TEMA 4

INDICE

Realización de consultas.

Caso práctico

Una de las cosas más importantes que ofrece una base de datos es la opción de poder consultar los datos que guarda, por eso Ana y Juan van a intentar sacar el máximo partido a las tablas que han guardado y sobre ellas van a obtener toda aquella información que su cliente les ha solicitado. Sabemos que dependiendo de quién consulte la base de datos, se debe ofrecer un tipo de información u otra. Es por esto que deben crear distintas consultas y vistas.

Ana sabe que existen muchos tipos de operadores con los que puede "jugar" para crear consultas y también tiene la posibilidad de crear campos nuevos donde podrán hacer cálculos e incluso trabajar con varias tablas relacionadas a la vez.

*Actualmente están con una base de datos en la que se ha almacenado información sobre los **empleados** de la empresa que tiene la página de juegos online, los **departamentos** en los que trabajan y los **estudios** de sus empleados. Se está guardando el **historial laboral y salarial** de todos los empleados. Ya que tienen una base de datos para sus clientes, han visto que también sería conveniente tener registrada esta otra información interna de la empresa.*

De este modo pueden llevar un control más exhaustivo de sus empleados, salario y especialización. Podrán conocer cuánto pagan en sueldos, qué departamento es el que posee mayor número de empleados, el salario medio, etc. Para obtener esta información necesitarán consultar la base utilizando principalmente el comando SELECT.

1.- Introducción.

Caso práctico

Juan quiere comenzar con consultas básicas a los datos, cosas bastante concretas y sencillas de manera que se obtenga información relevante de cada una de las tablas. También quieren realizar algunos cálculos como conocer el salario medio de cada empleado, o el mayor salario de cada departamento, o saber cuánto tiempo lleva cada empleado en la empresa.

En unidades anteriores has aprendido que SQL es un conjunto de sentencias u órdenes que se necesitan para acceder a los datos. Este lenguaje es utilizado por la mayoría de las aplicaciones donde se trabaja con datos para acceder a ellos. Es decir, es la vía de comunicación entre el usuario y la base de datos.

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

Codd. IBM aprovechó el modelo que planteaba Codd para desarrollar un lenguaje acorde con el recién nacido modelo relacional, a este primer lenguaje se le llamó SEQUEL (Structured English QUEry Language). Con el tiempo SEQUEL se convirtió en SQL (Structured Query Language). En 1979, la empresa Relational Software sacó al mercado la primera implementación comercial de SQL. Esa empresa es la que hoy conocemos como Oracle.

Actualmente SQL sigue siendo el estándar en lenguajes de acceso a base de datos relacionales.

En 1992, ANSI e ISO completaron la estandarización de SQL y se definieron las sentencias básicas que debía contemplar SQL para que fuera estándar. A este SQL se le denominó ANSI-SQL o SQL92.

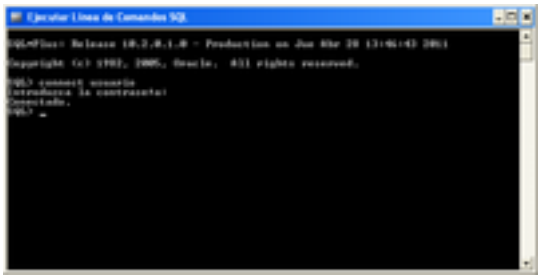
Hoy en día todas las bases de datos comerciales cumplen con este estándar, eso sí, cada fabricante añade sus mejoras al lenguaje SQL.

La primera fase del trabajo con cualquier base de datos comienza con sentencias DDL (en español Lenguaje de Definición de Datos), puesto que antes de poder almacenar y recuperar información debimos definir las estructuras donde agrupar la información: las tablas.

La siguiente fase será manipular los datos, es decir, trabajar con sentencias DML (en español Lenguaje de Manipulación de Datos). Este conjunto de sentencias está orientado a consultas y manejo de datos de los objetos creados. Básicamente consta de cuatro sentencias: SELECT, INSERT, DELETE y UPDATE. En esta unidad nos centraremos en una de ellas, que es la sentencia para consultas: SELECT.

Las sentencias SQL que se verán a continuación pueden ser ejecutadas desde el entorno web **Application Express** de Oracle utilizando el botón **SQL** en la página de inicio, y desplegando su lista desplegable elegir **Comandos SQL > Introducir Comando**.

También se pueden indicar las sentencias SQL desde el entorno de **SQL*Plus** que ofrece Oracle y que puedes encontrar en **Inicio > Todos los programas > Base de Datos Oracle Express Edition > Ejecutar Línea de Comandos SQL.**



Si optas por abrir esa aplicación (**Ejecutar Línea de Comandos SQL**), el primer paso que debe realizarse para manipular los datos de una determinada tabla, es conectarse utilizando un nombre de usuario con los permisos necesarios para hacer ese tipo de operaciones a la tabla deseada. Utiliza para ello la orden CONNECT seguida del nombre de usuario. Posteriormente, solicitará la contraseña correspondiente a dicho usuario.

Para ejecutar cualquiera de las sentencias SQL que aprenderás en los siguientes puntos, simplemente debes escribirla completa y pulsar **Intro** para que se inicie su ejecución.

¿Con qué sentencias se definen las estructuras donde agrupar la información, es decir, las tablas?

- ○ DML
- ◉ **DDL**
- ○ DCL

Así es, dentro del lenguaje de definición de datos está la creación de tablas.

2.- La sentencia SELECT.

Caso práctico

__Ana__ está trabajando con la tabla Partidas, de aquí quiere ver qué información es la más importante, para así crear las consultas más sencillas pero a la vez más frecuentes. Sabe que con SQL y utilizando el comando SELECT puede sacar provecho a los datos contenidos en una tabla.

¿Cómo podemos seleccionar los datos que nos interesen dentro de una base de datos? Para recuperar o seleccionar los datos, de una o varias tablas puedes valerte del lenguaje SQL, para ello utilizarás la sentencia SELECT, que consta de cuatro partes básicas:

- ✓ **Cláusula** SELECT seguida de la descripción de lo que se desea ver, es decir, de los nombres de las columnas que quieres que se muestren separadas por comas simples (" , "). Esta parte es obligatoria.
- ✓ **Cláusula** FROM seguida del nombre de las tablas de las que proceden las columnas de arriba, es decir, de donde vas a extraer los datos. Esta parte también es obligatoria.
- ✓ **Cláusula** WHERE seguida de un criterio de selección o condición. Esta parte es opcional.
- ✓ **Cláusula** ORDER BY seguida por un criterio de ordenación. Esta parte también es opcional.

Por tanto, una primera sintaxis quedaría de la siguiente forma:

```
SELECT [ALL | DISTINCT] columna1, columna2, ... FROM tabla1, tabla2, ... WHERE condición1, condición2, ... ORDER BY ordenación;
```

Las cláusulas ALL y DISTINCT son opcionales.

- ✓ **Si incluyes la cláusula ALL después de SELECT, indicarás que quieres seleccionar todas las filas estén o no repetidas. Es el valor por defecto y no se suele especificar.**
- ✓ **Si incluyes la cláusula DISTINCT después de SELECT, se suprimirán aquellas filas del resultado que tengan igual valor que otras.**

¿Qué se debe indicar a continuación de la cláusula FROM?

- ○ Las columnas que queremos seleccionar.
- ○ Los criterios con los que filtro la selección.
- ◉ **Las tablas de donde se van a extraer los datos.**
- ○ La ordenación ascendente.

Así es, Aparecerán todos los nombres de las tablas cuyas columnas estemos seleccionando, separadas por coma.

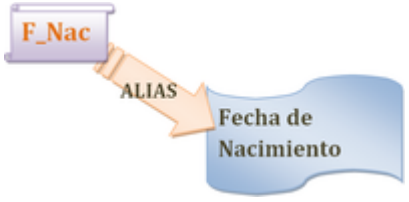
2.1.- Cláusula SELECT.

Ya has visto que a continuación de la sentencia SELECT debemos especificar cada una de las columnas que queremos seleccionar. Además, debemos tener en cuenta lo siguiente:

- ✓ **Se pueden nombrar** a las columnas anteponiendo el nombre de la tabla de la que proceden, pero esto es opcional y quedaría: NombreTabla.NombreColumna
- ✓ Si queremos **incluir todas las columnas** de una tabla podemos utilizar el comodín **asterisco** ("*"). Quedaría así: SELECT * FROM NombreTabla;
- ✓ También podemos **ponerle alias a los nombres** de las columnas. Cuando se consulta una base de datos, los nombres de las columnas se usan como cabeceras de presentación. Si éste resulta largo, corto o poco descriptivo, podemos usar un alias. Para ello a continuación del nombre de la columna ponemos entre comillas dobles el alias que demos a esa columna.

  Veamos un ejemplo:

```
SELECT F_Nacimiento "Fecha de Nacimiento" FROM USUARIOS;
```



- ✓ También podemos **sustituir el nombre** de las columnas por constantes, expresiones o funciones SQL. Un ejemplo:
- ✓ SELECT 4*3/100 "MiExpresión", Password FROM USUARIOS;

Si quieres conocer algo más sobre esta sentencia y ver algunos ejemplos del uso de SELECT aquí tienes el siguiente enlace:

http://www.devjoker.com/contenidos/Tutorial-SQL-/14/Consultar-datos-SELECT.aspx

Ejercicio resuelto

Si quieres practicar algunos ejercicios puedes ayudar a **Ana** con algunas consultas. Para ello te facilitamos las tablas que ha creado recientemente para la base de datos con la que actualmente están trabajando:

```
/* tabla empleados */
CREATE TABLE EMPLEADOS (
DNI NUMBER(8),
NOMBRE VARCHAR2(10) NOT NULL,
APELLIDO1 VARCHAR2(15) NOT NULL,
APELLIDO2 VARCHAR2(15),
SALARIO NUMBER(10,2),  /* podría ganar mucho */
DIRECC1 VARCHAR2(25),
DIRECC2 VARCHAR2(20),
CIUDAD VARCHAR2(20),
MUNICIPIO VARCHAR2(20),
COD_POSTAL VARCHAR2(5),
SEXO CHAR(1),
FECHA_NAC DATE,
CONSTRAINT PK EMPLEADOS PRIMARY KEY (DNI),
CONSTRAINT CK SEXO CHECK (SEXO IN ('H', 'M'))
);

/* tabla departamentos */
CREATE TABLE DEPARTAMENTOS (
DPTO COD NUMBER(5),
NOMBRE DPTO VARCHAR2(30) NOT NULL,
JEFE NUMBER(8),
PRESUPUESTO NUMBER(6) NOT NULL,
PRES_ACTUAL NUMBER(6),
CONSTRAINT PK_DEPARTAMENTOS PRIMARY KEY (DPTO_COD),
CONSTRAINT FK_DEPARTAMENTOS FOREIGN KEY (JEFE) REFERENCES EMPLEADOS (DNI)
);

/* Tabla universidades */
CREATE TABLE UNIVERSIDADES (
UNIV COD NUMBER(5),
NOMBRE UNIV VARCHAR2(25) NOT NULL,
CIUDAD VARCHAR2(20),
MUNICIPIO VARCHAR2(20),
COD_POSTAL VARCHAR2(5),
CONSTRAINT PK UNIVERSIDADES PRIMARY KEY (UNIV COD)
);
/* tabla trabajos */
CREATE TABLE TRABAJOS (
TRABAJO_COD NUMBER(5),
NOMBRE_TRAB VARCHAR2(20) NOT NULL UNIQUE,
SALARIO_MIN NUMBER(5) NOT NULL,
SALARIO MAX NUMBER(5) NOT NULL,
CONSTRAINT PK TRABAJOS PRIMARY KEY (TRABAJO COD)
);

/* tabla estudios */
CREATE TABLE ESTUDIOS (
EMPLEADO DNI NUMBER(8),
UNIVERSIDAD NUMBER(5),
AÑO NUMBER(4),
GRADO VARCHAR2(5),
ESPECIALIDAD VARCHAR2(20),
CONSTRAINT PK_ESTUDIOS PRIMARY KEY (EMPLEADO_DNI, AÑO, GRADO),
CONSTRAINT FK_ESTUDIOS_EMPLEADOS FOREIGN KEY (EMPLEADO_DNI) REFERENCES EMPLEADOS (DNI),
```

```
CONSTRAINT FK_ESTUDIOS_UNIVERSIDADES FOREIGN KEY (UNIVERSIDAD) REFERENCES UNIVERSIDADES
(UNIV_COD)
);

/* tabla historial_laboral */
CREATE TABLE HISTORIAL_LABORAL (
EMPLEADO_DNI NUMBER(8),
TRAB_COD NUMBER(5),
FECHA_INICIO DATE,
FECHA_FIN DATE,
DPTO_COD NUMBER(5),
SUPERVISOR_DNI NUMBER(8),
CONSTRAINT PK HISTORIAL LABORAL PRIMARY KEY (EMPLEADO DNI, FECHA INICIO),
CONSTRAINT FK HLABORAL EMPLEADOS FOREIGN KEY (EMPLEADO DNI) REFERENCES EMPLEADOS (DNI),
CONSTRAINT FK HLABORAL TRABAJOS FOREIGN KEY (TRAB COD) REFERENCES TRABAJOS (TRABAJO COD),
CONSTRAINT FK HLABORAL DEPARTAMENTOS FOREIGN KEY (DPTO COD) REFERENCES DEPARTAMENTOS
(DPTO_COD),
CONSTRAINT FK_HLABORAL_SUPERVISOR FOREIGN KEY (SUPERVISOR_DNI) REFERENCES EMPLEADOS (DNI),
CONSTRAINT CK HLABORAL FECHAS CHECK (FECHA FIN IS NULL OR FECHA INICIO < FECHA FIN)
);
/* tabla historial salarial */
CREATE TABLE HISTORIAL SALARIAL (
EMPLEADO_DNI NUMBER(8),
SALARIO NUMBER(5),
FECHA_COMIENZO DATE,
FECHA_FIN DATE,
CONSTRAINT PK HISTORIAL SALARIAL PRIMARY KEY (EMPLEADO DNI, FECHA COMIENZO),
CONSTRAINT FK_HISTORIAL_SALARIAL FOREIGN KEY (EMPLEADO_DNI) REFERENCES EMPLEADOS (DNI),
CONSTRAINT CK_FECHAS CHECK (FECHA_FIN IS NULL OR FECHA_COMIENZO < FECHA_FIN)
);

ALTER SESSION SET NLS DATE FORMAT='DD/MM/YYYY';
/* EMPLEADOS */
INSERT INTO EMPLEADOS VALUES( '12345','Jose', 'Mercé','López', '1500','C/Sol, 1', 'C/ Otra,
1', 'Cádiz','Cádiz', '11000', 'H', '5/01/74');
INSERT INTO EMPLEADOS VALUES( '22222','María', 'Rosal','Cózar','2000', '', '',
'Ubrique','Cádiz', '11600', 'M', '');
INSERT INTO EMPLEADOS VALUES( '33333','Pilar', 'Pérez','Rollán','1000', '', '', 'Cádiz','',
'11600', 'M', '2/8/73');

/* DEPARTAMENTOS */
INSERT INTO DEPARTAMENTOS VALUES( '001', 'INFORMÁTICA', '33333', 80000, 50000);

/* UNIVERSIDADES */
INSERT INTO UNIVERSIDADES VALUES('0001', 'UNED', 'MADRID', 'M', '41420');
INSERT INTO UNIVERSIDADES VALUES('0002', 'SEVILLA', 'SEVILLA', '', '55555');
INSERT INTO UNIVERSIDADES VALUES('0003', 'CÁDIZ', 'CÁDIZ', '', '11000');

/* TRABAJOS */
INSERT INTO TRABAJOS VALUES ('001', 'ADMINISTRATIVO', 900, 1000);
INSERT INTO TRABAJOS VALUES ('002', 'CONTABLE', 900, 1000);
INSERT INTO TRABAJOS VALUES ('003', 'INGENIERO TÉCNICO', 1000, 1200);
INSERT INTO TRABAJOS VALUES ('004', 'INGENIERO', 1200, 1800);

/* ESTUDIOS */
INSERT INTO ESTUDIOS VALUES( '12345', '0001', '1992', 'MED', 'ADMINISTRATIVO');
INSERT INTO ESTUDIOS VALUES( '22222', '0001', '1998', 'SUP', 'ING INFORMÁTICA');
INSERT INTO ESTUDIOS VALUES( '33333', '0002', '1997', 'SUP', 'LIC INFORMÁTICA');

/* HISTORIAL SALARIAL */
INSERT INTO HISTORIAL SALARIAL VALUES( '12345', 950, '5/01/2003', '');
INSERT INTO HISTORIAL_SALARIAL VALUES( '22222', 1000, '3/11/2004', '3/11/2005');
INSERT INTO HISTORIAL_SALARIAL VALUES( '22222', 1500, '3/11/2005', '');
INSERT INTO HISTORIAL_SALARIAL VALUES( '33333',1600, '15/01/2001', '');

/* HISTORIAL LABORAL */
INSERT INTO HISTORIAL LABORAL VALUES( '12345', '001', '5/01/2003','', '0001', '33333');
INSERT INTO HISTORIAL_LABORAL VALUES( '22222', '003', '3/11/2004', '3/11/2005', '001',
'33333');
INSERT INTO HISTORIAL LABORAL VALUES( '22222', '003', '3/11/2005', '', '001', '33333');
INSERT INTO HISTORIAL_LABORAL VALUES( '33333','004', '15/01/2001', '', '001', '33333');
```

También tienes algunos datos incluidos para probar las distintas consultas que crees. A partir de ahora nos referiremos a estos datos como tablas de la empresa JuegosCA.

