

Manual de SQL para bases de datos MySQL y Oracle

Introducción

Structure Query Language

Es un lenguaje de dominio específico utilizado en programación, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales

Tipos de datos SQL

Tipo de datos	Descripción
CHARACTER(n)	Cadena de caracteres. De longitud fija n
VARCHAR(n) CHARACTER VARYING(n)	or Cadena de caracteres. Longitud variable. Máxima longitud n
BINARY(n)	Cadena binaria. De longitud fija n
BOOLEAN	Almacena los valores VERDADERO o FALSO
VARBINARY(n) BINARY VARYING(n)	or Cadena binaria. Longitud variable. Máxima longitud n
INTEGER(p)	Numérica número entero (sin decimales). precisión p
SMALLINT	Numérica número entero (sin decimales). precisión 5

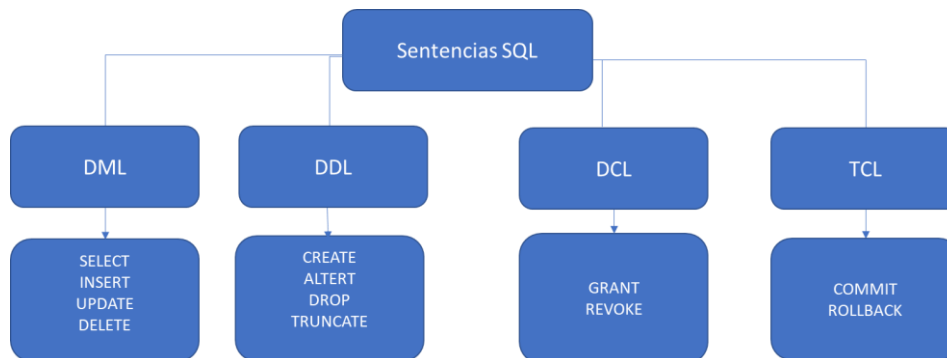
INTEGER	Numérica número entero (sin decimales). 10 precisión
BIGINT	Numérica número entero (sin decimales). 19 precisión
DECIMAL(p,s)	Numérica exacta, la precisión p, s escala. Ejemplo: decimal (5,2) es un número que tiene 3 dígitos antes del decimal y 2 dígitos después del punto decimal
NUMERIC(p,s)	Numérica exacta, la precisión p, s escala. (Igual que DECIMAL)
FLOAT(p)	Numérica aproximada, precisión mantisa p. Un número flotante en la base 10 de la notación exponencial. El argumento de tamaño de este tipo consiste en un único número que especifica el mínimo de precisión
REAL	Numérica aproximada, precisión mantisa 7
FLOAT	Numérica aproximada, precisión mantisa 16
DOUBLE PRECISION	Numérica aproximada, precisión mantisa 16
DATE	Tiendas el año, mes, día y valores
TIME	Tiendas de hora, minutos y segundos valores
TIMESTAMP	Tiendas el año, mes, día, hora, minuto y segundo valores
INTERVAL	Compuesto por una serie de campos de números enteros, lo que representa un período de tiempo, dependiendo del tipo de intervalo
ARRAY	Un conjunto de cuerpo y de colección ordenada de elementos

MULTISET	Una de longitud variable y la colección desordenada de elementos
XML	Almacena los datos XML

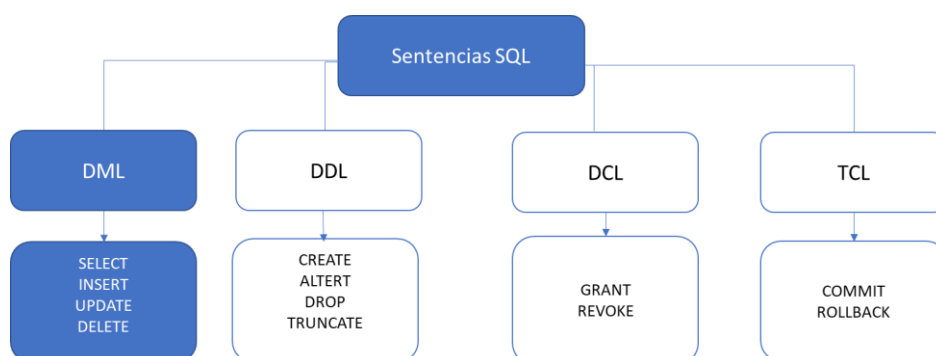
Sentencias SQL

Las sentencias SQL se agrupan en cuatro grupos:

- **DML**. Data Management Language. Sentencias para el manejo de Datos
- **DDL**. Data Definition Language. Sentencias para el manejo de la Estructura de Datos
- **DCL**. Data Control Language. Sentencias para el Control de acceso a Datos
- **TCL**. Transaction Control Language. Sentencias para el Control de las Transacciones



DML – Data Management Language



Las sentencias más importantes del lenguaje DML son **SELECT**, **INSERT**, **UPDATE** y **DELETE** ya que ejecutan el manejo básico de los datos:

- **SELECT**, con esta sentencia se ejecutan las consultas de los datos. Las bases de datos relacionales están orientadas a consultar la información relacional lo más eficazmente

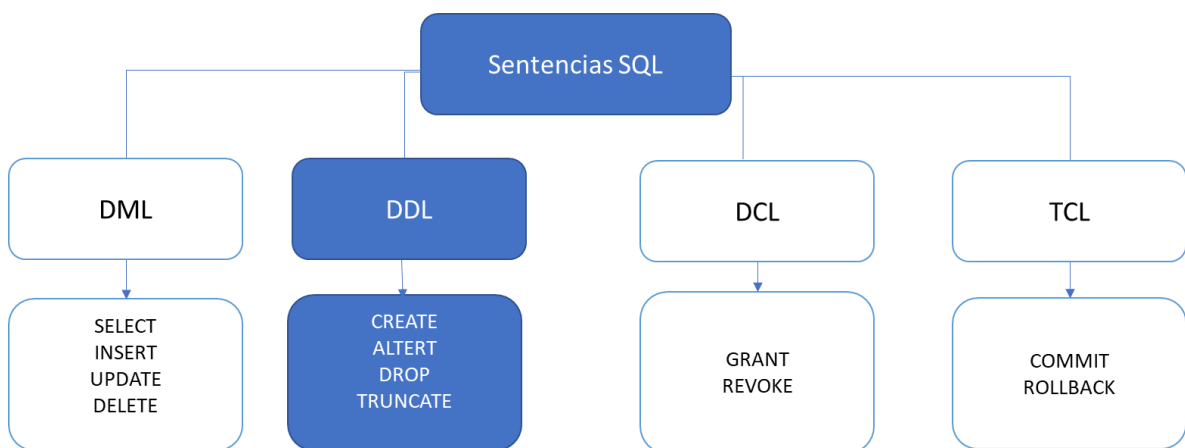
posible, por tanto, esta sentencia es la más optimizada y la que más usada en el manejo de datos en una base de datos relacional.

- INSERT, esta sentencia se utiliza para la inserción o almacenamiento de datos en una base de datos relacional
- UPDATE, para modificar los datos ya almacenados se utiliza esta sentencia.
- DELETE, si se quiere eliminar datos almacenados es la sentencia que se tiene que ejecutar.

En sí, cuando nos referimos a sentencias **DML** aunque SELECT es la más utilizada, nos referimos a **INSERT**, **UPDATE** y **DELETE** ya que modifican los datos propiamente dichos, SELECT quedaría aparte.

Hay otras sentencias DML pero son mucho menos utilizadas y quedan fuera de este manual, como por ejemplo MERGE, para la “mezcla” de datos y CALL, las llamadas a procedimientos almacenados.

DDL – Data Definition Language



Esta agrupación contiene las sentencias que modifican la estructura de los datos. ¿Qué es la estructura de los datos? En sí los objetos propios de las aplicaciones de bases de datos: esquemas, bases de datos, tablas, vistas, usuarios, procedimientos, funciones, índices, etc.

Todos estos objetos no son información almacenada, no son datos, pero estructuran y ayudan al almacenamiento y acceso a los mismos.

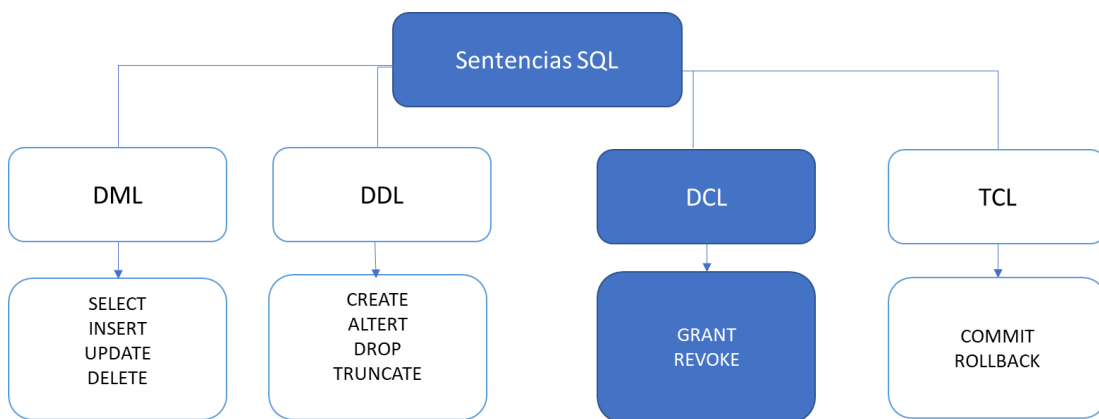
- **CREATE**, crea cualquier objeto de la base de datos, la más usada es CREATE TABLE, para generar tablas.

En una aplicación de bases de datos, según lo grande que sea pueden existir varios esquemas, que son agrupaciones lógicas y dentro de ellas bases de datos independientes unas de otras. En el caso de MySQL la relación esquema-base de datos es uno a uno, por tanto, es lo mismo crear un esquema que crear una base de datos, automáticamente nos generaría el otro elemento. En MySQL, CREATE SCHEMA es igual que CREATE DATABASE es. Sin embargo, en Oracle no, un esquema al menos debe contener una base de datos, pero puede contener varias.

- **ALTER**, modifica cualquier objeto en sí de la base de datos o crea, modifica y elimina elementos que dependan de la misma. Por ejemplo en una tabla, podemos, añadir, modificar y eliminar columnas.
- **DROP**, elimina cualquier objeto de la base de datos y con ello, todos los elementos que dependan del mismo.

Un caso especial es **TRUNCATE**, esta sentencia no modifica en sí la estructura, lo que hace cuando se ejecuta es eliminar todos los datos de la tabla, pero pertenece a este agrupación por cómo realiza esta acción: en sí no se eliminan los datos de la tabla, se elimina la referencia (puntero) que tiene la tabla referenciando a sus datos, al realizar esta acción, la tabla pierde la ubicación de sus datos en memoria, por ello cuando se la consulta está vacía.

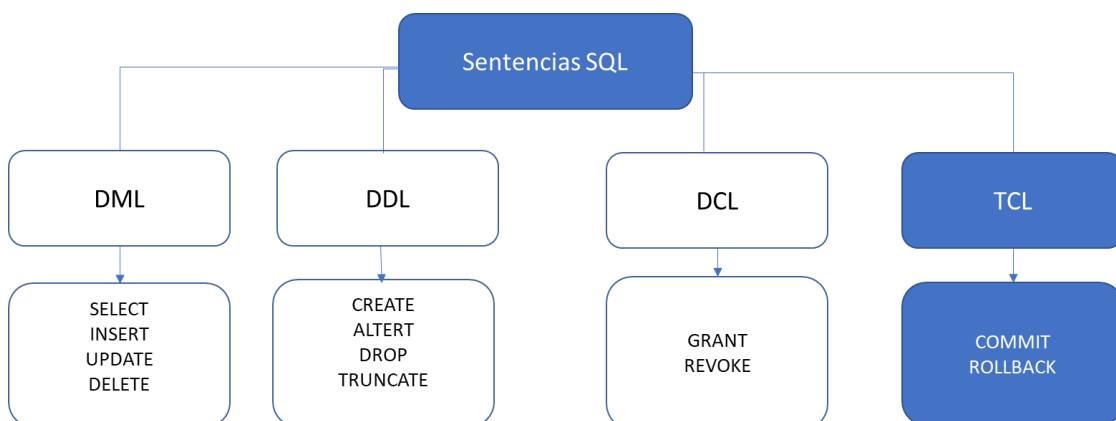
DCL – Data Control Language



El acceso a los datos almacenados debe estar protegido, no se puede dejar acceder a todo ni que realicen todas las acciones. Para controlar el acceso y las acciones a ejecutar se tienen las sentencias DCL que determinan el acceso de los diferentes usuarios del sistema a los datos y a la estructura de la base de datos.

- **GRANT**, esta sentencia otorga permisos al usuario.
- **REVOKE**, esta sentencia quita permisos al usuario.

TCL – Transaction Control Language



La base de una base de datos es la fiabilidad de la información que tenga almacenada, que en todo momento la información no sea incongruente y el sistema se mantenga estable.

Según como el uso de las bases de datos se fue propagando surgieron dos problemas, la concurrencia y la persistencia, ya que el uso de la información de la base de datos suele ser compartido por más de una conexión y/o usuario.

El problema de la concurrencia radica en que más de dos usuarios (o conexiones del mismo) quieran acceder al mismo recurso (suelen ser tablas) para ejecutar diferentes sentencias DML, en el caso de consultas, SELECT, no hay tanto problema, pero en el resto sí ya que tienen que tener un uso privativo del recurso, solo deberían acceder uno a la vez y el otro esperar.

El problema de la persistencia radica en el mismo hecho, pero orientado a la validez de los datos, ya que el primer usuario o conexión puede consultar o modificar los datos, pero el segundo ejecuta una sentencia DML sin tener en cuenta los cambios producidos o también si él los produce, el primero no tiene notificación y puede estar tratando datos que ya no son válidos.

El problema se complica más cuando cada usuario/conexión quiere ejecutar varias sentencias DML seguidas y necesita asegurarse que los recursos a los que acceda solo acceda él y que se ejecuten todas las sentencias o no se ejecute ninguna para asegurar que el sistema siempre este en una situación estable.

Para todo esto, se establecieron dos sentencias principales, COMMIT y ROLLBACK.

- **COMMIT**, fija todos los cambios producidos por sentencias DML desde el último COMMIT. Los cambios realizados en la base de datos no trascienden a otros usuarios o conexiones hasta que se ejecuta esta sentencia.
- **ROLLBACK**, en el caso que exista un error o cuando el usuario quiera, antes de ejecutar COMMIT, se puede ejecutar un ROLLBACK esto hace que cualquier cambio que se haya producido desde el último COMMIT quede descartado y se quede el sistema con los valores iniciales antes de los cambios.

COMMIT y ROLLBACK aplican únicamente a sentencias DML y concretamente a las que modifican los datos como puede ser insert,update,delete y merge.

En la mayoría de las aplicaciones de bases de datos, existe la función AUTOCOMMIT, que se puede configurar a true o false, si es true, automáticamente todas las sentencias DML ejecutadas se fijan.

Tabla DUAL

Todas las bases de datos proporcionan una tabla llamada DUAL con la que se permiten hacer pruebas. Esa tabla tiene una sola columna (llamada DUMMY) y una sola fila.

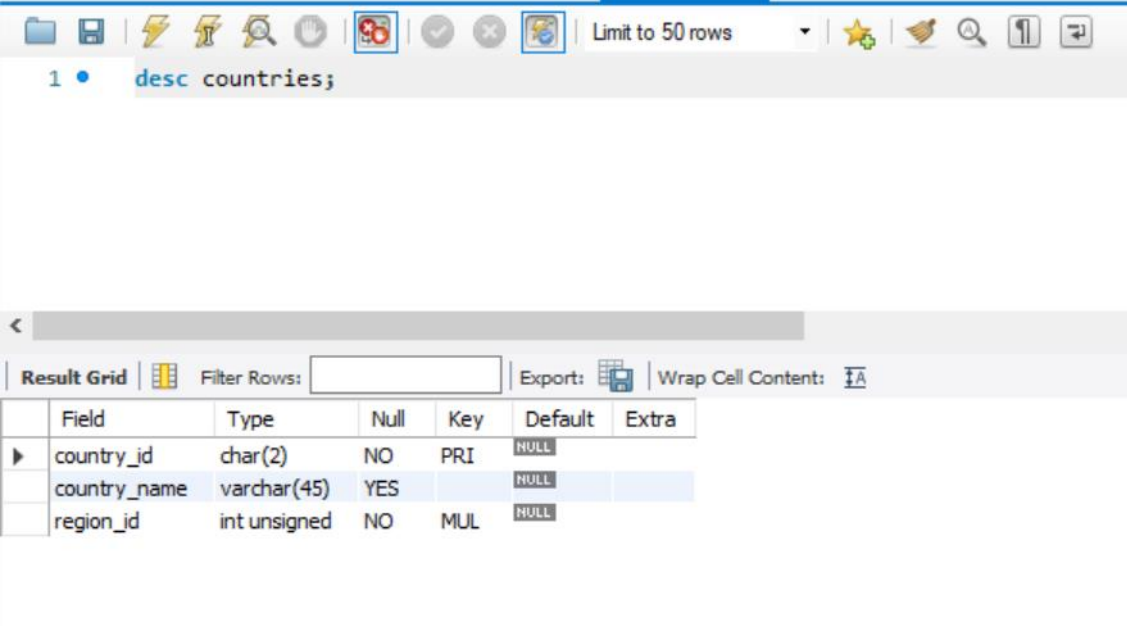
```
SELECT SQRT(5) FROM DUAL
```

Muestra una tabla con el contenido de ese cálculo (la raíz cuadrada de 5). En esa tabla solo habrá una columna con una fila, la cual mostrará el resultado del cálculo.

DUAL es una tabla interesante para hacer pruebas, ya que podemos utilizar funciones o cálculos sabiendo que devolverán un único resultado: eso sí esos cálculos se realizarán sobre valores independientes (no sobre valores de las tablas de la base de datos).

