forma básica:

```
# INSERT INTO "tabla" ("columna1", ["columna2,..."]) VALUES ("valor1", ["valor2",])
```

Las cantidades de columnas y valores deben ser iguales. Si una columna no se especifica, le será asignado el valor por omisión. Los valores especificados (o implícitos) por la sentencia INSERT deberán satisfacer todas las restricciones aplicables. Si ocurre un error de sintaxis o si alguna de las restricciones es violada, no se agrega la fila y se devuelve un error.

Ejemplo:

```
# INSERT INTO agenda_telefonica (nombre, numero) VALUES ('Roberto Jeldrez', 4886850);
```

Cuando se especifican todos los valores de una tabla, se puede utilizar la sentencia acortada:

```
# INSERT INTO "VALUES" ("valor1", ["valor2",])
```

## UPDATE

Una sentencia UPDATE de SQL es utilizada para modificar los valores de un conjunto de registros existentes en una tabla.

Ejemplo:

```
# UPDATE mi_tabla SET campo1 = 'nuevo valor campo1' WHERE campo2 = 'N';
```

## DELETE

Una sentencia DELETE de SQL borra uno o más registros existentes en una tabla. Forma básica:

```
# DELETE FROM 'tabla' WHERE 'columna1' = 'valor1'
```

Ejemplo:

```
# DELETE FROM My_table WHERE field2 = 'N';
```

Si bien el comando DELETE se debe usar con mucha precaución, es más, sólo el administrador de la base de datos debería ser capaz de eliminar datos de ella. Este comando es susceptible a errores, ya que si no implementamos correctamente la condición en el comando WHERE, podemos eliminar datos que no teníamos pensado borrar y no hay posibilidad de recuperarlos.

## Añadiendo o eliminado columnas

Utilizando la definición anterior de nuestra tabla Agenda , se busca agregar ahora una nota. Ante la eventualidad que deseemos añadir/remover una columna específica de una tabla, podemos implementar la siguiente sintaxis:

```
ALTER TABLE nombre_tabla ADD nueva_columna tipo_de_dato;
```

### Ejemplo

```
-- Declaramos que alteramos la tabla Agenda
ALTER TABLE Agenda
-- Definimos el campo nota con el tipo de dato cadena con un largo de 100 caracteres.
ADD nota VARCHAR(100);
```

Si deseamos eliminar alguna columna en específico, podemos implementar la instrucción DROP de manera análoga a como lo hicimos con ADD.

```
ALTER TABLE nombre_tabla DROP nombre_columna;
```

## Restricciones

Puede que existan casos en los que nuestras columnas necesitan reglas que cumplir, como que no haya valores repetidos o que no existan campos vacíos. Es por eso que aparece el concepto de restricciones. en este listado se muestran las principales con su respectiva definición:

**NOT NULL:** La columna no puede tener valores NULL( nulos).

**UNIQUE:** Todos los valores de la columna deben ser diferentes unos a otros.

**PRIMARY KEY:** Aplica la clave primaria.

**FOREIGN KEY:** Aplica la clave foránea.

**CHECK:** Todos los valores de una columna deben satisfacer una condición en específico.

**DEFAULT:** Le da un valor por defecto a aquellos registros que no tengan un valor asignado.

**INDEX:** Sirve para crear y recuperar datos de forma rápida.

Estos se aplican de la siguiente forma:

```
-- Creamos una tabla
CREATE TABLE nombre_tabla(
-- Declaramos una serie de restricciones a cada campo de dato creado columna1
tipo_de_dato1 restriccion,
columna2 tipo_de_dato2 restriccion,
columna3 tipo_de_dato3 restriccion
);
```

PRIMARY KEY y FOREIGN KEY se manejan de forma distinta, los explicaremos en la siguiente sección. Usaremos de ejemplo las mismas tablas que creamos antes, asignándoles restricciones:

Volveremos a crear las tablas Directorio\_telefónico y Agenda :

```
CREATE TABLE Directorio_telefonico(
nombre VARCHAR(25) NOT NULL,
apellido VARCHAR(25),
numero_telefonico VARCHAR(8) UNIQUE,
direccion VARCHAR(255),
edad INT );
```

```
CREATE TABLE Agenda(
nick VARCHAR(25) NOT NULL,
numero_telefonico VARCHAR(8) UNIQUE
);
```

En este caso, hicimos que número\_telefónico sea un valor único que no pueda repetirse, que nombre y nick no puedan tener valores NULL.

