

1 FN

Una tabla está en Primera Forma Normal si:

Todos los atributos son «atómicos». Por ejemplo, en el campo teléfono no tenemos varios teléfonos.

La tabla contiene una **clave primaria única**. Por ejemplo el DNI para personas, la patente para vehículos o un simple id autoincremental. Si no tiene clave, no es 1FN.

La clave primaria no contiene atributos nulos. No podemos tener filas para las que no haya clave (por ejemplo, personas sin DNI o vehículos sin patente).

No debe existir variación en el número de columnas. Si algunas filas tienen 8 columnas y otras 3, pues no estamos en 1FN.

Los campos no clave deben identificarse por la clave. Es decir, que los campos no clave dependen funcionalmente de la clave. Esto es prácticamente lo mismo que decir que existe clave primaria.

Debe Existir una independencia del orden tanto de las filas como de las columnas, es decir, si los datos cambian de orden no deben cambiar sus significados. Por ejemplo, si en la columna 1 tenemos el primer apellido y en la columna 2 tenemos el segundo, pues no estamos en 1FN. Igualmente si en la tercera fila tenemos el tercer mejor expediente y en la quinta fila el quinto, no estamos en 1FN.

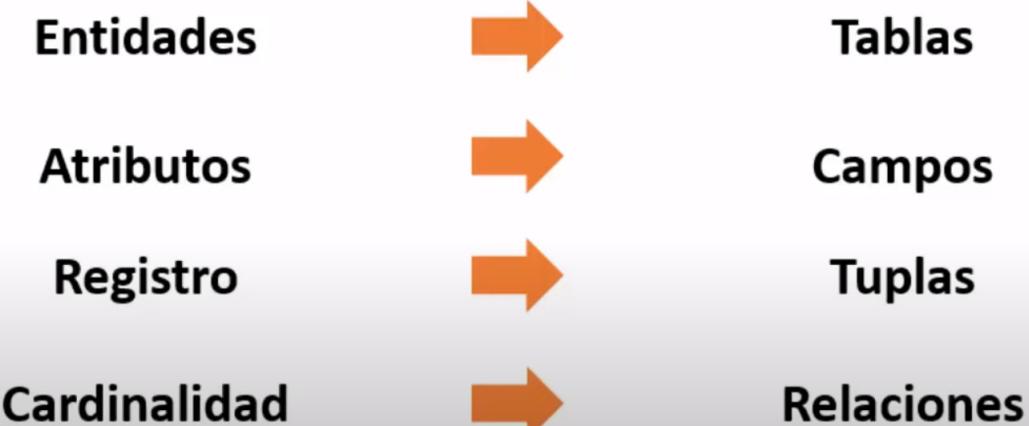
2 FN

Una tabla está en 2FN si además de estar en 1FN cumple que **los atributos no clave depende de TODA la clave principal**.

Por ejemplo, si tenemos una tabla con Personas, identificadas por su DNI y recogemos su empresa y dirección de trabajo, la clave sería CUIT-Empresa. Pero nos encontraremos con que una misma persona puede trabajar en varias empresas. Y vemos que la dirección de trabajo no depende de TODA la clave primaria, sino solo de la empresa. Por lo tanto, no estamos en 2FN

Una tabla está en 3FN si además de estar en 2FN **no existe ninguna dependencia transitiva entre los atributos que no son clave.**

Modelo Relacional



Creación de base de datos

Código SQL ANSI para crear una base de datos en MySQL

Otra manera de crear una **Base de Datos en Mysql** es a través de **código sql**. Para crear una nueva base de datos en MySQL, se usa la instrucción **CREATE DATABASE**, con la siguiente sintaxis:

```
MySQL
CREATE DATABASE nombreBasedeDatos;
```

Sintaxis para crear tablas en Mysql

La sintaxis de como crear una tabla en MySQL es la siguiente:

```
CREATE TABLE Nombre_Tabla
(
    Nombre_Columna1 Tipo_de_Dato (longitud),
    Nombre_Columna2 Tipo_de_Dato (longitud),
    Nombre_Columna3 Tipo_de_Dato (longitud),
    ...
);
```

Los parámetros “**Nombre_Columna**” especifican los nombres de las columnas que integran las tablas.
Los parámetros “**Tipo_de_Dato**” especifican que tipo de datos admitirá esa columna (ej. `varchar`, `integer`, `decimal`, `date`, etc.).

El parámetro “**Longitud**” especifica la longitud máxima de caracteres que admitirá la columna de la tabla.
Cabe aclarar que hay varias propiedades que pueden ser aplicadas a las columnas de la tabla, por ejemplo una de las más comunes es `primary key`, la cual nos permite indicar que columna será llave primaria, también podemos establecer que alguna columna no acepte valores nulos, a través de la propiedad `not null`.

Código para crear tablas en MySql

A continuación presentare un ejemplo de cómo crear una tabla llamada “**Alumnos**” con cinco columnas, el código es el siguiente:

```
MySQL
CREATE DATABASE Tutorial;

use Tutorial;

CREATE TABLE Alumnos
(
    IdAlumno int primary key not null,
    Nombre varchar (25),
    Apellido Varchar (25),
    Edad int,
    Direccion_Residencia varchar (50)
);
```

Inclusión de restricciones

- Las restricciones aplican reglas a nivel de tabla.
- Las restricciones impiden la supresión de una tabla si hay dependencias.
- Los siguientes tipos de restricciones son válidos:
 - NOT NULL
 - UNIQUE
 - PRIMARY KEY
 - FOREIGN KEY
 - CHECK



Inclusión de restricciones

Restricción	Descripción
NOT NULL	Especifica que la columna no puede contener un valor nulo.
UNIQUE	Especifica una columna o combinación de columnas cuyos valores deben ser únicos para todas las filas de la tabla.
PRIMARY KEY	Identifica de forma única cada fila de la tabla.
FOREIGN KEY	Establece y aplica la integridad referencial entre la columna y la columna de la tabla a la que se hace referencia; por ejemplo, busca los valores de una tabla que coinciden con los valores de la otra tabla.
CHECK	Especifica una condición que debe ser verdadera.