Sentencia DESC

Esta sentencia devuelve información el objeto sobre el que se ejecuta, según la base de datos nos va a devolver información diferente.



The screenshot shows a database management tool interface. At the top, there is a toolbar with various icons. Below the toolbar, the SQL command `desc countries;` is entered in a text area. The result is displayed in a table format. The table has the following structure:

Field	Type	Null	Key	Default	Extra
country_id	char(2)	NO	PRI	NULL	
country_name	varchar(45)	YES		NULL	
region_id	int unsigned	NO	MUL	NULL	

Sentencia USE

La sentencia USE se ejecuta cuando se quiere ejecutar las sentencias SQL en una base de datos determinada, con eso nos evitamos especificar en las sentencias el esquema objetivo dónde ejecutarlas. Digamos que nos permite conectarnos a esa base de datos para ejecutar los comandos sobre los objetos de esa base de datos.

SELECT básico

```
SELECT [DISTINCT] column1 [aliasColumna], column2, ... (lista de selección)
```

```
FROM [esquema.]table_name [aliasTabla]
```

```
[WHERE sentencias booleanas]
```

```
[ORDER BY columnas de ordenación[ASC|DESC]] ;
```

- **SELECT** sentencia DML para ejecutar consultas
- **FROM** específica de qué tablas se van a extraer los datos. Es obligatorio.
- **WHERE** filtra la información a consultar. Compuesta de una a varias condiciones booleanas, si el registro cumple las condiciones es seleccionado sino se descarta.
- **ORDER BY** especifica un orden según los valores de las columnas especificadas.

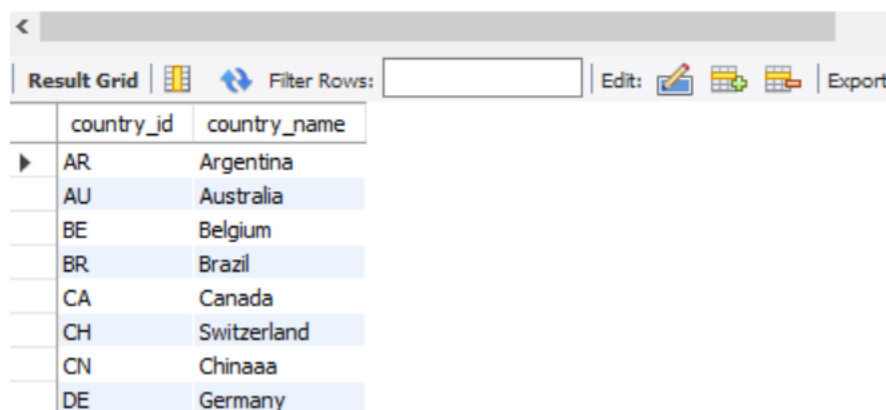
La partícula DISTINCT la veremos más adelante, pero se utiliza para que se muestren datos NO repetidos.

Ejemplo de Sentencia básica

Con enunciados de este tipo:

“Se necesitan los identificadores y nombres de los países” “Se seleccionan los identificadores y nombres de los países” “Se muestran los identificadores y nombres de los países”

```
1 select country_id, country_name from countries;
```



	country_id	country_name
▶	AR	Argentina
	AU	Australia
	BE	Belgium
	BR	Brazil
	CA	Canada
	CH	Switzerland
	CN	Chinaaa
	DE	Germany

En esta sentencia ni se filtra, por tanto, se van a mostrar todos los resultados, ni se ordena por tanto el sistema va a mostrar la información según el orden en la que la tenga almacenada, normalmente por el identificador.

Alias de columna

Por defecto como nombre de columna en los resultados se nos muestra el nombre de campo, función u operación que utilicemos, sin embargo, hay una forma de “renombrar” las columnas de los resultados mediante los [alias de columna](#).

Sin alias de columna:

```
1 select country_id, country_name from countries;
```

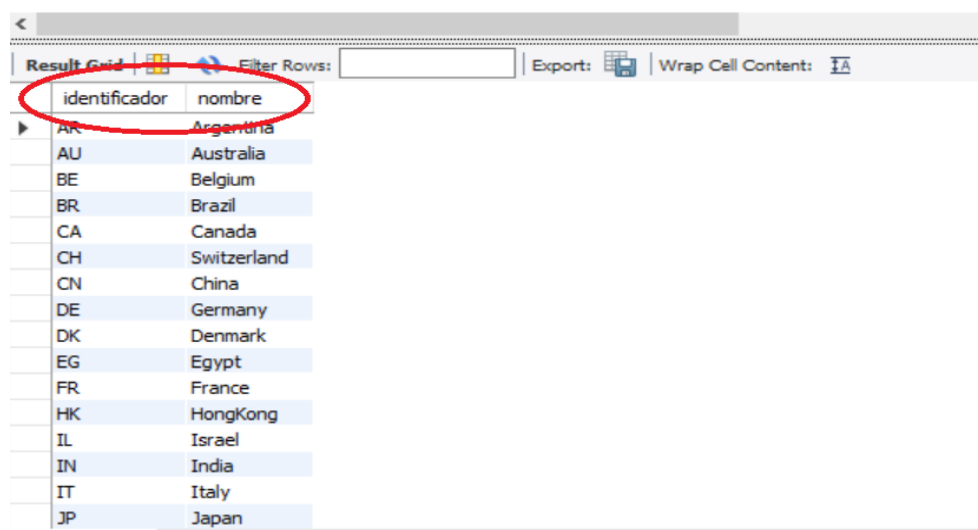


The screenshot shows a database query result grid. The header row contains two columns: 'country_id' and 'country_name', which are circled in red. Below the header, there are eight rows of data representing different countries.

country_id	country_name
AR	Argentina
AU	Australia
BE	Belgium
BR	Brazil
CA	Canada
CH	Switzerland
CN	Chinaaa
DE	Germany

Con alias de columna, se subrayan los alias y se indica la diferencia en el resultado.

```
1 SELECT country_id identificador, country_name nombre from countries;
```



The screenshot shows a database query result grid. The header row contains two columns: 'identificador' and 'nombre', which are circled in red. Below the header, there are sixteen rows of data representing different countries. The column names are underlined in the original image.

<u>identificador</u>	<u>nombre</u>
AR	Argentina
AU	Australia
BE	Belgium
BR	Brazil
CA	Canada
CH	Switzerland
CN	China
DE	Germany
DK	Denmark
EG	Egypt
FR	France
HK	HongKong
IL	Israel
IN	India
IT	Italy
JP	Japan

Los alias de columna se pueden indicar de varias formas según la base de datos.

En Oracle y MySQL:

- Sin comillas ni simples ni dobles siendo una única palabra

```
SELECT country_id identificador, country_name nombre from countries;
```

- Utilizando la partícula AS

```
SELECT country_id AS identificador, country_name AS nombre from countries;
```

- Utilizando comillas dobles para varias palabras

```
SELECT country_id AS "Identificador País", country_name AS "Nombre País" from countries;
```

```
SELECT country_id "Identificador País", country_name "Nombre País" from countries;
```

- Utilizando comillas simples para varias palabras

```
SELECT country_id AS 'Identificador País', country_name AS 'Nombre País' from countries;
```

```
SELECT country_id 'Identificador País', country_name 'Nombre País' from countries;
```

Selección de operaciones

Se pueden realizar operaciones por si mismas u operando con valores de los registros.

“Seleccionar el nombre y apellidos de los trabajadores, así como su salario y el diez por ciento del mismo más cien”

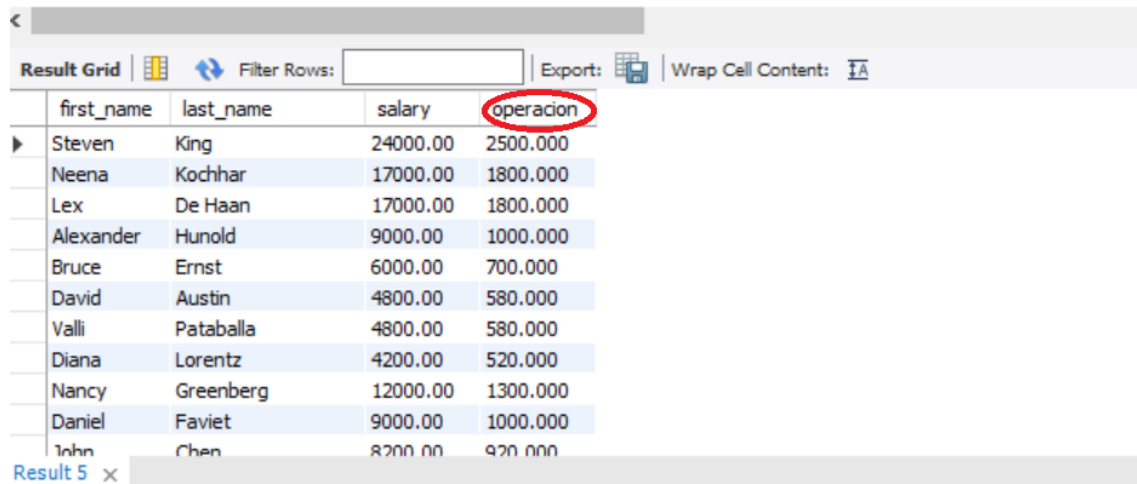
1 • `SELECT first_name,last_name,salary, salary*0.1+100 FROM hr.employees;`

	first_name	last_name	salary	salary*0.1+100
▶	Steven	King	24000.00	2500.000
	Neena	Kochhar	17000.00	1800.000
	Lex	De Haan	17000.00	1800.000
	Alexander	Hunold	9000.00	1000.000
	Bruce	Ernst	6000.00	700.000
	David	Austin	4800.00	580.000
	Valli	Pataballa	4800.00	580.000
	Diana	Lorentz	4200.00	520.000
	Nancy	Greenberg	12000.00	1300.000
	Daniel	Faviet	9000.00	1000.000
	John	Chen	8200.00	920.000

Result 4 x

Usando alias de columna :

```
1 • SELECT first_name,last_name,salary, salary*0.1+100 operacion FROM hr.employees;
```



The screenshot shows a database query result grid. The grid has four columns: first_name, last_name, salary, and operacion. The 'operacion' column is circled in red. The data is as follows:

first_name	last_name	salary	operacion
Steven	King	24000.00	2500.000
Neena	Kochhar	17000.00	1800.000
Lex	De Haan	17000.00	1800.000
Alexander	Hunold	9000.00	1000.000
Bruce	Ernst	6000.00	700.000
David	Austin	4800.00	580.000
Valli	Pataballa	4800.00	580.000
Diana	Lorentz	4200.00	520.000
Nancy	Greenberg	12000.00	1300.000
Daniel	Faviet	9000.00	1000.000
John	Chen	8200.00	920.000

Se puede seleccionar cualquier operativa numérica, de cadena, de fecha, cualquiera que permita el sistema.

Importante:

Si ejecutamos `select 'A' from countries` ¿Cuántas A se mostrarán? Tantas como registros existan

Si ejecutamos `select 1+1 from countries` ¿Cuántas veces se sumará uno más uno? Tantas veces como registros existan, se mostrarán 2 tantas veces como registros existan.

Cláusula WHERE - Filtro de consulta

Contiene expresiones booleanas de cualquier tipo, pueden ser simples:

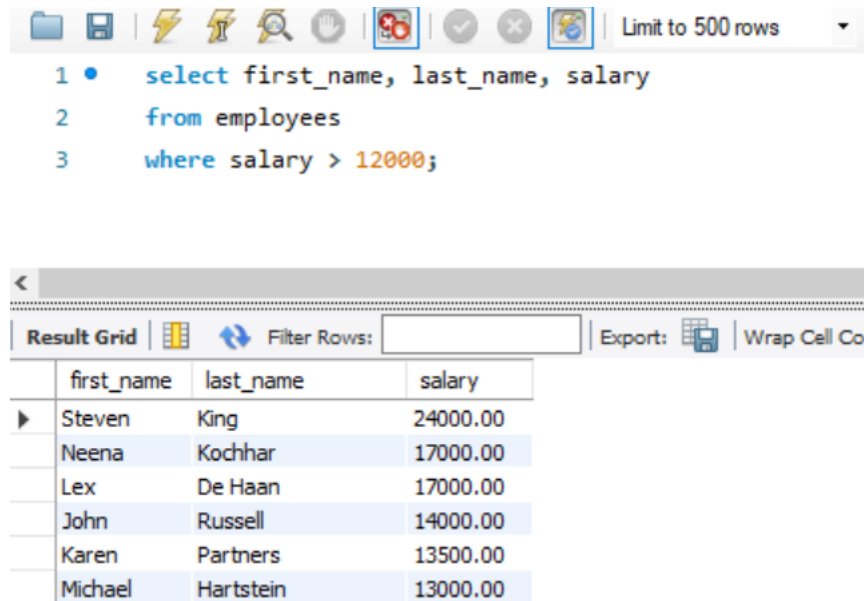
`WHERE salary >1200`

Hasta subconsultas complejas que devuelvan un único valor y este tenga algún tipo de comparativa, como veremos en el tema de subconsultas.

Esta cláusula es opcional y es única por cada sentencia SELECT.

Ejemplo de la cláusula WHERE:

“Seleccionar el nombre, apellido y salario de todos los empleado cuyo salario sea mayor de 12000”



The screenshot shows a SQL query editor with a toolbar at the top containing icons for file operations, execution, and a 'Limit to 500 rows' dropdown. The query text is as follows:

```
1 • select first_name, last_name, salary
2   from employees
3   where salary > 12000;
```

Below the query, a 'Result Grid' displays the results of the query. The grid has columns for 'first_name', 'last_name', and 'salary'. The results are sorted by salary in descending order.

	first_name	last_name	salary
▶	Steven	King	24000.00
	Neena	Kochhar	17000.00
	Lex	De Haan	17000.00
	John	Russell	14000.00
	Karen	Partners	13500.00
	Michael	Hartstein	13000.00

En la cláusula WHERE no se puede hacer referencia a los alias de columna, ya que la consulta todavía no ha realizado y los alias de columna se ejecutan cuando la consulta se ha realizado.

Cláusula ORDER BY – Ordenación de la consulta

El orden en el que se muestran los resultados de las consultas, en ausencia de la consulta ORDER BY, es en el que están los datos almacenados.

Si queremos de salida algún que otro orden dado por una o más columnas se debe utilizar la cláusula ORDER BY. El orden puede ser ascendente ([ASC](#)) o descendente ([DESC](#)) [por columna](#). El orden ascendente es el dado por defecto y no es necesario indicarlo.

También se puede indicar en vez del nombre del campo, su posición en la lista de selección, las dos soluciones dadas en el ejemplo siguiente son equivalentes.

Ejemplo

“Seleccionar el identificador de departamento, del trabajo y el nombre y apellidos de los empleados ordenados por el identificador de departamento, por el identificador de trabajo descendientemente y por el nombre ascendentemente”

```

1 • select department_id, job_id, first_name, last_name
2   from employees
3   order by department_id, job_id desc, first_name;

```

department_id	job_id	first_name	last_name
NULL	SA_REP	Kimberely	Grant
10	AD_ASST	Jennifer	Whalen
20	MK_REP	Pat	Fay
20	MK_MAN	Michael	Hartstein
30	PU_MAN	Den	Raphaely
30	PU_CLERK	Alexander	Khoo
30	PU_CLERK	Guy	Himuro
30	PU_CLERK	Karen	Colmenares
30	PU_CLERK	Shelli	Baida
30	PU_CLERK	Sigal	Tobias
40	HR_REP	Susan	Mavris

```

1 • select department_id, job_id, first_name, last_name
2   from employees
3   order by 1,2 desc, 3;

```

department_id	job_id	first_name	last_name
NULL	SA_REP	Kimberely	Grant
10	AD_ASST	Jennifer	Whalen
20	MK_REP	Pat	Fay
20	MK_MAN	Michael	Hartstein
30	PU_MAN	Den	Raphaely
30	PU_CLERK	Alexander	Khoo
30	PU_CLERK	Guy	Himuro
30	PU_CLERK	Karen	Colmenares
30	PU_CLERK	Shelli	Baida
30	PU_CLERK	Sigal	Tobias
40	HR_REP	Susan	Mavris

Las dos sentencias, primero ordenan por el identificador de departamento (10,20,30,etc.), después por el identificador de trabajo descendientemente (caso del departamento 30: PU_MAN, PU_CLERK,etc.) y después por el nombre ascendientemente (Alexander, Guy, Karen, etc.)

Partícula DISTINCT

La partícula DISTINCT se puede indicar en cualquier sentencia SELECT, simplemente indica que los datos a mostrar no pueden estar repetidos.

Si por ejemplo tenemos la consulta: “Mostrar los nombres de los empleados del sistema”

Vemos que existen valores en los resultados repetidos, sin embargo, si incluimos la partícula DISTINCT en la consulta: “Mostrar los nombres **distintos** de los empleados del sistema” los resultados no se repiten.

Uso de NULL

Si un dato tiene valor Null, significa que no existe, que no está determinado.

Vacío ≠ Null ≠ 0

Aunque una cadena este vacía no significa que sea nula.

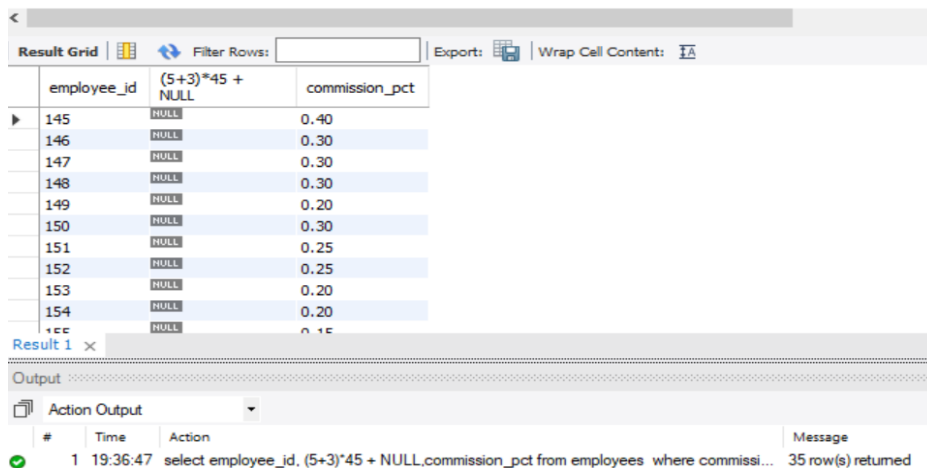
Cualquier operación aritmética sobre un valor NULL tendrá valor NULL.