## Restricciones a nivel de PRIMARY KEY y FOREIGN KEY

Como mencionamos antes, las claves primarias sirven para identificar un registro único en una tabla, y la clave foránea sirve para referenciar esa columna desde otra tabla. La forma de aplicarlas es la siguiente:

```
CREATE TABLE tabla1(  
columna1 tipo_de_dato1,  
columna2 tipo_de_dato2,  
columna3 tipo_de_dato3,  
PRIMARY KEY (columna1) );  
  
CREATE TABLE tabla2(  
columna4 tipo_de_dato4,  
columna5 tipo_de_dato5,  
FOREIGN KEY (columna4) REFERENCES tabla1(columna1)  
);
```

Volvamos al ejemplo de Directorio\_telefonico y Agenda , ambos usan los mismos números telefónicos para referirse a las mismas personas. La diferencia aquí, es que obtenemos los números desde Directorio\_telefonico para llevarlos a Agenda .

Es por esto, que numero\_telefonico será clave primaria (PRIMARY KEY) en Directorio\_telefonico y clave foránea (FOREIGN KEY) en Agenda . Esto se puede ver reflejado así:

```
CREATE TABLE Directorio_telefonico(  
nombre VARCHAR(25),  
apellido VARCHAR(25),  
numero_telefonico VARCHAR(8),  
direccion VARCHAR(255),  
edad INT, PRIMARY KEY (numero_telefonico)  
);
```

```
CREATE TABLE Agenda(  
nick VARCHAR(25),  
numero_telefonico VARCHAR(8),  
nota VARCHAR(100),  
FOREIGN KEY (numero_telefonico) REFERENCES  
Directorio_telefonico(numero_telefonico)  
);
```

## Unidad 5 Agrupaciones y consultas

### Competencias

- Realizar agrupaciones en consultas
- Realizar consultas a una base de datos.

### Consultas de selección con funciones de agrupación

Es conocido cómo hacer cálculos con los datos de una consulta, e incluso como utilizar funciones en esos cálculos. Pero hasta ahora las funciones utilizadas solo podían usar información procedente de datos de la misma fila. Es decir, no se podían sumar, por ejemplo, datos procedentes de distintas filas. Sin embargo, hacer cálculos con datos de diferentes filas es una necesidad muy habitual.

Las funciones de cálculo con grupos son las encargadas de realizar cálculos en vertical (usando datos de diferentes filas) en lugar de en horizontal (usando datos procedentes de la misma fila en la que vemos el resultado).

Las funciones de cálculos de totales para datos agrupados son las siguientes:

Función	Significado
<b>COUNT</b> (expresión)	Cuenta los elementos de un grupo. Se suele indicar un asterisco ( <b>COUNT(*)</b> ) en lugar de una expresión, ya que la cuenta no varía por indicar una expresión concreta; el resultado siempre es el número de elementos del grupo. Hay que tener en cuenta que esta función ignora los valores nulos a la hora de contar, por lo que la expresión <i>COUNT(telefono)</i> cuenta la cantidad de teléfonos que hay (ignorando los nulos).
<b>SUM</b> (expresión)	Suma los valores de la expresión
<b>AVG</b> (expresión)	Calcula la media aritmética sobre la expresión indicada
<b>MIN</b> (expresión)	Mínimo valor que toma la expresión indicada
<b>MAX</b> (expresión)	Máximo valor que toma la expresión indicada
<b>STDDEV</b> (expresión)	Calcula la desviación estándar
<b>VARIANCE</b> (expresión)	Calcula la varianza

Todas ellas requieren trabajar con grupos (más adelante se explica cómo agrupar filas), si no se indican grupos, las funciones trabajan sobre todos los datos de una tabla.

Por ejemplo si se tiene esta tabla:

Cod_trabajador	Nombre	Salario
1	Pedro	1200
2	Ana	1500
3	Raquel	1650
4	Sebastián	980
5	Marcos	1200
6	Mónica	1650
7	Raúl	1200
8	Mireia	980
9	Gorka	1650
10	Maia	1650

Suponiendo que son los datos de la tabla de nombre “trabajadores”, la consulta:

```
SELECT MAX(salario) FROM trabajadores;
```

devuelve el valor 1650, que es el máximo salario. Mientras que la consulta:

```
SELECT COUNT(*) FROM trabajadores;
```

devolvería el valor 10, puesto que hay 10 trabajadores en la tabla.

### Manejo de los valores nulos

Las funciones de cálculo agrupado ignoran los valores NULL. Si no se desea ignorarlos, se deben utilizar funciones de nulos como COALESCE, NVL o NVL2 para manejar apropiadamente el valor nulo.

Sin embargo, la función COUNT, se suele usar con un asterisco (\*) como objeto de cálculo, lo que hace que cuente filas independientemente de su contenido. Si usamos una expresión, como COUNT(salario), no contaría las filas que tengan un salario nulo en ellas.