Definición de restricciones

- ✓ Ejemplo de una restricción a nivel de columna:

```
CREATE TABLE employees (
    employee_id  NUMBER(6)
        CONSTRAINT emp_emp_id_pk PRIMARY KEY,
    first_name    VARCHAR2(20),
    ...);
```

1

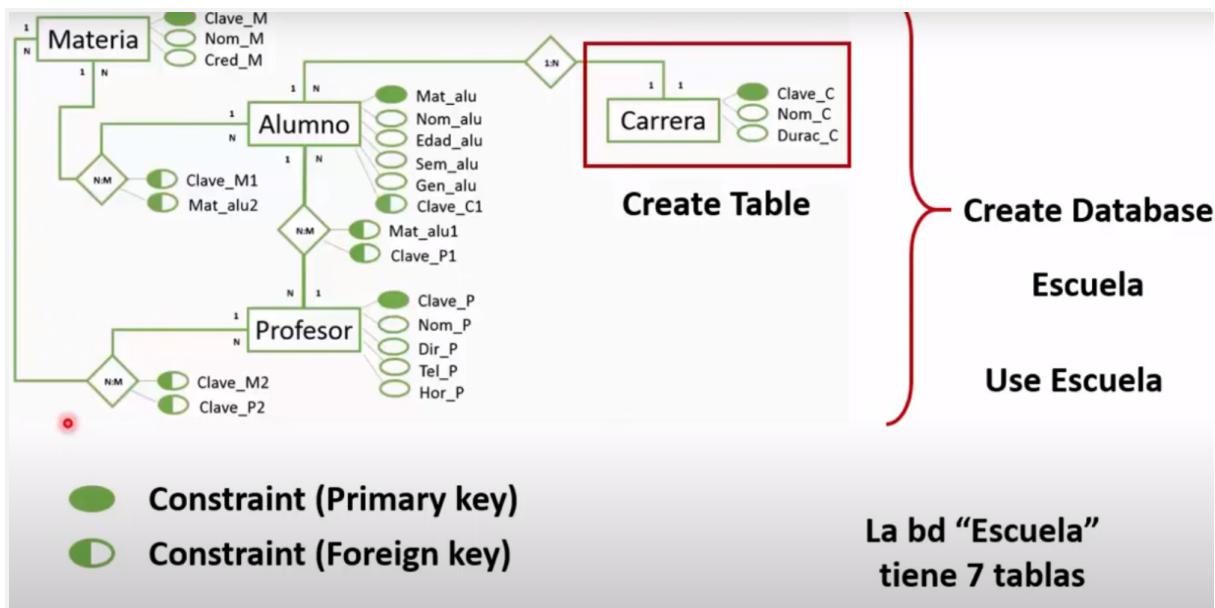
- ✓ Ejemplo de una restricción a nivel de tabla:

```
CREATE TABLE employees (
    employee_id  NUMBER(6),
    first_name    VARCHAR2(20),
    ...
    job_id        VARCHAR2(10) NOT NULL,
    CONSTRAINT emp_emp_id_pk
        PRIMARY KEY (EMPLOYEE_ID));
```

2

Restricción FOREIGN KEY: Palabras Clave

- FOREIGN KEY: define la columna en la tabla secundaria a nivel de restricción de tabla
- REFERENCES: identifica la tabla y la columna en la tabla principal
- ON DELETE CASCADE: suprime las filas dependientes de la tabla secundaria cuando se suprime una fila de la tabla principal
- ON DELETE SET NULL: convierte los valores de clave ajena dependiente en nulos.



DML Insert y select



Insertar, actualizar y borrar datos en MySQL



NombreTabla1	
Columna1	INT
Columna2	VARCHAR(45)
Columna3	VARCHAR(45)
Columna4	DATE
Indexes	

NombreTabla2	
Columna1	INT
Columna2	VARCHAR(45)
Columna3	VARCHAR(45)
Columna4	DATE
Indexes	

Para agregar datos a las tablas se utiliza la instrucción **Insert into**, este comando es uno de los más usados en los diferentes gestores de Base de Datos. Para insertar los registros se puede hacer de uno en uno, o agregar varios registros a través de una misma instrucción **insert into**.

Sintaxis de Insert into en MySQL

La sintaxis para usar la instrucción `insert into` en una tabla de MySQL es la siguiente:

```
INSERT INTO "NombreTabla" ("PrimeraColumna", "SegundaColumna", etc)
VALUES ("Dato1", "Dato2", etc);
```

Sintaxis de las Sentencias **INSERT**

NombreTabla: Es el nombre de la tabla en la que insertaremos registros.

PrimeraColumna, SegundaColumna,..: Son las columnas de la tabla en la que vamos a insertar registros.

"Dato1", "Dato2" ,..: Son los valores que vamos a guardar en cada columna especificada.

Es importante mencionar que la sintaxis anterior se puede reducir en los casos que vamos agregar datos a todas las columnas, ya que podríamos plantearlo de la forma siguiente:

```
INSERT INTO "NombreTabla" VALUES ("Dato1", "Dato2", etc);
```

Cuando aplicamos esta sintaxis, debemos respetar la estructura de la tabla y además después del comando `values` enviar todos los datos para cada columna en el orden correcto, ya que dicha sintaxis indica que vamos a insertar registros a todas las columnas, por lo tanto debemos enviar los datos exactamente como los hemos especificado al momento de crearla.

Existe otra opción de insertar registros y es mencionando columnas específicas, para este caso debemos tomar en cuenta que las columnas que omitimos puedan quedar nulas, es decir que le hayamos agregado la propiedad `not null`.