Para comparaciones de filtro se utiliza la palabra clave NULL, no se utiliza el operador de igualdad ya que no se iguale el contenido, no es que el campo tenga valor nulo es que ES nulo, no se reserva memoria para su valor.

Ejemplo:

“Seleccionar el identificador de empleado y la operación aritmética $(5+3)*45 + \text{NULL}$ dónde la comisión sea NO nula”

```
1 • select employee_id, (5+3)*45 + NULL, commission_pct
2   from employees
3  where commission_pct is not null;
```



The screenshot shows a database query result grid. The query is: `select employee_id, (5+3)*45 + NULL, commission_pct from employees where commission_pct is not null;`. The result grid has three columns: `employee_id`, `(5+3)*45 + NULL`, and `commission_pct`. The `(5+3)*45 + NULL` column contains NULL values for all rows. The `commission_pct` column contains various values (0.40, 0.30, 0.20, 0.25). The result grid is displayed in a table format with alternating row colors. Below the result grid, there is an 'Output' section showing the execution details: '1 19:36:47 select employee_id, (5+3)*45 + NULL, commission_pct from employees where commission_pct is not null; 35 row(s) returned'.

employee_id	(5+3)*45 + NULL	commission_pct
145	NULL	0.40
146	NULL	0.30
147	NULL	0.30
148	NULL	0.30
149	NULL	0.20
150	NULL	0.30
151	NULL	0.25
152	NULL	0.25
153	NULL	0.20
154	NULL	0.20
155	NULL	0.15

Result 1 x

Output

Action Output

#	Time	Action	Message
1	19:36:47	select employee_id, (5+3)*45 + NULL, commission_pct from employees where commission_pct is not null;	35 row(s) returned

Se seleccionan únicamente 35 registros porque son los registros que tienen comisión.

Y todos los campos de la operación salen nulos, ya que cualquier operación con NULL da como resultado NULL.

Nota: Probar a realizar la misma consulta con la operación $(5+3)*\text{salary} + \text{NULL}$, tendríamos el mismo resultado.

Operadores

Operadores lógicos

AND : "A and B" devuelve cierto si A y B valen cierto, y falso en cualquier otro caso.

OR : "A or B" devuelve cierto si A o B valen cierto, y falso únicamente cuando tanto A como B valen falso.

NOT : "not A" devuelve falso si A vale cierto, y cierto si A vale falso.

Operadores de comparación

> : "A > B" devuelve cierto si A es estrictamente **mayor que** B, de lo contrario devuelve falso.

< : "A < B" devuelve cierto si A es estrictamente **menor que** B, de lo contrario devuelve falso.

= : "A = B" devuelve cierto si A es **igual a** B, de lo contrario devuelve falso.

>= : "A >= B" devuelve cierto si A es **mayor o igual a** B, de lo contrario devuelve falso.

<= : "A <= B" devuelve cierto si A es **menor o igual a** B, de lo contrario devuelve falso.

!= : "A != B" devuelve cierto si A es **distinto a** B, de lo contrario devuelve falso.

Operadores BETWEEN e IN

Se pueden aplicar a campos numéricos principalmente y también se pueden usar aunque es menos común es campo de cadena y de fechas.

BETWEEN: establece un rango de valores incluidos ambos. Es equivalente a: "campo mayor e igual a valor inicial y campo menor e igual a valor final".

IN: el valor se encuentra en una lista de valores. Es equivalente a: "campo igual a un valor o campo igual a otro valor o campo igual a otro valor...etc."

Ejemplo BETWEEN con valores numéricos

"Seleccionar el identificador, nombre y apellidos de los empleados que tienen un salario entre 9000 y 9600 ordenados por salario"

```

1 • SELECT employee_id, first_name, last_name, salary
2 FROM employees
3 WHERE salary BETWEEN 9000 AND 9600
4 order by salary;

```

Result Grid	Filter Rows:	Edit:	Export/Im
employee_id	first_name	last_name	salary
103	Alexander	Hunold	9000.00
109	Daniel	Faviet	9000.00
152	Peter	Hall	9000.00
158	Allan	McEwen	9000.00
151	David	Bernstein	9500.00
157	Patrick	Sully	9500.00
163	Danielle	Greene	9500.00
170	Taylor	Fox	9600.00
NULL	NULL	NULL	NULL

Ejemplo BETWEEN con fechas

“Seleccionar a los empleados (identificador, nombre y apellido) que se incorporación en el año 1997 ordenados por fecha de incorporación (o de contratación)”

```

1 • SELECT employee_id, first_name, last_name, hire_date
2 FROM employees
3 WHERE hire_date BETWEEN '1997-01-01' AND '1997-12-31'
4 order by hire_date;
5

```

Result Grid	Filter Rows:	Edit:	Export/Im
employee_id	first_name	last_name	hire_date
146	Karen	Partners	1997-01-05
142	Curtis	Davies	1997-01-29
150	Peter	Tucker	1997-01-30
131	James	Marlow	1997-02-16
185	Alexis	Bull	1997-02-20
193	Britney	Everett	1997-03-03
147	Alberto	Errazuriz	1997-03-10
159	Lindsey	Smith	1997-03-10
168	Lisa	Ozer	1997-03-11
175	Alyssa	Hutton	1997-03-19
151	David	Bernstein	1997-03-24
121	Adam	Fripp	1997-04-10
188	Kelly	Chung	1997-06-14

Ejemplo BETWEEN con cadenas

“Seleccionar los empleados (identificador, nombre y apellido) que como nombre empiezan desde la ‘A’ hasta la ‘AL’ ordenados por nombre”

```
1 • SELECT employee_id, first_name, last_name
2 FROM employees
3 WHERE first_name BETWEEN 'A' AND 'ALZ'
4 order by first_name;
5
```

Result Grid			
Filter Rows: <input type="text"/>			
Edit:			
employee_id	first_name	last_name	
121	Adam	Fripp	
196	Alana	Walsh	
147	Alberto	Errazuriz	
103	Alexander	Hunold	
115	Alexander	Khoo	
185	Alexis	Bull	
158	Allan	McEwen	
175	Alyssa	Hutton	
NULL	NULL	NULL	

Ejemplo de IN

“Seleccionar los empleados de los departamentos 10,20 y 30, seleccionar además del identificador del empleado, su nombre y apellido, el identificador del departamento y ordenar por este dato”

```
1 • SELECT employee_id, first_name, last_name, department_id
2 FROM employees
3 WHERE department_id IN (10,20,30)
4 order by department_id;
5
```

Result Grid			
Filter Rows: <input type="text"/>			
Edit:			
Export/Import:			
employee_id	first_name	last_name	department_id
200	Jennifer	Whalen	10
201	Michael	Hartstein	20
202	Pat	Fay	20
114	Den	Raphaely	30
115	Alexander	Khoo	30
116	Shelli	Baida	30
117	Sigal	Tobias	30
118	Guy	Himuro	30
119	Karen	Colmenares	30
NULL	NULL	NULL	NULL

Operador LIKE

Se aplica a campos de cadena y se podría traducir a “se parece a”.

Se utiliza principalmente con dos comodines:

% Se utiliza como comodín y sustituye a 0 o más caracteres de la cadena.

_ Se utiliza como comodín y sustituye a 1 carácter de la cadena.

Ejemplos:

campo LIKE 'a' El campo debe ser una 'a', equivalente al operador =

campo LIKE '%a' El campo finaliza con una 'a'.

campo LIKE 'a%' El campo empieza con una 'a'

campo LIKE '%a%' El campo contiene una 'a'

campo LIKE '_a%' La segunda letra del campo es una 'a'.

Caracteres de escape

Si quisiéramos buscar en la cadena un %, tenemos que utilizarlo con el sentido de carácter no de comodín, para ello utilizamos un carácter de escape para “escapar” su sentido de comodín y tratarlo como un simple carácter. Todas las bases de datos tienen un carácter de escape por defecto, pero también podemos indicar cual se quiere usar.

Ejemplo:

Campo LIKE '%\%%' ESCAPE '\ ' El campo contiene %.

Se establece el carácter de escape como \. El primer y último % tienen carácter de comodín, pero el segundo % lo escapamos colocando delante el carácter de escape y lo tratamos como un carácter.

Ejemplo carácter escape por defecto '\'

“Seleccionar los empleados (identificador, nombre y apellido) donde el teléfono contenga '505.', seleccionar y ordenar por este campo también”

```
1 • SELECT employee_id, first_name, last_name, phone_number
2 FROM employees
3 WHERE phone_number LIKE '%505\.%'
4 order by phone_number;
5
```

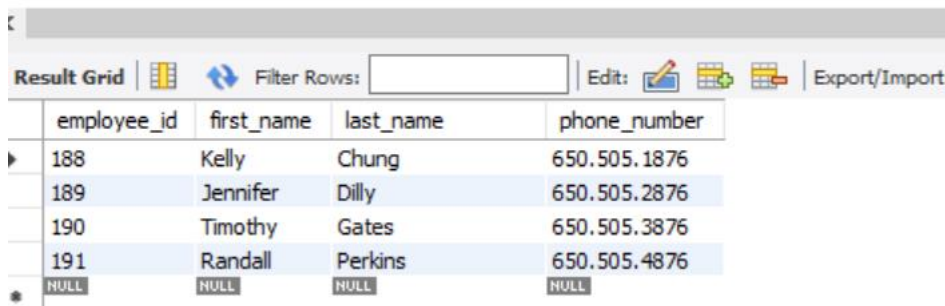
employee_id	first_name	last_name	phone_number
188	Kelly	Chung	650.505.1876
189	Jennifer	Dilly	650.505.2876
190	Timothy	Gates	650.505.3876
191	Randall	Perkins	650.505.4876
NULL	NULL	NULL	NULL

Ejemplo carácter escape

Igual que el anterior, pero establecemos el carácter de escape a '&'.

“Seleccionar los empleados (identificador, nombre y apellido) donde el teléfono contenga '505.', seleccionar y ordenar por este campo también”

```
1 • SELECT employee_id, first_name, last_name, phone_number
2 FROM employees
3 WHERE phone_number LIKE '%505&.%' ESCAPE '&'
4 order by phone_number;
5
```



The screenshot shows a database interface with a 'Result Grid' tab. The grid displays the results of the SQL query, showing columns for employee_id, first_name, last_name, and phone_number. The results are ordered by phone_number, showing four employees with phone numbers starting with 650.505.

employee_id	first_name	last_name	phone_number
188	Kelly	Chung	650.505.1876
189	Jennifer	Dilly	650.505.2876
190	Timothy	Gates	650.505.3876
191	Randall	Perkins	650.505.4876
NULL	NULL	NULL	NULL

Funciones SQL de fila simple

En realidad, hay dos tipos de funciones:

- **Funciones que operan con datos de la misma fila.** Estas funciones devuelven un resultado por cada fila de la consulta. Operan con expresiones relativas a los datos de cada fila. Se ejecutan por cada fila o registro seleccionado o que exista en la tabla
- **Funciones que operan con datos de varias filas diferentes (funciones de agrupación).** Las veremos en el capítulo de “Agrupaciones”

Dentro de las funciones SQL de fila simple, comentaremos:

- **Funciones numéricas.** Incluyen las de redondeo, truncado y las propias de matemáticas.
- **Funciones de caracteres**
 - **Funciones de conversión de texto.** De mayúsculas y minúsculas y minúsculas.
 - **Funciones de transformación.** Quitar espacios, concatenar, cortar cadenas, buscar caracteres, reemplazar caracteres.
 - **Otras funciones.** De número a caracteres, de caracteres a números y funciones de sonido.
 - **De trabajo con nulos.** sustitución de valores nulos, reemplazar valores por nulos.
- **Funciones de fecha.** Todas las funciones que tratan con campos tipo fecha o tiempo
- **Funciones de conversión.** Las funciones de conversión son las que cambian los valores de fecha a cadena, de numero a cadena y de cadena a numero y cadena a fecha.

- **Funciones condicionales.** Clausulas CASE y DECODE, cambian los valores en los resultados según las condiciones indicadas.

La documentación oficial de Oracle para la funciones de fila simple se puede encontrar en:

[Single-Row Functions \(oracle.com\)](https://docs.oracle.com/cd/E11882_01/server.112/e41084/functions002.htm#SQLRF51178)

https://docs.oracle.com/cd/E11882_01/server.112/e41084/functions002.htm#SQLRF51178

Y para MySQL:

[MySQL :: Manual de referencia de MySQL 8.0 :: 12 funciones y operadores](https://dev.mysql.com/doc/refman/8.0/en/functions.html)

<https://dev.mysql.com/doc/refman/8.0/en/functions.html>

Funciones numéricas

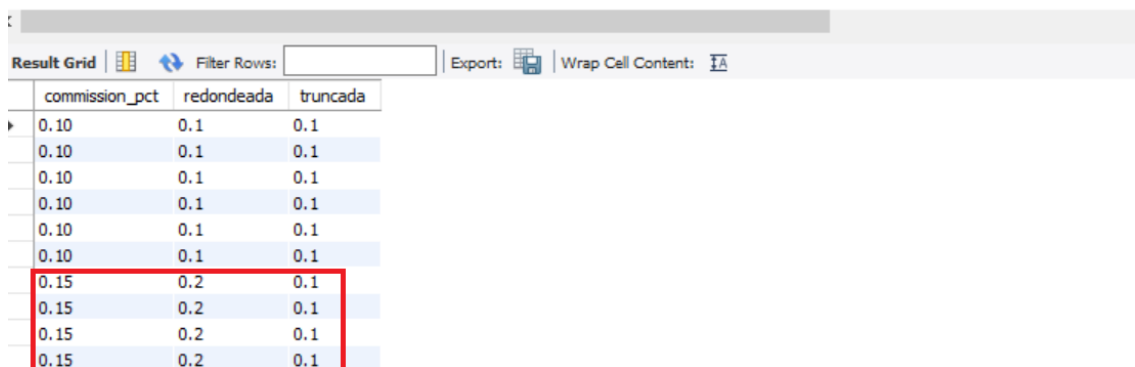
Redondeo y truncado

En MySQL las funciones son round(<valor>,<numero decimales>) y truncate(<valor>,<numero decimales>). En Oracle son round(<valor>,<numero decimales>) y trunc(<valor>,<numero decimales>).

Redondear: redondea el valor al número de decimales especificado y

Trucar: corta el valor al número de decimales especificado.

```
1 • select commission_pct, round(commission_pct,1) redondeada, truncate(commission_pct,1) truncada
2   from employees
3  where commission_pct is not null
4  order by commission_pct;
```



commission_pct	redondeada	truncada
0.10	0.1	0.1
0.10	0.1	0.1
0.10	0.1	0.1
0.10	0.1	0.1
0.10	0.1	0.1
0.10	0.1	0.1
0.15	0.2	0.1
0.15	0.2	0.1
0.15	0.2	0.1
0.15	0.2	0.1

Como se puede ver en la figura el valor 0,15 si se redondea a un decimal toma el valor 0,2, mientras si se trunca el valor toma el valor 0,1 independientemente del valor que tomen los decimales después.