## Uso de DISTINCT en las funciones de totales

Las funciones anteriores admiten que se anteponga el término DISTINCT a la expresión. De esa forma solo se tienen en cuenta los valores distintos. Por ejemplo en:

```
SELECT COUNT(DISTINCT salario);
```

El resultado es 4, dado que sólo hay 4 salarios distintos. Lo mismo ocurre con el resto de funciones, ya que DISTINCT hace que se ignoren los valores repetidos.

## Agrupaciones

Lo normal es utilizar las funciones anteriores, no para una tabla completa sino para grupos de filas en base a un criterio. Esta técnica es la que se conoce como agrupación de filas y provoca que, de cada grupo, se muestre una sola fila.

Para ello se utiliza la cláusula **GROUP BY** que permite indicar en base a qué registros se realiza la agrupación.

En el apartado GROUP BY, se indica el nombre de las columnas (o expresiones más complejas) por las que se agrupa. La función de este apartado es crear una única fila por cada valor distinto en las columnas del grupo. Si por ejemplo se hace una agrupación en base a las columnas tipo y modelo en una tabla de existencias, se creará un único registro por cada tipo y modelo distintos:

Si la tabla de existencias sin agrupar es:



Tipo	Modelo	N_Almacen	Cantidad
AR	6	1	2500
AR	6	2	5600
AR	6	3	2430
AR	9	1	250
AR	9	2	4000
AR	9	3	678
AR	15	1	5667
AR	20	3	43
BI	10	2	340
BI	10	3	23
BI	38	1	1100
BI	38	2	540
BI	38	3	700

Si ahora se ejecuta la siguiente instrucción:

```
SELECT tipo,modelo
FROM existencias
GROUP BY tipo,modelo;
```

Se obtendrá este resultado:

Tipo	Modelo
AR	6
AR	9
AR	15
AR	20
BI	10
BI	38

Los datos se resumen. Por cada tipo y modelo distintos se creará un grupo de modo que solo aparecerá una fila por cada grupo. Los datos n\_almacen y cantidad no están disponibles ya que varían en cada grupo. Solo se podrá mostrar los datos agrupados o datos sobre los que se realicen cálculos.

Es decir, esta consulta es errónea:

```
SELECT tipo,modelo, cantidad
FROM existencias
GROUP BY tipo,modelo;
SELECT tipo,modelo, cantidad
```

\*

**ERROR** en línea 1:

ORA-00979: **no** es una expresión **GROUP BY**

La razón es el uso de la columna cantidad en el SELECT; puesto que no se ha agrupado por ella y no es un cálculo sobre tipo y/o modelo (las columnas por la que se está agrupando), se produce el error.

### Uso de funciones de totales con grupos

Lo normal es agrupar para obtener cálculos sobre cada grupo. Por ejemplo es posible modificar la consulta anterior de esta forma:

```
SELECT tipo,modelo, cantidad, SUM(Cantidad)
FROM existencias
GROUP BY tipo,modelo;
```

y se obtiene este resultado:

Tipo	Modelo	SUM(Cantidad)
AR	6	10530
AR	9	4928
AR	15	5667
AR	20	43
BI	10	363
BI	38	1740

Se suman las cantidades para cada grupo. Igualmente se podría realizar cualquier otro tipo de cálculo (AVG, COUNT, etc.).

## Condiciones HAVING

A veces se desea restringir el resultado de una expresión agrupada; por ejemplo se podría tener esta idea:

```
SELECT tipo,modelo, cantidad, SUM(Cantidad)
FROM existencias
WHERE SUM(Cantidad)>500
GROUP BY tipo,modelo;
```

Lo que se espera es que solo aparezcan los grupos cuya suma de la cantidad sea menor de 500. Pero, en su lugar, Oracle devolvería este error:

```
WHERE SUM(Cantidad)>500
      *
```

```
ERROR en línea 3:
ORA-00934: función de grupo no permitida aquí
```

La razón reside en el orden en el que se ejecutan las cláusulas de la instrucción SELECT. Oracle calcula primero el WHERE y luego los grupos (cláusula GROUP BY); por lo que no se puede usar en el WHERE la función de cálculo de grupos SUM, porque es imposible en ese momento conocer el resultado de la SUMA al no haberse establecido aún los grupos.

La solución es utilizar otra cláusula: HAVING, que se ejecuta una vez realizados los grupos. Es decir, si se quiere ejecutar condiciones sobre las funciones de totales, se debe hacer en la cláusula HAVING.

La consulta anterior quedaría:

```
SELECT tipo,modelo, cantidad, SUM(Cantidad)
FROM existencias
GROUP BY tipo,modelo
HAVING SUM(Cantidad)>500;
```

Eso no implica que no se pueda usar WHERE. Ésta expresión sí es válida:

```
SELECT tipo,modelo, cantidad, SUM(Cantidad)
FROM existencias
WHERE tipo!='AR'
GROUP BY tipo,modelo
HAVING SUM(Cantidad)>500;
```

En definitiva, el orden de ejecución de la consulta marca lo que se puede utilizar con WHERE y lo que se puede utilizar con HAVING.