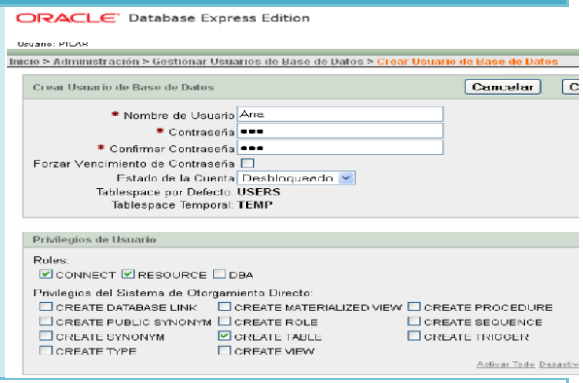
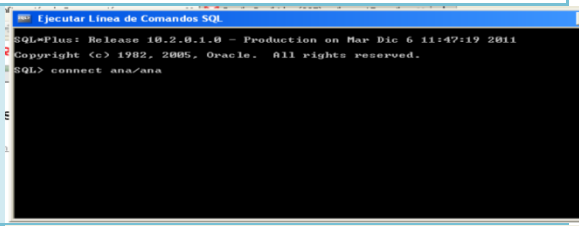
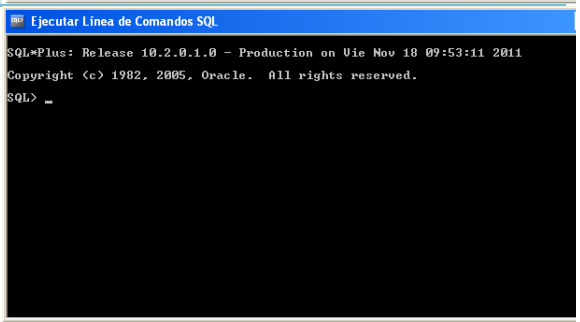
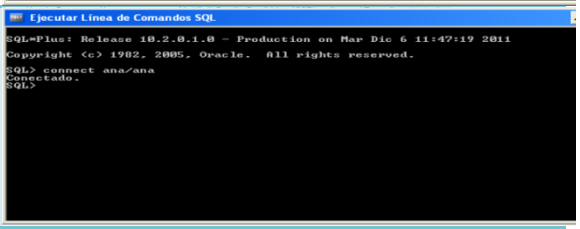
Por tanto lo primero que tienes que hacer es abrir el editor de SQL, para ello debes ir a Base de Datos de Oracle 10g Express y a continuación pulsar en Ejecutar Línea de Comandos SQL. Aparecerá una pantalla donde tienes que realizar los siguientes pasos:

1. Conectarte a través de un usuario.
2. Ejecutar el archivo que has bajado, para ello debes escribir :

```
@Ruta_donde_se_encuentra_el_archivo/BD04_CONT_R07_02.sql
```

En este ejercicio te pedimos que ejecutes el archivo y crees las tablas necesarias para poder realizar ejercicios posteriores.

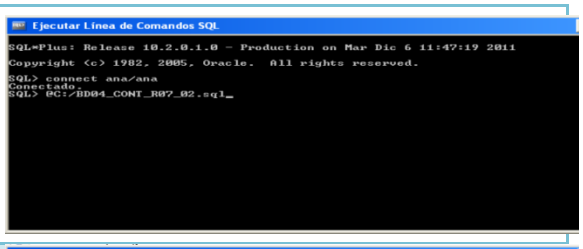
Resultado:
El resultado puedes verlo en la siguiente tabla.

Creación de tablas y preparación de Oracle.	
Para comenzar crearemos a partir de un usuario con privilegios el usuario ANA y pondremos una contraseña. Además, le daremos posibilidad de crear tablas.	
Desconectamos a este usuario, ya que queremos crear las tablas e insertar los datos en el nuevo usuario creado. Vamos a utilizar la línea de comandos de SQL para ejecutar el archivo descargado, para ello seguiremos los pasos que aparecen a continuación.	
Vamos a Base de Datos de Oracle 10g Express.	
Pulsamos en Ejecutar Línea de Comandos SQL. Aparecerá la siguiente pantalla	
Ejecutamos la instrucción: connect ana/ana. Cuando ejecutamos el comando debe decirnos que ya está conectado	

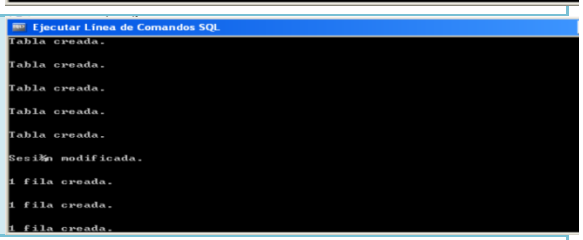
Ahora ya podemos ejecutar el archivo del siguiente modo:

```
@Ruta_donde_se_encuentra_el_archivo/BD04_CONT_R07_02.sql
```

En nuestro caso, el archivo está guardado directamente en la unidad C para que nos resulte más fácil localizarlo



Si todo es correcto, deberían crearse las tablas e insertarse los datos que contiene el archivo



A partir de aquí ya tienes un usuario con tablas y datos inlcuidos para poder practicar a la vez que Ana.

Puedes hacerlo a través de línea de comandos o entrando a entorno web **Application Express** de Oracle utilizando el botón **SQL** en la página de inicio, y desplegando su lista desplegable elegir **Comandos SQL** > **Introducir Comando**.

2.2.- Cláusula FROM.

Al realizar la consulta o selección has visto que puedes elegir las columnas que necesites, pero ¿de dónde extraigo la información?

En la sentencia `SELECT` debemos establecer de dónde se obtienen las columnas que vamos a seleccionar, para ello disponemos en la sintaxis de la cláusula `FROM`.

Por tanto, en la cláusula `FROM` **se definen los nombres de las tablas** de las que proceden las columnas. Si se utiliza más de una, éstas deben aparecer separadas por comas. A este tipo de consulta se denomina **consulta combinada** o `join`. Más adelante verás que para que la consulta combinada pueda realizarse, necesitaremos aplicar una condición de combinación a través de una cláusula `WHERE`.

También puedes añadir el nombre del usuario que es **propietario** de esas tablas, indicándolo de la siguiente manera:

```
USUARIO.TABLA
```

de este modo podemos distinguir entre las tablas de un usuario y otro (ya que esas tablas pueden tener el mismo nombre).

También puedes asociar un alias a las tablas para abreviar, en este caso **no es necesario que lo encierres entre comillas**.

Pongamos un ejemplo:

```
SELECT * FROM USUARIOS U;
```

2.3.- Cláusula WHERE.

¿Podríamos desear seleccionar los datos de una tabla que cumplan una determinada condición? Hasta ahora hemos podido ver la sentencia `SELECT` para obtener todas o un subconjunto de columnas de una o varias tablas. Pero esta selección afectaba a todas las filas (registros) de la tabla. Si queremos restringir esta selección a un subconjunto de filas debemos especificar una condición que

deben cumplir aquellos registros que queremos seleccionar. Para poder hacer esto vamos a utilizar la cláusula `WHERE`.

A continuación de la palabra `WHERE` será donde pongamos la condición que han de cumplir las filas para salir como resultado de dicha consulta.

El criterio de búsqueda o condición puede ser más o menos sencillo y para crearlo se pueden conjugar operadores de diversos tipos, funciones o expresiones más o menos complejas.
Si en nuestra tabla USUARIOS, necesitáramos un listado de los usuarios que son mujeres, bastaría con crear la siguiente consulta:

```
SELECT nombre, apellidos
FROM USUARIOS
WHERE sexo = 'M';
```

Más adelante te mostraremos los operadores con los que podrás crear condiciones de diverso tipo.

Aquí te adelantamos los operadores para que vayas conociéndolos. Con ellos trabajarás cuando hayas adquirido algunos conocimientos más:
http://www.desarrolloweb.com/articulos/1870.php

2.4.- Ordenación de registros. Cláusula ORDER BY.

En la consulta del ejemplo anterior hemos obtenido una lista de nombres y apellidos de las usuarias de nuestro juego. Sería conveniente que aparecieran ordenadas por apellidos, ya que siempre quedará más profesional además de más práctico. De este modo, si necesitáramos localizar un registro concreto la búsqueda sería más rápida. ¿Cómo lo haremos? Para ello usaremos la cláusula `ORDER BY`.

`ORDER BY` se utiliza para **especificar el criterio de ordenación** de la respuesta a nuestra consulta. Tendríamos:

```
SELECT [ALL | DISTINCT] columna1, columna2, ...
FROM tabla1, tabla2, ...
WHERE condición1, condición2, ...
ORDER BY columna1 [ASC | DESC], columna2 [ASC | DESC], …, columnaN [ASC | DESC];
```

Después de cada columna de ordenación se puede incluir el tipo de ordenación (ascendente o descendente) utilizando las palabras reservadas `ASC` o `DESC`. Por defecto, y si no se pone nada, la ordenación es ascendente.

Debes saber que es **posible ordenar por más de una columna**. Es más, puedes ordenar no solo por columnas sino a través de una expresión creada con columnas, una constante (aunque no tendría mucho sentido) o funciones SQL.

En el siguiente ejemplo, ordenamos por apellidos y en caso de empate por nombre:

```
SELECT nombre, apellidos
FROM USUARIOS
ORDER BY apellidos, nombre;
```

Puedes colocar el número de orden del campo por el que quieres que se ordene en lugar de su nombre, es decir, referenciar a los campos por su posición en la lista de selección. Por ejemplo, si queremos el resultado del ejemplo anterior ordenado por localidad:

```
SELECT nombre, apellidos, localidad
FROM usuarios
ORDER BY 3;
```

Si colocamos un número mayor a la cantidad de campos de la lista de selección, aparece un mensaje de error y la sentencia no se ejecuta.

¿Se puede utilizar cualquier tipo de datos para ordenar? No todos los tipos de campos te servirán para ordenar, únicamente aquellos de tipo **carácter, número o fecha**.

Relaciona cada cláusula de la sentencia SELECT con la información que debe seguirle:

Ejercicio de relacionar

Cláusula	Relación	Información que le sigue.
WHERE	**4**	1. Ordenación.
ORDER BY	**1**	2. Columnas.
FROM	**3**	3. Tablas.
SELECT	**2**	4. Condiciones.

Ejercicio resuelto

Utilizando las tablas y datos de la empresa **JuegosCA** descargados anteriormente, vamos a realizar una consulta donde obtengamos de la tabla ESTUDIOS, DNI de los empleados ordenados por Universidad descendente y año de manera ascendente.

Respuesta:

```
SELECT EMPLEADO_DNI
FROM ESTUDIOS
ORDER BY UNIVERSIDAD DESC, AÑO;
```

3.- Operadores.

Caso práctico

*En el proyecto en el que actualmente trabajan **Ana** y **Juan**, tendrán que realizar consultas que cumplan unos criterios concretos, por ejemplo, obtener el número de jugadores que tienen cierto número de créditos o aquellos que son mujeres e incluso conocer el número de usuarios que son de una provincia y además sean hombres.*

*Para poder realizar este tipo de consultas necesitaremos utilizar operadores que sirvan para crear las expresiones necesarias. **Ana** y **Juan** conocen los 4 tipos de operadores con los que se puede trabajar: relacionales, aritméticos, de concatenación y lógicos.*

Veíamos que en la cláusula `WHERE` podíamos incluir expresiones para filtrar el conjunto de datos que queríamos obtener. Para crear esas expresiones necesitas utilizar distintos operadores de modo que puedas comparar, utilizar la lógica o elegir en función de una suma, resta, etc.

Los operadores son símbolos que permiten realizar operaciones matemáticas, concatenar cadenas o hacer comparaciones.

Oracle reconoce 4 tipos de operadores:

1. Relacionales o de comparación.
2. Aritméticos.
3. De concatenación.
4. Lógicos.

¿Cómo se utilizan y para qué sirven? En los siguientes apartados responderemos a estas cuestiones.

Si quieres conocer un poco más sobre los operadores visita este enlace:
http://deletesql.com/viewtopic.php?f=5&t=10

3.1.- Operadores de comparación.

Los puedes conocer con otros nombres como **relacionales**, nos **permitirán comparar expresiones**, que pueden ser valores concretos de campos, variables, etc.

Los operadores de comparación son símbolos que se usan como su nombre indica para comparar dos valores. Si el resultado de la comparación es correcto la expresión considerada es verdadera, en caso contrario es falsa.

Tenemos los siguientes operadores y su operación:

Operadores y su significado.	
OPERADOR	**SIGNIFICADO**
=	Igualdad.
!=, < >, ^=	Desigualdad.
<	<
>	Mayor que.
<=	Menor o igual que.
>=	Mayor o igual que.
IN	Igual que cualquiera de los miembros entre paréntesis.
NOT IN	Distinto que cualquiera de los miembros entre paréntesis.
BETWEEN	Entre. Contenido dentro del rango.
NOT BETWEEN	Fuera del rango.
LIKE '_abc%'	Se utiliza sobre todo con textos y permite obtener columnas cuyo valor en un

	campo cumpla una condición textual. Utiliza una cadena que puede contener los símbolos "**%**" que sustituye a un conjunto de caracteres o "**_**" que sustituye a un carácter.
IS NULL	Devuelve verdadero si el valor del campo de la fila que examina es nulo.