Funciones matemáticas

Hay multitud en ambas bases de datos, el siguiente cuadro expone una relación de las funciones matemáticas de Oracle:

Función	Descripción
---------	-------------

MOD(<i>n1</i> , <i>n2</i>)	Devuelve el resto resultado de dividir <i>n1</i> entre <i>n2</i>
POWER(valor,exponente)	Eleva el valor al exponente indicado
SQRT(<i>n</i>)	Calcula la raíz cuadrada de <i>n</i>
SIGN(<i>n</i>)	Devuelve 1 si <i>n</i> es positivo, cero si vale cero y -1 si es negativo
ABS(<i>n</i>)	Calcula el valor absoluto de <i>n</i>
EXP(<i>n</i>)	Calcula <i>eⁿ</i> , es decir el exponente en base <i>e</i> del número <i>n</i>
LN(<i>n</i>)	Logaritmo neperiano de <i>n</i>
LOG(<i>n</i>)	Logaritmo en base 10 de <i>n</i>
SIN(<i>n</i>)	Calcula el seno de <i>n</i> (<i>n</i> tiene que estar en radianes)
COS(<i>n</i>)	Calcula el coseno de <i>n</i> (<i>n</i> tiene que estar en radianes)
TAN(<i>n</i>)	Calcula la tangente de <i>n</i> (<i>n</i> tiene que estar en radianes)
ACOS(<i>n</i>)	Devuelve en radianes el arco coseno de <i>n</i>
ASIN(<i>n</i>)	Devuelve en radianes el arco seno de <i>n</i>
ATAN(<i>n</i>)	Devuelve en radianes el arco tangente de <i>n</i>
SINH(<i>n</i>)	Devuelve el seno hiperbólico de <i>n</i>
COSH(<i>n</i>)	Devuelve el coseno hiperbólico de <i>n</i>
TANH(<i>n</i>)	Devuelve la tangente hiperbólica de <i>n</i>

En MySQL

Función	Descripción
<u>ABS()</u>	Devolver el valor absoluto
<u>ACOS()</u>	Devolver el coseno de arco
<u>ASIN()</u>	Devolver el arco sinusoidal
<u>ATAN()</u>	Devolver la tangente de arco
<u>ATAN2(), ATAN()</u>	Devolver la tangente de arco de los dos argumentos
<u>CEIL()</u>	Devolver el valor entero más pequeño no inferior al argumento
<u>CEILING()</u>	Devolver el valor entero más pequeño no inferior al argumento
<u>CONV()</u>	Convertir números entre diferentes bases numéricas
<u>COS()</u>	Devolver el coseno
<u>COT()</u>	Devolver la cotangente
<u>CRC32()</u>	Calcular un valor de comprobación de redundancia cíclica
<u>DEGREES()</u>	Convertir radianes en grados
<u>DIV</u>	División entera
<u>EXP()</u>	Elevarse al poder de
<u>FLOOR()</u>	Devolver el valor entero más grande no mayor que el argumento
<u>LN()</u>	Devolver el logaritmo natural del argumento
<u>LOG()</u>	Devolver el logaritmo natural del primer argumento
<u>LOG10()</u>	Devolver el logaritmo base-10 del argumento
<u>LOG2()</u>	Devolver el logaritmo base-2 del argumento
<u>MOD()</u>	Devolver el resto
<u>PI()</u>	Devolver el valor de pi
<u>POW()</u>	Devolver el argumento planteado a la potencia especificada
<u>POWER()</u>	Devolver el argumento planteado a la potencia especificada
<u>RADIANS()</u>	Argumento return convertido en radianes
<u>RAND()</u>	Devolver un valor aleatorio de coma flotante
<u>SIGN()</u>	Devolver el signo del argumento

<u>SIN()</u>	Devolver el seno del argumento
<u>SQRT()</u>	Devolver la raíz cuadrada del argumento
<u>TAN()</u>	Devolver la tangente del argumento

Funciones de cadena

Existen también muchas funciones de cadena en ambas bases de datos, se recomienda visitar la documentación oficial indicada al principio de este capítulo. Vamos a destacar las más utilizadas:

Para Oracle:

Función	Descripción
LOWER(texto)	Convierte el texto a minúsculas (funciona con los caracteres españoles)
UPPER(texto)	Convierte el texto a mayúsculas
INITCAP(texto)	Coloca la primera letra de cada palabra en mayúsculas
RTRIM(texto)	Elimina los espacios a la derecha del texto
LTRIM(texto)	Elimina los espacios a la izquierda que posea el texto
TRIM(texto)	Elimina los espacios en blanco a la izquierda y la derecha del texto y los espacios dobles del interior.
TRIM(caracteres FROM texto)	Elimina del texto los caracteres indicados. Por ejemplo TRIM('h' FROM nombre) elimina las haches de la columna nombre que estén a la izquierda y a la derecha
SUBSTR(texto,n[,m])	<p>Obtiene los m siguientes caracteres del texto a partir de la posición n (si m no se indica se cogen desde n hasta el final). Ejemplo:</p> <pre>SELECT FROM DUAL;</pre> <p>Escribiría el texto rue (ya que son las tres siguientes letras desde la posición dos).</p> <p>Si el parámetro n es un número negativo, entonces la posición inicial del texto a extraer se calcula de derecha a izquierda así por ejemplo la instrucción:</p> <pre>SELECT FROM DUAL;</pre> <p>Mostraría el texto: eb</p>
LENGTH(texto)	Obtiene el tamaño del texto. Así: LENGTH('Hola') devolvería cuatro.
REPLACE(texto, textoA Buscar, [textoReemplazo])	<p>Buscar el texto a buscar en un determinado texto y lo cambia por el indicado como texto de reemplazo.</p> <p>Si no se indica texto de reemplazo, entonces esta función elimina el texto que se busca.</p>
LPAD(texto, anchuraMáxima, [carácterDeRelleno])	<p>Rellena el texto a la izquierda (LPAD) o a la derecha (RPAD) con el carácter indicado para ocupar la anchura indicada.</p> <p>Si el texto es más grande que la anchura indicada, el texto se recorta.</p>

RPAD(texto, anchuraMáxima, [caracterDeRelleno])	<p>Si no se indica carácter de relleno se rellena el espacio marcado con espacios en blanco.</p> <p>Ejemplo: LPAD('Hola',10,'-')</p> <p>devuelve como resultado: -----Hola</p>
REVERSE(texto)	Invierte el texto (le da la vuelta)
INSTR(texto, textoBuscado [,posInicial [, nAparición]])	<p>Obtiene la posición en la que se encuentra el texto buscado en el texto inicial. Ejemplo:</p> <pre>SELECT INSTR('es 'es') FROM DUAL;</pre> <p>Escribe el valor: 0</p> <p>Se puede empezar a buscar a partir de una posición inicial concreta, por ejemplo para buscar desde el tercer carácter.</p> <pre>SELECT INSTR('es 'es',3) FROM DUAL;</pre> <p>Devuelve 11</p> <p>La posición inicial podría ser negativa y en ese caso buscaríamos desde el lado derecho.</p> <pre>SELECT INSTR('es 'es',1) FROM DUAL;</pre> <p>También devuelve 11</p> <p>Incluso podemos indicar el número de aparición del texto buscado:</p> <pre>SELECT INSTR('es 'es',1,2) FROM DUAL;</pre> <p>Vuelve a mostrar 11 (posición de la segunda vez que aparece la palabra es). Si no se encuentra el texto a buscar, la función devuelve el número cero.</p>
TRANSLATE(texto, caracteresACambiar, caracteresSustitutivos)	<p>Potentísima función que permite transformar caracteres. Se cambia una serie de caracteres por otros que se indiquen en el mismo orden.</p> <p>De tal modo que, el primer carácter a cambiar se cambia por el primer carácter sustitutivo, el segundo por el segundo y así sucesivamente. Ejemplo:</p> <pre>SELECT 'wx') FROM DUAL;</pre> <p>El resultado sería el texto prwxba, de tal forma que la u se cambia por la w y la e por la x.</p>

	Si la segunda cadena es más corta, los caracteres de la primera que no encuentran sustituto, se eliminan. Ejemplo: SELECT FROM DUAL; Da como resultado prwba
ASCII(carácter)	Devuelve el código ASCII del carácter indicado. Si se le pasa más de un carácter, devuelve el código ASCII del primer carácter-
CHR(número)	Devuelve el carácter correspondiente al código ASCII indicado

En MySQL:

Nombre	Descripción
<u>ASCII()</u>	Valor numérico devuelto del carácter más a la izquierda
<u>CHAR()</u>	Devolver el carácter de cada entero pasado
<u>CHAR_LENGTH()</u>	Devolver el número de caracteres en el argumento
<u>CHARACTER_LENGTH()</u>	Sinónimo de CHAR_LENGTH()
<u>CONCAT()</u>	Devolver cadena concatenada
<u>CONCAT_WS()</u>	Retorno concatenado con separador
<u>FIELD()</u>	Índice (posición) del primer argumento en los argumentos posteriores
<u>FORMAT()</u>	Devolver un número con formato al número especificado de decimales
<u>HEX()</u>	Representación hexadecimal del valor decimal o de cadena
<u>INSERT()</u>	Insertar subcadena en la posición especificada hasta el número especificado de caracteres
<u>INSTR()</u>	Devolver el índice de la primera aparición de la subcadena
<u>LCASE()</u>	Sinónimo de LOWER()
<u>LEFT()</u>	Devolver el número de caracteres más a la izquierda especificado
<u>LENGTH()</u>	Devolver la longitud de una cadena en bytes
<u>LOCATE()</u>	Devolver la posición de la primera aparición de la subcadena
<u>LOWER()</u>	Devolver el argumento en minúsculas
<u>LPAD()</u>	Devolver el argumento string, acolchado a la izquierda con la cadena especificada
<u>LTRIM()</u>	Quitar espacios principales
<u>MATCH()</u>	Realizar búsqueda de texto completo
<u>MID()</u>	Devolver una subcadena a partir de la posición especificada
<u>POSITION()</u>	Sinónimo de LOCATE()
<u>REGEXP_INSTR()</u>	Índice inicial de la expresión regular coincidente de subcadenas
<u>REGEXP_LIKE()</u>	Si la cadena coincide con la expresión regular
<u>REGEXP_REPLACE()</u>	Reemplazar subcadenas que coincidan con la expresión regular
<u>REGEXP_SUBSTR()</u>	Devolver subcadena que coincide con la expresión regular
<u>REPEAT()</u>	Repita una cadena el número especificado de veces
<u>REPLACE()</u>	Reemplazar apariciones de una cadena especificada
<u>REVERSE()</u>	Invertir los caracteres de una cadena
<u>RIGHT()</u>	Devolver el número de caracteres especificado a la derecha
<u>RPAD()</u>	Anexar cadena el número especificado de veces
<u>RTRIM()</u>	Eliminar espacios finales
<u>STRCMP()</u>	Comparar dos cadenas

<u>SUBSTR()</u>	Devolver la subcadena como se especifica
<u>SUBSTRING()</u>	Devolver la subcadena como se especifica
<u>SUBSTRING INDEX()</u>	Devolver una subcadena de una cadena antes del número especificado de apariciones del delimitador
<u>TRIM()</u>	Eliminar espacios principales y finales
<u>UCASE()</u>	Sinónimo de UPPER()
<u>UPPER()</u>	Convertir a mayúsculas

Funciones de tratamiento de nulos

Son funciones que o nos encuentran los valores nulos y sustituyen por otro valor, o según el valor de una condición nos fijan un valor nulo, etc.

Son ampliamente usadas en cualquier sistema.

Para Oracle:

Función	Descripción
NVL(valor,sustituto)	Si el <i>valor</i> es NULL, devuelve el valor <i>sustituto</i> ; de otro modo, devuelve valor
NVL2(valor,sustituto1, sustituto2)	Variante de la anterior, devuelve el valor <i>sustituto1</i> si <i>valor</i> no es nulo. Si <i>valor</i> es nulo devuelve el <i>sustituto2</i>
NULLIF(valor1,valor2)	Devuelve nulo si <i>valor1</i> es igual a <i>valor2</i> . De otro modo devuelve <i>valor1</i>

En MySQL:

Función	Descripción
<u>IFNULL ()</u>	Null if/else construct
<u>NULLIF ()</u>	Devolver NULL si expr1 = expr2

Funciones de fechas

Este grupo es uno de los que más funciones tienen, ya que el trabajo con fechas es muy importante y muy pesado, las funciones tienen como objetivo facilitar la definición y ejecución de las queries.

Para Oracle:

Función	Descripción
SYSDATE	Obtiene la fecha y hora actuales
SYSTIMESTAMP	Obtiene la fecha y hora actuales en formato TIMESTAMP

Función	Descripción
ADD_MONTHS(fecha,n)	Añade a la fecha el número de meses indicado por <i>n</i>
MONTHS_BETWEEN(fecha1, fecha2)	Obtiene la diferencia en meses entre las dos fechas (puede ser decimal)

NEXT_DAY(fecha,día)	Indica cual es el día que corresponde a añadir a la fecha el día indicado. El día puede ser el texto 'Lunes', 'Martes', 'Miércoles',... (si la configuración está en español) o el número de día de la semana (1=lunes, 2=martes,...)
LAST_DAY(fecha)	Obtiene el último día del mes al que pertenece la fecha. Devuelve un valor DATE
EXTRACT(valor FROM fecha)	Extrae un valor de una fecha concreta. El valor puede ser day (día), month (mes), year (año), etc.
GREATEST(fecha1, fecha2,...)	Devuelve la fecha más moderna la lista
LEAST(fecha1, fecha2,...)	Devuelve la fecha más antigua la lista
ROUND(fecha [, 'formato'])	Redondea la fecha al valor de aplicar el formato a la fecha. El formato puede ser: 'YEAR' Hace que la fecha refleje el año completo 'MONTH' Hace que la fecha refleje el mes completo más cercano a la fecha 'HH24' Redondea la hora a las 00:00 más cercanas 'DAY' Redondea al día más cercano
TRUNC(fecha [,formato])	Igual que el anterior pero trunca la fecha en lugar de redondearla.

En MySQL:

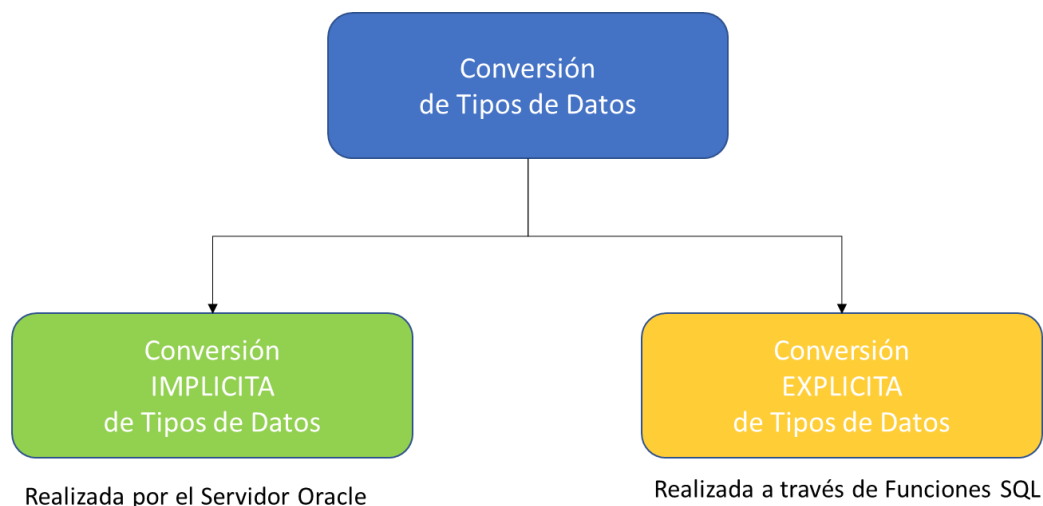
Nombre	Descripción
<u>ADDDATE ()</u>	Agregar valores de tiempo (intervalos) a un valor de fecha
<u>ADDTIME ()</u>	Añadir tiempo
<u>CONVERT TZ ()</u>	Convertir de una zona horaria a otra
<u>CURDATE ()</u>	Devolver la fecha actual
<u>CURRENT DATE (), CURRENT DATE</u>	Sinónimos de CURDATE()
<u>CURRENT TIME (), CURRENT TIME</u>	Sinónimos de CURTIME()
<u>CURRENT_TIMESTAMP (), CURRENT_TIMESTAMP</u>	Sinónimos de NOW()
<u>CURTIME ()</u>	Devolver la hora actual
<u>DATE ()</u>	Extraer la parte de fecha de una expresión de fecha o fecha y hora
<u>DATE_ADD ()</u>	Agregar valores de tiempo (intervalos) a un valor de fecha
<u>DATE_FORMAT ()</u>	Dar formato a la fecha especificada
<u>DATE_SUB ()</u>	Restar un valor de tiempo (intervalo) de una fecha
<u>DATEDIFF ()</u>	Restar dos fechas
<u>DAY ()</u>	Sinónimo de DAYOFMONTH()
<u>DAYNAME ()</u>	Devolver el nombre del día laborable
<u>DAYOFMONTH ()</u>	Devolver el día del mes (0-31)
<u>DAYOFWEEK ()</u>	Devolver el índice de días laborables del argumento
<u>DAYOFYEAR ()</u>	Regresa el día del año (1-366)
<u>EXTRACT ()</u>	Extraer parte de una fecha
<u>HOURL ()</u>	Extraer la hora
<u>LAST DAY</u>	Devolver el último día del mes para el argumento
<u>LOCALTIME (), HORA LOCAL</u>	Sinónimo de NOW()
<u>LOCALTIMESTAMP, LOCALTIMESTAM MP ()</u>	Sinónimo de NOW()

<u>MINUTE ()</u>	Devolver el minuto del argumento
<u>MONTH ()</u>	Devolver el mes a partir de la fecha de paso
<u>MONTHNAME ()</u>	Devolver el nombre del mes
<u>NOW ()</u>	Devolver la fecha y hora actuales
<u>SECOND ()</u>	Vuelve el segundo (0-59)
<u>SUBTIME ()</u>	Restar tiempos
<u>SYSDATE ()</u>	Devolver la hora a la que se ejecuta la función
<u>TIME ()</u>	Extraer la parte de tiempo de la expresión pasada
<u>TIME TO SEC ()</u>	Devolver el argumento convertido a segundos
<u>TIMEDIFF ()</u>	Restar tiempo
<u>TIMESTAMP ()</u>	Con un solo argumento, esta función devuelve la expresión date o datetime; con dos argumentos, la suma de los argumentos
<u>TIMESTAMPADD ()</u>	Agregar un intervalo a una expresión datetime
<u>TIMESTAMPDIFF ()</u>	Restar un intervalo de una expresión datetime
<u>TO_DAYS ()</u>	Devolver el argumento de fecha convertido a días
<u>TO_SECONDS ()</u>	Devolver el argumento date o datetime convertido en segundos desde el año 0
<u>WEEK ()</u>	Devolver el número de semana
<u>WEEKDAY ()</u>	Devolver el índice de días laborables
<u>WEEKOFYEAR ()</u>	Devolver la semana natural de la fecha (1-53)
<u>YEAR ()</u>	Devolver el año
<u>YEARWEEK ()</u>	Devolver el año y la semana

Funciones de conversión de datos

Hay dos tipos de conversiones en todo sistema SGDB (DBMS):

- **Implícita**, la realiza el propio sistema: normalmente de cadena a número y de número a cadena y de cadena a fecha. Estas conversiones se realizan sin formato en el caso de los números y en el caso de las fechas, con el formato por defecto que tenga cada sistema.
- **Explícita**, se van a utilizar funciones de conversión de datos, que normalmente conllevan un formato específico de datos.



Ejemplo de conversión implícita en Oracle

Oracle es capaz de convertir datos automáticamente a fin de que la expresión final tenga sentido.

```
SELECT 5+'3' FROM DUAL --El resultado es 8
```

```
SELECT 5 || '3' FROM DUAL -- El resultado es 53
```

También ocurre eso con la conversión de textos a fechas.