### Orden de ejecución de instrucción SELECT

Para evitar problemas conviene conocer la forma de trabajar de una instrucción SELECT. Estos son los pasos en la ejecución de una instrucción de agrupación por parte del gestor de bases de datos:

1. Seleccionar las filas deseadas utilizando WHERE. Esta cláusula eliminará columnas en base a la condición indicada.
2. Se establecen los grupos indicados en la cláusula GROUP BY.
3. Se calculan los valores de las funciones de totales (COUNT, SUM, AVG, etc.).
4. Se filtran los registros que cumplen la cláusula HAVING.
5. El resultado se ordena en base al apartado ORDER BY.

### Funciones de agrupación anidadas

Es posible obtener resultados muy interesantes gracias a la posibilidad de anidar funciones de totales. Así se puede ejecutar esta instrucción:

```
SELECT AVG(SUM(Cantidad))
FROM existencias
GROUP BY tipo, modelo;
```

Para ello se debe tener en cuenta que la consulta funciona así:

1. Se calcula la suma de la cantidad para cada grupo, el resultado es una cantidad por cada tipo y modelo distintos en la tabla.

2. A partir de ese resultado se calcula la media de las cantidades obtenidas en el punto anterior.

Por lo tanto el resultado es la media de la suma de cantidades de cada grupo.

## Unidad 6 Modelos de datos y relaciones

### Competencias

- Reconocer distintos modelos de datos.
- Conocer el modelo entidad relación.
- Describir y utilizar relaciones entre tablas.
- Asociaciones y queries anidadas.

### Qué es un modelo de datos y cómo leerlo

Un modelo de datos es un conjunto de herramientas conceptuales para la descripción de los datos y las relaciones entre ellos, su semántica y las restricciones de consistencia.

De aquí en más se utilizará el modelo de datos relacional, esto dado que su simplicidad facilita el trabajo del programador en comparación con otros modelos.

Sin embargo, primero se desarrollará un modelo basado en Entidad-Relación el cual después será transformado finalmente en el Relacional, ya que E-R sirve para modelar el problema y reglas de negocio.

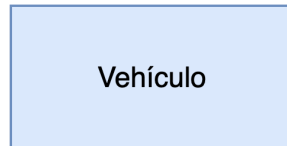
También a continuación se muestran algunos conceptos importantes relacionados con el diseño del modelo de bases de datos y también con respecto a integridad en las relaciones y base de datos antes de iniciar con el diseño del modelo.

### Modelo Entidad-Relación a Relacional

Este modelo es solamente un método que se aprovecha para diseñar los esquemas que posteriormente se deben implementar en la Base de datos. Este modelo se representa a través de diagramas y está formado por varios elementos que se analizarán a continuación.

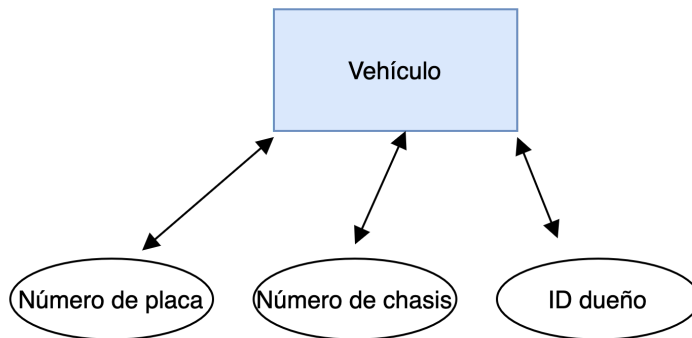
Este modelo además de tener un diagrama que ayuda a entender los datos y como se relacionan entre ellos, tiene que ser completado con la lista de los atributos y relaciones de cada elemento.

**Entidad:** Cada entidad representa cosas y objetos ya sean reales o abstractos que se diferencian entre sí. En un diagrama las entidades se representan con rectángulos.



Entidad Vehículo.

**Atributos:** Los atributos definen las características de las entidades, son las propiedades de cada una. Cada entidad contiene distintos atributos, que dan información sobre esta entidad. Estos atributos pueden ser de distintos tipos (numéricos, texto, fecha...). En un diagrama los atributos se representan con círculos que descienden de la entidad.