Insertar datos en una tabla Mysql

Ya habiendo conocido la sintaxis ahora nos interesa poder agregar los registros, para ello lo haremos con la tabla Alumnos, vamos a tomar en cuenta que tiene los siguientes campos:

IdAlumno

Nombres

Apellidos

Edad

Direccion_Residencia

El código necesario para insertar un único registro en la tabla Alumnos es el siguiente:

```
MySQL
INSERT INTO Alumnos (IdAlumno, Nombres, Apellidos, Edad, Direccion_Residencia) VALUES
('0101', 'Franklin1', 'Garcia', '25', 'avenida 01');
```

Código Ansi SQL para insertar registros en Mysql

Si queremos agregar varios registros a través de un mismo `insert`, basta con agregar una coma en los valores que le enviamos en `values`, y especificar los datos a insertar.

```
MySQL
INSERT INTO Alumnos (IdAlumno, Nombres, Apellidos, Edad, Direccion_Residencia)
VALUES
('0102', 'Franklin1', 'Garcia', '25', 'avenida 01'),
('0103', 'Franklin2', 'Garcia', '25', 'avenida 02'),
('0104', 'Franklin3', 'Garcia', '25', 'avenida 03');
```

Actualizar datos en una tabla en Mysql

Sintaxis de Sentencias UPDATE

- ✓ Modificar los valores existentes en una tabla con la sentencia **UPDATE**
- ✓ Actualizar más de una fila cada vez (si es necesario).

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

En la sintaxis:

table es el nombre de la tabla.

column es el nombre de la columna de la tabla que se debe llenar.

value es el valor o *subconsulta* correspondiente para la columna.

condition identifica las filas que se deben actualizar y se compone de nombres de columna, expresiones, constantes, *subconsultas* y operadores de comparación.

Para confirmar la operación de actualización, consulte la tabla para visualizar las filas actualizadas.

Actualización de filas en una tabla

- ✓ Si se especifica la cláusula **WHERE**, se modifican los valores de una fila o varias filas específicas:

```
UPDATE employees|
SET department_id = 50
WHERE employee id = 113;
1 rows updated
```

- ✓ Si se omite la cláusula **WHERE**, se modifican los valores de todas las filas de la tabla:

```
UPDATE copy_emp
SET department_id = 110;
22 rows updated
```

- ✓ Especificar **SET column_name= NULL** para actualizar un valor de columna a **NULL**.

Borrado de filas en una tabla en Mysql

Sentencia DELETE

Puede eliminar filas existentes de una tabla mediante la sentencia `DELETE`

```
DELETE [FROM] table  
[WHERE condition];
```

En la sintaxis:

`table` es el nombre de la tabla.

`condition` identifica las filas que se deben suprimir y se compone de nombres de columna, expresiones, constantes, subconsultas y operadores de comparación.

Nota: si no se suprime ninguna fila, se devuelve el mensaje "0 rows deleted" (en el separador Script Output de Workbench).

Sentencia DELETE

Supresión de filas de tablas

- ✓ Se suprimen filas concretas si se especifica la cláusula

```
DELETE FROM departments  
WHERE department_name = 'Finance';  
1 rows deleted
```

- ✓ Se suprimen todas las filas de la tabla si omite la cláusula `WHERE`

```
DELETE FROM copy_emp;  
22 rows deleted
```

Sentencia TRUNCATE

- ✓ Elimina todas las filas de una tabla, dejando la tabla vacía y la estructura de la misma intacta.
- ✓ Es una sentencia de lenguaje de definición de datos (DDL) en lugar de una sentencia DML; no se puede deshacer fácilmente.

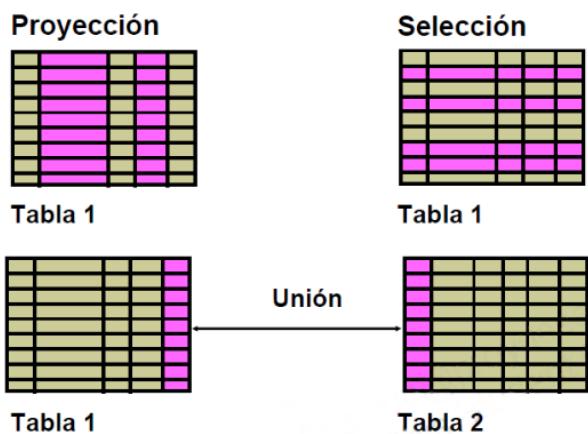
Sintaxis:

```
TRUNCATE TABLE table_name;
```

- ✓ Ejemplo:

```
TRUNCATE TABLE copy_emp;
```

Capacidades de las Sentencias SQL SELECT



Una sentencia SELECT recupera información de la base de datos. Con una sentencia SELECT, se puede hacer lo siguiente:

Proyección: seleccione las columnas de una tabla devueltas por una consulta. Seleccione tantas columnas como sea necesario.

Selección: seleccione las filas de una tabla devueltas por una consulta. Se pueden utilizar diferentes criterios para restringir las filas recuperadas.

Uniones: reúna los datos almacenados en diferentes tablas especificando el enlace entre ellas. Las uniones SQL se tratan de forma más detallada en la lección titulada "Visualización de Datos de Varias Tablas mediante Uniones".

Sentencia SELECT Básica

```
SELECT * | { [DISTINCT] column|expression [alias],... }  
FROM    table;
```

- ✓ `SELECT` identifica las `columnas` que se van a mostrar.
- ✓ `FROM` identifica la tabla que contiene estas columnas.

En la sintaxis:

<code>SELECT</code>	es una lista de una o más columnas.
<code>*</code>	selecciona todas las columnas.
<code>DISTINCT</code>	suprime los duplicados.
<code>column expression</code>	selecciona la columna o expresión especificada.
<code>Alias</code>	proporciona diferentes cabeceras de las columnas seleccionadas.
<code>FROM table</code>	especifica la tabla que contiene las columnas.