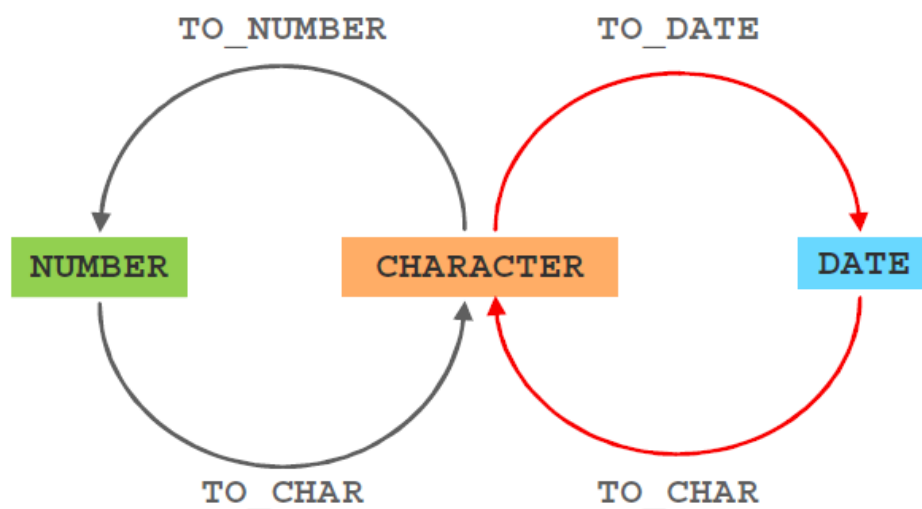
```
SELECT SYSDATE - 1/1/2000 FROM DUAL;
```

Nos devolverá los días que han pasado desde el año 2000 ya que entiende 1/1/2000 como una fecha y no como la división de tres números.

No obstante, para las fechas deberemos saber cómo especificarlas y eso además depende de cada sistema.

Por ello lo más seguro es utilizar funciones de conversión, ya podremos pasar de unos tipos de datos a otros y además en la forma que especifiquemos

Para Oracle:



TO_CHAR

Permite convertir una fecha o un número a texto . Su sintaxis es:

```
TO_CHAR(expresión [,modeloDeFormato])
```

El primer parámetro es la expresión que queremos convertir. El segundo (aunque es opcional es raro no utilizarle) es un texto (por lo que va entrecomillado) que contiene la forma de convertir deseada. Los códigos posibles se explican más adelante.

Ejemplo

```
SELECT TO_CHAR(SYSDATE,'dd/month/yyyy') FROM DUAL;
```

TO_DATE

Funciona como las anteriores, pero se encarga de convertir un texto en una fecha. Imprescindible para añadir fechas a las tablas.

Ejemplos

```
SELECT TO_CHAR(SYSDATE, 'DD/MONTH/YYYY, DAY HH:MI:SS')  
FROM DUAL ;
```

Sale : 16/AGOSTO /2021, LUNES 08:35:15, por ejemplo

En la función TO_CHAR es posible indicar texto literal. Para ello se le encierra entre comillas dobles.

```
SELECT TO_CHAR(SYSDATE, 'Day DD "de" MONTH "de" YYYY')  
FROM DUAL ;
```

Sale, por ejemplo: Lunes 16 de Agosto de 2021

Patrones de Fecha

El modelo de Formato se denomina patrón y en el caso de Oracle para cambiar de fecha a cadena o de cadena a fecha son los siguientes.

Función	Descripción
RR	Formato de año en dos cifras. De modo que si el año actual es superior al 50 del siglo en el que estemos, si especificamos un año inferior al 50, se referirá al siglo anterior. Si estamos en un año inferior al 50, los años indicamos con cifras superiores a 50 se referirán al siglo anterior. Es el formato de dos cifras más natural y más pensado para el futuro.
Y	Última cifra del año
YY	Año en formato de dos cifras
YYY	Año en formato de tres cifras
YYYY	Año en formato de cuatro cifras
SYYYY	Igual que el anterior, pero si el año es anterior a Cristo muestra un signo negativo
I IY IYY IYYY	Año en formato de una, dos, tres o cuatro cifras según la notación ISO.
YEAR	Año en formato hablado (en inglés). Ejemplo: <pre>SELECT TO_CHAR(SYSDATE,'day/month/year') FROM DUAL;</pre> Resulta:

	sábado/abril/twenty fourteen
CC	Siglo (por ejemplo 21)
SCC	Siglo pero si es anterior a Cristo lo indica con un signo menos
BC o AD	Indica BC (en español usará AC) o AD (en español DC) si el año es anterior o posterior, respectivamente, al nacimiento de Jesucristo
MM	Mes en formato de dos cifras
MON	Las tres primeras letras del mes
MONTH	Nombre completo del mes
RM	Mes en romano
DY	Día de la semana en tres letras
DAY	Día completo de la semana
D	Día de la semana (del 1 al 7)
DD	Día del mes en formato de dos cifras (del 1 al 31)
DDD	Día del año
Q	Trimestre
WW o W	Semana del año
IW	Semana del año en formato ISO
J	Día juliano. Número de días transcurridos desde el 31 de Diciembre del 4713 A.C.
HH12	Hora en formato de 1 a 12
HH24	Hora en formato 24 horas (de 0 a 23)
MI	Minutos (0 a 59)
SS	Segundos (0 a 59)
SSSS	Segundos desde medianoche
AM o PM	Indicador de meridiano utilizado para mostrar la hora en formato de 12 horas. Mostrará AM o PM según corresponda
/ . , ; ' ,	Posición de los separadores, donde se pongan estos símbolos aparecerán en el resultado

TO_NUMBER

El funcionamiento es igual que TO_CHAR, sólo que la labor de TO_NUMBER es la contraria, convierte un texto en un número. Se utiliza para introducir valores en las tablas, normalmente.

Patrones de Números

Función	Descripción
9	Representa una posición para un dígito numérico
0	Representa una posición para un dígito numérico; si en esa posición no hay número, se rellenará con un cero
\$	Usa formato dólar
L	Usa el símbolo local de la moneda
S	Posición para el signo del número
D	Posición del símbolo decimal (en español, la coma)
G	Posición del separador de grupo (en español el punto)
.	Punto decimal (utilizado para números en formato británico)
,	Coma de separador de grupo (utilizado para números en formato británico)
MI	Signo menos (se usa sólo a la derecha del número)
PR	Muestra los números negativos entre paréntesis
EEEE	Número en notación científica

U	Símbolo del euro
V	Multiplica al número por 10 tantas veces como se indique
B	Muestra el valor cero como espacio y no como cero

En MySQL:

Se tienen las mismas conversiones, pero se utilizan las siguientes funciones:

- [STR_TO_DATE](#): para convertir una cadena a fecha
- [DATE_FORMAT](#): para convertir una fecha a cadena

Se utilizan del mismo modo que TO_CHAR y TO_DATE, el primer parámetro es la cadena o la fecha y el segundo en el patrón de formato

Patrones de fechas en MySQL

Especificador	Descripción
%a	Nombre abreviado del día de la semana (..SunSat)
%b	Nombre abreviado del mes (..JanDec)
%c	Mes, numérico (..012)
%D	Día del mes con sufijo inglés (,,,...)0th1st2nd3rd
%d	Día del mes, numérico (..0031)
%e	Día del mes, numérico (..031)
%f	Microsegundos (..000000999999)
%H	Hora (..0023)
%h	Hora (..0112)
%I	Hora (..0112)
%i	Minutos, numéricos (..0059)
%j	Día del año (..001366)
%k	Hora (..023)
%l	Hora (..112)
%M	Nombre del mes (..JanuaryDecember)
%m	Mes, numérico (..0012)
%p	AM o PM
%r	Tiempo, 12 horas (hh:mm:ss seguido de o AMPM)
%S	Segundos (..0059)
%s	Segundos (..0059)
%T	Tiempo, 24 horas (hh:mm:ss)
%U	Semana (..), donde el domingo es el primer día de la semana; <u>WEEK()</u> modo 00053
%u	Semana (..), donde el lunes es el primer día de la semana; <u>SEMANA()</u> modo 10053
%V	Semana (..), donde el domingo es el primer día de la semana; <u>SEMANA()</u> modo 2; utilizado con 0153%X
%v	Semana (..), donde el lunes es el primer día de la semana; <u>SEMANA()</u> modo 3; utilizado con 0153%x
%W	Nombre del día laborable (..SundaySaturday)
%w	Día de la semana (=Domingo.. =Sábado)06
%X	Año para la semana donde el domingo es el primer día de la semana, numérico, de cuatro dígitos; utilizado con %V
%x	Año para la semana, donde el lunes es el primer día de la semana, numérico, de cuatro dígitos; utilizado con %v
%Y	Año, numérico, cuatro dígitos

%Y	Año, numérico (dos dígitos)
%%	Un carácter literal%
%x	x, para cualquier "x" no mencionado anteriormente

Conversiones avanzadas CAST

CAST es una función muy versátil que permite convertir el resultado a un tipo concreto.

CAST(expresión AS tipoDatos)

Ejemplos

```
SELECT CAST(4-6 AS SIGNED);
```

Resultado: -2

```
SELECT CAST(4-6 AS UNSIGNED);
```

Resultado: 18446744073709551614

Mas información:

[MySQL: CAST Function \(techonthenet.com\)](https://techonthenet.com/mysql/cast-function/)

[Oracle CAST Function Explained with Examples \(databasestar.com\)](https://databasestar.com/oracle/cast-function-explained-with-examples/)

Funciones Condicionales

Función Case

Realmente CASE no es una función al uso. Pero su utilidad es la misma, al final devuelve un valor.

Se trata de un elemento que da mucha potencia a las instrucciones SQL ya que simula una estructura de tipo if-then-else que es una de las bases de la programación en lenguajes de ordenador. La sintaxis es:

CASE [expresión]

WHEN *expresión_comparación1* THEN *valor_devuelto1*

[WHEN *expresión_comparación2* THEN *valor_devuelto2*

...

[ELSE *valor_devuelto_else*]]

END;

Pasos:

1. Se evalúa la expresión
2. Se compara su valor con el de la primera expresión de comparación
3. Si coinciden, se devuelve el valor con el del primer THEN

4. Si no, se compara con la expresión del siguiente WHEN (si le hay) y así sucesivamente para todos los WHEN
5. Si la expresión no coincide con la de ningún WHEN, entonces se devuelve el valor que posee el apartado ELSE (si le hay)

Ejemplo

```
SELECT nombre,  
  
       CASE commission    WHEN 1 THEN cuota*0.85  
                           WHEN 2 THEN cuota *0.93  
                           WHEN 3 THEN cuota *0.96  
                           ELSE cuota  
  
       END  
  
FROM empleados;
```

En el ejemplo, se calcula una columna a partir de la comisión de modo que el cálculo varía en función de lo que vale la columna comisión. En el caso de que esa columna no valga ni uno, ni dos, ni tres, se mostrará la cuota tal cual (para eso sirve el apartado ELSE).

Función DECODE

Función que permite realizar condiciones en una consulta. Simplifica la función anterior (aunque tiene menos posibilidades que la anterior). Se evalúa una expresión y se colocan a continuación pares valor-resultado de forma que si se la expresión equivale al valor, se obtiene el resultado indicado.

Se puede indicar un último parámetro con el resultado a efectuar en caso de no encontrar ninguno de los valores indicados (equivalente al apartado ELSE de la función anterior).

**DECODE(expresión, valor1, resultado1
 [,valor2, resultado2,...]
 [,valorPordefecto])**

Ejemplo

```
SELECT nombre,  
  
       DECODE(comisión,1, cuota*0.85,  
              2,cuota * 0.93,  
              3,cuota * 0.96,  
              salario)  
  
FROM empleados;
```

En el ejemplo dependiendo de la cotización se muestra rebajado la cuota: un 85% si la comisión es uno, un 93 si es dos y un 96 si es tres. Si la comisión no es ni uno ni dos ni tres, sencillamente se muestra la cuota sin más.

Consultas Multi-tabla

La mayoría de las consultas se realizan sobre más de una tabla, estas tablas deben estar previamente relacionadas mediante un modelo Entidad Relación.

Extraer información sobre tablas que no estén relacionadas de algún modo entre sí, se puede hacer, pero no se debe, ya que la información resultante no tendría valor.

La relación puede ser lógica o física, ambas deberían coincidir.

Se establecen relaciones entre tablas a través de claves secundarias o foreign-keys. Una tabla puede contener múltiples claves secundarias que la relacionan con otras tablas y así mismo los campos de esa tabla, normalmente su identificador, puede ser claves secundarias en otras.

Las “uniones” o relaciones que especificamos entre tablas a la hora de realizar consultas es lo que se denomina JOINS.

Alias de tabla

Para hacer referencia a las tablas en las consultas y evitar campos ambiguos, que no se sepa a que tabla puedan pertenecer, se usan los alias de tabla.

Los alias de tabla se indican con una palabra(o letra) al lado derecho de la tabla y todos los campos de la misma se deben referir a esa palabra o letra.

```
SELECT e.empleados
FROM empleados e
WHERE e.salary >9000
```

Suele ser la primera inicial de la tabla pero podría ser cualquier palabra.

```
SELECT pepe.empleados
FROM empleados pepe
WHERE pepe.salary >9000
```

Se evitan ambigüedades, el primer campo department_id es de la tabla empleados y el segundo campo department_id es el de la tabla de departamentos.

```
SELECT e.department_id, d.department_id
FROM employees e, departments d
```

Producto cartesiano

Si se consultan dos tablas sin indicar la relación o JOIN con la que se quiere relacionar, exista o no la relación, se produce el producto cartesiano de resultados, esto significa que se muestran tantos resultados como la multiplicación de los registros de una tabla por los registros de la otra. Ya que se seleccionan todos los registros de una tabla por cada registro de la otra.

Ejemplo

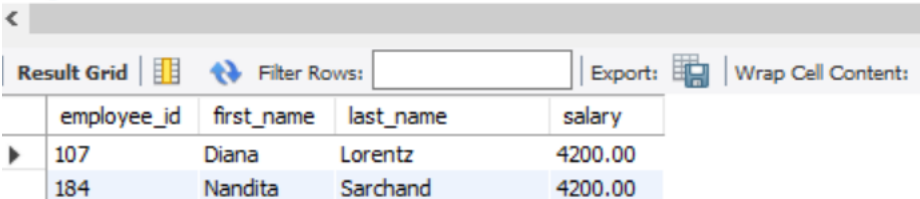
Vamos a suponer que la tabla empleados tiene 107 registros y la tabla departamentos 28, entonces, la siguiente consulta, devolvería $107 \times 28 = 2996$ registros

```
SELECT e.employee_id, d.department_id  
FROM employees e, departments d;
```

Los productos cartesianos se utilizan para consultas que hacen referencia a la propia tabla.

“Mostrar la información de los empleados que tienen el mismo salario que el empleado Diana Lorentz “

```
1 • SELECT e.employee_id, e.first_name, e.last_name, e.salary  
2 FROM employees e, employees e1  
3 WHERE e1.first_name='Diana'  
4 AND e1.last_name='Lorentz'  
5 AND e.salary=e1.salary;  
6
```



The screenshot shows a SQL query result grid with the following data:

employee_id	first_name	last_name	salary
107	Diana	Lorentz	4200.00
184	Nandita	Sarchand	4200.00

En estos casos usamos el producto cartesianos porque sabemos que por cada registro de una tabla se van a seleccionar todos los registros de la otra. Siguiendo esta cualidad, se van a usar dos juegos de datos de la tabla empleados uno para seleccionar los empleados en sí y otra para encontrar el empleado que nos indican y encontrar su salario, que vamos a utilizar para filtrar los datos de los empleados a buscar.

Vamos a fijarnos en el filtro, los dos primeros filtran el segundo juego de empleados (empleados e1) para determinar el empleado: nombre = 'Diana' y apellido 'Lorentz', con esto el juego de datos (empleado e1) seleccionaría un único campo. Ahora filtramos el primero con el valor del salario del empleado 'Diana Lorentz' por tanto se nos seleccionan todos los empleados del primer juego (empleados e).

Ejemplo

“Mostrar la información de los empleados que trabajan en el mismo departamento que el empleado Diana Lorentz “

```

1 • SELECT e.employee_id,e.first_name, e.last_name,e.department_id
2 FROM employees e, employees e1
3 WHERE e1.first_name='Diana'
4 AND e1.last_name='Lorentz'
5 AND e.department_id=e1.department_id;
6

```

employee_id	first_name	last_name	department_id
103	Alexander	Hunold	60
104	Bruce	Ernst	60
105	David	Austin	60
106	Valli	Pataballa	60
107	Diana	Lorentz	60

Este ejemplo se trata exactamente igual que el anterior, pero hay que tener cuidado en el filtro “e.department_id=e1.department_id” porque como veremos a continuación al igualar identificadores puede verse como un enlace (join) entre tablas, pero no, es un filtro, simplemente iguala el valor de dos campos, no existe un enlace entre la misma tabla(employees e y employees e1) por el campo department_id.

Un join se denomina cuando las tablas tienen un enlace físico o lógico entre ellas, que se pudiera establecer una clave ajena(foreign key) . En el caso de dos tablas de empleados, por ejemplo por los campos employee_id y manager_id, que sí que tiene establecida una clave ajena.

Equal Join

Los equal join se les denomina asociaciones simples. Se va a igualar mediante filtros los identificadores de la clave ajena, porque deben tener los mismos valores.

“Seleccionar los datos del empleado (identificador, nombre y apellido) y el nombre de departamento de cada empleado”

```

1 • SELECT e.employee_id,e.first_name, e.last_name,d.department_name
2 FROM employees e, departments d
3 WHERE d.department_id=e.department_id
4 ORDER BY e.employee_id;
5

```

employee_id	first_name	last_name	department_name
100	Steven	King	Executive
101	Neena	Kochhar	Executive
102	Lex	De Haan	Executive
103	Alexander	Hunold	IT
104	Bruce	Ernst	IT
105	David	Austin	IT
106	Valli	Pataballa	IT
107	Diana	Lorentz	IT
108	Nancy	Greenberg	Finance
109	Daniel	Faviet	Finance
110	John	Chen	Finance
111	Ismael	Sciarra	Finance
112	Jose Manuel	Urman	Finance

El enlace entre tablas se realiza en la sección de filtros.

Inner Join

Los inner join se denominan cuando los enlaces deben estar obligatoriamente establecidos, se van a seleccionar únicamente los registros que tienen establecida esa relación. Son equivalentes a los equal join, aunque se supone que son más óptimos, ya que se establecen en la sección de FROM donde se establece de donde se van a sacar los datos, y no se tendría tanta información en memoria, no como los equal join que tendrían todos los datos de las tablas en memoria, hasta que en la siguiente sección la de filtro (WHERE) que es donde se eliminarían los registros sobrantes.