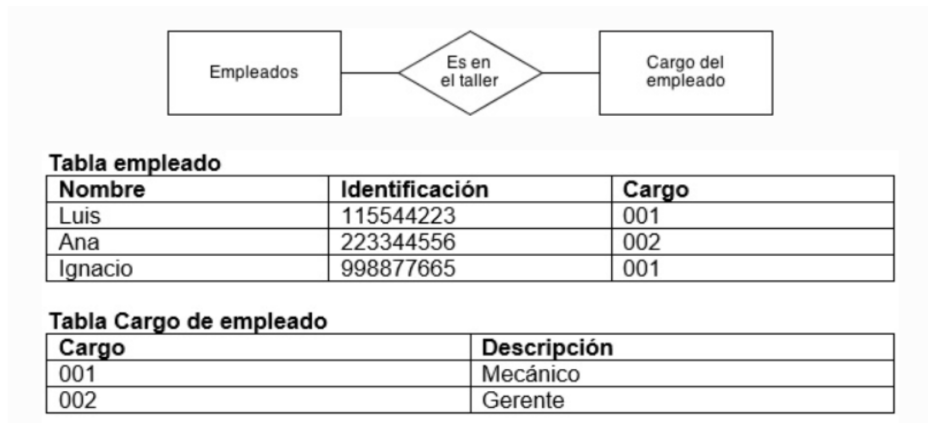
**Tupla:** La tupla son elementos formados por una fila de una tabla. Cada fila de la entidad estaría compuesta por los atributos de la entidad correspondiente.

### El concepto de relación

Una relación es lo que permite definir qué tipo de dependencia se debe dar entre entidades, es decir, permite decir que ciertas entidades comparten ciertos atributos de manera necesaria.

Para manejarlo de una manera más gráfica, se puede ver en un diagrama y en una tabla, los diagramas siempre se van a definir según las reglas del negocio.

**Ejemplo:** los empleados de un taller tienen un cargo asociado. Es decir, si un atributo de la entidad "Empleados" especifica qué cargo tiene en el taller, el cargo que ejerce debe coincidir con el que existe en la tabla de "cargos".



## Llaves en las relaciones

Una llave es un atributo de una entidad, la cual hace distinguir a esta entidad de los demás registros. En primer lugar, es necesario saber cuáles son los tipos de llaves que existen en una entidad y cómo se pueden relacionar con otras entidades.

Una forma de representar en diagramas o textos la llave primaria es poniendo “#” como prefijo de la llave primaria (PK), o subrayando el nombre de la llave primaria

### Ejemplo:

Estudiante (#ID, Nombre, Apellido) Estudiante (ID, Nombre, Apellido)

**Llave primaria:** Es un atributo (o columna) que restringe y distingue a las tuplas para que no se repitan en la misma entidad. La clave primaria es única. En algunos casos puede ser dos atributos lo cual conlleva al siguiente concepto.

**Llave primaria compuesta:** Como su nombre lo dice, es una llave primaria compuesta por varios atributos de una tabla; generalmente una llave primaria compuesta está formada por dos llaves foráneas de la tabla.

**Llave foránea:** es el atributo de una entidad que existe como dependencia en otra entidad, cuyos valores en las tuplas deben coincidir con valores de una llave que debe ser primaria de las tuplas de otra relación.

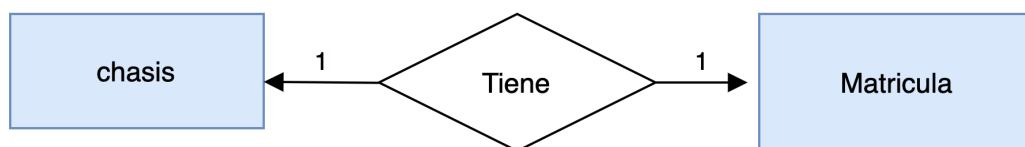


## Tipos de relaciones

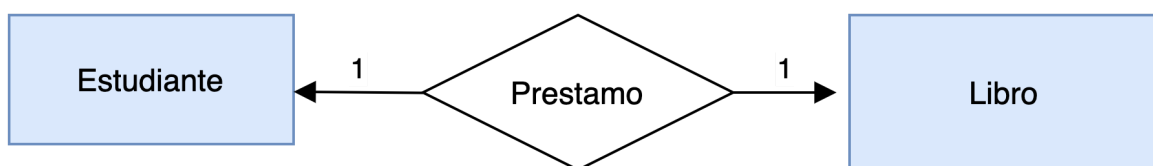
Existen diferentes tipos de relaciones según tengan parte las entidades en ellas. Las relaciones están dadas por el enunciado, como en el ejemplo anterior se puede analizar que cada empleado puede tener solamente un cargo, pero varios empleados pueden tener el mismo cargo.

La cardinalidad se mide con un mínimo y un máximo al lado de la entidad, en ese orden respectivamente (min,max).