Escritura de sentencias Select

- ✓ Las sentencias `SQL` no son sensibles a mayúsculas/minúsculas.
- ✓ Las sentencias `SQL` se pueden introducir en una o más líneas.
- ✓ Las palabras clave no se pueden abreviar o dividir entre líneas.
- ✓ Las cláusulas se suelen colocar en líneas independientes.
- ✓ El sangrado se utiliza para mejorar la legibilidad.
- ✓ En `MySQL Workbench`, las sentencias `SQL` también pueden terminar con un punto y coma (`;`). Los puntos y comas son necesarios si ejecuta varias sentencias `SQL`.

Limitación de filas seleccionadas

- ✓ Restringir las filas devueltas al utilizar la cláusula WHERE:

```
SELECT * | { [DISTINCT] column|expression [alias],... }  
FROM   table  
[WHERE condition(s) ] ;
```

- ✓ La cláusula WHERE sigue a la cláusula FROM.

Puede restringir las filas que devuelve la consulta al utilizar la cláusula WHERE. Una cláusula WHERE contiene una condición que se debe cumplir e, inmediatamente después, le sigue la cláusula FROM. Si la condición es verdadera, se devolverá la fila que cumpla con la condición.

En la sintaxis:

WHERE restringe la consulta a filas que cumplan con una condición.
condition está compuesto por nombres de columna, expresiones, constantes y un operador de comparación. Una condición especifica una combinación de una o más expresiones y operadores lógicos (booleanos) y devuelve un valor de TRUE, FALSE o UNKNOWN.
La cláusula WHERE puede comparar valores en columnas, literales, expresiones aritméticas o funciones. Consta de tres elementos:

- Nombre de la columna
- Condición de comparación
- Nombre de la columna, constante o lista de valores

Uso de la cláusula WHERE

```
SELECT employee_id, last_name, job_id, department_id  
FROM employees  
WHERE department_id = 90;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

En el ejemplo, la sentencia `SELECT` recupera el ID de empleado, apellido, ID de cargo y número de departamento de todos los empleados del departamento 90.

Nota: puede utilizar el alias de columna en la cláusula `WHERE`.

Fechas y cadenas de caracteres

- ✓ Las cadenas de caracteres y valores de fecha se incluyen entre comillas simples.
- ✓ Los valores de caracteres son sensibles a mayúsculas/minúsculas y los valores de datos son sensibles a formato.
- ✓ El formato de visualización de la fecha por defecto es `DD-MON-RR`.

```
SELECT last_name, job_id, department_id  
FROM employees  
WHERE last_name = 'Whalen' ;
```

```
SELECT last_name  
FROM employees  
WHERE hire_date = '17-FEB-96' ;
```

Operadores de comparación

Operador	Significado
=	Igual que
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
<>	Distinto de
BETWEEN ... AND ...	Entre dos valores (ambos incluidos)
IN (set)	Coincide con cualquiera de los
LIKE	Coincide con un patrón de
IS NULL	Es un valor nulo

Operadores de comparación

Los operadores de comparación se utilizan en condiciones que comparan una expresión con otra expresión o valor. Se utilizan en la cláusula WHERE en el siguiente formato:

Sintaxis

... WHERE expr operator value

Ejemplo

```
... WHERE hire_date = '01-JAN-95'  

... WHERE salary >= 6000  

... WHERE last_name = 'Smith'
```

Recuerde, un alias no se puede utilizar en la cláusula WHERE.

Nota: los símbolos != y ^= también pueden representar la condición not equal to.

Uso de Condiciones de Rango mediante el Operador BETWEEN

Utilizar el operador **BETWEEN** para mostrar las filas basadas en un rango de valores:

```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 2500 AND 3500 ;
```

Límite inferior Límite superior

	LAST_NAME	SALARY
1	Rajs	3500
2	Davies	3100
3	Matos	2600
4	Vargas	2500

Puede utilizar las filas basadas en un rango de valores utilizando la condición de rango **BETWEEN**. El rango que especifique contiene un límite inferior y un límite superior. La sentencia **SELECT** de la diapositiva devuelve filas de la tabla **EMPLOYEES** para cualquier empleado cuyo salario esté entre 2.500 y 3.500 dólares.

Condición de Miembro mediante el Operador IN

Utilizar el operador **IN** para probar los valores de una lista:

```
SELECT employee_id, last_name, salary, manager_id
FROM employees
WHERE manager_id IN (100, 101, 201) ;
```

	EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
1	201 Hartstein	13000	100	
2	101 Kochhar	17000	100	
3	102 De Haan	17000	100	
4	124 Mourgos	5800	100	
5	149 Zlotkey	10500	100	
6	200 Whalen	4400	101	
7	205 Higgins	12000	101	
8	202 Fay	6000	201	

Para probar valores de un juego especificado de valores, utilice el operador **IN**. La condición definida mediante el operador **IN** también se denomina *condición de miembro*. El ejemplo de la diapositiva muestra los números de empleado, apellidos, salarios y números de empleado de los gestores de todos los empleados cuyo número de empleado del gestor sea 100, 101 o 201.

Coincidencia de Patrones mediante el Operador LIKE

- ✓ Utilizar el operador **LIKE** para realizar búsquedas con comodines de valores de cadena de búsqueda válidos.
- ✓ Las condiciones de búsqueda pueden contener caracteres literales o números:
 - % indica cero o varios caracteres.
 - _ indica un carácter.