El valor `NULL` significaba valor inexistente o desconocido y por tanto es tratado de forma distinta a otros valores. Si queremos verificar que un valor es `NULL` no serán validos los operadores que acabamos de ver. Debemos utilizar los valores `IS NULL` como se indica en la tabla o `IS NOT NULL` que devolverá verdadero si el valor del campo de la fila no es nulo.

Además, cuando se utiliza un `ORDER BY`, los valores `NULL` se presentarán en primer lugar si se emplea el modo ascendente y al final si se usa el descendente.

Si queremos obtener aquellos empleados cuyo salario es superior a 1000€ podemos crear la siguiente consulta:

```
SELECT nombre FROM EMPLEADOS WHERE SALARIO > 1000;
```

Ahora queremos aquellos empleados cuyo apellido comienza por R:

```
SELECT nombre FROM EMPLEADOS WHERE APELLIDO1 LIKE 'R%';
```

Ejercicio resuelto

Utilizando las tablas y datos de la empresa **JuegosCA** descargados anteriormente, vamos a realizar una consulta donde obtengamos las universidades de Sevilla o Cádiz.

Resultado:

```
SELECT UNIV_COD, NOMBRE_UNIV FROM UNIVERSIDADES
WHERE CIUDAD IN ('SEVILLA', 'CÁDIZ');
```

Fíjate que buscará aquellas ciudades que coincidan textualmente con las que ponemos entre comillas.

Los operadores que se utilizan en MySQL puedes verlos en el siguiente enlace:
http://dev.mysql.com/doc/refman/5.0/es/comparison-operators.html

3.2.- Operadores aritméticos y de concatenación.

Aprendimos que los operadores son símbolos que permiten realizar distintos tipos de operaciones. Los operadores aritméticos permiten realizar cálculos con valores numéricos. Son los siguientes:

Operadores aritméticos y su significado.	
OPERADOR	**SIGNIFICADO**
+	Suma
-	Resta
*****	Multiplicación
/	División

Utilizando expresiones con operadores **es posible obtener** salidas en las cuales **una columna sea el resultado de un cálculo** y no un campo de una tabla.

Mira este ejemplo en el que obtenemos el salario aumentado en un 5% de aquellos empleados que cobran menos de 1000€:

```
SELECT SALARIO*1,05
FROM EMPLEADOS
WHERE SALARIO<=1000;
```

Cuando una expresión aritmética se calcula sobre valores `NULL`, el resultado es el propio valor `NULL`.
Para concatenar cadenas de caracteres existe el operador de concatenación (" **||** "). Oracle puede convertir automáticamente valores numéricos a cadenas para una concatenación.

En la tabla EMPLEADOS tenemos separados en dos campos el primer y segundo apellido de los empleados, si necesitáramos mostrarlos juntos podríamos crear la siguiente consulta:

```
SELECT Nombre, Apellido1 || Apellido2
FROM EMPLEADOS;
```

Si queremos dejar un espacio entre un apellido y otro, debemos concatenar también el espacio en blanco de la siguiente manera:

```
SELECT Nombre, Apellido1 || ' ' ||Apellido2
FROM EMPLEADOS;
```

Los operadores que se utilizan en MySQL puedes verlos en el siguiente enlace:
http://dev.mysql.com/doc/refman/5.0/es/arithmetic-functions.html

3.3.- Operadores lógicos.

Habrá ocasiones en las que tengas que evaluar más de una expresión y necesites verificar que se cumple una única condición, otras veces comprobar si se cumple una u otra o ninguna de ellas. Para poder hacer esto utilizaremos los operadores lógicos.

Tenemos los siguientes:

Operadores lógicos y su significado.	
OPERADOR	**SIGNIFICADO**
AND	Devuelve verdadero si sus expresiones a derecha e izquierda son ambas verdaderas.
OR	Devuelve verdadero si alguna de sus expresiones a derecha o izquierda son verdaderas.
NOT	Invierte la lógica de la expresión que le precede, si la expresión es verdadera devuelve falsa y si es falsa devuelve verdadera.

Fíjate en los siguientes ejemplos:
Si queremos obtener aquellos empleados en cuyo historial salarial tengan sueldo menor o igual a 800€ o superior a 2000€:

```
SELECT empleado_dni
FROM HISTORIAL_SALARIAL
WHERE salario<=800 OR salario>2000;
```

Ejercicio resuelto

Utilizando las tablas y datos de la empresa JuegosCA descargados anteriormente, vamos a realizar una consulta donde obtengamos todos nombres de trabajos menos el de contable.

Respuesta:

```
SELECT NOMBRE_TRAB
FROM TRABAJOS
WHERE NOMBRE_TRAB NOT IN ('CONTABLE');
```

3.4.- Precedencia.

Con frecuencia utilizaremos la sentencia SELECT acompañada de expresiones muy extensas y resultará difícil saber que parte de dicha expresión se evaluará primero, por ello es conveniente conocer el orden de precedencia que tiene Oracle:

1. Se evalúa la multiplicación (*****) y la división (**/**) al mismo nivel.
2. A continuación sumas (**+**) y restas (**-**).
3. Concatenación (**| |**).
4. Todas las comparaciones (**<**, **>**, …).
5. Después evaluaremos los operadores **IS NULL**, **IN NOT NULL**, **LIKE**, **BETWEEN**.
6. **NOT**.
7. **AND**.
8. **OR**.

Si quisiéramos variar este orden necesitaríamos utilizar paréntesis.

En la siguiente consulta:

```
SELECT APELLIDOS
FROM JUGADORES
WHERE APELLIDOS LIKE 'A%S%';
```

¿Qué estaríamos seleccionando?

- ○ Aquellos jugadores cuyos apellidos contienen la letra A y la S
- ◉ **Aquellos jugadores cuyos apellidos comienzan por la letra A y contienen la letra S**
- ○ Aquellos jugadores cuyos apellidos no contienen ni la letra A ni la S
- ○ Todos los apellidos de todos los jugadores menos los que su apellido comienza por S

También tenemos orden de precedencia en MySQL:
http://dev.mysql.com/doc/refman/5.0/es/operator-precedence.html

4.- Consultas calculadas.

Caso práctico

*A la empresa ha llegado **Carlos** que está en fase de prácticas y como anda un poco desubicado ha comenzado su trabajo revisando la teoría y práctica que han dado en clase. No recuerda bien como se creaban campos nuevos a partir de otros ya existentes en la base de datos. Sabe que es algo sencillo pero no quiere meter la pata ya que está ayudando a **Juan** en un proyecto que acaba de entrar.*

Lo que hará será practicar a partir de una tabla que tenga bastantes campos numéricos de manera que pueda manipular la información sin modificar nada.

En clase trabajaban con la tabla ARTICULOS que tenía, entre otros, los campos Precio y Cantidad. A partir de ellos podría realizar consultas calculadas para obtener el precio con IVA incluido, un descuento sobre el precio e incluso aumentar ese precio en un porcentaje concreto. Seguro que se pone al día rápidamente.

En algunas ocasiones es interesante realizar operaciones con algunos campos para obtener información derivada de éstos. Si tuviéramos un campo Precio, podría interesarnos calcular el precio incluyendo el IVA o si tuviéramos los campos Sueldo y Paga Extra, podríamos necesitar obtener la suma de los dos campos. Estos son dos ejemplos simples pero podemos construir expresiones mucho más complejas. Para ello haremos uso de la creación de campos calculados.

Los operadores aritméticos se pueden utilizar para hacer cálculos en las consultas.

Estos **campos calculados** se obtienen a través de la sentencia `SELECT` poniendo a continuación la expresión que queramos. Esta consulta **no modificará los valores originales** de las columnas ni de la tabla de la que se está obteniendo dicha consulta, únicamente mostrará una columna nueva con los valores calculados. Por ejemplo:

```
SELECT Nombre, Credito, Credito + 25
FROM USUARIOS;
```

Con esta consulta hemos creado un campo que tendrá como nombre la expresión utilizada. Podemos ponerle un alias a la columna creada añadiéndolo detrás de la expresión junto con la palabra AS. En nuestro ejemplo quedaría de la siguiente forma:

```
SELECT Nombre, Credito, Credito + 25 AS CreditoNuevo
FROM USUARIOS;
```

Los campos calculados pueden ir en:

- [x] La cláusula SELECT
- [x] La cláusula WHERE
- [ ] La cláusula FROM

Así es, y también podemos añadir un alias a ese campo poniéndolo a continuación.

5.- Funciones.

Caso práctico

__Juan__ le ha pedido a __Ana__ que calcule la edad actual de los usuarios que tienen registrados en la base de datos pues sería interesante realizar estadísticas mensuales sobre los grupos de edad que acceden al sistema y en función de ello obtener algunos resultados interesantes para la empresa. Para realizar el cálculo de la edad tendríamos que echar mano a funciones que nos ayuden con los cálculos. Existen funciones que nos facilitarán la tarea y nos ayudarán a obtener información que de otro modo resultaría complicado.

¿Has pensado en todas las operaciones que puedes realizar con los datos que guardas en una base de datos? Seguro que son muchísimas. Pues bien, en casi todos los Sistemas Gestores de Base de Datos existen funciones ya creadas que facilitan la creación de consultas más complejas. Dichas funciones varían según el SGBD, veremos aquí las que utiliza Oracle.

Las funciones son realmente operaciones que se realizan sobre los datos y que realizan un determinado cálculo. Para ello necesitan unos datos de entrada llamados parámetros o argumentos y en función de éstos, se realizará el cálculo de la función que se esté utilizando. Normalmente los parámetros se especifican entre paréntesis.

Las funciones se pueden incluir en las cláusulas `SELECT`, `WHERE` y `ORDER BY`.

Las funciones se especifican de la siguiente manera:

```
NombreFunción [(parámetro1, [parámetro2, …)]
```

Puedes anidar funciones dentro de funciones.

Existe una gran variedad para cada tipo de datos:
- ✓ numéricas,
- ✓ de cadena de caracteres,
- ✓ de manejo de fechas,
- ✓ de conversión,
- ✓ otras

Oracle proporciona una tabla con la que podemos hacer pruebas, esta tabla se llama Dual y contiene un único campo llamado DUMMY y una sola fila.

Podremos utilizar la tabla Dual en algunos de los ejemplos que vamos a ver en los siguientes apartados.

5.1.- Funciones numéricas.

¿Cómo obtenemos el cuadrado de un número o su valor absoluto? Nos referimos a valores numéricos y por tanto necesitaremos utilizar funciones numéricas.

Para trabajar con campos de tipo número tenemos las siguientes funciones:

Funciones Numéricas	
ABS(n)	Calcula el valor ***absoluto*** de un número **n** `SELECT ABS(-17) FROM DUAL; -- Resultado: 17`
EXP(n)	Calcula $\mathbf{e}^{\mathbf{n}}$, es decir, el ***exponente*** en base **e** del número **n** `SELECT EXP(2) FROM DUAL; -- Resultado: 7,38`
CEIL(n)	Calcula el valor ***entero*** inmediatamente ***superior*** o igual al

	argumento **n** `SELECT CEIL(17.4) FROM DUAL; -- Resultado: 18`
FLOOR(n)	Calcula el valor ***entero*** inmediatamente ***inferior*** o igual al parámetro **n** `SELECT FLOOR(17.4) FROM DUAL; --Resultado: 17`
MOD(m,n)	Calcula el ***resto*** resultante de dividir **m** entre **n** `SELECT MOD(15, 2) FROM DUAL; --Resultado: 1`
POWER(valor, exponente)	***Eleva*** el valor ***al exponente*** indicado `SELECT POWER(4, 5) FROM DUAL; -- Resultado: 1024`
ROUND(n, decimales)	**Redondea** el número **n** al siguiente número con el número de decimales que se indican. `SELECT ROUND(12.5874, 2) FROM DUAL; -- Resultado: 12.59`
SQRT(n)	Calcula la ***raíz cuadrada*** de **n**. `SELECT SQRT(25) FROM DUAL; --Resultado: 5`

TRUNC(m,n)	***Trunca*** un número a la cantidad de decimales especificada por el segundo argumento. Si se omite el segundo argumento, se truncan todos los decimales. Si "**n**" es negativo, el número es truncado desde la parte entera.
	`SELECT TRUNC(127.4567, 2) FROM DUAL; -- Resultado: 127.45`
	`SELECT TRUNC(4572.5678, -2) FROM DUAL; -- Resultado: 4500`
	`SELECT TRUNC(4572.5678, -1) FROM DUAL; -- Resultado: 4570`
	`SELECT TRUNC(4572.5678) FROM DUAL; -- Resultado: 4572`

SIGN(n)	Si el argumento "**n**"es un valor ***positivo***, retorna ***1***, si es ***negativo***, devuelve ***-1*** y 0 si es 0. `SELECT SIGN(-23) FROM DUAL; - Resultado: -1`

Aquí encontrarás las funciones que has visto y algunas más.
http://sites.google.com/site/josepando/home/funciones-sql/funciones-que-devuelven-una-valor-nico-para-cada-fila-de-una-consulta-o-vista/funciones-numricas

5.2.- Funciones de cadena de caracteres.

Ya verás como es muy común manipular campos de tipo carácter o cadena de caracteres. Como resultado podremos obtener caracteres o números. Estas son las funciones más habituales:

Funciones de cadena de caracteres	
CHR(n)	Devuelve el carácter cuyo valor codificado es **n**. `SELECT CHR(81) FROM DUAL; --Resultado: Q`
ASCII(n)	Devuelve el ***valor ASCII*** de **n** `SELECT ASCII('O') FROM DUAL; --Resultado: 79`
CONCAT(cad1, cad2)	Devuelve ***las dos cadenas unidas***. Es equivalente al operador **||** `SELECT CONCAT('Hola', 'Mundo') FROM DUAL; --Resultado: HolaMundo`

LOWER(cad)	Devuelve la cadena **cad** con ***todos*** sus ***caracteres*** en ***minúsculas***.
	`SELECT LOWER('En MInúsculAS') FROM DUAL;`
	`--Resultado: en minúsculas`

UPPER(cad)	Devuelve la cadena **cad** con ***todos*** sus ***caracteres*** en ***mayúsculas***.
	`SELECT UPPER('En MAyúsculAS') FROM DUAL;`
	`--Resultado: EN MAYÚSCULAS`

INITCAP(cad)	Devuelve la cadena **cad** con su ***primer carácter*** en ***mayúscula***. `SELECT INITCAP('hola') FROM DUAL; --Resultado: Hola`
LPAD(cad1, n, cad2)	Devuelve **cad1** con longitud **n**, ajustada a la derecha, ***rellenando por la izquierda*** con **cad2**. `SELECT LPAD('M', 5, '*') FROM DUAL; --Resultado: ****M`

RPAD(cad1, n, cad2)	Devuelve **cad1** con longitud **n**, ajustada a la izquierda, ***rellenando por la derecha*** con **cad2**. `SELECT RPAD('M', 5, '*') FROM DUAL; --Resultado: M****`

REPLACE(cad, ant, nue)	Devuelve ***cad*** en la que cada ocurrencia de la cadena ***ant*** ha sido ***sustituida por*** la cadena ***nue***.
	`SELECT REPLACE('correo@gmail.es', 'es', 'com') FROM DUAL;`
	`--Resultado: correo@gmail.com`

SUBSTR(cad, m, n)	Devuelve la cadena ***cad*** compuesta por ***n caracteres a partir*** de la posición ***m***. `SELECT SUBSTR('1234567', 3, 2) FROM DUAL; --Resultado: 34`
LENGTH(cad)	Devuelve la ***longitud de cad*** `SELECT LENGTH('hola') FROM DUAL; --Resultado: 4`

TRIM(cad)	***Elimina*** los ***espacios*** en blanco a la ***izquierda y*** la ***derecha*** de ***cad*** y los espacios dobles del interior.
	`SELECT TRIM(' Hola de nuevo ') FROM DUAL;`
	`--Resultado: Hola de nuevo`

LTRIM(cad)	***Elimina*** los ***espacios*** a la ***izquierda*** que posea ***cad***. `SELECT LTRIM(' Hola') FROM DUAL; --Resultado: Hola`
RTRIM(cad)	***Elimina*** los ***espacios*** a la ***derecha*** que posea cad. `SELECT RTRIM('Hola ') FROM DUAL; --Resultado: Hola`

INSTR(cad, cadBuscada [, posInicial [, nAparición]])	Obtiene la ***posición*** en la que se encuentra la ***cadena buscada en*** la ***cadena inicial cad***. Se puede comenzar a buscar desde una posición inicial concreta e incluso indicar el número de aparición de la cadena buscada. Si no encuentra nada devuelve cero.
	`SELECT INSTR('usuarios', 'u') FROM DUAL; --Resultado: 1`
	`SELECT INSTR('usuarios', 'u', 2) FROM DUAL; --Resultado: 3`
	`SELECT INSTR('usuarios', 'u', 2, 2) FROM DUAL; --Resultado: 0`

En la siguiente consulta: SELECT LENGTH("Adiós") FROM DUAL; ¿qué obtendríamos?