**Relación uno a uno:** una entidad se relaciona únicamente con otra entidad y viceversa. Por ejemplo, si se tiene una entidad chasis y otra con matrículas del chasis del auto, se debe determinar que cada chasis solo puede tener una matrícula y una matrícula solo puede existir en un chasis específico.



**Relación uno a varios:** una entidad puede estar asociada con varias entidades, pero en esta segunda entidad, la primera puede existir solo una vez. Un ejemplo concreto sería:



Las reglas de negocio se leerían como: A un estudiante se le pueden prestar varios libros. Varios libros pueden ser prestados únicamente a un estudiante.

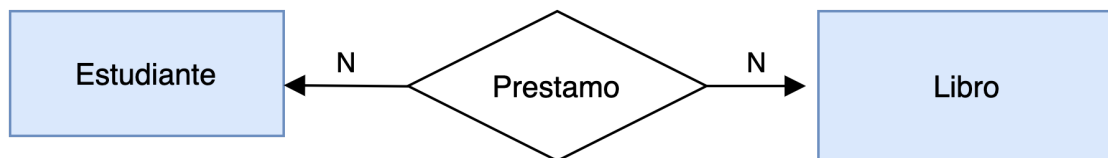
Al pasar de una relación de modelo entidad-relación a modelo relacional, se toma como llave primaria de la relación, la llave primaria de la relación de cardinalidad N.

Si se tiene la tabla Estudiante(ID, Nombre, Apellido) y Libro(Código, Nombre, Autor),

la llave primaria de la relación va a ser Código de libro.

### Relación varios a varios:

una entidad puede estar asociada con otra con ninguno o varias entidades y viceversa.



En este caso se pueden leer las reglas de negocio como: A varios estudiantes se les puede prestar varios libros. Varios libros pueden ser prestados a varios estudiantes.

Al pasar este modelo entidad-relación a las bases de datos relacionales, se deben cambiar las cardinalidades de las relaciones, los modelos relaciones soportan cardinalidades de 1 a N.

Al pasar de una relación de N a N a modelo relacional, la llave primaria de la relación es una llave primaria compuesta conformada por las llaves primarias de las relaciones.

Si tenemos Estudiante(ID, Nombre, Apellido) y Libro(Código, Nombre, Autor), la llave primaria de la relación va a ser (ID, Código) de libro.

### Consultas de selección con tablas relacionadas

Las bases de datos relacionales almacenan sus datos en varias tablas. Lo normal, en casi cualquier consulta, es requerir datos de varias tablas a la vez. Esto es posible porque los datos de las tablas están asociados por columnas que contienen claves secundarias o externas que permiten relacionar los datos de esa tabla con datos de otra tabla.

Como ejemplo, se considerará esta tabla de departamentos:

Cod_dep	Departamento
1	Ventas

2	Producción
3	Calidad
4	Dirección

Por otro lado se tiene una tabla de empleados:

Cod_emp	Nombre	Apellido	Cod_dep	edad
1	Marisa	León	1	54
2	Arturo	Crespo	1	58
3	Ana	Diez	2	43
4	Pau	Cabanillas	2	29
5	Luisa	Rodríguez	3	34
6	Anxo	Olivenza	4	21
7	Pedro	Anderez		40

La columna cod\_dep en la tabla de empleados es una clave secundaria. A través de ella se sabe que Marisa León, por ejemplo, es del departamento de ventas.

**Actividad propuesta. Para verificar por tu cuenta los resultados de las siguientes consultas, crea las tablas mencionadas anteriormente en SQL developer.**

### Asociaciones simples

Vamos a realizar la siguiente consulta, Asociar a los empleados con sus departamentos. Esto sería de la siguiente forma.

```
SELECT nombre, apellido, departamento
FROM departamentos,empleados
WHERE departamentos.cod_dep=empleados.cod_dep
ORDER BY nombre,apellido,departamento;
```

Nótese que se utiliza la notación tabla.columna para evitar la ambigüedad. Tanto la

tabla de departamentos como la de empleados tienen una columna llamada cod\_dep, por ello hay que distinguirla anteponiendo el nombre de la tabla a la que pertenece.

Para evitar repetir continuamente el nombre de la tabla, se puede utilizar un alias de tabla:

```
SELECT nombre, apellido, departamento
FROM departamentos d,empleados e
WHERE d.cod_dep=e.cod_dep
ORDER BY nombre,apellido,departamento;
```

En cualquier caso el resultado muestra realmente los empleados y los departamentos a los que pertenecen.