```
SELECT first_name
FROM employees
WHERE first_name LIKE 'S%';
```

Puede que no siempre conozca el valor exacto que debe buscar. Puede seleccionar filas que coincidan con un patrón de caracteres utilizando la condición **LIKE**. Se hace referencia a la operación de coincidencia de patrón de caracteres como búsqueda con *comodines*.

Para crear la cadena de búsqueda se pueden utilizar dos símbolos.

La sentencia **SELECT** de la diapositiva devuelve el nombre del empleado de la tabla **EMPLOYEES** de cualquier empleado cuyo nombre empiece por la letra “S”. Observe que se trata de la “S” mayúscula. No se devolverán los nombres que empiecen por “s” minúscula.

Uso de las Condiciones NULL

Probar condiciones nulas con el operador **IS NULL**.

```
SELECT last_name, manager_id
FROM employees
WHERE manager_id IS NULL;
```

LAST_NAME	MANAGER_ID
King	(null)

Las condiciones **NULL** incluyen las condiciones **IS NULL** e **IS NOT NULL**.

La condición **IS NULL** prueba las condiciones nulas.

Un valor nulo significa que el valor no está disponible, no está asignado, se desconoce o no es aplicable. Por lo tanto, no puede probar con = porque un valor nulo no puede ser igual o desigual a cualquier valor. El ejemplo de la diapositiva recupera los apellidos y gestores de todos los empleados que no tienen un gestor.

Definición de Condiciones mediante los Operadores Lógicos

Operador	Significado
AND	Devuelve TRUE si <i>ambas</i> condiciones de componente son verdaderas
OR	Devuelve TRUE si <i>cualquier</i> condición de componente es verdadera
NOT	Devuelve TRUE si la condición es falsa

Una condición lógica combina el resultado de dos condiciones de componentes para producir un resultado único basado en dichas condiciones o invierte el resultado de una condición única. Se devuelve una fila sólo si el resultado global de la condición es verdadera.

En SQL, están disponibles tres operadores lógicos:

- ✓ AND
- ✓ OR
- ✓ NOT

Todos los ejemplos indicados hasta ahora han especificado sólo una condición en la cláusula WHERE. Puede utilizar varias condiciones en una única cláusula WHERE mediante los operadores AND y OR.

Tabla de Verdad AND

La siguiente tabla muestra los resultados de combinar dos expresiones con AND:

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

Tabla de verdad OR

La siguiente tabla muestra los resultados de combinar dos expresiones con OR:

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL



Uso del Operador NOT

```
SELECT last_name, job_id
FROM employees
WHERE job_id
NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');
```

#	LAST_NAME	JOB_ID
1	De Haan	AD_VP
2	Fay	MK_REP
3	Gietz	AC_ACCOUNT
4	Hartstein	MK_MAN
5	Higgins	AC_MGR
6	King	AD_PRES
7	Kochhar	AD_VP
8	Mourgos	ST_MAN
9	Whalen	AD_ASST
10	Zlotkey	SA_MAN

En el ejemplo de la diapositiva se muestra el apellido y el ID de trabajo de todos los empleados cuyo ID de trabajo *no sea* IT_PROG, ST_CLERK o SA_REP.

Tabla de Verdad NOT

La siguiente tabla muestra el resultado de aplicar el operador **NOT** a una condición:

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

Nota: el operador **NOT** también se puede utilizar con otros operadores **SQL**, como **BETWEEN**, **LIKE** y **NULL**.

```
... where job_id    NOT IN ('AC ACCOUNT' , 'AD VP')
... where salary    NOT BETWEEN 10000 AND 15000
... where last_name NOT LIKE '%A%'
... where commission_pct IS NOT NULL
```

Uso de la Cláusula ORDER BY

- ✓ Ordenar las filas recuperadas con la cláusula **ORDER BY**:
 - **ASC**: orden ascendente, valor por defecto
 - **DESC**: orden descendente
- ✓ La cláusula **ORDER BY** es la última en una sentencia **SELECT**:

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY hire_date ;
```

	LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
1	King	AD_PRES	90	17-JUN-87
2	Whalen	AD_ASST	10	17-SEP-87
3	Kochhar	AD_VP	90	21-SEP-89
4	Hunold	IT_PROG	60	03-JAN-90
5	Ernst	IT_PROG	60	21-MAY-91
6	De Haan	AD_VP	90	13-JAN-93

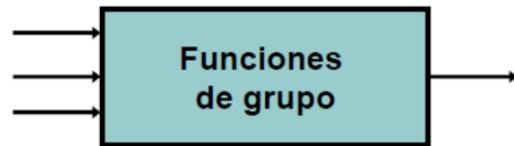
Ordenación

El orden por defecto es el orden ascendente:

- ✓ Los valores numéricos se muestran con los valores más bajos primero (por ejemplo, de 1 a 999).
- ✓ Los valores de fecha se muestran con el primer valor en primer lugar (por ejemplo, 01-ENE-92 antes de 01-ENE-95).
- ✓ Los valores de caracteres se muestran en orden alfabético (por ejemplo, primero la "A" y por último la "Z").
- ✓ Los valores nulos se muestran al final para las secuencias ascendentes y al principio para las secuencias descendentes.
- ✓ Puede ordenar por una columna que no esté en la lista **SELECT**.

Tipos de Funciones de Grupos

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE



A diferencia de las funciones de una sola fila, las funciones de grupo funcionan en juegos de filas para proporcionar un resultado por grupo. Estos juegos pueden formar la tabla completa o la tabla dividida en grupos.