- ○ 5
- ○ 4
- ○ 6
- ◉ **Nos devolvería un error.**

Cierto, las cadenas van con comillas simples.

5.3.- Funciones de manejo de fechas.

La fecha de emisión de una factura, de llegada de un avión, de ingreso en una web, podríamos seguir poniendo infinidad de ejemplos, lo que significa que es una información que se requiere en muchas situaciones y es importante guardar.

En los SGBD se utilizan mucho las fechas. Oracle tiene dos tipos de datos para manejar fechas, son `DATE` y `TIMESTAMP`.

- ✓ `DATE` almacena **fechas concretas incluyendo a veces la hora.**
- ✓ `TIMESTAMP` almacena **un instante de tiempo más concreto** que puede incluir hasta fracciones de segundo.

Podemos realizar operaciones numéricas con las fechas:

- ✓ Le podemos **sumar números** y esto se entiende como sumarles días, si ese número tiene decimales se suman días, horas, minutos y segundos.
- ✓ La **diferencia** entre dos fechas también nos dará un número de días.

En Oracle tenemos las siguientes funciones más comunes:

Funciones de fecha	
SYSDATE	Devuelve la ***fecha y hora actuales*** `SELECT SYSDATE FROM DUAL; --Resultado: 26/07/11`

SYSTIMESTAMP	Devuelve la ***fecha y hora actuales*** en formato `TIMESTAMP`
	`SELECT SYSTIMESTAMP FROM DUAL;`
	`--Resultado: 26-JUL-11 08.32.59,609000 PM +02:00`

ADD_MONTHS(fecha, n)	***Añade a la fecha*** el número de ***meses*** indicado con ***n***
	`SELECT ADD_MONTHS('27/07/11', 5) FROM DUAL;`
	`--Resultado: 27/12/11`

MONTHS_BETWEEN(fecha1, fecha2)	Devuelve el ***número de meses*** que hay ***entre fecha1*** y ***fecha2***
	`SELECT MONTHS_BETWEEN('12/07/11','12/03/11') FROM DUAL;`
	`--Resultado: 4`

LAST_DAY(fecha)	Devuelve el ***último día del mes*** al que pertenece la fecha. El valor devuelto es tipo `DATE` `SELECT LAST_DAY('27/07/11') FROM DUAL; --Resultado: 31/07/11`

NEXT_DAY(fecha, d)	Indica el ***día*** que corresponde ***si añadimos*** a la fecha el día **d**. El día devuelto puede ser texto ('Lunes', Martes', ..) o el número del día de la semana (1=lunes, 2=martes, ..) dependiendo de la configuración.
	`SELECT NEXT_DAY('31/12/11','LUNES') FROM DUAL;`
	`--Resultado: 02/01/12`

EXTRACT(valor FROM fecha)	Extrae un valor de una fecha concreta. El valor puede ser `day`, `month`, `year`, `hours`, etc. `SELECT EXTRACT(MONTH FROM SYSDATE) FROM DUAL; --Resultado: 7`

En Oracle: Los operadores aritméticos "**+**" (más) y "**-**" (menos) pueden emplearse para las fechas. Por ejemplo:

```
SELECT SYSDATE - 5;
```

Devolvería la fecha correspondiente a 5 días antes de la fecha actual.

Se pueden emplear estas funciones enviando como argumento el nombre de un campo de tipo fecha.

¿Cuáles de estas afirmaciones sobre funciones de manejo de fechas son ciertas?

- ☑ **Existen dos tipos de fechas de datos con las que podemos trabajar, DATE y TIMESTAMP.**
- ☐ Se puede poner como argumento el nombre de un campo de cualquier tipo.
- ☑ **Le podemos sumar o restar números, lo cual se entiende como sumarle o restarle días.**
- ☑ **La diferencia entre dos fechas nos dará un número de días.**

5.4.- Funciones de conversión.

Los SGBD tienen funciones que pueden pasar de un tipo de dato a otro. Oracle convierte automáticamente datos de manera que el resultado de una expresión tenga sentido. Por tanto, de manera automática se pasa de texto a número y al revés. Ocurre lo mismo para pasar de tipo texto a

fecha y viceversa. Pero existen ocasiones en que queramos realizar esas conversiones de modo explícito, para ello contamos con funciones de conversión.

- ✓ TO_NUMBER(cad, formato) Convierte textos en números. Se suele utilizar para dar un formato concreto a los números. Los formatos que podemos utilizar son los siguientes:

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 un número o fecha d a cadena de caracteres, se utiliza normalmente para fechas ya que de número a texto se hace de forma implícita como hemos visto antes.
- ✓ `TO DATE( cad, formato)` Convierte textos a fechas. Podemos indicar el formato con el que queremos que aparezca.

Para las funciones `TO CHAR` y `TO DATE`, en el caso de fechas, indicamos el formato incluyendo los siguientes símbolos:

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 **PM**	Indicador a.m. Indicador p.m.
HH12 **HH24**	Hora de 1 a 12 Hora de 0 a 23
MI	Minutos de 0 a 59
SS **SSSS**	Segundos dentro del minuto Segundos dentro desde las 0 horas

5.5.- Otras funciones: NVL y DECODE.

¿Recuerdas que era el valor `NULL`? Cualquier columna de una tabla podía contener un valor nulo independientemente al tipo de datos que tuviera definido. Eso sí, esto no era así en los casos en que definíamos esa columna como no nula (`NOT NULL`), o que fuera clave primaria (`PRIMARY KEY`).

Cualquier operación que se haga con un valor `NULL` devuelve un `NULL`. Por ejemplo, si se intenta dividir por `NULL`, no nos aparecerá ningún error sino que como resultado obtendremos un `NULL` (no se producirá ningún error tal y como puede suceder si intentáramos dividir por cero).

También es posible que el resultado de una función nos de un valor nulo.

Por tanto, es habitual encontrarnos con estos valores y es entonces cuando aparece la necesidad de poder hacer algo con ellos. Las **funciones con nulos** nos permitirán hacer algo en caso de que aparezca un valor nulo.

- ✓ `NVL(valor, expr1)`

  Si valor es **NULL**, entonces devuelve **expr1**. Ten en cuenta que **expr1** debe ser del mismo tipo que **valor**.

  ¿Y no habrá alguna función que nos permita evaluar expresiones? La respuesta es afirmativa y esa función se llama `DECODE`.

- ✓ `DECODE(expr1, cond1, valor1 [, cond2, valor2, ...], default )`

  Esta función evalúa una expresión **expr1**, si se cumple la primera condición (**cond1**) devuelve el **valor1**, en caso contrario evalúa la siguiente condición y así hasta que una de las condiciones se cumpla. Si no se cumple ninguna condición se devuelve el valor por defecto que hemos llamado **default**.

Si en la tabla EMPLEADOS queremos un listado de sus direcciones, podemos pedir que cuando una dirección no exista, aparezca el texto **No tiene dirección**, para ello podemos utilizar la siguiente consulta:

```
SELECT NVL(DIRECC1, 'No tiene dirección conocida') FROM EMPLEADOS;
```

Obtendremos:

Resultado de la sentencia SELECT

DIRECCIONES

No tiene dirección conocida

C/Sol, 1

¿Qué función convierte un número o fecha a cadena de caracteres?

- ○ TO_DATE.
- ◉ **TO_CHAR.**
- ○ DECODE.
- ○ TO_NUMBER.

Así es, se utiliza normalmente para fechas ya que de número a texto se hace de forma implícita

6.- Consultas de resumen.

Caso práctico

__Ada__ le ha pedido a __Juan__ que le eche una mano en otro de los proyectos en los que está inmersa la empresa. Necesita que cree varias consultas de resumen sobre unas tablas de empleados de banca. Está interesada en obtener el salario medio de los empleados clasificado por tipo de empleado, y quiere también que obtenga el total de empleados por sucursal.

Realmente no es un trabajo difícil ya que las consultas de resumen son muy fáciles de crear, pero __Ada__ está tan ocupada que no tiene tiempo para esos detalles.

Seguro que alguna vez has necesitado realizar cálculos sobre un campo para obtener algún resultado global, por ejemplo, si tenemos una columna donde estamos guardando las notas que obtienen unos alumnos o alumnas en Matemáticas, podríamos estar interesados en saber cual es la nota máxima que han obtenido o la nota media.

La sentencia `SELECT` nos va a permitir **obtener resúmenes de los datos de modo vertical**. Para ello consta de una serie de cláusulas específicas (`GROUP BY`, `HAVING`) y tenemos también unas **funciones** llamadas **de agrupamiento o de agregado** que son las que nos dirán qué cálculos queremos realizar sobre los datos (sobre la columna).

Hasta ahora las consultas que hemos visto daban como resultado un subconjunto de filas de la tabla de la que extraíamos la información. Sin embargo, este tipo de consultas que vamos a ver no corresponde con ningún valor de la tabla sino un **total calculado** sobre los datos de la tabla. Esto hará que las consultas de resumen tengan limitaciones que iremos viendo.

Las funciones que podemos utilizar se llaman de agrupamiento (de agregado). Éstas **toman un grupo de datos** (una columna) y **producen un único dato que resume el grupo**. Por ejemplo, la función `SUM()` acepta una columna de datos numéricos y devuelve la suma de estos.

El simple hecho de utilizar una función de agregado en una consulta la convierte en consulta de resumen.

Todas las funciones de agregado tienen una estructura muy parecida: `FUNCIÓN ([ALL| DISTINCT] Expresión)` y debemos tener en cuenta que:

- ✓ La palabra `ALL` indica que se tienen que tomar todos los valores de la columna. Es el valor por defecto.
- ✓ La palabra `DISTINCT` indica que se considerarán todas las repeticiones del mismo valor como uno solo (considera valores distintos).
- ✓ El grupo de valores sobre el que actúa la función lo determina el resultado de la expresión que será el nombre de una columna o una expresión basada en una o varias columnas. Por tanto, en la expresión nunca puede aparecer ni 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 (excepto `COUNT`) ignoran los valores `NULL`.
- ✓ Podemos encontrar una función de agrupamiento dentro de una lista de selección en cualquier sitio donde pudiera aparecer el nombre de una columna. Es por eso que puede formar parte de una expresión pero no se pueden anidar funciones de este tipo.
- ✓ No se pueden mezclar funciones de columna con nombres de columna ordinarios, aunque hay excepciones que veremos más adelante.

Ya estamos preparados para conocer cuáles son estas funciones de agregado (o agrupamiento). Las veremos a continuación.

Puedes acceder a este enlace si quieres conocer más sobre este tipo de consultas.
http://www.aulaclic.es/sql/t_4_1.htm

6.1.- Funciones de agregado: SUM y COUNT.

Sumar y contar filas o datos contenidos en los campos es algo bastante común. Imagina que para nuestra tabla Usuarios necesitamos sumar el número de créditos total que tienen nuestros jugadores. Con una función que sumara los valores de la columna crédito sería suficiente, siempre y cuando lo agrupáramos por cliente, ya que de lo contrario lo que obtendríamos sería el total de todos los clientes jugadores.

- ✓ La función `SUM`:
  - `SUM([ALL|DISTINCT] expresión)`
  - Devuelve la suma de los valores de la expresión.
  - Sólo puede utilizarse con columnas cuyo tipo de dato sea número. El resultado será del mismo tipo aunque puede tener una precisión mayor.
  - Por ejemplo,
    ```
    SELECT SUM( credito) FROM Usuarios;
    ```
- ✓ La función `COUNT`:
  - `COUNT([ALL|DISTINCT] expresión)`
  - Cuenta los elementos de un campo. Expresión contiene el nombre del campo que deseamos contar. Los operandos de expresión pueden incluir el nombre del campo, una constante o una función.
  - Puede contar cualquier tipo de datos incluido texto.
  - `COUNT` simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan.
  - La función `COUNT` no cuenta los registros que tienen campos `NULL` a menos que **expresión** sea el carácter comodín **asterisco** (*).
  - Si utilizamos `COUNT(*)`, calcularemos el total de filas, incluyendo aquellas que contienen valores `NULL`.
  - Por ejemplo,
    ```
    SELECT COUNT(nombre) FROM Usuarios;
    SELECT COUNT(*) FROM Usuarios;
    ```

Ejercicio resuelto

Utilizando las tablas y datos de la empresa **JuegosCA** descargados anteriormente, vamos a realizar una consulta donde contemos el número de empleados que son mujeres.

Resultado:

```
SELECT COUNT(Nombre) FROM EMPLEADOS WHERE SEXO='M';
```

6.2.- Funciones de agregado: MIN y MAX.

¿Y si pudiéramos encontrar el valor máximo y mínimo de una lista enormemente grande? Esto es lo que nos permiten hacer las siguientes funciones.

- ✓ **Función** `MIN`:
  - `MIN ([ALL| DISTINCT] expresión)`
  - Devuelve el valor mínimo de la expresión sin considerar los nulos (`NULL`).
  - En expresión podemos incluir el nombre de un campo de una tabla, una constante o una función (pero no otras funciones agregadas de SQL).

- Un ejemplo sería:

```
SELECT MIN(credito) FROM Usuarios;
```

- ✓ **Función `MAX`:**
  - `MAX ([ALL| DISTINCT] expresión)`
  - Devuelve el valor máximo de la expresión sin considerar los nulos (`NULL`).
  - En expresión podemos incluir el nombre de un campo de una tabla, una constante o una función (pero no otras funciones agregadas de SQL).
  - Un ejemplo,

```
SELECT MAX (credito) FROM Usuarios;
```

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

Quizás queramos obtener datos estadísticos de los datos guardados en nuestra base de datos. Para ello podemos hacer uso de las funciones que calculan el promedio, la varianza y la desviación típica.

- ✓ **Función `AVG`**
  - `AVG ([ALL| DISTINCT] expresión)`
  - Devuelve el promedio de los valores de un grupo, para ello se omiten los valores nulos (`NULL`).
  - El grupo de valores será el que se obtenga como resultado de la expresión y ésta puede ser un nombre de columna o una expresión basada en una columna o varias de la tabla.
  - Se aplica a campos tipo número y el tipo de dato del resultado puede variar según las necesidades del sistema para representar el valor.
- ✓ **Función `VAR`**
  - `VAR ([ALL| DISTINCT] expresión)`
  - Devuelve la varianza estadística de todos los valores de la expresión.
  - Como tipo de dato admite únicamente columnas numéricas. Los valores nulos (`NULL`) se omiten.
- ✓ **Función `STDEV`**
  - `STDEV ([ALL| DISTINCT] expresión)`
  - Devuelve la desviación típica estadística de todos los valores de la expresión.
  - Como tipo de dato admite únicamente columnas numéricas. Los valores nulos (`NULL`) se omiten.