Los enlaces entre tablas siempre se tienen que realizar entre campos que tengan campos de tipos compatibles.

La estructura de los inner join 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.columa=tabla2.columna)]
```

Natural Join

```
SELECT tabla1.columna1, tabla1.columna2,...  
tabla2.columna1, tabla2.columna2,... FROM tabla1  
NATURAL JOIN tabla2
```

Un NATURAL JOIN se enlaza entre todos los campos que se denominan igual entre dos tablas sin que tengamos que especificar nada en la consulta.

Esta característica es el criterio final para usarlo o no, ya que por un lado es muy cómodo de utilizar, pero por otro puede llevar fácilmente a error.

Es el enlace más restrictivo, donde se pueda utilizar Natural, se puede utilizar USING y ON.

Todo Natural JOIN debe cumplir: [Los enlaces entre tablas siempre se tienen que realizar entre campos que tengan campos de tipos compatibles.](#)

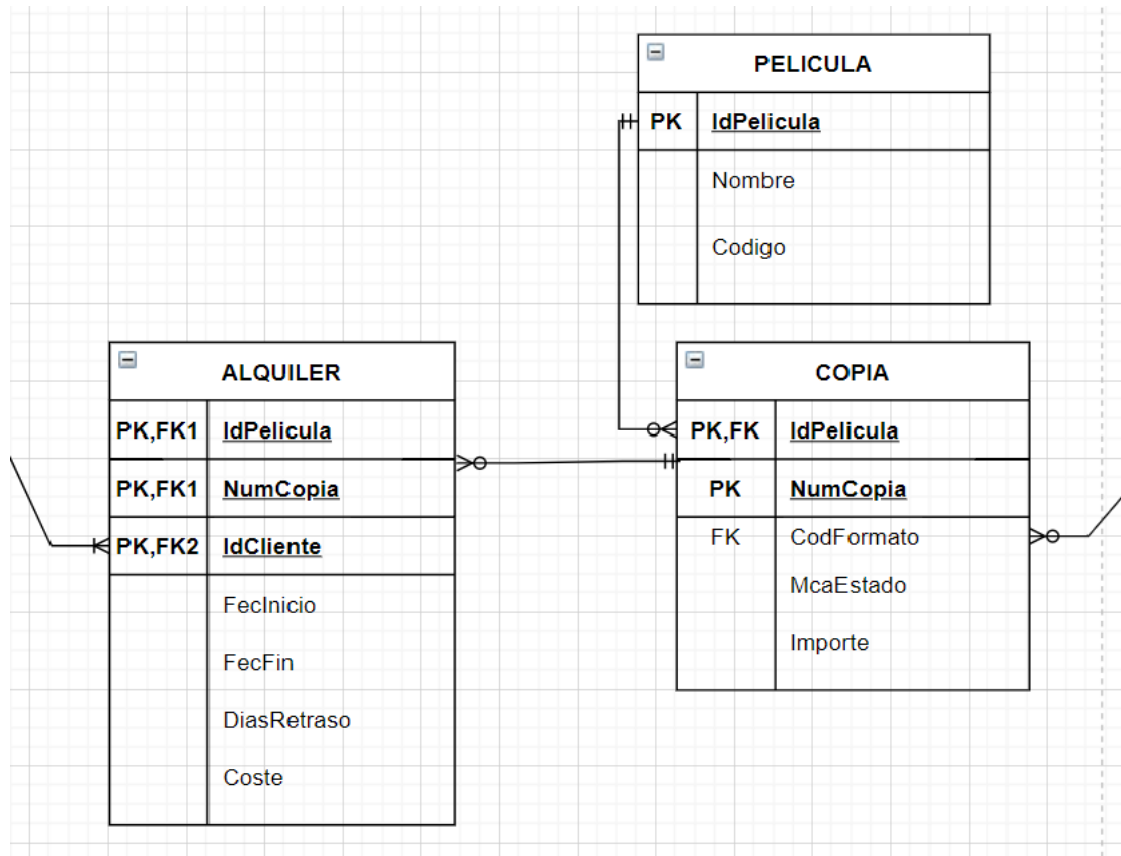
El Natural Join está pensado para unir la clave de primaria de una tabla, ya sea de un campo o sea compuesta, otra tabla donde estos campos estén definidos como clave ajena. Si fuera una clave compuesta, valdría también que uno solo de los campos estuviera en otra tabla como clave ajena.

Para entender este concepto vamos a repasarlo con ejemplos:

Ejemplo de buen uso

- Con dos Campos

Tenemos el siguiente modelo relacional:



“Conseguir la información de los alquileres de las copias de un videoclub”

La sentencia a ejecutar sería la siguiente:

```
SELECT A.* FROM ALQUILER A NATURAL JOIN COPIA C;
```

Esta sentencia va a seleccionar todos los alquileres que pertenecen a una copia determinada, la unión se realiza por dos campos: IdPelicula y NumCopia que identifican la pk de Copia y al mismo tiempo son campos fk de Alquiler (independientemente de que también formen parte de su pk, no tendrían porqué, simplemente ser foreign keys)

- Con un Campo

“Conseguir los datos de dirección de los departamentos, seleccionar el identificador del departamento y todos los datos de su dirección, ordenar por identificador de departamento”

```

1 • select d.department_id, l.*
2   from departments d natural join locations l
3   order by d.department_id;

```

department_id	location_id	street_address	postal_code	city	state_province	country_id
10	1700	2004 Charade Rd	98199	Seattle	Washington	US
20	1800	147 Spadina Ave	M5V 2L7	Toronto	Ontario	CA
30	1700	2004 Charade Rd	98199	Seattle	Washington	US
40	2400	8204 Arthur St	NULL	London	NULL	UK
50	1500	2011 Interiors Blvd	99236	South San Francisco	California	US
60	1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US
70	2700	Schwanthalerstr. 7031	80925	Munich	Bavaria	DE
80	2500	Magdalen Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK
90	1700	2004 Charade Rd	98199	Seattle	Washington	US

Ejemplo de mal uso

“Conseguir los datos de empleado(id,nombre y apellido) y el nombre de su departamento, ordenar por identificador de empleado”

```

1 • select e.employee_id, e.first_name, e.last_name, d.department_name
2   from employees e NATURAL JOIN departments d
3   order by employee_id;

```

employee_id	first_name	last_name	department_name
101	Neena	Kochhar	Executive
102	Lex	De Haan	Executive
104	Bruce	Ernst	IT
105	David	Austin	IT
106	Valli	Pataballa	IT
107	Diana	Lorentz	IT

Esta query no es correcta por que la tabla Employees y la tabla Departments tienen dos campos que tienen el nombre igual y los tipos compatibles (department_id y manager_id) pero únicamente department_id identifica la relación, manager_id sin embargo tiene significados diferentes y no están enlazados entre sí. El campo manager_id de Employees identifica al jefe de ese empleado y el campo manager_id de Departments identifica al jefe del departamento.

Nota: Hay que tener cuidado porque cuando se da este caso los sistemas no informan que el NATURAL JOIN está mal construido y no es correcto.

Clausula Usign

SELECT tabla1.columna1, tabla1.columna2,...

tabla2.columna1, tabla2.columna2,... FROM tabla1

JOIN tabla2 USING(columna)

Un USIGN JOIN se enlaza entre uno o varios campos especificados en la cláusula USIGN que se deben denominar igual en las dos tablas.




Todo USIGN JOIN debe cumplir: Los enlaces entre tablas siempre se tienen que realizar entre campos que tengan campos de tipos compatibles.

Donde se pueda utilizar USIGN, se puede utilizar ON.

Ejemplo

“Conseguir los datos de empleado(id,nombre y apellido) y el nombre de su departamento, ordenar por identificador de empleado”

```
1 • select e.employee_id, e.first_name, e.last_name, d.department_name
2   from employees e JOIN departments d USING(department_id)
3   order by employee_id;
```

Result Grid  Filter Rows: <input type="text"/> Export:  Wrap Cell Content: 				
	employee_id	first_name	last_name	department_name
▶	100	Steven	King	Executive
	101	Neena	Kochhar	Executive
	102	Lex	De Haan	Executive
	103	Alexander	Hunold	IT
	104	Bruce	Ernst	IT
	105	David	Austin	IT
	106	Valli	Pataballa	IT
	107	Diana	Lorentz	IT
	108	Nancy	Greenberg	Finance
	109	Daniel	Faviet	Finance
	110	John	Chen	Finance
	111	Ismael	Sciarra	Finance
	112	Jose Manuel	Urman	Finance
	113	Luis	Popp	Finance
	114	Den	Ranhaelv	Purchasing

Clausula On

SELECT tabla1.columna1, tabla1.columna2,...

tabla2.columna1, tabla2.columna2,... FROM tabla1

JOIN tabla2

ON (tabla1.columa=tabla2.columna)

Un ON JOIN se enlaza entre uno o varios campos especificados entre dos tablas que no tiene porqué denominarse igual.

Todo ON JOIN debe cumplir: Los enlaces entre tablas siempre se tienen que realizar entre campos que tengan campos de tipos compatibles.

Ejemplo

“Conseguir los datos de empleado(id,nombre y apellido) y el nombre de su departamento, ordenar por identificador de empleado”

```
1 • select e.employee_id, e.first_name, e.last_name, d.department_name
2 from employees e JOIN departments d ON(d.department_id=e.department_id)
3 order by employee_id;
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
employee_id	first_name	last_name	department_name
100	Steven	King	Executive
101	Neena	Kochhar	Executive
102	Lex	De Haan	Executive
103	Alexander	Hunold	IT
104	Bruce	Ernst	IT
105	David	Austin	IT
106	Valli	Pataballa	IT
107	Diana	Lorentz	IT
108	Nancy	Greenberg	Finance
109	Daniel	Faviet	Finance
110	John	Chen	Finance
111	Ismael	Sciarra	Finance
112	Jose Manuel	Urman	Finance
113	Luis	Popp	Finance
114	Den	Raphaely	Purchasing
115	Alexander	Khoo	Purchasing

Uso Indebido de ON

En sí en la clausula ON estamos especificando un filtro, tiene exactamente la misma sintaxis que los equals join con la salvedad de que estos lo realizan dentro del WHERE, por tanto, TODOS los filtros especificados en la clausula ON van a funcionar, pero, aunque se ejecute correctamente, incumple de pleno la filosofía de ejecución de las consultas: En la clausula ON se especifica los enlaces entre tablas, su objetivo NO es filtrar datos. Para filtrar datos esta la sección WHERE.

Cross Join

```
SELECT tabla1.columna1, tabl1.columna2,...  
tabla2.columna1, tabla2.columna2,... FROM tabla1  
CROSS JOIN tabla2
```

Es el menos usado, ya que si se obvia tenemos el mismo resultado, en sí simplemente sirve para especificar que se va a realizar un producto cartesiano entre tablas.

Outer Join

Los Outer Join realizan exactamente los mismos enlaces que los Inner Join, y siguen las mismas reglas para su uso, pero permiten seleccionar los elementos que no estuvieran enlazados.

Determinan de qué tabla se van a seleccionar los elementos no enlazados. En estos enlaces la posición de las tablas en los enlaces es básica.

```
SELECT tabla1.columna1, tabl1.columna2,...  
tabla2.columna1, tabla2.columna2,... FROM tabla1  
[CROSS JOIN tabla2]  
[NATURAL JOIN tabla2]  
[[LEFT|RIGHT|FULL OUTER] JOIN tabla2 USING(columna)]  
[[LEFT|RIGHT|FULL OUTER] JOIN tabla2  
ON (tabla1.columa=tabla2.columna)]
```

Left Join

El Left Join selecciona los elementos enlazados al igual que los inner join y además los elementos no enlazados de la tabla situada a la izquierda del JOIN

Ejemplo

“Seleccionar los datos de los empleados (identificador,nombre y apellido) y nombre del departamento de TODOS los empleados”

Con un inner join normal, nos mostraría únicamente los empleados que tuvieran departamento con este enlace podemos seleccionar también los empleados que no tienen departamento (que su clave ajena dada por el campo department_id es nula).

```

1 • select e.employee_id, e.first_name, e.last_name, d.department_name
2   from employees e LEFT JOIN departments d ON(d.department_id=e.department_id)
3   order by employee_id;

```

employee_id	first_name	last_name	department_name
169	Harrison	Bloom	Sales
170	Taylor	Fox	Sales
171	William	Smith	Sales
172	Elizabeth	Bates	Sales
173	Sundita	Kumar	Sales
174	Ellen	Abel	Sales
175	Alyssa	Hutton	Sales
176	Jonathon	Taylor	Sales
177	Jack	Livingston	Sales
178	Kimberely	Grant	NULL
179	Charles	Johnson	Sales
180	Winston	Taylor	Shipping
181	Jean	Fleaur	Shipping
182	Martha	Sullivan	Shipping

Right Join

El Right Join selecciona los elementos enlazados al igual que los inner join y además los elementos no enlazados de la tabla situada a la derecha del JOIN

Ejemplo



“Seleccionar los datos de los empleados (identificador, nombre y apellido) y nombre del departamento de TODOS los departamentos”

Con un inner join normal, nos mostraría únicamente los departamentos que tuvieran empleados asociados con este enlace podemos seleccionar también los departamentos que no tienen ningún empleado, los que no aparecen en el campo department_id de la tabla empleados.

```

1 • select e.employee_id, e.first_name, e.last_name, d.department_name
2    from employees e RIGHT JOIN departments d ON(d.department_id=e.department_id)
3    order by employee_id;

```

Result Grid			
Filter Rows: <input type="text"/>			
Export:  Wrap Cell Content: 			
employee_id	first_name	last_name	department_name
NULL	NULL	NULL	Treasury
NULL	NULL	NULL	Corporate Tax
NULL	NULL	NULL	Control And Credit
NULL	NULL	NULL	Shareholder Services
NULL	NULL	NULL	Benefits
NULL	NULL	NULL	Manufacturing
NULL	NULL	NULL	Construction
NULL	NULL	NULL	Contracting
NULL	NULL	NULL	Operations
NULL	NULL	NULL	IT Support
NULL	NULL	NULL	NOC
NULL	NULL	NULL	IT Helpdesk
NULL	NULL	NULL	Government Sales
NULL	NULL	NULL	Retail Sales
NULL	NULL	NULL	Recruiting
NULL	NULL	NULL	Payroll
100	Steven	King	Executive

Full Join

El Full Join selecciona los elementos enlazados al igual que los inner join y además los elementos no enlazados de la tabla situada a la derecha y a la izquierda del JOIN.

Nota: Full Join no existe en MySQL

Ejemplo

“Seleccionar los datos de los empleados (identificador, nombre y apellido) y nombre del departamento de TODOS los departamentos y de TODOS los empleados”

```

1 select e.employee_id, e.first_name, e.last_name, d.department_name
2 from employees e FULL JOIN departments d ON(d.department_id=e.department_id)
3 order by employee_id;

```

Resultado de la Consulta x

Todas las Filas Recuperadas: 123 en 0,016 segundos

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_NAME
75	174 Ellen	Abel	Sales
76	175 Alyssa	Hutton	Sales
77	176 Jonathon	Taylor	Sales
78	177 Jack	Livingston	Sales
79	178 Kimberely	Grant	(null)
80	179 Charles	Johnson	Sales
81	180 Winston	Taylor	Shipping
82	181 Jean	Fleaur	Shipping
83	182 Martha	Sullivan	Shipping
84	183 Girard	Geoni	Shipping
85	184 Nandita	Sarchand	Shipping

Y además:

107	206 William	Gietz	Accounting
108	(null) (null)	(null)	Contracting
109	(null) (null)	(null)	Construction
110	(null) (null)	(null)	Corporate Tax
111	(null) (null)	(null)	Payroll
112	(null) (null)	(null)	Treasury
113	(null) (null)	(null)	Operations
114	(null) (null)	(null)	Recruiting
115	(null) (null)	(null)	Control And Credit
116	(null) (null)	(null)	Retail Sales
117	(null) (null)	(null)	Shareholder Services
118	(null) (null)	(null)	Benefits
119	(null) (null)	(null)	IT Support
120	(null) (null)	(null)	Government Sales
121	(null) (null)	(null)	Manufacturing
122	(null) (null)	(null)	IT Helpdesk
123	(null) (null)	(null)	NOC

Con un inner join normal, nos mostraría únicamente los departamentos que tuvieran empleados asociados y los empleados que tuvieran departamento, con este enlace podemos seleccionar también los departamentos que no tienen ningún empleado, los que no aparecen en el campo department_id de la tabla empleados y los empleados que no tienen asociado departamento, departamento, que su clave ajena dada por el campo department_id es nula.

Operadores SET

Cada consulta que ejecutamos se le denomina conjunto de datos. Los operadores SET trabajan sobre un par de conjunto de datos, por ejemplo:

Conjunto 1 “ Seleccionar los datos de empleados (identificador, nombre y apellido) y nombre del departamento del departamento 80”

```
SELECT e.employee_id, e.first_name, e.last_name, d.department_name
FROM employees e, departments d
WHERE d.department_id=e.department_id
AND e.department_id=80;
```

Conjunto 2 “Seleccionar los datos de los empleados (identificador, nombre y apellido) y salario que ganan más de 10000”

```
SELECT e.employee_id, e.first_name, e.last_name, e.salary
FROM employees e
WHERE e.salary>10000;
```

Normas que deben cumplir los conjuntos de datos:

- Deben seleccionar el mismo número de columnas
- Los tipos de datos de cada columna deben ser compatibles entre ellos.

Con los conjuntos de datos antes establecidos no podríamos operar ya que tenemos en el primer conjunto, el nombre del departamento que no aparece en la otra y en el segundo, el salario que tampoco aparece en la primera.

Entonces nos quedan dos posibilidades: o seleccionar en los dos conjuntos el dato que falta, o hacer el valor de la columna compatible. Vamos a hacer un ejemplo de cada: en el primer conjunto vamos a seleccionar el salario también y vamos a sacar en vez del nombre del departamento, simplemente el identificador y en la segunda vamos a indicar también el identificador del departamento.