Tipos de Funciones de Grupos

Cada una de las funciones acepta un argumento. La siguiente tabla identifica las opciones que se pueden utilizar en la sintaxis:

Función	Descripción
AVG ([DISTINCT ALL] <i>n</i>)	Valor medio de <i>n</i> ; ignora los valores nulos
COUNT ({ * [DISTINCT ALL] <i>expr</i> })	Número de filas donde <i>expr</i> evalúa otros valores que no son nulos (tiene en cuenta todas filas seleccionadas con *, incluyendo duplicados y filas con valores nulos)
MAX ([DISTINCT ALL] <i>expr</i>)	Valor máximo de <i>expr</i> ; ignora los valores nulos
MIN ([DISTINCT ALL] <i>expr</i>)	Valor mínimo de <i>expr</i> ; ignora los valores nulos
STDDEV ([DISTINCT ALL] <i>n</i>)	Desviación estándar de <i>n</i> ; ignora los valores nulos
SUM ([DISTINCT ALL] <i>n</i>)	Valores de suma de <i>n</i> ; ignora los valores nulos
VARIANCE ([DISTINCT ALL] <i>n</i>)	Varianza de <i>n</i> ; ignora los valores nulos

Funciones de Grupos Sintaxis

```
SELECT      group_function(column), ...
FROM        table
[WHERE      condition]
[ORDER BY   column];
```

La función de grupo se coloca después de la palabra clave SELECT. Puede que tenga varias funciones de grupo separadas por comas.
Instrucciones para utilizar las funciones de grupo:

- ✓ DISTINCT hace que la función considere sólo los valores no duplicados; ALL hace que considere cada valor, incluyendo los duplicados. El valor por defecto es ALL y, por lo tanto, no es necesario especificarlo.
- ✓ Los tipos de datos para las funciones con el argumento expr pueden ser CHAR, VARCHAR2, NUMBER o DATE.
- ✓ Todas las funciones de grupo ignoran los valores nulos.

Uso de las Funciones AVG y SUM

Puede utilizar AVG y SUM para datos numéricos.

```
SELECT AVG(salary), MAX(salary),  
       MIN(salary), SUM(salary)  
  FROM employees  
 WHERE job_id LIKE '%REP%';
```

	Avg(Salary)	Max(Salary)	Min(Salary)	Sum(Salary)
1	8150	11000	6000	32600

Puede utilizar las funciones AVG, SUM, MIN y MAX en las columnas que pueden almacenar datos numéricos. El ejemplo de la diapositiva muestra la media, el valor más alto, más bajo y la suma de los salarios mensuales de todos los vendedores.

Nota: las funciones AVG, SUM, VARIANCE y STDDEV se pueden utilizar sólo con los tipos de dato numéricos. MAX and MIN no se pueden utilizar con tipos de dato LOB o LONG.

Uso de la Función COUNT

COUNT (*) devuelve el número de filas en una tabla:

1

```
SELECT COUNT(*)  
  FROM employees  
 WHERE department_id = 50;
```

	COUNT(*)
1	5

COUNT (expr) devuelve el número de filas con valores no nulos para la expresión *expr*:

2

```
SELECT COUNT(commission_pct)  
  FROM employees  
 WHERE department_id = 80;
```

	COUNT(COMMISSION_PCT)
1	3

Creación de Grupos de Datos

Sintaxis de la cláusula **Group By**

Puede dividir las filas de una tabla en grupos más pequeños utilizando la cláusula **GROUP BY**.

```
SELECT      column, group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY  group_by_expression]
[ORDER BY  column];
```

Puede utilizar la cláusula **GROUP BY** para dividir las filas de la tabla en grupos. A continuación puede utilizar las funciones de grupo para devolver información de resumen de cada grupo.

En la sintaxis:

group_by_expression Especifica columnas cuyos valores determinan la base para agrupar filas

Uso de la cláusula **Group By** en varias columnas

Puede devolver resultados de resumen para grupos y subgrupos mostrando varias columnas **GROUP BY**. La cláusula **GROUP BY** agrupa filas pero no garantiza el orden del juego de resultados. Para ordenar los agrupamientos, utilice la cláusula **ORDER BY**.

En el ejemplo de la diapositiva, la sentencia **SELECT** que contiene una cláusula **GROUP BY** se evalúa de la siguiente forma:

- ✓ La cláusula **SELECT** especifica la columna que se van a recuperar:
 - ID de departamento en la tabla **EMPLOYEES**
 - ID de cargo de la tabla **EMPLOYEES**
 - Suma de todos los salarios del grupo especificada en la cláusula **GROUP BY**
- ✓ La cláusula **FROM** especifica las dos tablas a las que la base de datos debe acceder: tabla **EMPLOYEES**.
- ✓ La cláusula **WHERE** reduce el juego de resultados a aquellas filas en las que el **ID** de departamento es mayor de 40.
- ✓ La cláusula **GROUP BY** especifica cómo debe agrupar las filas resultantes:
 - En primer lugar, las filas se agrupan por **ID** de departamento.
 - En segundo lugar, las filas se agrupan por **ID** de cargo en los grupos de **ID** de departamento.
- ✓ La cláusula **ORDER BY** ordena los resultados por **ID** de departamento.

Nota: la función **SUM** se aplica a la columna de salario de todos los **ID** de cargo en el juego de resultados de cada grupo de **ID** de departamento. Además, tenga en cuenta que la fila **SA_REP** no se devuelve. El **ID** de departamento para esta fila es **NULL** y, por lo tanto, no cumple la condición **WHERE**.

Uso de la cláusula HAVING

```
SELECT      department_id, MAX(salary)
FROM        employees
GROUP BY    department_id
HAVING      MAX(salary)>10000 ;
```

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	90	24000
3	110	12000
4	80	11000

El ejemplo de la diapositiva muestra los número de departamento y los salarios máximos de los departamentos cuyo salario máximo sea superior a 10.000 dólares.

Puede utilizar la cláusula `GROUP BY` sin utilizar una función de grupo en la lista `SELECT`. Si restringe las filas según el resultado de una función de grupo, debe tener una cláusula `GROUP BY` y una cláusula `HAVING`.

El siguiente ejemplo muestra los números de departamento y los salarios medios de los departamentos cuyo salario máximo sea superior a 10.000 dólares:

```
SELECT      department_id, AVG(salary)
FROM        employees
GROUP BY    department_id
HAVING      max(salary)>10000;
```

	DEPARTMENT_ID	Avg(Salary)
1	20	9500
2	90	19333.3333333333...
3	110	10150
4	80	10033.3333333333...