Ejercicio resuelto

Utilizando las tablas y datos de la empresa **JuegosCA** descargados anteriormente, vamos a realizar una consulta donde obtengamos la media del salario mínimo y máximo de la tabla TRABAJOS.

Resultado:

```
SELECT AVG(SALARIO_MIN), AVG(SALARIO_MAX) FROM TRABAJOS;
```

¿Cuáles de las siguientes afirmaciones sobre las consultas de resumen son ciertas?

- [x] Toman un grupo de datos de una columna.
- [x] Producen un único dato que resume el grupo.
- [x] Utilizar una función de agregado en una consulta la convierte en consulta de resumen.
- [ ] Dan como resultado un subconjunto de filas de la tabla.

7.- Agrupamiento de registros.

Caso práctico

__Juan__ ha estado realizando algunas consultas de resumen y ahora quiere continuar sacando todo el jugo posible a las tablas realizando operaciones como las anteriores pero agrupándolas por algún campo. Hay veces que se obtiene mucha información si estudiamos los datos por grupos, como puede ser el número de jugadores por provincia, o el saldo medio según el sexo del jugador para así poder obtener conclusiones sobre la información guardada.

Hasta aquí las consultas de resumen que hemos visto obtienen totales de todas las filas de un campo o una expresión calculada sobre uno o varios campos. Lo que hemos obtenido ha sido una única fila con un único dato.

Ya verás como en muchas ocasiones en las que utilizamos consultas de resumen nos va a interesar calcular **totales parciales**, es decir, **agrupados según un determinado campo**.

De este modo podríamos obtener de una tabla EMPLEADOS, en la que se guarda su sueldo y su actividad dentro de la empresa, el valor medio del sueldo en función de la actividad realizada en la empresa. También podríamos tener una tabla clientes y obtener el número de veces que ha realizado un pedido, etc.

En todos estos casos en lugar de una única fila de resultados necesitaremos una fila por cada actividad, cada cliente, etc.

Podemos obtener estos **subtotales** utilizando la cláusula `GROUP BY`.

La sintaxis es la siguiente:

```
SELECT columna1, columna2, ...
FROM tabla1, tabla2, ...
WHERE condición1, condición2, …
GROUP BY columna1, columna2, …
HAVING condición
ORDER BY ordenación;
```

En la cláusula `GROUP BY` se colocan las columnas por las que vamos a agrupar. En la cláusula `HAVING` se especifica la condición que han de cumplir los grupos para que se realice la consulta.

Es muy importante que te fijes bien en el orden en el que se ejecutan las cláusulas:

1. `WHERE` que filtra las filas según las condiciones que pongamos.
2. `GROUP BY` que crea una tabla de grupos nueva.
3. `HAVING` filtra los grupos.
4. `ORDER BY` que ordena o clasifica la salida.

Las columnas que aparecen en el `SELECT` y que no aparezcan en la cláusula `GROUP BY` deben tener una función de agrupamiento. Si esto no se hace así producirá un error. Otra opción es poner en la claúsula `GROUP BY` las mismas columnas que aparecen en `SELECT`.

Veamos un par de ejemplos:

```
SELECT provincia, SUM(credito) FROM Usuarios GROUP BY provincia;
```

Obtenemos la suma de créditos de nuestros usuarios agrupados por provincia. Si estuviéramos interesados en la suma de créditos agrupados por provincia pero únicamente de las provincias de Cádiz y Badajóz nos quedaría:

PROVINCIA	COUNT(CREDITO)
BADAJOZ	3
CÁDIZ	4

```
SELECT provincia, SUM(credito) FROM Usuarios GROUP BY provincia HAVING provincia = 'CÁDIZ' OR
provincia= 'BADAJOZ';
```

Relaciona cada cláusula con su orden de ejecución:

Cláusula.	Relación.	Función.
WHERE	**1**	1. PRIMERO
ORDER BY	**4**	2. SEGUNDO
HAVING	**3**	3. TERCERO
GROUP BY	**2**	4. CUARTO

8.- Consultas multitablas.

Caso práctico

*Hasta ahora **Juan** ha estado haciendo uso de consultas a una única tabla, pero eso limita la obtención de resultados. Si esas tablas están relacionadas **Juan** podrá coger información de cada una de ellas según lo que le interese. En las tablas de la empresa **JuegosCA**, tiene por un lado la tabla que recoge los datos del empleado y por otra su historial laboral. En esta última tabla, lo único que recogemos de los empleados es su código. Como a **Juan** le interesa obtener el historial laboral incluyendo nombres y apellidos de sus empleados, debe utilizar la información que viene en ambas tablas.*

Recuerda que una de las propiedades de las bases de datos relacionales era que distribuíamos la información en varias tablas que a su vez estaban relacionadas por algún campo común. Así evitábamos repetir datos. Por tanto, también será frecuente que tengamos que consultar datos que se encuentren distribuidos por distintas tablas.

Si disponemos de una tabla USUARIOS cuya clave principal es Login y esta tabla a su vez está relacionada con la tabla PARTIDAS a través del campo **Cod_Creación**. Si quisiéramos obtener el nombre de los usuarios y las horas de las partidas de cada jugador necesitaríamos coger datos de ambas tablas pues las horas se guardan en la tabla PARTIDAS. Esto significa que cogeremos filas de una y de otra.

Imagina también que en lugar de tener una tabla USUARIOS, dispusiéramos de dos por tenerlas en servidores distintos. Lo lógico es que en algún momento tendríamos que unirlas.

Hasta ahora las consultas que hemos usado se referían a una sola tabla, pero también es posible hacer consultas usando varias tablas en la misma sentencia `SELECT`. Esto permitirá realizar distintas operaciones como son:

- ✓ La composición interna.
- ✓ La composición externa.

En la versión SQL de 1999 se especifica una nueva sintaxis para consultar varias tablas que Oracle incorpora, así que también la veremos. La razón de esta nueva sintaxis era separar las condiciones de asociación respecto a las condiciones de selección de registros.

La sintaxis es la siguiente:

```
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.

¿Qué ocurre si combinamos dos o más tablas sin ninguna restricción? El resultado será un producto cartesiano.

El producto cartesiano entre dos tablas da como resultado todas las combinaciones de todas las filas de esas dos tablas.

Se indica poniendo en la cláusula FROM las tablas que queremos componer separadas por comas. Y puedes obtener el producto cartesiano de las tablas que quieras.

Como lo que se obtiene son todas las posibles combinaciones de filas, debes tener especial cuidado con las tablas que combinas. Si tienes dos tablas de 10 filas cada una, el resultado tendrá 10x10 filas,

a medida que aumentemos el número de filas que contienen las tablas, mayor será el resultado final, con lo cual se puede considerar que nos encontraremos con una operación costosa.

Esta operación no es de las más utilizadas ya que coge una fila de una tabla y la asocia con todos y cada uno de las filas de la otra tabla, independientemente de que tengan relación o no. Lo más normal es que queramos seleccionar los registros según algún criterio.

Necesitaremos **discriminar** de alguna forma para que únicamente aparezcan filas de una tabla que estén relacionadas con la otra tabla. A esto se le llama **asociar tablas** (`JOIN`).

Para hacer una composición interna se parte de un producto cartesiano y se eliminan aquellas filas que no cumplen la condición de composición.

Lo importante en las composiciones internas es emparejar los campos que han de tener valores iguales.

Las reglas para las composiciones son:
- ✓ Pueden combinarse tantas tablas como se desee.
- ✓ El criterio de combinación puede estar formado por más de una pareja de columnas.
- ✓ En la cláusula `SELECT` pueden citarse columnas de ambas tablas, condicionen o no, la combinación.
- ✓ Si hay columnas con el mismo nombre en las distintas tablas, deben identificarse especificando la tabla de procedencia o utilizando un alias de tabla.

Las columnas que aparecen en la cláusula `WHERE` se denominan **columnas de emparejamiento** ya que son las que permiten emparejar las filas de las dos tablas. Éstas no tienen por qué estar incluidas en la lista de selección. Emparejaremos tablas que estén relacionadas entres sí y además, una de las columnas de emparejamiento será clave principal en su tabla. Cuando emparejamos campos debemos especificar de la siguiente forma:

```
NombreTabla1. Camporelacionado1 = NombreTabla2.Camporelacionado2.
```

Puedes combinar una tabla consigo misma pero debes poner de manera obligatoria un alias a uno de los nombres de la tabla que vas a repetir.

Veamos un ejemplo, si queremos obtener el historial laboral de los empleados incluyendo nombres y apellidos de los empleados, la fecha en que entraron a trabajar y la fecha de fin de trabajo si ya no continúan en la empresa, tendremos:

```
SELECT Nombre, Apellido1, Apellido2, Fecha_inicio, Fecha_fin
FROM EMPLEADOS, HISOTRIAL LABORAL
WHERE HISTORIAL_LABORAL.Empleado_DNI= EMPLEADOS.DNI;
```

Vamos a obtener el historial con los nombres de departamento, nombre y apellidos del empleado de todos los departamentos:

```
SELECT Nombre_Dpto, Nombre, Apellido1, Apellido2
FROM DEPARTAMENTOS, EMPLEADOS, HISTORIAL LABORAL
WHERE EMPLEADOS.DNI= HISTORIAL_LABORAL. EMPLEADO_DNI
AND HISTORIAL_LABORAL.DPTO_COD = DEPARTAMENTOS. DPTO_COD;
```

Ejercicio resuelto

Utilizando las tablas y datos de la empresa JuegosCA descargados anteriormente, vamos a realizar una consulta donde obtengamos el nombre de los empleados junto a su salario.

Respuesta:

```
SELECT Nombre, Apellido1, Apellido2, Salario
FROM EMPLEADOS, HISTORIAL_SALARIAL
WHERE HISTORIAL_SALARIAL.Empleado_DNI.= EMPLEADOS.DNI;
```

Ejercicio resuelto

Obtener un listado con el histórico laboral de un empleador cuyo DNI sea '12345'. En dicho listado interesa conocer el nombre del puesto, así como el rango salarial.

Respuesta:

```
SELECT    T.NOMBRE_TRAB,    T.SALARIO_MIN,    T.SALARIO_MAX,    HL.EMPLEADO_DNI,    HL.TRAB_COD,
HL.FECHA INICIO, HL.FECHA FIN, HL.DPTO COD, HL.SUPERVISOR DNI
FROM TRABAJOS T, HISTORIAL LABORAL HL
WHERE T.TRABAJO_COD=HL.TRAB_COD AND
EMPLEADO_DNI='12345';
```

8.2.- Composiciones externas.

¿Has pensado que puede que te interese seleccionar algunas filas de una tabla aunque éstas no tengan correspondencia con las filas de la otra tabla? Esto puede ser necesario.

Imagina que tenemos en una base de datos guardadas en dos tablas la información de los empleados de la empresa (**Cod_empleado, Nombre, Apellidos, salario y Cod_dpto**) por otro lado los departamentos (**Codigo_dep, Nombre**) de esa empresa. Recientemente se ha remodelado la empresa y se han creado un par de departamentos más pero no se les ha asignado los empleados. Si tuviéramos que obtener un informe con los datos de los empleados por departamento, seguro que deben aparecer esos departamentos aunque no tengan empleados. Para poder hacer esta combinación usaremos las composiciones externas.

¿Cómo es el formato? Muy sencillo, añadiremos un signo más entre paréntesis (+) en la igualdad entre campos que ponemos en la cláusula WHERE. El carácter (+) irá detrás del nombre de la tabla en la que deseamos aceptar valores nulos.

En nuestro ejemplo, la igualdad que tenemos en la cláusula WHERE es Cod_dpto (+)= Codigo_dep ya que es en la tabla empleados donde aparecerán valores nulos.

Ejercicio resuelto

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.

Resultado:

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

Si queremos incluir aquellas filas que no tienen aún correspondencia con la tabla relacionada, tendremos que poner un signo más entre paréntesis:

- Delante del nombre de la tabla en la cláusula FROM.
- Delante del nombre del campo que relaciona donde sabemos que hay valores nulos.

- (●) **Detrás del nombre del campo que relaciona donde sabemos que hay valores nulos.**
- ( ) Delante del nombre del campo que relaciona donde sabemos que no hay valores nulos.

8.3.- Composiciones en la versión SQL99.

Como has visto, SQL incluye en esta versión mejoras de la sintaxis a la hora de crear composiciones en consultas. Recuerda que la sintaxis es la siguiente:

```
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)];
```

`CROSS JOIN`: creará un producto cartesiano de las filas de ambas tablas por lo que podemos olvidarnos de la cláusula `WHERE`.

`NATURAL JOIN`: detecta automáticamente las claves de unión, basándose en el nombre de la columna que coincide en ambas tablas. Por supuesto, se requerirá que las columnas de unión tengan el mismo nombre en cada tabla. Además, esta característica funcionará incluso si no están definidas las claves primarias o ajenas.

`JOIN USING`: las tablas pueden tener más de un campo para relacionar y no siempre queremos que se relacionen por todos los campos. Esta cláusula permite establecer relaciones indicando qué campo o campos comunes se quieren utilizar para ello.

`JOIN ON`: se utiliza para unir tablas en la que los nombres de columna no coinciden en ambas tablas o se necesita establecer asociaciones más complicadas.

`OUTER JOIN`: se puede eliminar el uso del signo (+) para composiciones externas utilizando un `OUTER JOIN`, de este modo resultará más fácil de entender.

`LEFT OUTER JOIN`: es una composición externa izquierda, todas las filas de la tabla de la izquierda se devuelven, aunque no haya ninguna columna correspondiente en las tablas combinadas.

`RIGTH OUTER JOIN`: es una composición externa derecha, todas las filas de la tabla de la derecha se devuelven, aunque no haya ninguna columna correspondiente en las tablas combinadas.

`FULL OUTER JOIN`: es una composición externa en la que se devolverán todas las filas de los campos no relacionados de ambas tablas.

Podríamos transformar algunas de las consultas con las que hemos estado trabajando:

Queríamos obtener el historial laboral de los empleados incluyendo nombres y apellidos de los empleados, la fecha en que entraron a trabajar y la fecha de fin de trabajo si ya no continúa en la empresa. Es una consulta de composición interna, luego utilizaremos `JOIN ON`:

```
SELECT E.Nombre, E.Apellido1, E.Apellido2, H.Fecha_inicio, H.Fecha_fin
FROM EMPLEADOS E JOIN HISTORIAL_LABORAL H ON (H.Empleado_DNI= E.DNI);
```

Queríamos también, 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. Aquí estamos ante una composición externa, luego podemos utilizar `OUTER JOIN`:

```
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);
```

En MySQL también se utilizan las composiciones, aquí puedes verlo:
http://mysql.conclase.net/curso/?cap=012a#MUL_JOIN

9.- Otras consultas multitablas: Unión, Intersección y diferencia de consultas.

Caso práctico

Ana le cuenta a Carlos que ya tienen terminado casi todo el trabajo, pero que no le importa enseñarle otros tipos de consultas que no han necesitado utilizar en esta ocasión pero que es conveniente conocer, se refiere al uso de uniones, intersecciones y diferencia de consultas. Le explicará que es muy parecido a la teoría de conjuntos que recordará de haber terminado hace poco sus estudios.

Seguro que cuando empieces a trabajar con bases de datos llegará un momento en que dispongas de varias tablas con los mismos datos guardados para distintos registros y quieras unirla en una única tabla. ¿Esto se puede hacer? Es una operación muy común junto a otras. Al fin y al cabo, una consulta da como resultado un conjunto de filas y con conjuntos podemos hacer entre otras, tres tipos de operaciones comunes como son: unión, intersección y diferencia.

`UNION`: combina las filas de un primer `SELECT` con las filas de otro `SELECT`, desapareciendo las filas duplicadas.

`INTERSECT`: examina las filas de dos `SELECT` y devolverá aquellas que aparezcan en ambos conjuntos. Las filas duplicadas se eliminarán.

`MINUS`: devuelve aquellas filas que están en el primer `SELECT` pero no en el segundo. Las filas duplicadas del primer `SELECT` se reducirán a una antes de comenzar la comparación.

Para estas tres operaciones es muy **importante** que utilices en los dos `SELECT` el **mismo número** y tipo de **columnas** y en el **mismo orden**.

Estas operaciones se pueden combinar anidadas, pero es conveniente utilizar paréntesis para indicar que operación quieres que se haga primero.

Veamos un ejemplo de cada una de ellas.

`UNIÓN`**: Obtener los nombres y ciudades de todos los proveedores y clientes de Alemania.**

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

I

`NTERSECCIÓN`**: Una academia de idiomas da clases de inglés, frances y portugues; almacena los datos de los alumnos en tres tablas distintas una llamada "ingles", en una tabla denominada "frances" y los que aprenden portugues en la tabla "portugues". La academia necesita el nombre y domicilio de todos los alumnos que cursan los tres idiomas para enviarles iformación sobre los exámenes.**

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

`DIFERENCIA`**: Ahora la academia necesita el nombre y domicilio solo de todos los alumnos que cursan inglés (no quiere a los que ya cursan portugués pues va a enviar publicidad referente al curso de portugués).**

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

¿Cuáles de las siguientes afirmaciones son correctas?

- [x] La unión combina las filas de un primer SELECT con las filas de otro SELECT, desapareciendo las filas duplicadas.

- [x] **La diferencia devuelve aquellas filas que están en el primer SELECT pero no en el segundo. Correcta.**
- [x] **La intersección examina las filas de un SELECT y de otro y devolverá aquellas que aparezcan en ambos conjuntos.**
- [x] **En uniones, intersecciones y diferencias, los dos SELECT deben tener el mismo número pero no tienen por qué tener el mismo tipo de columnas.**

10.- Subconsultas.

Caso práctico

*— ¿Es posible consultar dentro de otra consulta? — pregunta **Carlos**.*

*Ha estado pensando que a veces va a necesitar filtrar los datos en función de un resultado que a priori desconoce. **Ana** se pone manos a la obra porque ve que ha llegado el momento de explicarle a **Carlos** las subconsultas.*

A veces tendrás que utilizar en una consulta los resultados de otra que llamaremos **subconsulta**. La sintaxis es:

```
SELECT listaExpr
FROM tabla
WHERE expresión OPERADOR
     ( SELECT listaExpr
     FROM tabla);
```

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

El **OPERADOR** puede ser **>**, **<**, **>=**, **<=**, **!=**, **=** o **IN**. Las subconsultas que se utilizan con estos operadores devuelven un único valor, si la subconsulta devolviera más de un valor devolvería un error.

Como puedes ver en la sintaxis, las subconsultas deben ir entre paréntesis y a la derecha del operador.

Pongamos un ejemplo:

```
SELECT Nombre_empleado, sueldo
FROM EMPLEADOS
WHERE SUELDO <
     (SELECT SUELDO FROM EMPLEADOS
     WHERE Nombre_emple = 'Ana');
```

Obtendríamos el nombre de los empleados y el sueldo de aquellos que cobran menos que Ana.

Los tipos de datos que devuelven la subconsulta y la columna con la que se compara ha de ser el mismo.

¿Qué hacemos si queremos comparar un valor con varios, es decir, si queremos que la subconsulta devuelva más de un valor y comparar el campo que tenemos con dichos valores? Imagina que queremos ver si el sueldo de un empleado que es administrativo es mayor o igual que el sueldo medio de otros puestos en la empresa. Para saberlo deberíamos calcular el sueldo medio de las demás ocupaciones que tiene la empresa y éstos compararlos con la de nuestro empleado. Como ves, el resultado de la subconsulta es más de una fila. ¿Qué hacemos?

Cuando el resultado de la subconsulta es más de una fila, SQL utiliza **instrucciones especiales** entre el operador y la consulta. Estas instrucciones son:

- ✓ `ANY`. Compara con cualquier fila de la consulta. La instrucción es válida si hay un registro en la subconsulta que permite que la comparación sea cierta.
- ✓ `ALL`. Compara con todas las filas de la consulta. La instrucción resultará cierta si es cierta toda la comparación con los registros de la subconsulta.
- ✓ `IN`. No utiliza comparador, lo que hace es comprobar si el valor se encuentra en el resultado de la subconsulta.
- ✓ `NOT IN`. Comprueba si un valor no se encuentra en una subconsulta.

En la siguiente consulta obtenemos el empleado que menos cobra:

```
SELECT nombre, sueldo
FROM EMPLEADOS
WHERE sueldo <= ALL (SELECT sueldo FROM EMPLEADOS);
```

Relaciona cada instrucción con su función:

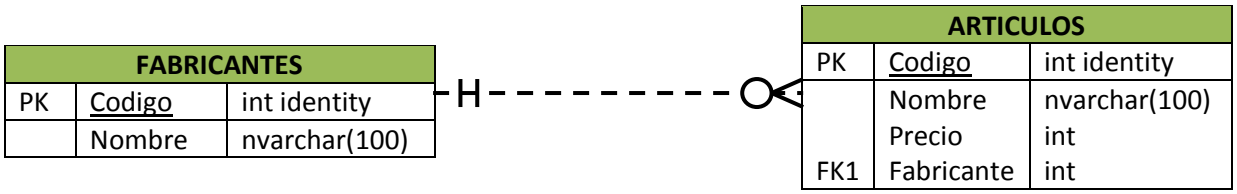
Instrucción.	Relación.	Función.
ANY	**1**	1. Compara con cualquier fila de la consulta.
ALL	**3**	2. Comprueba si el valor se encuentra en el resultado de la subconsulta.
IN	**2**	3. Compara con todas las filas de la consulta.
OT IN	**4**	4. Comprueba si un valor no se encuentra en una subconsulta.

¿Quieres más ejemplos con los que practicar?

http://superalumnos.net/ejercicios_resueltos_de_sql

Varios ejercicios SQL resueltos

La tienda de informática



1. Obtener los nombres de los productos de la tienda

```
SELECT Nombre FROM ARTICULOS
```

2. Obtener los nombres y los precios de los productos de la tienda

```
SELECT Nombre, Precio FROM ARTICULOS
```

3. Obtener el nombre de los productos cuyo precio sea menor o igual a 200€

```
SELECT Nombre FROM ARTICULOS WHERE Precio > 200
```

4. Obtener todos los datos de los artículos cuyo precio esté entre los 60€ y los 120€ (ambas cantidades incluidas)

```
/* Con AND */
SELECT * FROM ARTICULOS
   WHERE Precio >= 60 AND Precio <= 120

/* Con BETWEEN */
SELECT * FROM ARTICULOS
   WHERE Precio BETWEEN 60 AND 120
```

5. Obtener el nombre y el precio en pesetas (es decir, el precio en euros multiplicado por 166,386)

```
/* Sin AS */
SELECT Nombre, Precio * 166.386 FRO ARTICULOS

/* Con AS */
SELECT Nombre, Precio * 166.386 AS PrecioPtas FROM ARTICULOS
```

6. Seleccionar el precio medio de todos los productos

```
SELECT AVG(Precio) FROM ARTICULOS
```

7. Obtener el precio medio de los artículos cuyo código de fabricante sea 2

```
SELECT AVG(Precio) FROM ARTICULOS WHERE Fabricante=2
```

8. Obtener el número de artículos cuyo precio sea mayor o igual a 180€

```
SELECT COUNT(*) FROM ARTICULOS WHERE Precio >= 180
```

9. Obtener el nombre y precio de los artículos cuyo precio sea mayor o igual a 180€ y ordenarlos descendentemente por precio, y luego ascendentemente por nombre

```
SELECT Nombre, Precio FROM ARTICULOS
   WHERE Precio >= 180
   ORDER BY Precio DESC, Nombre
```

10. Obtener un listado completo de artículos, incluyendo por cada artículo los datos del artículo y de su fabricante

```
/* Sin INNER JOIN */
SELECT * FROM ARTICULOS, FABRICANTES
   WHERE ARTICULOS.Fabricante = FABRICANTES.Codigo

/* Con INNER JOIN */
```

```
SELECT *
   FROM ARTICULOS INNER JOIN FABRICANTES
   ON ARTICULOS.Fabricante = FABRICANTES.Codigo
```

11. Obtener un listado de artículos, incluyendo el nombre del artículo, su precio, y el nombre de su fabricante

```
/* Sin INNER JOIN */
SELECT ARTICULOS.Nombre, Precio, FABRICANTES.Nombre
   FROM ARTICULOS, FABRICANTES
   WHERE ARTICULOS.Fabricante = FABRICANTES.Codigo

/* Con INNER JOIN */
SELECT ARTICULOS.Nombre, Precio, FABRICANTES.Nombre
   FROM ARTICULOS INNER JOIN FABRICANTES
   ON ARTICULOS.Fabricante = FABRICANTES.Codigo
```

12. Obtener el precio medio de los productos de cada fabricante, mostrando solo los códigos de fabricante

```
SELECT AVG(Precio), Fabricante FROM ARTICULOS
   GROUP BY Fabricante
```

13. Obtener el precio medio de los productos de cada fabricante, mostrando el nombre del fabricante

```
/* Sin INNER JOIN */
SELECT AVG(Precio), FABRICANTES.Nombre
   FROM ARTICULOS, FABRICANTE
   WHERE ARTICULOS.Fabricante = FABRICANTES.Codigo
   GROUP BY FABRICANTES.Nombre

/* Con INNER JOIN */
SELECT AVG(Precio), FABRICANTES.Nombre
   FROM ARTICULOS INNER JOIN FABRICANTES
   ON ARTICULOS.Fabricante = FABRICANTES.Codigo
   GROUP BY FABRICANTES.Nombre
```

14. Obtener los nombres de los fabricantes que ofrezcan productos cuyo precio medio sea mayor o igual a 150€

```
/* Sin INNER JOIN */
SELECT AVG(Precio), FABRICANTES.Nombre
   FROM ARTICULOS, FABRICANTES
   WHERE ARTICULOS.Fabricante = FABRICANTES.Codigo
   GROUP BY FABRICANTES.Nombre
   HAVING AVG(Precio) >= 150

/* Con INNER JOIN */
SELECT AVG(Precio), FABRICANTES.Nombre
   FROM ARTICULOS INNER JOIN FABRICANTES
   ON ARTICULOS.Fabricante = FABRICANTES.Codigo
   GROUP BY FABRICANTES.Nombre
   HAVING AVG(Precio) >= 150
```

15. Obtener el nombre y precio del artículo más barato

```
SELECT Nombre, Precio
   FROM ARTICULOS
   WHERE Precio = (SELECT MIN(Precio) FROM ARTICULOS)
```

16. Obtener una lista con el nombre y precio de los artículos más caros de cada proveedor (incluyendo el nombre del proveedor)

```
/* Sin INNER JOIN */
SELECT A.Nombre, A.Precio, F.Nombre
   FROM ARTICULOS A, FABRICANTES F
   WHERE A.Fabricante = F.Codigo
      AND A.Precio =
      (
         SELECT MAX(A.Precio)
            FROM ARTICULOS A
```

```
            WHERE A.Fabricante = F.Codigo
      )

/* Con INNER JOIN */
SELECT A.Nombre, A.Precio, F.Nombre
   FROM ARTICULOS A INNER JOIN FABRICANTES F
   ON A.Fabricante = F.Codigo
      AND A.Precio =
      (
         SELECT MAX(A.Precio)
            FROM ARTICULOS A
            WHERE A.Fabricante = F.Codigo
      )
```

17. Añadir un nuevo producto: Altavoces de 70€ (del fabricante 2)

```
INSERT INTO ARTICULOS(Nombre, Precio, Fabricante)
   VALUES ( 'Altavoces', 70, 2 )
```

18. Cambiar el nombre del producto 8 a 'Impresora Laser'

```
UPDATE ARTICULOS
   SET Nombre = 'Impresora Laser'
   WHERE Codigo = 8
```

19. Aplicar un descuento del 10% (multiplicar el precio por 0,9) a todos los productos

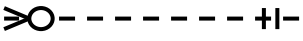
```
UPDATE ARTICULOS
   SET Precio = Precio * 0.9
```

20. Aplicar un descuento de 10€ a todos los productos cuyo precio sea mayor o igual a 120€

```
UPDATE ARTICULOS
   SET Precio = Precio - 10
   WHERE Precio >= 120
```

Empleados

EMPLEADOS		
PK	DNI	varchar(8)
	Nombre	nvarchar(100)
	Apellidos	nvarchar(255)
FK1	Departamento	int



DEPARTAMENTOS		
PK	Codigo	int
	Nombre	nvarchar(100)
	Presupuesto	int

1. Obtener los apellidos de los empleados

```
SELECT Apellidos FROM EMPLEADOS
```

2. Obtener los apellidos de los empleados sin repeticiones

```
SELECT DISTINCT Apellidos FROM EMPLEADOS
```

3. Obtener todos los datos de los empleados que se apellidan 'López'

```
SELECT * FROM EMPLEADOS WHERE Apellidos = 'López'
```

4. Obtener todos los datos de los empleados que se apellidan 'López' y los que se apellidan 'Pérez'

```
/* Con OR */
SELECT * FROM EMPLEADOS
   WHERE Apellidos = 'López' OR Apellidos = 'Pérez'

/* Con IN */
SELECT * FROM EMPLEADOS
   WHERE Apellidos IN ('López' , 'Pérez')
```

5. Obtener todos los datos de los empleados que trabajan para el departamento 14

```
SELECT * FROM EMPLEADOS WHERE Departamento = 14
```

6. Obtener todos los datos de los empleados que trabajan para el departamento 37 y para el departamento 77

```
/* Con OR */
SELECT * FROM EMPLEADOS
   WHERE Departamento = 37 OR Departamento = 77

/* Con IN */
SELECT * FROM EMPLEADOS
   WHERE Departamento IN (37,77)
```

7. Obtener todos los datos de los empleados cuyo apellido comience por 'P'

```
SELECT * FROM EMPLEADOS
   WHERE Apellidos LIKE 'P%'
```

8. Obtener el presupuesto total de todos los departamentos

```
SELECT SUM(Presupuesto) FROM DEPARTAMENTOS
```

9. Obtener el número de empleados en cada departamento

```
SELECT Departamento, COUNT(*)
   FROM EMPLEADOS
   GROUP BY Departamento
```

10. Obtener un listado completo de empleados, incluyendo por cada empleado los datos del empleado y de su departamento

```
SELECT *
   FROM EMPLEADOS INNER JOIN DEPARTAMENTOS
   ON EMPLEADOS.Departamento = DEPARTAMENTOS.Codigo
```

11. Obtener un listado completo de empleados, incluyendo el nombre y apellidos del empleado junto al nombre y presupuesto de su departamento.