Nombre	Apellido	Departamento
Ana	Díez	Producción
Anxo	Olivenza	Dirección
Arturo	Crespo	Ventas
Luisa	Rodríguez	Calidad
Marisa	León	Ventas
Pau	Cabanillas	Producción

Al apartado WHERE se le pueden añadir condiciones encadenándose con el operador AND. Ejemplo:

```
SELECT nombre, apellido, departamento
FROM departamentos d,empleados e
WHERE d.cod_dep=e.cod_dep AND nombre='Ana'
ORDER BY nombre,apellido,departamento;
```

### Asociar más de una tabla

Por supuesto es posible asociar datos de más de dos tablas. Por ejemplo se considerará que se tiene estas tablas, mostradas desde su diseño relacional:



Se desea mostrar el nombre y apellidos de los alumnos junto con el número de curso a los que están asociados y las fecha de inicio y fin de los mismos.

Aunque los datos que se necesitan están en dos tablas (alumnos y cursos), no hay relación entre ambas, la relación entre esas tablas es la indicada por la tabla asistir, por lo que necesitamos indicar esa tabla. La consulta resultante sería:

```
SELECT nombre,apellido1,apellido2,
       c.n_curso, fecha_inicio,fecha_fin
FROM alumnos a, asistir s, cursos c
WHERE a.dni=s.dni AND c.n_curso=s.n_curso;
```

Por lo tanto, se pueden asociar más de dos tablas sin ningún problema. Al hacer consultas sobre varias tablas hay que tener en cuenta estos detalles:

- Se deben añadir las tablas que contienen los datos que se necesiten
- Además se debe añadir las tablas necesarias para asociar correctamente a las tablas anteriores

• Por último, de todas las anteriores, se debe usar las mínimas tablas. Cuantas menos mejor. Especialmente en los inicios de un desarrollador en SQL, conviene tener en cuenta este último punto. Añadir tablas de más puede cambiar el resultado.

Por ejemplo, si en el ejemplo anterior se usa y asocia una tabla con información sobre los profesores de los cursos, solo aparecen datos de cursos que tienen asignado al menos un profesor, y eso puede cambiar notablemente el resultado.

## Integridad referencial

Cuando se define una columna como clave foránea, las filas de la tabla pueden contener en esa columna o bien el valor nulo (ningún valor), o bien un valor que existe en la otra

tabla; un error sería asignar a un habitante una población que no está en la tabla de poblaciones. Eso es lo que se denomina integridad referencial, y consiste en que los datos que referencian otros (claves foráneas) deben ser correctos. La integridad referencial hace que el sistema gestor de la base de datos se asegure de que no hayan en las claves foráneas valores que no estén en la tabla principal.

La integridad referencial se activa en cuanto se crea una clave foránea, y a partir de ese momento se comprueba cada vez que se modifiquen datos que puedan alterarla.

¿ Cuándo se pueden producir errores en los datos?

- Cuando se inserta una nueva fila en la tabla secundaria y el valor de la clave foránea no existe en la tabla principal. Se inserta un nuevo habitante y en la columna población se escribe un código de población que no está en la tabla de poblaciones (una población que no existe).
- Cuando se modifica el valor de la clave principal de un registro que tiene 'hijos'.
- Cuando se modifica el valor de la clave foránea, el nuevo valor debe existir en la tabla principal.
- Cuando se desea borrar una fila de la tabla principal y ese registro tiene 'hijos'.

Asociada a la integridad referencial están los conceptos de actualizar los registros en cascada y eliminar registros en cascada.

Actualización y borrado en cascada

El actualizar y/o eliminar registros en cascada, son opciones que se definen cuando se define la clave foránea, y le indican al sistema gestor qué hacer en los casos comentados en el punto anterior.

Actualizar registros en cascada

Esta opción le indica al sistema gestor de la base de datos que cuando se cambie un valor del campo clave de la tabla principal, automáticamente cambiará el valor de la clave foránea de los registros relacionados en la tabla secundaria.

Eliminar registros en cascada

Esta opción le indica al sistema gestor de la base de datos que cuando se elimina un registro de la tabla principal automáticamente se borran también los registros relacionados en la tabla secundaria.

## Querys anidadas

Se analizó anteriormente que una subconsulta puede reemplazar una expresión. Dicha subconsulta debe devolver un valor escalar o una lista de valores de un campo; las subconsultas que retornan una lista de valores reemplazan a una expresión en una cláusula "where" que contiene la palabra clave "in".

El resultado de una subconsulta con "in" (o "not in") es una lista. Luego que la subconsulta retorna resultados, la consulta exterior los usa.