Por tanto, quedarían los conjuntos:

Conjunto 1:

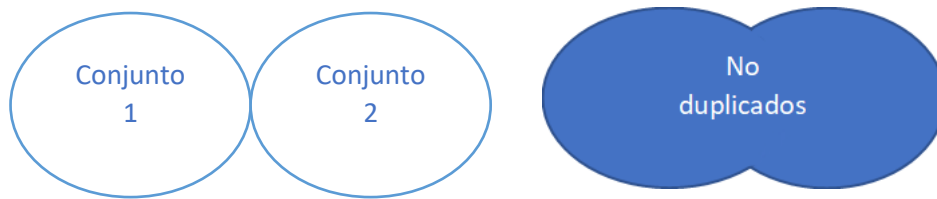
```
SELECT e.employee_id, e.first_name, e.last_name, e.salary, e.department_id
FROM employees e
WHERE e.department_id=80;
```

Conjunto 2

```
SELECT e.employee_id, e.first_name, e.last_name, e.salary, e.department_id
FROM employees e
WHERE e.salary>10000;
```

Con estos cambios, ya podríamos operar con las consultas

UNION



Todas las filas distintas seleccionadas por cualquiera de las dos consultas. Los datos nunca se repiten.

```
1 • SELECT e.employee_id, e.first_name, e.last_name, e.salary, e.department_id
2 FROM employees e
3 WHERE e.department_id=80
4 UNION
5 SELECT e.employee_id, e.first_name, e.last_name, e.salary, e.department_id
6 FROM employees e
7 WHERE e.salary>10000
8 order by employee_id;
```

employee_id	first_name	last_name	salary	department_id
100	Steven	King	24000.00	90
101	Neena	Kochhar	17000.00	90
102	Lex	De Haan	17000.00	90
108	Nancy	Greenberg	12000.00	100
114	Den	Raphaely	11000.00	30
145	John	Russell	14000.00	80
146	Karen	Partners	13500.00	80
147	Alberto	Errazuriz	12000.00	80
148	Gerald	Cambrault	11000.00	80
149	Eleni	Zlotkey	10500.00	80
150	Peter	Tucker	10000.00	80
151	David	Bernstein	9500.00	80
152	Peter	Hall	9000.00	80

Como ejemplo, el empleado 146, no se repite dos veces aunque cumpla las dos select.

Nota: En ORACLE, los resultados se ordenan por defecto por la primera columna seleccionada.

La consulta resultante es como si realizáramos un OR junto con un distinct para eliminar resultados.

UNION ALL



Todas las filas seleccionadas por cualquiera de las dos consultas, incluidos todos los duplicados

Los resultados no se muestran ordenados, a no ser que se indique explícitamente.

```

1 • SELECT e.employee_id, e.first_name, e.last_name, e.salary, e.department_id
2 FROM employees e
3 WHERE e.department_id=80
4 UNION ALL
5 SELECT e.employee_id, e.first_name, e.last_name, e.salary, e.department_id
6 FROM employees e
7 WHERE e.salary>10000
8 order by employee_id;

```

employee_id	first_name	last_name	salary	department_id
100	Steven	King	24000.00	90
101	Neena	Kochhar	17000.00	90
102	Lex	De Haan	17000.00	90
108	Nancy	Greenberg	12000.00	100
114	Den	Raphaely	11000.00	30
145	John	Russell	14000.00	80
145	John	Russell	14000.00	80
146	Karen	Partners	13500.00	80
146	Karen	Partners	13500.00	80
147	Alberto	Errazuriz	12000.00	80
147	Alberto	Errazuriz	12000.00	80
148	Gerald	Cambrault	11000.00	80
148	Gerald	Cambrault	11000.00	80

La consulta resultante es como si realizáramos un OR, como ejemplo, el empleado 146, se repite dos veces porque cumple las dos selects.

INTERSECT



Todas las filas distintas seleccionadas por ambas consultas, por tanto, no hay repetidos.

En MySQL esta cláusula no existe.


```

1 SELECT e.employee_id, e.first_name, e.last_name, e.salary, e.department_id
2 FROM employees e
3 WHERE e.department_id=80
4 INTERSECT
5 SELECT e.employee_id, e.first_name, e.last_name, e.salary, e.department_id
6 FROM employees e
7 WHERE e.salary>10000
8 order by employee_id;

```

Resultado de la Consulta x

Todas las Filas Recuperadas: 8 en 0,005 segundos

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	DEPARTMENT_ID
1	145	John	Russell	14000	80
2	146	Karen	Partners	13500	80
3	147	Alberto	Errazuriz	12000	80
4	148	Gerald	Cambrault	11000	80
5	149	Eleni	Zlotkey	10500	80
6	162	Clara	Vishney	10500	80
7	168	Lisa	Ozer	11500	80
8	174	Ellen	Abel	11000	80

La consulta resultante es como si realizásemos un AND.

MINUS



Todas las filas seleccionadas por la primera sentencia SELECT y no seleccionadas en la segunda sentencia SELECT. Es el único operador SET en el que orden de las consultas se tiene que tener en cuenta.

En MySQL esta cláusula no existe.

```

1 SELECT e.employee_id, e.first_name, e.last_name, e.salary, e.department_id
2 FROM employees e
3 WHERE e.department_id=80
4 MINUS
5 SELECT e.employee_id, e.first_name, e.last_name, e.salary, e.department_id
6 FROM employees e
7 WHERE e.salary>10000
8 order by employee_id;

```

Resultado de la Consulta x

SQL | Todas las Filas Recuperadas: 26 en 0,008 segundos

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	DEPARTMENT_ID
1	150	Peter	Tucker	10000	80
2	151	David	Bernstein	9500	80
3	152	Peter	Hall	9000	80
4	153	Christopher	Olsen	8000	80
5	154	Nanette	Cambrault	7500	80
6	155	Oliver	Tuvault	7000	80
7	156	Janette	King	10000	80
8	157	Patrick	Sully	9500	80
9	158	Allan	McEwen	9000	80
10	159	Lindsey	Smith	8000	80

Nota: Comprobar que faltan exactamente los empleados seleccionados por INTERSEC

Sentencias DML

En las sentencias DML se incluyen las sentencias: INSERT, UPDATE y DELETE y SELECT.

Las sentencias DML modifican los datos, la sentencia SELECT esta incluida en este grupo, pero únicamente hace consulta no modifica ningún dato y devuelve una lista de registros (tabla de resultados).

Las sentencias DML afectan a una única tabla, no se van a poder insertar, modificar o eliminar datos de dos tablas al tiempo, solo de una. Otra característica es que estas sentencias devuelven un entero numérico con el número de registros afectados.

En las sentencias DML contamos con “segunda oportunidad” ya que los cambios no se fijan hasta que no se realiza un COMMIT y mientras que no se haya realizado este podemos realizar ROLLBACK para volver a la situación inicial. La mayoría de las bases de datos tienen una funcionalidad llamada Autocommit, que si esta activada (a TRUE) fija los cambios cada vez que se ejecute una sentencia DML, como si cada vez que se ejecutase la sentencia también se ejecutara un COMMIT. Si la funcionalidad Autocommit esta a FALSE la ejecución del COMMIT es manual y también el ROLLBACK.

Inserción de datos – Insert

Esta sentencia va a insertar datos en la base de datos, esta orientada a registros ya que se van a insertar uno o más registros cuando se ejecute.

INSERT INTO tabla [(listaDeColumnas)]

VALUES (valor1 [,valor2 ...])

La tabla representa la tabla a la que queremos añadir el registro y los valores que siguen a la cláusula VALUES, son los valores que damos a los distintos campos del registro. Si no se especifica la lista de campos, la lista de valores debe seguir el orden de las columnas según fueron creados (para conocer ese orden basta invocar al comando DESCRIBE).

La lista de campos a rellenar se indica si no queremos rellenar todas las columnas.

Las columnas no rellenadas explícitamente con la orden INSERT, se rellenan con su valor por defecto (DEFAULT) o bien con NULL si no se indicó valor por defecto alguno. Si alguna columna tiene restricción de obligatoriedad (NOT NULL), ocurrirá un error si no indicamos un valor para dicha columna.

Si tenemos claves ajenas establecidas el valor de estos campos tiene que existir en la tabla y campo referenciado.

Esta sentencia no tiene filtro, no tiene parte WHERE.

Inserción de varios registros

En la misma sentencia de insert se pueden insertar varios registros, identificados entre paréntesis y separados por comas.

INSERT INTO tabla [(listaDeColumnas)]

VALUES (valor1 [,valor2 ...]), (valor1 [,valor2 ...]), (valor1 [,valor2 ...]);

Mediante una sentencia select

INSERT INTO tabla [(listaDeColumnas)]

SELECT valor1, valor2,... FROM OtraTabla [WHERE...]

La select tiene que seleccionar los campos en el mismo orden y de ser de tipos compatibles con la lista de columnas indicadas en la inserción.

Actualización de Datos - UPDATE

Esta sentencia va a modificar los datos en la base de datos, esta orientada a columna ya que se van a poder modificar datos de varios registros pero no se modifica el registro completo, solo columnas determinadas.

UPDATE tabla

SET columna1=valor1 [,columna2=valor2...]

[WHERE condición];

Se modifican las columnas indicadas en el apartado SET con los valores indicados. La cláusula WHERE permite especificar qué registros serán modificados.

En las sentencias UPDATE se pueden utilizar subconsultas para filtrar los datos a actualizar o conseguir los valores a los que tienen que ser actualizados los campos.

Eliminación de datos - DELETE

Esta sentencia se utiliza para eliminar los registros de una tabla.

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

Esta instrucción, elimina las filas de la tabla que cumplan la condición indicada.

En las sentencias DELETE se pueden utilizar subconsultas para filtrar los datos a eliminar.

Sentencias DDL

Las sentencias DDL se aplican sobre la estructura del sistema de gestión de base de datos (SGBD), o sea sobre los objetos propios de la base de datos: esquemas, base de datos, tablas, vistas, usuarios, conexiones, funciones, procedimientos, índices, etc.

Las sentencias DDL no tienen “segunda oportunidad” cuando se ejecutan los cambios quedan fijados, no como en las sentencias DML que se fijan cuando se realiza un commit o pueden descartarse mediante un rollback.

La ejecución de las sentencias DDL también va muy unida a los permisos que tenga un usuario, porque dependiendo de los mismos va a poder realizar estas acciones, no todo el mundo puede modificar la estructura del SGDB, normalmente son usuarios con permisos de DBA.

Crear elementos – CREATE

Es la sentencia que se ejecuta para crear elementos dentro del SGDB;

Crear esquema

```
CREATE SCHEMA HR;
```

Nos genera un nuevo esquema HR.

Un esquema es la agrupación lógica donde se encuentran las bases de datos.

Crear base de datos

```
CREATE DATABASE HR;
```

Nos genera una nueva base de datos HR;

Nota: En MySQL la relación esquema-base de datos es uno a uno y además da lo mismo crear un esquema o la base de datos, cuando se genera uno el otro automáticamente también se genera.

Crear Tabla

```
CREATE TABLE [esquema.] nombreDeTabla (  
    nombreDeLaColumna1 tipoDeDatos [DEFAULT valor]  
    [restricciones] [, ...]  
);
```

Es la orden SQL que permite crear una tabla. Por defecto será almacenada en el espacio y esquema del usuario que crea la tabla.

Ejemplo común a ambas bases de datos

```
CREATE TABLE Alumno (  
    id INTEGER NOT NULL,  
    nombre VARCHAR(50) NOT NULL,  
    cursoId INT,  
    PRIMARY KEY(id),  
    CONSTRAINT FK_AlumnoCurso FOREIGN KEY (cursoId) REFERENCES Curso(id)  
);
```

Ejemplo MySQL

```
CREATE TABLE Alumnos (  
    id int NOT NULL,  
    nombre varchar(50) NOT NULL,  
    cursoId int,  
    PRIMARY KEY (id),  
    FOREIGN KEY (cursoId) REFERENCES Curso(id)  
);
```

En este ejemplo los sistemas nombraran ellos automáticamente a la Foreign Key.

Ejemplo Oracle

```
CREATE TABLE Alumnos (  
    id int NOT NULL PRIMARY KEY,  
    nombre int NOT NULL,
```

```
cursoId int FOREIGN KEY REFERENCES Curso(id)
);
```

En este ejemplo los sistemas nombraran ellos automáticamente a la Foreign Key.

Mediante una sentencia select

```
SELECT column1, column2, column3, ...
```

```
INTO newtable [IN externaldb]
```

```
FROM oldtable
```

```
WHERE condition;
```

La nueva tabla se creará con los nombres de columna y los tipos definidos en la tabla anterior. Puede crear nuevos nombres de columna utilizando la cláusula AS.

Clausula DEFAULT

Default se utiliza para dar un valor por defecto a la columna, independientemente de que puede ser nula o no, aunque normalmente se indica para las columnas no nulas.

Clausula AUTOINCREMENT

Válida únicamente en MySQL, sirve para que el campo, normalmente la primary key de la tabla aumente su valor automáticamente de 1 en 1. La secuencia siempre empieza con el valor 1.

Restricciones

Una restricción es una condición de obligado cumplimiento para una o más columnas de la tabla.

Existen estos tipos de restricciones:

- **Not Null.** El campo debe tomar algún valor.
- **Unique.** El campo debe tener un valor único respecto a otros valores del mismo campo de la tabla. Crear esta restricción conlleva crear un índice en este campo.
- **Primary Key.** El campo toma las dos restricciones anteriores, debe ser Not Null y Unique, por tanto tiene asociado un índice automáticamente, es el campo que referencia al resto del registro.
- **Foreign Key.** El campo identifica la relación con un campo Primary Key de otra tabla, identifica una relación M-1 o 1-1 con otra tabla. No tiene porqué tener asociado un índice en Oracle, pero en MySQL sí.
- **Check.** El campo tiene que cumplir una condición.

Nota: El nombre de las restricciones (constraints) debe ser único en el sistema.
--

No puede repetirse el nombre, en el momento de crear una restricción si existe el nombre no dejará crearla.

Las restricciones se pueden realizar cuando estamos creando (CREATE) o modificando (ALTER) una tabla. En realidad, hay dos maneras de poner restricciones:

- Poner una restricción de columna. En ese caso la restricción se pone seguido a la definición de la columna. Sintaxis:

```
columna tipo [DEFAULT expresión] [CONSTRAINT nombre] tipo,
```

```
CREATE TABLE Alumnos (
    id int NOT NULL PRIMARY KEY,
    nombre varchar(50) NOT NULL,
    apellidos varchar(50) UNIQUE,
    edad int
);
```

- Poner una restricción de tabla. En ese caso se ponen al final de la lista de columnas. La única restricción que no se puede definir de esta forma es la de tipo NOT NULL. El resto se harían siguiendo esta sintaxis:

```
columna1 definición1,
...,
últimaColumna últimaDefinición,
[CONSTRAINT nombre] tipo(listaColumnas)
[,...otras restricciones...]
```

```
CREATE TABLE Alumnos (
    id int NOT NULL,
    nombre varchar(255) NOT NULL,
    apellidos varchar(255),
    edad int,
    provincia varchar(255),
    CONSTRAINT PK_Alumno PRIMARY KEY (id),
    CONSTRAINT CHK_Alumno CHECK (edad>=18 AND provincia='Madrid'),
    CONSTRAINT UC_Alumno UNIQUE (nombre,apellidos)
);
```

La diferencia está en que en el primer caso no se especifica la lista de columnas al definir la restricción: lógico porque se entiende perfectamente que las restricciones de columna se aplicarán a la columna en la que se definen.

Crear Vista

```
CREATE VIEW [esquema.] nombreDeVista (
    SELECT columna1, columna2, ...
FROM table_name
```

[WHERE logical expression(s)]

[GROUP BY column1, column2, ...]

[ORDER BY column1, column2, ... ASC|DESC];

Una Vista es un objeto de la base de datos, que también se le denomina consulta almacenada, ya que se guarda la información de resultado de la consulta como si fuera una tabla.

Una vista puede ser ejecutada por el usuario manualmente, siempre que tenga permisos.

Nota: Una vista se va a ejecutar automáticamente en el momento en el que alguna de las tablas que la componen sufra alguna modificación (insert, update o delete) en sus campos.

Modificar elementos - tabla

La modificación de elementos se realiza con ALTER.

Las sentencias más utilizadas de alteración son las de tabla:

- Para modificar su nombre
- Para añadir columnas
- Para modificar columnas
- Para eliminar columnas
- Para crear restricciones
- Para eliminar restricciones

Modificar nombre de tabla

Añadir columnas

```
ALTER TABLE Alumnos
```

```
ADD claseId INT;
```

Simplemente hay que indicar el nombre de la columna y el tipo, si se quiere también se puede especificar NOT NULL, UNIQUE o DEFAULT.

Modificar columnas

La sentencia para modificar una columna difiere según la base de datos.

Vamos a modificar la columna nombre para aumentar su tamaño y darle un valor por defecto.

En MySQL:

```
ALTER TABLE Alumnos
```

```
CHANGE COLUMN nombre nombre VARCHAR(100) NOT NULL DEFAULT 'Sin nombre';
```

En Oracle:

```
ALTER TABLE Alumnos
```

```
MODIFY nombre VARCHAR(100) NOT NULL DEFAULT 'Sin nombre';
```


Eliminar columnas

```
ALTER TABLE Alumno
```

```
DROP COLUMN claseId;
```

Hay que indicar únicamente el nombre de la columna. Si la columna tiene asociada una Foreign Key no nos dejará eliminarla.

Crear restricción

Para crear una restricción debemos alterar la tabla

```
ALTER TABLE Alumnos
```

```
ADD CONSTRAINT FK_AlumnosClase
```

```
FOREIGN KEY (ClaseId) REFERENCES Clase(id);
```

Eliminar restricciones

En MySQL:

```
ALTER TABLE Alumnos
```

```
DROP FOREIGN KEY FK_AlumnosClase;
```

En Oracle:

```
ALTER TABLE Alumnos
```

```
DROP CONSTRAINT FK_AlumnosClase;
```

Eliminar elementos

La sentencia para eliminar elementos es DROP.