```
/* Sin etiquetas */
SELECT EMPLEADOS.Nombre, Apellidos, DEPARTAMENTOS.Nombre, Prespuesto
   FROM EMPLEADOS INNER JOIN DEPARTAMENTOS
   ON EMPLEADOS.Departamento = DEPARTAMENTOS.Codigo

/* Con etiquetas */
SELECT E.Nombre, Apellidos, D.Nombre, Presupuesto
   FROM EMPLEADOS E INNER JOIN DEPARTAMENTOS D
   ON E.Departamento = D.Codigo
```

12. Obtener los nombres y apellidos de los empleados que trabajan en departamentos cuyo presupuesto sea mayor de 60.000€

```
/* Sin subconsulta */
SELECT EMPLEADOS.Nombre, Apellidos
   FROM EMPLEADOS INNER JOIN DEPARTAMENTOS
   ON EMPLEADOS.Departamento = DEPARTAMENTOS.Codigo
      AND DEPARTAMENTOS.Presupuesto > 60000

/* Con subconsulta */
SELECT Nombre, Apellidos FROM EMPLEADOS
   WHERE Departamento IN
   (SELECT Codigo FROM DEPARTAMENTOS WHERE Presupuesto > 60000)
```

13. Obtener los datos de los departamentos cuyo presupuesto es superior al presupuesto medio de todos los departamentos

```
SELECT * FROM DEPARTAMENTOS
   WHERE Presupuesto >
   (
      SELECT AVG(Presupuesto)
      FROM DEPARTAMENTOS
```

```
   )
```

14. Obtener los nombres (únicamente los nombres) de los departamentos que tienen más de dos empleados

```
/* Con subconsulta */
SELECT Nombre FROM DEPARTAMENTOS
   WHERE Codigo IN
   (
      SELECT Departamento
         FROM EMPLEADOS
         GROUP BY Departamento
         HAVING COUNT(*) > 2
   )

/* Con UNION. No funciona si dos departamentos tienen el mismo nombre */
SELECT DEPARTAMENTOS.Nombre
   FROM EMPLEADOS INNER JOIN DEPARTAMENTOS
   ON Departamento = Codigo
   GROUP BY DEPARTAMENTOS.Nombre
   HAVING COUNT(*) > 2
```

15. Añadir un nuevo departamento: 'Calidad', con presupuesto de 40.000€ y código 11. Añadir un empleado vinculado al departamento recién creado: Esther Vázquez, DNI: 89267109

```
INSERT INTO DEPARTAMENTOS
   VALUES(11,'Calidad',40000)

INSERT INTO EMPLEADOS
   VALUES('89267109','Esther','Vázquez',11)
```

16. Aplicar un recorte presupuestario del 10% a todos los departamentos

```
UPDATE DEPARTAMENTOS SET Presupuesto = Presupuesto * 0.9
```

17. Reasignar a los empleados del departamento de investigación (código 77) al departamento de informática (código 14)

```
UPDATE EMPLEADOS SET Departamento = 14 WHERE Departamento = 77
```

18. Despedir a todos los empleados que trabajan para el departamento de informática (código 14)

```
DELETE FROM EMPLEADOS
   WHERE Departamento = 14
```

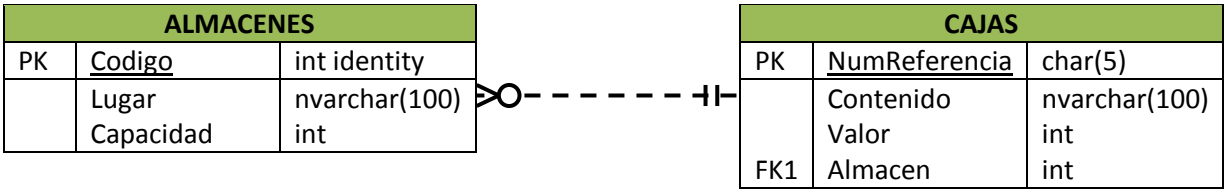
19. Despedir a todos los empleados que trabajen para departamentos cuyo presupuesto sea superior a los 60.000€

```
DELETE FROM EMPLEADOS
   WHERE Departamento IN
   (
      SELECT Codigo FROM DEPARTAMENTO
         WHERE Presupuesto >= 60000
   )
```

20. Despedir a todos los empleados

```
DELETE FROM EMPLEADOS
```

Los Almacenes



1. Obtener todos los almacenes

```
SELECT * FROM ALMACENES
```

2. Obtener todas las cajas cuyo contenido tenga un valor superior a 150€

```
SELECT * FROM CAJAS WHERE Valor > 150
```

3. Obtener los tipos de contenidos de las cajas

```
SELECT DISTINCT Contenido FROM CAJAS
```

4. Obtener el valor medio de todas las cajas

```
SELECT AVG(Valor) FROM CAJAS
```

5. Obtener el valor medio de las cajas de cada almacén

```
SELECT Almacen, AVG(Valor)
   FROM CAJAS
   GROUP BY Almacen
```

6. Obtener los códigos de los almacenes en los cuales el valor medio de las cajas sea superior a 150€

```
SELECT Almacen, AVG(Valor)
   FROM CAJAS
   GROUP BY Almacen
   HAVING AVG(Valor > 150
```

7. Obtener el número de referencia de cada caja junto con el nombre de la ciudad en la que se encuentra.

```
SELECT NumReferencia, Lugar
   FROM ALMACENES INNER JOIN CAJAS
   ON ALMACENES.Codigo = CAJAS.Almacen
```

8. Obtener el número de cajas que hay en cada almacén

```
/* Esta consulta no tiene en cuenta los almacenes vacíos */
SELECT Almacen, COUNT(*)
   FROM CAJAS
   GROUP BY Almacen

/* Esta consulta tiene en cuenta los almacenes vacíos */
SELECT Codigo, COUNT(NumReferencia)
   FROM ALMACENES LEFT JOIN CAJAS
   ON ALMACENES.Codigo = CAJAS.Almacen
   GROUP BY Codigo
```

9. Obtener los códigos de los almacenes que están saturados (los almacenes donde el número de cajas es superior a la capacidad)

```
SELECT Codigo
   FROM ALMACENES
   WHERE Capacidad <
   (
      SELECT COUNT(*)
         FROM CAJAS
         WHERE Almacen = Codigo
   )
```

10. Obtener los números de referencia de las cajas que están en Bilbao

```
/* Sin subconsultas */
SELECT NumReferencia
   FROM ALMACENES LEFT JOIN CAJAS
   ON ALMACENES.Codigo = CAJAS.Almacen
   WHERE Lugar = 'Bilbao'

/* Con subconsultas */
SELECT NumReferencia
   FROM CAJAS
   WHERE Almacen IN
   (
      SELECT Codigo
         FROM ALMACENES
         WHERE Lugar = 'Bilbao'
   )
```

11. Insertar un nuevo almacén en Barcelona con capacidad para 3 cajas

```
INSERT INTO ALMACENES(Lugar,Capacidad) VALUES('Barcelona',3)
```

12. Insertar una nueva caja, con número de referencia 'H5RT', con contenido 'Papel, valor 200, y situada en el almacén 2

```
INSERT INTO CAJAS
   VALUES('H5RT','Papel',200,2)
```

13. Rebajar el valor de todas las cajas un 15%

```
UPDATE CAJAS SET Valor = Valor * 0.85
```

14. Rebajar un 20% el valor de todas las cajas cuyo valor sea superior al valor medio de todas las cajas

```
UPDATE CAJAS SET Valor = Valor * 0.80
   WHERE Valor > (SELECT AVG(Valor) FROM CAJAS)
```

15. Eliminar todas las cajas cuyo valor sea inferior a 100€

```
DELETE FROM CAJAS WHERE Valor < 100
```

16. Vaciar el contenido de los almacenes que están saturados

```
DELETE FROM CAJAS WHERE Almacen IN
   (
      SELECT Codigo
         FROM ALMACENES
         WHERE Capacidad <
         (
            SELECT COUNT(*)
               FROM CAJAS
                  WHERE Almacen = Codigo
         )
   )
```

Películas y Salas

SALAS		
PK	Codigo	int identity
	Nombre	nvarchar(100)
FK1	Película	int

PELICULAS		
PK	Codigo	int identity
	Nombre	nvarchar(100)
	CalificacionEdad	int

1. Mostrar el nombre de todas las películas

```
SELECT Nombre FROM PELICULAS
```

2. Mostrar las distintas calificaciones de edad que existen

```
SELECT DISTINCT CalificacionEdad FROM PELICULAS
```

3. Mostrar todas las películas que no han sido calificadas

```
SELECT * FROM PELICULAS WHERE CalificacionEdad IS NULL
```

4. Mostrar todas las salas que no proyectan ninguna película

```
SELECT * FROM SALAS WHERE Pelicula IS NULL
```

5. Mostrar la información de todas las salas y, si se proyecta alguna película en la sala, mostrar también la información de la película

```
SELECT *
   FROM SALAS LEFT JOIN PELICULAS
   ON SALAS.Pelicula = PELICULAS,Codigo
```

6. Mostrar la información de todas las películas y, si se proyecta en alguna sala, mostrar también la información de la sala

```
SELECT *
   FROM SALAS RIGHT JOIN PELICULAS
   ON SALAS.Pelicula = PELICULAS.Codigo
```

7. Mostrar los nombres de las películas que no se proyectan en ninguna sala

```
/* Con JOIN */
SELECT PELICULAS.Nombre
   FROM SALAS RIGHT JOIN PELICULAS
   ON SALAS.Pelicula = PELICULAS.Codigo
   WHERE SALAS.Pelicula IS NULL

/* Con Subconsulta */
SELECT Nombre FROM PELICULAS
   WHERE Codigo NOT IN
   (
      SELECT Pelicula FROM SALAS
      WHERE Pelicula IS NOT NULL
   )
```

8. Añadir una nueva película 'Uno, Dos, Tres', para mayores de 7 años

```
INSERT INTO PELICULAS(Nombre,CalificacionEdad) VALUES('Uno, Dos, Tres', 7)
```

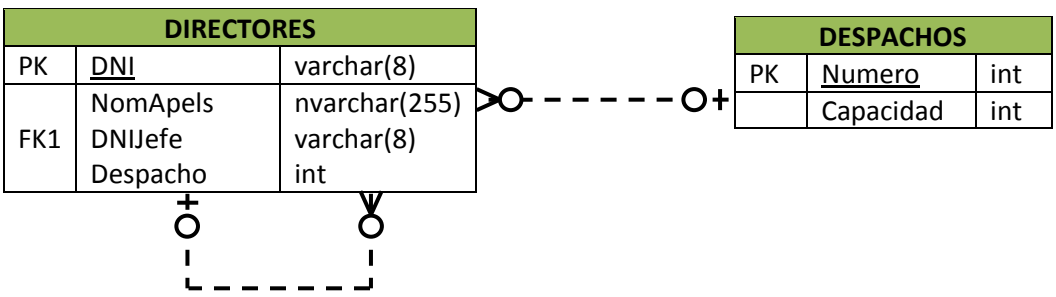
9. Hacer constar que todas las películas no calificadas han sido calificadas 'no recomendables para menores de 13 años'

```
UPDATE PELICULAS SET CalificacionEdad=13 WHERE CalificacionEdad IS NULL
```

10. Eliminar todas las salas que proyectan películas recomendadas para todos los públicos

```
DELETE FROM SALAS WHERE Pelicula IN
   (SELECT Codigo FROM PELICULAS WHERE CalificacionEdad = 0)
```

Los Directores



1. Mostrar el DNI, nombre y apellidos de todos los directores

```
SELECT DNI, NomApels FROM DIRECTORES
```

2. Mostrar los datos de los directores que no tienen jefes

```
SELECT * FROM DIRECTORES WHERE DNIJefe IS NULL
```

3. Mostrar el nombre y apellidos de cada director, junto con la capacidad del despacho en el que se encuentra

```
SELECT NomApels, Despacho, Capacidad
   FROM DIRECTORES INNER JOIN DESPACHOS
   ON DIRECTORES.Despacho = DESPACHOS.Numero
```

4. Mostrar el número de directores que hay en cada despacho

```
/* Sin tener en cuenta despachos vacíos */
SELECT Despacho, COUNT(*)
   FROM DIRECTORES
   GROUP BY Despacho

/* Teniendo en cuenta despachos vacíos */
SELECT Numero, COUNT(DNI)
   FROM DESPACHOS LEFT JOIN DIRECTORES
   ON DESPACHOS.Numero = DIRECTORES.Despacho
   GROUP BY Numero
```

5. Mostrar los datos de los directores cuyos jefes no tienen jefes

```
SELECT * FROM DIRECTORES
   WHERE DNIJefe IN
   (SELECT DNI FROM DIRECTORES WHERE DNIJefe IS NULL)
```

6. Mostrar los nombres y apellidos de los directores junto con los de su jefe

```
/* Con INNER JOIN. No muestra directores que no tienen jefes */
SELECT d1.NomApels, d2.NomApels
   FROM DIRECTORES d1 INNER JOIN DIRECTORES d2
   ON d1.DNIJefe = d2.DNI

/* Con LEFT JOIN. Si muestra directores sin jefe */
SELECT d1.NomApels, d2.NomApels
   FROM DIRECTORES d1 LEFT JOIN DIRECTORES d2
   ON d1.DNIJefe = d2.DNI
```

7. Mostrar el número de despachos que están sobreutilizados

```
SELECT Numero FROM DESPACHOS
   WHERE Capacidad <
   (
      SELECT COUNT(*)
      FROM DIRECTORES
      WHERE Despacho = Numero
   )
```

8. Añadir un nuevo director llamado Paco Pérez, DNI 28301700, sin jefe, y situado en el despacho 124

```
INSERT INTO DIRECTORES VALUES('28301700', 'Paco Pérez', NULL, 124)
```

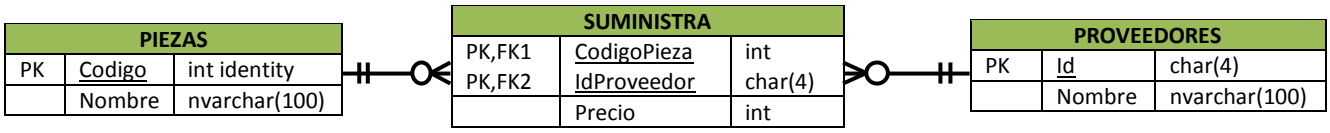
9. Asignar a todos los empleados apellidados Pérez un nuevo jefe con DNI 74568521

```
UPDATE DIRECTORES SET DNIJefe = '74568521' WHERE NomApels LIKE '%Pérez%'
```

10. Despedir a todos los directores, excepto a los que no tienen jefe

```
DELETE FROM DIRECTORES WHERE DNIJefe IS NOT NULL
```

Piezas y Proveedor



1. Obtener los nombres de todas las piezas

```
SELECT Nombre FROM PIEZAS
```

2. Obtener todos los datos de todos los proveedores

```
SELECT * FROM PROVEEDORES
```

3. Obtener el precio medio al que se nos suministran las piezas

```
   SELECT CodigoPieza, AVG(Precio)
      FROM SUMINISTRA
      GROUP BY CodigoPieza
```

4. Obtener los nombres de los proveedores que suministran la pieza 1

```
/* Sin subconsulta */
SELECT PROVEEDORES.Nombre
   FROM PROVEEDORES INNER JOIN SUMINISTRA
   ON PROVEEDORES.Id = SUMINISTRA.IdProveedor
      AND SUMINISTRA.CodigoPieza = 1

/* Con subconsulta */
SELECT Nombre
   FROM PROVEEDORES
   WHERE Id IN
   (SELECT IdProveedor FROM SUMINISTRA WHERE CodigoPieza = 1)
```

5. Obtener los nombres de las piezas suministradas por el proveedor coyo código es HAL

```
/* Sin subconsulta */
SELECT PIEZAS.Nombre
   FROM PIEZAS INNER JOIN SUMINISTRA
   ON PIEZAS.Codigo = SUMINISTRA.CodigoPieza
      AND SUMINISTRA.IdProveedor = 'HAL'

/* Con subconsultas IN */
SELECT Nombre
   FROM PIEZAS
   WHERE Codigo IN
      (SELECT CodigoPieza FROM SUMINISTRA WHERE IdProveedor = 'HAL')

/* Con subconsulta EXISTS */
SELECT Nombre
   FROM PIEZAS
   WHERE EXISTS
   (
      SELECT * FROM SUMINISTRA
         WHERE IdProveedor = 'HAL'
            AND CodigoPieza = PIEZAS.Codigo
   )
```

6. Obtener los nombres de los proveedores que suministran las piezas más caras indicando el nombre de la pieza y el precio al que la suministran

```
SELECT p1.Nombre, pr1.Nombre, Precio
   FROM PIEZAS p1 INNER JOIN
      (SUMINISTRA s1 INNER JOIN PROVEEDORES pr1
      ON s1.IdProveedor = pr1.Id)
   ON p1.Codigo = s1.CodigoPieza
   WHERE Precio IN
   (
      SELECT MAX(Precio) FROM SUMINISTRA s2
      GROUP BY s2.CodigoPieza
      HAVING s2.CodigoPieza = p1.Codigo
```

```
   )
```

7. Hacer constar en la base de datos que la empresa “Skellington Supplies” (código TNBC) va a empezar a suministrarnos tuercas (códio 1) a 7 pesetas cada tuerca.