Es posible averiguar si un valor de la consulta externa pertenece o no al conjunto devuelto por una subconsulta empleando "in" y "not in". La sintaxis básica es la siguiente:

```
...where EXPRESION in (SUBCONSULTA);
```

Este ejemplo muestra los nombres de las editoriales que ha publicado libros de un determinado autor:

```
select nombre
from editoriales
where codigo in
(select codigoeditorial
from libros
where autor='Richard Bach');
```

La subconsulta (consulta interna) retorna una lista de valores de un solo campo (codigoeditorial) que la consulta exterior luego emplea al recuperar los datos. Se averigua si el código devuelto por la consulta externa se encuentra dentro del conjunto de valores retornados por la consulta interna.

Se recomienda probar las subconsultas antes de incluirlas en una consulta exterior, así puede verificar que retorna lo necesario, porque a veces resulta difícil verlo en consultas anidadas.

También se puede buscar valores no coincidentes con una lista de valores que retorna una subconsulta; por ejemplo, las editoriales que no han publicado libros de un autor específico:

```
select nombre
from editoriales
where codigo not in
(select codigoeditorial
```

```
from libros  
where autor='Richard Bach');
```

## Querys con distintos tipos de JOIN

Los JOINS de Oracle se utilizan para recuperar datos de varias tablas. Un JOIN de Oracle se realiza siempre que dos o más tablas se unen en una declaración SQL.

Hay 4 tipos diferentes de combinaciones de Oracle:

- INNER JOIN de Oracle (o, a veces, llamado unión simple)
- Oracle LEFT OUTER JOIN (o algunas veces llamado LEFT JOIN)
- Oracle RIGHT OUTER JOIN (o algunas veces llamado RIGHT JOIN)
- Oracle FULL OUTER JOIN (o algunas veces llamado FULL JOIN)

### INNER JOIN (unión simple)

Es el tipo de unión más común. Oracle INNER JOIN devuelve todas las filas de varias tablas donde se cumple la condición de unión.

La sintaxis para INNER JOIN en Oracle / PLSQL es:

```
SELECT columns  
FROM table1  
INNER JOIN table2  
ON table1.column = table2.column;
```

Este ejemplo de Oracle INNER JOIN devolvería todas las filas de las tablas de proveedores y pedidos donde hay un valor de ID de proveedor coincidente en las tablas de proveedores y pedidos.

### LEFT OUTER JOIN

Otro tipo de combinación se llama Oracle LEFT OUTER JOIN. Este tipo de combinación devuelve todas las filas de la tabla de la IZQUIERDA especificada en la condición ON y solo aquellas filas de la otra tabla donde los campos combinados son iguales (se cumple la condición de combinación).



La sintaxis para Oracle LEFT OUTER JOIN es:

```
SELECT columns  
FROM table1  
LEFT [OUTER] JOIN table2  
ON table1.column = table2.column;
```

En algunas bases de datos, las palabras clave LEFT OUTER JOIN se reemplazan por LEFT JOIN.

Ejemplo: Uso de LEFT OUTER JOIN de Oracle:

```
SELECT suppliers.supplier_id, suppliers.supplier_name,  
orders.order_date  
FROM suppliers  
LEFT OUTER JOIN orders  
ON suppliers.supplier_id = orders.supplier_id;
```

Este ejemplo de LEFT OUTER JOIN devolvería todas las filas de la tabla de proveedores y solo aquellas filas de la tabla de pedidos donde los campos combinados son iguales. Si un valor de provider\_id en la tabla de proveedores no existe en la tabla de pedidos, todos los campos de la tabla de pedidos se mostrarán como <nulo > en el conjunto de resultados.

## RIGHT OUTER JOIN

Otro tipo de unión se denomina RIGHT OUTER JOIN. Este tipo de combinación devuelve todas las filas de la tabla de la DERECHA especificada en la condición ON y solo aquellas filas de la otra tabla donde los campos combinados son iguales (se cumple la condición de combinación).

La sintaxis de Oracle RIGHT OUTER JOIN es:

```
SELECT columns  
FROM table1  
RIGHT [OUTER] JOIN table2  
ON table1.column = table2.column;
```

En algunas bases de datos, las palabras clave RIGHT OUTER JOIN se reemplazan por RIGHT JOIN.

Aquí hay un ejemplo de una RIGHT OUTER JOIN de Oracle:

```
SELECT orders.order_id, orders.order_date,  
suppliers.supplier_name  
FROM suppliers  
RIGHT OUTER JOIN orders  
ON suppliers.supplier_id = orders.supplier_id;
```

Este ejemplo de RIGHT OUTER JOIN devolvería todas las filas de la tabla de pedidos y solo aquellas filas de la tabla de proveedores donde los campos combinados son iguales.

Si un valor de provider\_id en la tabla de pedidos no existe en la tabla de proveedores, todos los campos de la tabla de proveedores se mostrarán como <nulo> en el conjunto de resultados.

#### **Referencias:**

[Oracle - Funciones](#)

[Oracle: Modelo de datos](#)

[Oracle: Joins](#)