Borrar tabla

```
DROP TABLE personas;
```

La orden DROP TABLE seguida del nombre de una tabla, permite eliminar la tabla en cuestión.

- Al borrar una tabla:
- Desaparecen todos los datos
- Cualquier vista y sinónimo referente a la tabla seguirá existiendo, pero ya no funcionará (conviene eliminarlos)
- Las transacciones pendientes son aceptadas (COMMIT), en aquellas bases de datos que tengan la posibilidad de utilizar transacciones.

Consultas agrupadas

Las consultas agrupadas se utilizan para conseguir datos como conteos, medias, sumas, valores mínimo o máximos de conjunto de datos, estos conjuntos de datos pueden ser el conjunto completo de los datos de la tabla o agrupados por uno o varios campos.

La sintaxis de estas consultas es la siguiente:

```
SELECT column1, column2, ...  
FROM table_name  
[WHERE logical expression(s)]  
[GROUP BY column1, column2, ...]  
[HAVING logical expression(s)]  
[ORDER BY column1, column2, ... ASC|DESC];
```

GROUP BY nos indica los campos por los que se va a realizar la agrupación.

La filas que tengan el mismo valor en ese campo quedarían agrupadas.

La agrupación se realiza cuando el bloque SELECT ya se ha ejecutado, por tanto vamos a poder tener acceso a partir de este punto a los alias de columna.

Las columnas agrupadas no tienen por qué estar en la lista de SELECT, pero tampoco pueden aparecer columnas que no estén agrupadas que difieran en valor para cada grupo.

Las funciones de agrupación actuarán siempre sobre los datos agrupados.

HAVING nos indica las condiciones que tiene que cumplir la agrupación.

La cláusula WHERE indica las condiciones que tiene que cumplir la fila y la cláusula HAVING la que tiene que cumplir la agrupación.

En HAVING se deben utilizar los alias de columna porque si queremos filtrar, por ejemplo por la media de salario o volvemos a invocar a la función de media o utilizamos su alias para hacer referencia a la misma, es más óptimo no volverla a invocar y simplemente hacer referencia con su alias.

Funciones de agrupación

El uso de las funciones de agrupación es el verdadero objetivo de la agrupación de consultas

Las funciones de agrupación más importantes son las siguientes:

Count – Contador

Cuenta el número de registros de cada agrupación, no comprueba el valor, únicamente la existencia del registro, si el campo es nulo, no lo cuenta.

Para contar el número de registros podemos ver las siguientes sentencias:

- Count(<campo>) Nos cuenta el número de campos no nulos. Si el campo es la pk este conteo va a ser equivalente a los siguientes.

- Count(*) Cuenta el número de registros completos, muy utilizado, pero no es óptimo ya que coloca mucha información en memoria.
- Count(1) Cuenta el número de unos (el número de registros), ya que se seleccionan tantos unos como registros existan y es el más óptimo ya que solo pone en memoria una lista de unos.

Sum – Suma

Suma los valores de los registros de cada agrupación sin tener en cuenta los campos a NULL.

Avg – Media

Hace la media de los registros que tienen valor, sin tener en cuenta los NULL. En este caso hay que tener cuidado ya que no es lo mismo: “Hacer la media de las comisiones de todos los empleados” a “Hacer la media de los empleados que tengan comisión”.

El primer enunciado (“Hacer la media de las comisiones de todos los empleados”) debemos contar con el número total de los empleados, para que la media nos salga correcta tenemos que pasar los valores NULL a 0: `avg(nvl(commission_pct,0))`

Con el segundo enunciado (“Hacer la media de los empleados que tengan comisión”) podemos utilizar la media directa sin hacer ningún tratamiento del dato: `avg(commission_pct)`

Max – Valor máximo

Nos devuelve el valor máximo de cada agrupación.

Min – Valor mínimo

Nos devuelve el valor mínimo de cada agrupación.

Uso de las funciones de agrupación

Con el conjunto completo de la tabla

En este caso no hace falta agrupar, ya que los registros de la tabla son el conjunto en sí mismo agrupado. No se utilizan ni la cláusula GROUP BY ni HAVING

Ejemplo

“Mostrar el número de empleados, la media de salario, el máximo salario, el mínimo salario, el número de empleados que tienen comisión, la suma de las comisiones, la media de comisión de estos empleados y la media de comisión para todos los empleados”

```

1 • select count(1) nEmpleado, avg(e.salary) mediaSalario, max(e.salary) maxSalario,
2       min(e.salary) minSalary, count(e.commission_pct) nComision, sum(e.commission_pct) sumaComision,
3       avg(e.commission_pct) mediaComision, avg(ifnull(e.commission_pct,0)) mediaComisionEmpleados
4 from employees e;

```

Result Grid Filter Rows: Export: Wrap Cell Content:							
nEmpleado	mediaSalario	maxSalario	minSalary	nComision	sumaComision	mediaComision	mediaComisionEmpleados
107	6461.682243	24000.00	2100.00	35	7.80	0.222857	0.072897

Con agrupación por campos

En este caso si existe agrupación por uno o varios campos, se utilizan la cláusula GROUP BY y la cláusula HAVING si se quiere filtrar por datos de la agrupación.

“Mostrar el número de empleados, la media de salario, el máximo salario, el mínimo salario de cada departamento”

```

1 • select e.department_id, count(1) nEmpleado, avg(e.salary) mediaSalario,
2       max(e.salary) maxSalario, min(e.salary) minSalary
3 from employees e
4 group by department_id;

```

Result Grid Filter Rows: Export: Wrap Cell Content:					
department_id	nEmpleado	mediaSalario	maxSalario	minSalary	
NULL	1	7000.000000	7000.00	7000.00	
10	1	4400.000000	4400.00	4400.00	
20	2	9500.000000	13000.00	6000.00	
30	6	4150.000000	11000.00	2500.00	
40	1	6500.000000	6500.00	6500.00	
50	45	3475.555556	8200.00	2100.00	
60	5	5760.000000	9000.00	4200.00	
70	1	10000.000000	10000.00	10000.00	
80	34	8955.882353	14000.00	6100.00	
90	3	19333.333333	24000.00	17000.00	
100	6	8600.000000	12000.00	6900.00	
110	2	10150.000000	12000.00	8300.00	

“Mostrar el número de empleados, la media de salario, el máximo salario, el mínimo salario de cada departamento y por cada trabajo que contenga”

```

1 • select e.department_id,e.job_id,count(1) nEmpleado, avg(e.salary) mediaSalario,
2       max(e.salary) maxSalario, min(e.salary) minSalary
3 from employees e
4 group by department_id,job_id
5 order by department_id,job_id;

```

Result Grid Filter Rows: <input type="text"/> Export: Wrap Cell Content:					
department_id	job_id	nEmpleado	mediaSalario	maxSalario	minSalary
NULL	SA_REP	1	7000.000000	7000.00	7000.00
10	AD_ASST	1	4400.000000	4400.00	4400.00
20	MK_MAN	1	13000.000000	13000.00	13000.00
20	MK_REP	1	6000.000000	6000.00	6000.00
30	PU_CLERK	5	2780.000000	3100.00	2500.00
30	PU_MAN	1	11000.000000	11000.00	11000.00
40	HR_REP	1	6500.000000	6500.00	6500.00
50	SH_CLERK	20	3215.000000	4200.00	2500.00
50	ST_CLERK	20	2785.000000	3600.00	2100.00
50	ST_MAN	5	7280.000000	8200.00	5800.00
60	IT_PROG	5	5760.000000	9000.00	4200.00
70	PR_REP	1	10000.000000	10000.00	10000.00
80	SA_MAN	5	12200.000000	14000.00	10500.00
80	SA_REP	29	8396.551724	11500.00	6100.00
90	AD_PRES	1	24000.000000	24000.00	24000.00

Subconsultas

El uso de subconsultas es una técnica que permite utilizar el resultado de una tabla SELECT en otra consulta SELECT. Permite solucionar consultas complejas mediante el uso de resultados previos conseguidos a través de otra consulta.

El SELECT que se coloca en el interior de otro SELECT se conoce con el término de SUBSELECT. Ese SUBSELECT se puede colocar dentro de las cláusulas SELECT, FROM o JOIN , WHERE y HAVING.

Las subconsultas nunca se colocan en las cláusulas GROUP BY ni ORDER BY.

En las consultas vistas hasta ahora cogemos la información de Tablas, elementos físicos permanentes que están modelados según ciertos criterios.

¿Pero y si se necesitará que los datos existentes estuvieran estructurados de diferente forma para consultarlos? Para esto se utilizan las subqueries o subconsultas:

- Se realiza una subconsulta para que datos que no tenemos estructurados juntos, lo estén.
- necesitamos un dato para una consulta que no tenemos directamente.

Las subconsultas en las Cláusulas FROM o JOIN suelen del primer caso y las subconsultas en las cláusulas WHERE y HAVING del segundo tipo.

La tipificación de las subconsultas va a depender de dónde se encuentren localizadas dentro de la query:

- **Subconsultas de selección:** Si se encuentran en el SELECT, este tipo de subconsultas son las menos comunes, y sirven para seleccionar un dato que no se encuentre en la consulta principal. Se ejecutan tantas veces como registros seleccionados existan.
- **Subconsultas de tabla:** Se encuentran en el FROM, identifican tablas de datos, la subconsulta normalmente selecciona múltiples datos, que son tratados como una tabla normal.
- **Subconsultas de filtro:** Se encuentran en el WHERE o en el HAVING, suelen ser subconsultas escalares (devuelven un único valor: un único registro con un único campo), aunque también pueden devolver varios datos. Se utilizan para filtrar la información de la consulta principal.

Subconsultas escalares: Devuelven un único registro con un único campo, por tanto, devuelven un único valor. Utilizadas masivamente en las subconsultas de filtro.

Subconsultas de selección

Se utilizan para seleccionar un único dato, en cada uno de los registros.

Es el tipo de subconsulta menos utilizada ya que siempre se puede sustituir por subconsultas de tabla, que son más óptimas en su ejecución.

Ejemplo

“Seleccionar nombre, salario y número de empleados del departamento de cada empleado”

```
1 • select e.first_name, e.salary, (select count(1) from employees e1
2                                where e1.department_id=e.department_id) numEmpleados
3 from employees e;
```

Result Grid			
Filter Rows: <input type="text"/>			
Export: Wrap Cell Content:			
	first_name	salary	numEmpleados
▶	Steven	24000.00	3
	Neena	17000.00	3
	Lex	17000.00	3
	Alexander	9000.00	5
	Bruce	6000.00	5
	David	4800.00	5
	Valli	4800.00	5
	Diana	4200.00	5
	Nancy	12000.00	6
	Daniel	9000.00	6

Subconsultas de tabla

Las subconsultas de tabla son las más utilizadas, las más óptimas en su ejecución y son más versátiles ya que en cualquier momento podemos conseguir más datos de la subquerys.

Las subquerys de tabla se utilizan en la sección FROM de la select principal y se pueden tratar como cualquier tabla normal, incluso estableciendo enlaces.

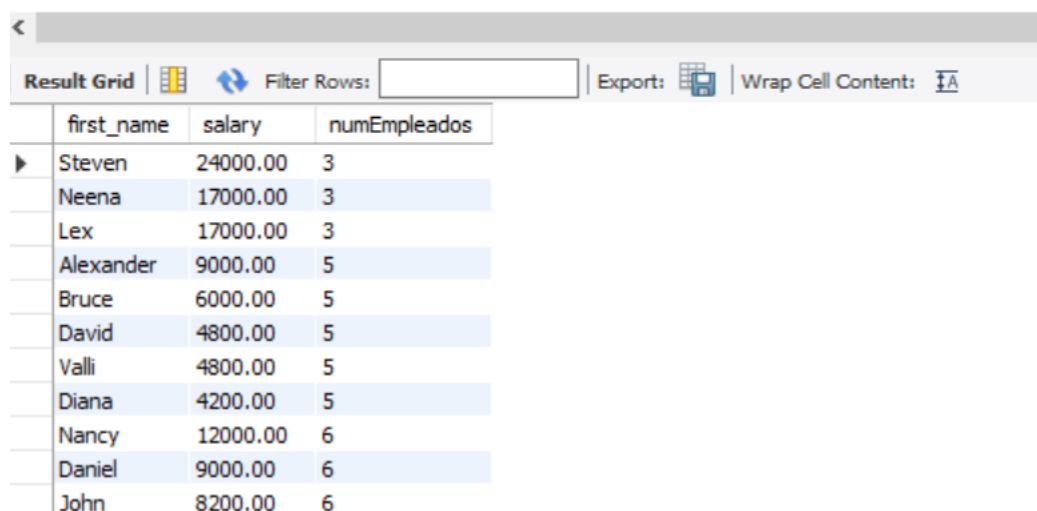
La gran mayoría de las subquerys de selección y de filtro se pueden pasar a una subquery de tabla.

Ejemplos

La subquery anterior se puede convertir fácilmente en una subquery de tabla:

“Seleccionar nombre, salario y número de empleados del departamento de cada empleado”

```
1 • select e.first_name, e.salary, t.numEmpleados
2   from employees e, (select department_id, count(1) numEmpleados
3                      from employees e1
4                      group by e1.department_id) t
5  where t.department_id=e.department_id
6  order by employee_id;
```



The screenshot shows a database interface with a SQL query editor at the top and a 'Result Grid' below it. The query is a SELECT statement that joins the 'employees' table with a subquery that counts the number of employees per department. The results are displayed in a table with columns 'first_name', 'salary', and 'numEmpleados'.

	first_name	salary	numEmpleados
▶	Steven	24000.00	3
	Neena	17000.00	3
	Lex	17000.00	3
	Alexander	9000.00	5
	Bruce	6000.00	5
	David	4800.00	5
	Valli	4800.00	5
	Diana	4200.00	5
	Nancy	12000.00	6
	Daniel	9000.00	6
	John	8200.00	6

Subconsultas de filtro

Subconsultas simples

Las subconsultas simples son aquellas que devuelven una única fila. Si además devuelven una única columna, se las llama subconsultas escalares, ya que devuelven un único valor. Utilizadas en las subconsultas de selección y en las subconsultas de filtro

La sintaxis es:

SELECT listaExpresiones

FROM tabla

WHERE expresión OPERADOR

(SELECT listaExpresiones

FROM tabla);

El operador puede ser >, <, >=, <=, !=, = o IN.

Estas susconsultas son propias de cláusulas WHERE o HAVING

Subconsultas de múltiples filas

En el apartado anterior se comentaba que las subconsultas pueden devolver sólo una única fila. Estas subconsultas son las utilizadas en las subconsultas de tabla y también en las de filtro.

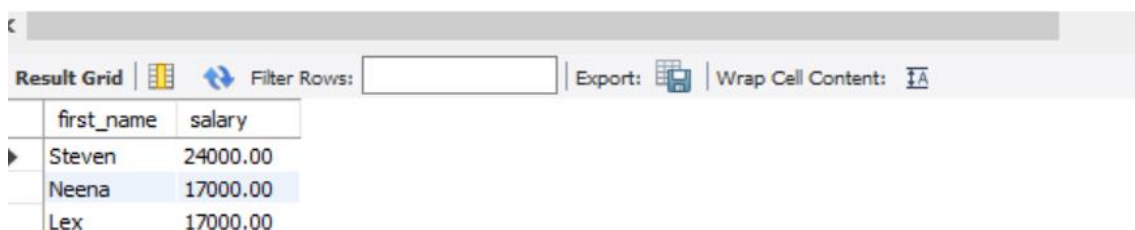
Operadores de múltiples filas:

- ANY, SOME. Devuelve TRUE (selecciona el registro) si CUALQUIERA de los valores de la subconsultante cumple la condición
- ALL. Devuelve TRUE si TODOS los valores de la subconsultante cumplen la condición
- IN. Devuelve TRUE si el valor está contenido en la lista de valores de IN
- EXISTS. Este operador devuelve TRUE si el valor del campo existe en los datos seleccionados de la subconsulta, si no, no lo selecciona.

Todos estos operadores devuelven un valor booleano como resultado, quiere decir que seleccionan o no seleccionan el registro de la tabla de la que depende la subquery.

Un ejemplo sería como subconsulta de filtro sería : “Mostrar el sueldo y nombre de los empleados cuyo sueldo supera al de todos los empleados del departamento 80”

```
1 • select e.first_name, e.salary
2   from employees e
3  where e.salary >ALL (select salary from employees where department_id=80);
```



	first_name	salary
▶	Steven	24000.00
	Neena	17000.00
	Lex	17000.00

La subconsulta necesaria para ese resultado mostraría todos los sueldos del departamento de ventas. Pero no podremos utilizar un operador de comparación directamente ya que esa subconsulta devuelve más de una fila. La solución a esto es utilizar instrucciones especiales entre el operador y la consulta, que permiten el uso de subconsultas de varias filas.

Ejemplos

SELECT nombre, sueldo

FROM empleados

WHERE sueldo >= ALL (SELECT sueldo FROM empleados);

La consulta anterior obtiene el empleado que más cobra. Otro ejemplo:

SELECT nombre FROM empleados


```
WHERE dni IN (SELECT dni FROM directivos);
```

En ese caso se obtienen los nombres de los empleados cuyos dni están en la tabla de directivos.

Si se necesita comparar dos columnas en una consulta IN, se hace de esta forma:

```
SELECT nombre FROM empleados
```

```
WHERE (cod1,cod2) IN (SELECT cod1,cod2 FROM directivos);
```

Esta última sentencia selecciona únicamente las piezas que tienen existencias:

```
SELECT tipo, modelo, precio_venta
```

```
FROM piezas p
```

```
WHERE EXISTS (
```

```
    SELECT tipo, modelo FROM existencias
```

```
    WHERE tipo=p.tipo AND modelo=p.modelo);
```

Esta consulta devuelve las piezas que se encuentran en la tabla de existencias.