Anidamiento de Funciones

Mostrar el salario máximo medio:

```
SELECT MAX(AVG(salary))  
FROM employees  
GROUP BY department id;
```

MAX(AVG(SALARY))

Las funciones de grupo se pueden anidar en una profundidad de dos.

El ejemplo de la diapositiva calcula el salario medio para cada `department_id` y, a continuación, muestra el salario máximo medio.

Tenga en cuenta que la cláusula `GROUP BY` es obligatoria al anidar funciones de grupo.

Cualificación de nombres de columnas ambiguas

Al unir dos o más tablas, debe cualificar los nombres de las columnas con el nombre de la tabla para evitar ambigüedad. Sin los prefijos de tabla, la columna `DEPARTMENT_ID` de la lista `SELECT` puede provenir de la tabla `DEPARTMENTS` o de la tabla `EMPLOYEES`. Es necesario agregar el prefijo de tabla para ejecutar la consulta. Si no existen nombres de columna comunes entre las dos tablas, no es necesario cualificar las columnas. Sin embargo, el uso del prefijo de tabla mejora el rendimiento, ya que indica al servidor de Oracle dónde encontrar exactamente las columnas.

Sin embargo, la cualificación de nombres de columna con nombres de tabla puede llevar bastante tiempo, especialmente si los nombres de tabla son largos. En su lugar, puede utilizar *alias de tabla*. Igual que un alias de columna proporciona otro nombre a una columna, un alias de tabla proporciona otro nombre a una tabla. Los alias de tabla ayudan a mantener el código `SQL` más pequeño y, por lo tanto, menos uso de memoria.

El nombre de tabla se especifica por completo, seguido de un espacio y del alias de tabla. Por ejemplo, a la tabla `EMPLOYEES` se le puede proporcionar el alias `e` y, a la tabla `DEPARTMENTS` el alias `d`.

Instrucciones

- Los alias de tabla pueden tener hasta 30 caracteres de longitud, pero los alias más cortos son mejores que los largos.
- Si se utiliza un alias de tabla para un nombre de tabla determinado en la cláusula `FROM`, el alias de tabla se deberá sustituir por el nombre de tabla mediante la sentencia `SELECT`.
- Los alias de tabla deben ser significativos.
- El alias de tabla es válido sólo para la sentencia actual `SELECT`.

Recuperación de registros con cláusula `ON`

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
  FROM employees e JOIN departments d
 WHERE (e.department_id = d.department_id);
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200 Whalen	10	10	1700
2	201 Hartstein	20	20	1800
3	202 Fay	20	20	1800
4	144 Vargas	50	50	1500
5	143 Matos	50	50	1500
6	142 Davies	50	50	1500
7	141 Raji	50	50	1500
8	124 Moursund	50	50	1500
9	103 Hunold	60	60	1400
10	104 Ernst	60	60	1400
11	107 Lorentz	60	60	1400

En este ejemplo, las columnas `DEPARTMENT_ID` en la tabla `EMPLOYEES` y `DEPARTMENTS` se unen con la cláusula `ON`. Cuando un ID de departamento de la tabla `EMPLOYEES` sea igual al ID de departamento de la tabla `DEPARTMENTS`, se devolverá una fila. El alias de tabla es necesario para cualificar los `column names` coincidentes.

También puede utilizar la cláusula `ON` para unir columnas que tienen nombres diferentes. Los paréntesis de las columnas unidas, como se muestra en el ejemplo de la diapositiva, `(e.department_id = d.department_id)` son opcionales.

Incluso, `ON e.department_id = d.department_id` funcionará.

Creación de uniones en 3 direcciones con la cláusula ON

```
SELECT employee_id, city, department_name
FROM employees e
JOIN departments d
ON d.department_id = e.department_id
JOIN locations l
ON d.location_id = l.location_id;
```

	EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100	Seattle	Executive
2	101	Seattle	Executive
3	102	Seattle	Executive
4	103	Southlake	IT
5	104	Southlake	IT
6	107	Southlake	IT
7	124	South San Francisco	Shipping
8	141	South San Francisco	Shipping
9	142	South San Francisco	Shipping

...

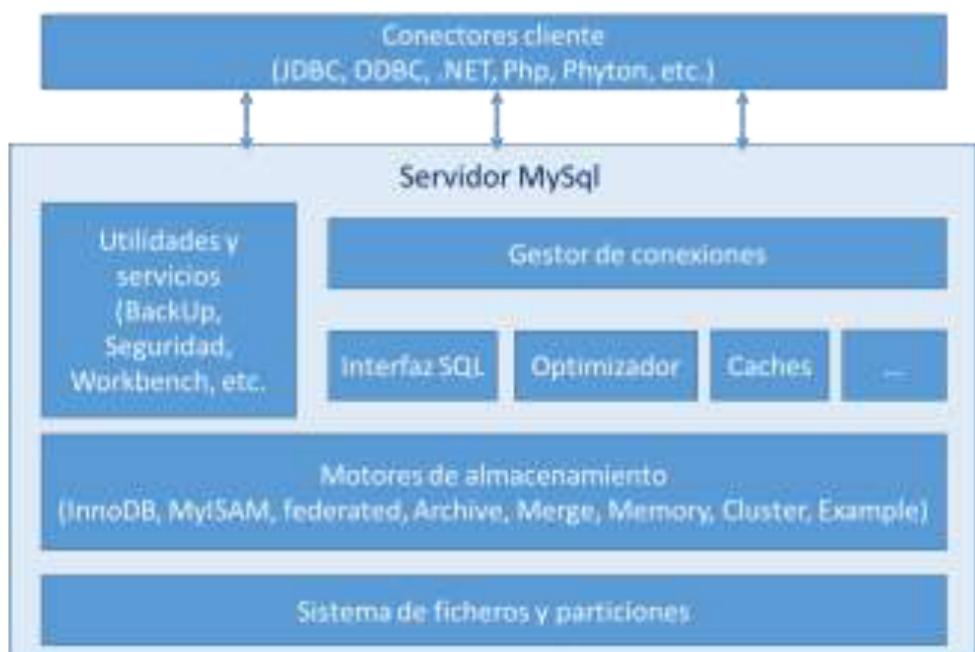
Una unión en tres direcciones es una unión de tres tablas. En la sintaxis compatible con SQL:1999, las uniones se realizan de izquierda a derecha. Por lo tanto, la primera unión que se realiza es ~~EMPLOYEES JOIN DEPARTMENTS~~. La primera condición de unión puede hacer referencia a las columnas de ~~EMPLOYEES~~ y ~~DEPARTMENTS~~, pero no puede hacer referencia a las columnas de ~~LOCATIONS~~. La segunda condición de unión puede hacer referencia a las columnas de las tres tablas.