```
INSERT INTO SUMINISTRA VALUES('TNBC',1,7)
```

8. Aumentar los precios en una unidad

```
UPDATE SUMINISTRA SET Precio = Precio + 1
```

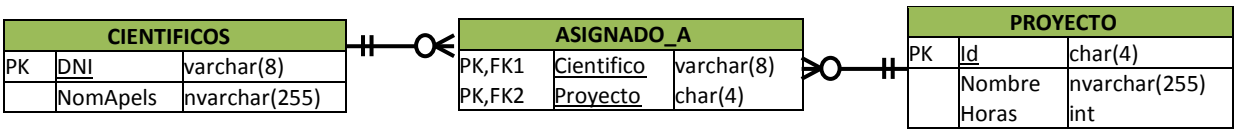
9. Hacer constar en la base de datos que la empresa “Susan Calvin Corp” (RBT) no va a suministrarnos ninguna pieza (aunque la empresa en sí va a seguir constando en nuestra base de datos)

```
DELETE FROM SUMINISTRA WHERE IdProveedor = 'RBT'
```

10. Hacer constar en la base de datos que la empresa “Susan Calvin Corp.” (RBT) ya no va a suministrarnos clavos (código 4)

```
DELETE FROM SUMINISTRA
   WHERE IdProveedor = 'RBT'
      AND CodigoPieza = 4
```

Los Científicos



1. Sacar una relación completa de los científicos asignados a cada proyecto. Mostrar DNI, Nombre del científico, Identificador del proyecto y nombre del proyecto

```
/* Sin JOIN */
SELECT DNI, NomApels, Id, Nombre
   FROM CIENTIFICOS C, ASIGNADO_A A, PROYECTO P
   WHERE C.DNI = A.Cientifico
   AND A.Proyecto = P.Id

/* Con JOIN */
SELECT DNI, NomApels, Id, Nombre
   FROM CIENTIFICOS C INNER JOIN
   (ASIGNADO_A A INNER JOIN PROYECTO P
   ON A.Proyecto = P.Id)
   ON C.DNI = A.Cientifico
```

2. Obtener el número de proyectos al que está asignado cada científico (mostrar el DNI y el nombre)

```
SELECT DNI,NomApels, COUNT(Proyecto)
   FROM CIENTIFICOS LEFT JOIN ASIGNADO_A
   ON CIENTIFICOS.DNI = ASIGNADO_A.Cientifico
   GROUP BY DNI, NomApels
```

3. Obtener el número de científicos asignados a cada proyecto (mostrar el identificador de proyecto y el nombre del proyecto)

```
SELECT Id, Nombre, COUNT(Proyecto)
   FROM PROYECTO LEFT JOIN ASIGNADO A
   ON PROYECTO.Id = ASIGNADO A.Proyecto
   GROUP BY Id, Nombre
```

4. Obtener el número de horas de dedicación de cada científico

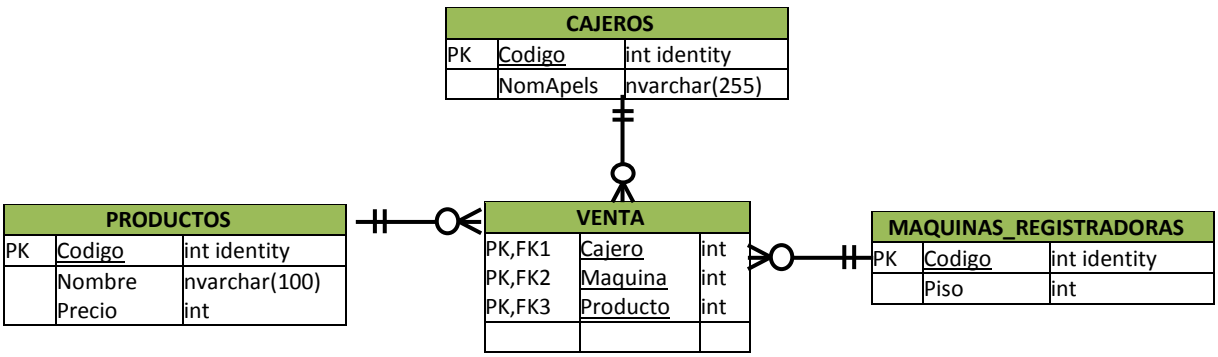
```
SELECT DNI,NomApels, SUM(Horas)
   FROM CIENTIFICOS C LEFT JOIN
   (ASIGNADO A A INNER JOIN PROYECTO P
   ON A.Proyecto = P.Id)
   ON C.DNI = A.Cientifico
   GROUP BY DNI,NomApels
```

5. Obtener el DNI y nombre de los científicos que se dedican a más de un proyecto y cuya dedicación media a cada proyecto sea superior a las 80 horas

```
/* Con dos subconsultas */
SELECT DNI, NomApels
   FROM CIENTIFICOS C
   WHERE 1 <
   (
      SELECT COUNT(*) FROM ASIGNADO_A
      WHERE Cientifico = C.DNI
   )
   AND 80 <
   (
      SELECT AVG(Horas)
      FROM PROYECTO INNER JOIN ASIGNADO A
      ON PROYECTO.Id = ASIGNADO A.Proyecto
      WHERE Cientifico = C.DNI
   )

/* Juntando tablas y con HAVING */
SELECT DNI, NomApels
   FROM CIENTIFICOS C, ASIGNADO A A, PROYECTO P
   WHERE C.DNI = A.Cientifico
   AND P.Id = A.Proyecto
   GROUP BY DNI, NomApels
   HAVING COUNT(Proyecto) > 1 and AVG(Horas) > 80
```

Grandes Almacenes



1. Mostrar el número de ventas de cada producto, ordenado de más a menos ventas

```
SELECT Codigo, Nombre, COUNT(VENTA.Producto)
   FROM PRODUCTOS LEFT JOIN VENTA
   ON PRODUCTOS.Codigo = VENTA.Producto
   GROUP BY Codigo, Nombre
   ORDER BY COUNT(VENTA.Producto) DESC
```

2. Obtener un informe completo de ventas, indicando el nombre del cajero que realizó la venta, nombre y precios de los productos vendidos, y piso en el que se encuentra la máquina registradora donde se realizó la venta

```
/* Sin JOIN */
SELECT NomApels, Nombre, Precio, Piso
   FROM VENTA V, CAJEROS C, PRODUCTOS P, MAQUINAS REGISTRADORAS M
   WHERE V.Cajero = C.Codigo
   AND V.Producto = P.Codigo
```

```
   AND V.Maquina = M.Codigo

/* Con JOIN */
SELECT NomApels, Nombre, Precio, Piso
   FROM CAJEROS C INNER JOIN
      (PRODUCTOS P INNER JOIN
         (MAQUINAS_REGISTRADORAS M INNER JOIN VENTA V
         ON V.Maquina = M.Codigo)
      ON V.Producto = P.Codigo)
   ON V.Cajero = C.Codigo

Obtener las ventas totales realizadas en cada piso
SELECT Piso, SUM(Precio)
   FROM VENTA V, PRODUCTOS P, MAQUINAS REGISTRADORAS M
   WHERE V.Producto = P.Codigo
   AND V.Maquina = M.Codigo
   GROUP BY Piso
```

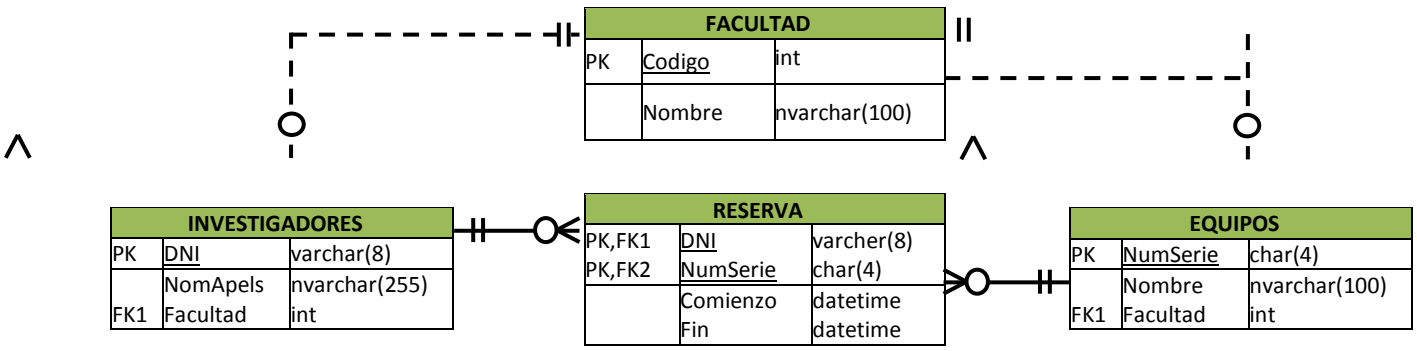
3. Obtener el código y nombre de cada empleado junto con el importe total de sus ventas

```
SELECT C.Codigo, C.NomApels, SUM(Precio)
   FROM PRODUCTOS P INNER JOIN
      (CAJEROS C LEFT JOIN VENTA V
         ON V.Cajero = C.Codigo)
   ON V.Producto = P.Codigo
   GROUP BY C.Codigo, NomApels
```

4. Obtener el código y nombre de aquellos cajeros que hayan realizado ventas en pisos cuyas ventas totales sean inferiores a los 500€

```
SELECT Codigo, NomApels FROM CAJEROS
   WHERE Codigo IN
   (
      SELECT Cajero FROM VENTA
         WHERE Maquina IN
         (
            SELECT Codigo FROM MAQUINAS REGISTRADORAS
               WHERE Piso IN
               (
                  SELECT Piso
                  FROM VENTA V, PRODUCTOS P, MAQUINAS_REGISTRADORAS M
                  WHERE V.Producto = P.Codigo
                  AND V.Maquina = M.Codigo
                  GROUP BY Piso
                  HAVING SUM(Precio) < 500
               )
         )
   )
```

Los investigadores



1. Obtener el DNI y nombre de aquellos investigadores que han realizado más de una reserva

```
/* Juntando tablas */
```

```
SELECT I.DNI, NomApels
   FROM INVESTIGADORES I LEFT JOIN RESERVA R
   ON R.DNI = I.DNI
   GROUP BY I.DNI, NomApels
   HAVING COUNT(R.DNI) > 1

/* Con subconsulta */
SELECT DNI, NomApels
   FROM INVESTIGADORES
   WHERE DNI IN
   (
      SELECT DNI FROM RESERVA
      GROUP BY DNI
      HAVING COUNT(*) > 1
   )
```

2. Obtener un listado completo de reservas, incluyendo los siguientes datos:
   - ✓ DNI y nombre del investigador, junto con el nombre de su facultad
   - ✓ Numero de serie y nombre del equipo reservado, junto con el nombre de la facultad a la que pertenece
   - ✓ Fecha de comienzo y fin de la reserva

```
SELECT I.DNI, NomApels, F INV.Nombre, E.NumSerie, E.Nombre, F EQUIP.Nombre, Comienzo, Fin
   FROM RESERVA R, INVESTIGADORES I, EQUIPOS E, FACULTAD F INV, FACULTAD F EQUIP
   WHERE R.DNI = I.DNI
   AND R.NumSerie = E.NumSerie
   AND I.Facultad = F INV.Codigo
   AND E.Facultad = F_EQUIP.Codigo
```

3. Obtener el DNI y el nombre de los investigadores que han reservado equipos que no son de su facultad

```
/* Juntando tablas */
SELECT DISTINCT I.DNI,NomApels
   FROM RESERVA R, INVESTIGADORES I, EQUIPOS E
   WHERE R.DNI = I.DNI
   AND R.NumSerie = E.NumSerie
   AND I.Facultad <> E.Facultad

/* Con EXISTS */
SELECT DNI, NomApels
   FROM INVESTIGADORES I
   WHERE EXISTS
   (
      SELECT * FROM RESERVA R INNER JOIN EQUIPOS E
         ON R.NumSerie = E.NumSerie
         WHERE R.DNI = I.DNI
         AND I.Facultad <> E.Facultad
   )
```

4. Obtener los nombres de las facultades en las que ningún investigador ha realizado una reserva

```
SELECT Nombre FROM FACULTAD
   WHERE Codigo IN
   (
      SELECT Facultad FROM INVESTIGADORES I LEFT JOIN RESERVA R
         ON I.DNI = R.DNI
         GROUP BY Facultad
         HAVING COUNT(R.DNI) = 0
   )
```

5. Obtener los nombres de las facultades con investigadores ‘ociosos’ (investigadores que no han realizado ninguna reserva)

```
SELECT Nombre FROM FACULTAD
   WHERE Codigo IN
   (
      SELECT Facultad FROM INVESTIGADORES
      WHERE DNI NOT IN
      (
         SELECT DNI FROM RESERVA
      )
```

```
   )
```

6. Obtener el número de serie y nombre de los equipos que nunca han sido reservados

```
/* Juntando tablas */
SELECT E.NumSerie, Nombre
   FROM EQUIPOS E LEFT JOIN RESERVA R
   ON R.NumSerie = E.NumSerie
   GROUP BY E.NumSerie, Nombre
   HAVING COUNT(R.NumSerie) = 0

/* Con subconsult IN */
SELECT NumSerie, Nombre FROM EQUIPOS
   WHERE NumSerie NOT IN
   (
      SELECT NumSerie FROM RESERVA
   )

/* Con EXISTS */
SELECT NumSerie, Nombre
   FROM EQUIPOS E
   WHERE NOT EXISTS
   (
      SELECT * FROM RESERVA R
      WHERE R.NumSerie = E.NumSerie
   )
```