Unidad 6

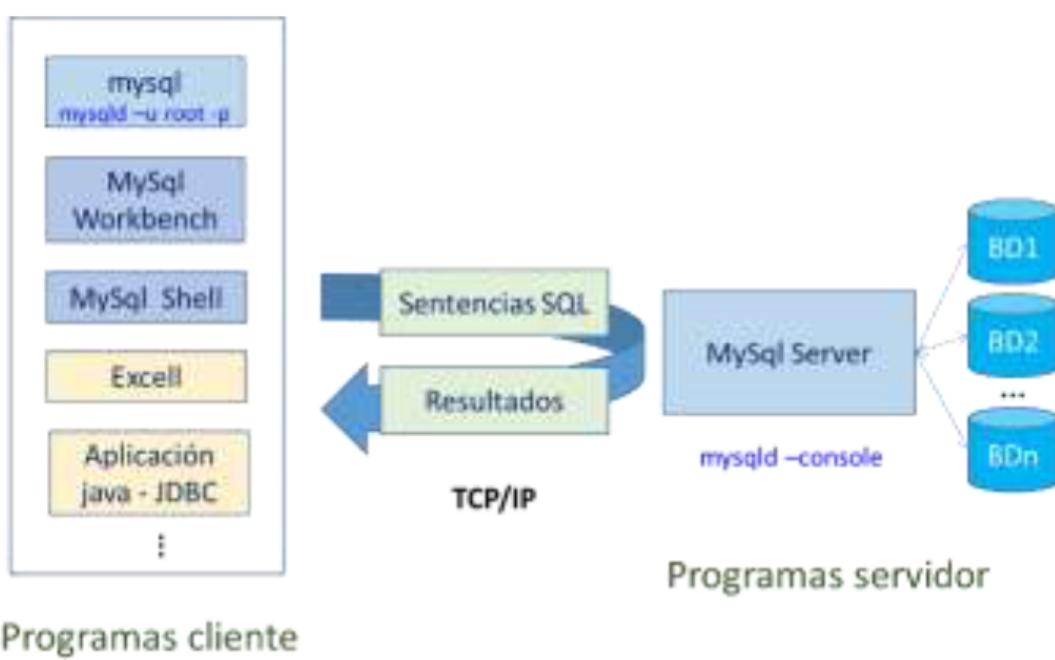
SQL es un lenguaje de modelado de datos universal a la hora de manipular bases de datos relacionales.

Lo interesante de este lenguaje es que si ustedes deciden utilizar como motor de base de datos MySql, SQL Server, DB2, SQLite, Oracle o cualquier otro motor de base de datos relacional, las consultas a la base de datos se ejecutan de la misma manera haciendo uso del lenguaje SQL.

MySQL, es el sistema de gestión de bases de datos SQL de código abierto más popular. Actualmente desarrollado, distribuido y respaldado por Oracle Corporation que ofrece una versión comercial (MySql Enterprise). Está escrito en C y C++ y está disponible para múltiples plataformas. La siguiente figura muestra la arquitectura de MySql:



Un servidor MySQL puede gestionar múltiples bases de datos y conexiones cliente a las mismas:



Además de MySQL Workbench, existe otro programa cliente popular llamado PhpMyAdmin que también permite administrar y gestionar bases de datos MySQL en diferentes sistemas operativos.

MySQL Server dispone de varios motores siendo InnoDB el más utilizado:

Engine	Descripción
InnoDB	Soporta transacciones, bloqueo a nivel de fila y claves foráneas. Es el motor por defecto
MyISAM	Utilizada para trabajo de solo lectura o principalmente de lectura
MEMORY	Los datos se almacenan en memoria principal.
CSV	Tablas almacenadas en formato de valores separados por comas
ARCHIVE	Motor de almacenamiento de archivo
EXAMPLE	Motor de ejemplo, de utilidad para desarrolladores.
FEDERATED	Motor de almacenamiento que accede a tablas remotas.
HEAP	Sinónimo del motor MEMORY.
MERGE	Colección de tablas MyISAM usadas como una sola. También conocido como MRG MyISAM.

Introducción a SQL

Introducción a SQL

Este bloque es el más relevante cuando se trata de la manipulación de bases de datos desde un punto de vista práctico, aunque es muy importante haber pasado por los capítulos anteriores para tener las bases necesarias para abordar este.

Aquí trataremos el lenguaje DDL (Lenguaje de definición de datos) y el DML, (Lenguaje de Manipulación de datos).

Tengan presente que la gran diferencia entre estos es que DDL nos permite definir y modificar la base de datos a nivel estructura, es decir, crear bases de datos, crear tablas, modificarlas, etc, mientras que DML son aquellas sentencias que nos permiten manipular los datos propiamente dichos, por ejemplo agregar registros a una tabla, modificar estos registros o mostrarlos.