

BASE DE DATOS I

PROCESUAL HITO 3

¿QUE SIGNIFICA DDL?

LENGUAJE DE DEFINICIÓN DE DATOS (DDL)

Es un lenguaje de programación para definir estructuras de datos, proporcionado por los sistemas gestores de bases de datos, en este caso PostgreSQL. En inglés, Data Definition Language, de ahí sus siglas DDL.

Para definir la estructura disponemos de tres sentencias:

CREATE, se usa para crear una base de datos, tabla, vistas, etc.

ALTER, se utiliza para modificar la estructura, por ejemplo añadir o borrar columnas de una tabla.

DROP, con esta sentencia, podemos eliminar los objetos de la estructura, por ejemplo un índice o una secuencia.

¿QUE SIGNIFICA DML?

LENGUAJE DE MANIPULACIÓN DE DATOS (DML)

También es un lenguaje proporcionado por los sistemas gestores de bases de datos. En inglés, Data Manipulation Language (DML).

Utilizando instrucciones de SQL, permite a los usuarios introducir datos para posteriormente realizar tareas de consultas o modificación de los datos que contienen las Bases de Datos.

Los elementos que se utilizan para manipular los datos, son los siguientes:

SELECT, esta sentencia se utiliza para realizar consultas sobre los datos.

INSERT, con esta instrucción podemos insertar los valores en una base de datos.

UPDATE, sirve para modificar los valores de uno o varios registros.

DELETE, se utiliza para eliminar las filas de una tabla.

PRIMARY KEY

La clave primaria, **PRIMARY KEY**, identifica de manera única cada fila de una tabla.

La columna definida como clave primaria (PRIMARY KEY) debe ser **UNIQUE** (valor único) y **NOT NULL** (no puede contener valores nulos).

Cada tabla sólo puede tener una clave primaria (PRIMARY KEY).

Ejemplo **PRIMARY KEY**, clave primaria en MySQL

```
CREATE TABLE personas { identificador int NOT NULL, nombre varchar(255) NOT NULL, apellido1  
varchar(255) NOT NULL, PRIMARY KEY (identificador) }
```


FOREIGN KEY

Un **FOREIGN KEY** es una columna o grupo de columnas en una tabla de base de datos relacional que proporciona un vínculo entre los datos de dos tablas. Actúa como una referencia cruzada entre tablas porque hace referencia a la clave principal de otra tabla, estableciendo así un vínculo entre ellas.

FOREIGN KEY

Ejemplo de FOREIGN KEY

Tabla "departamentos", con la clave primaria "dep"

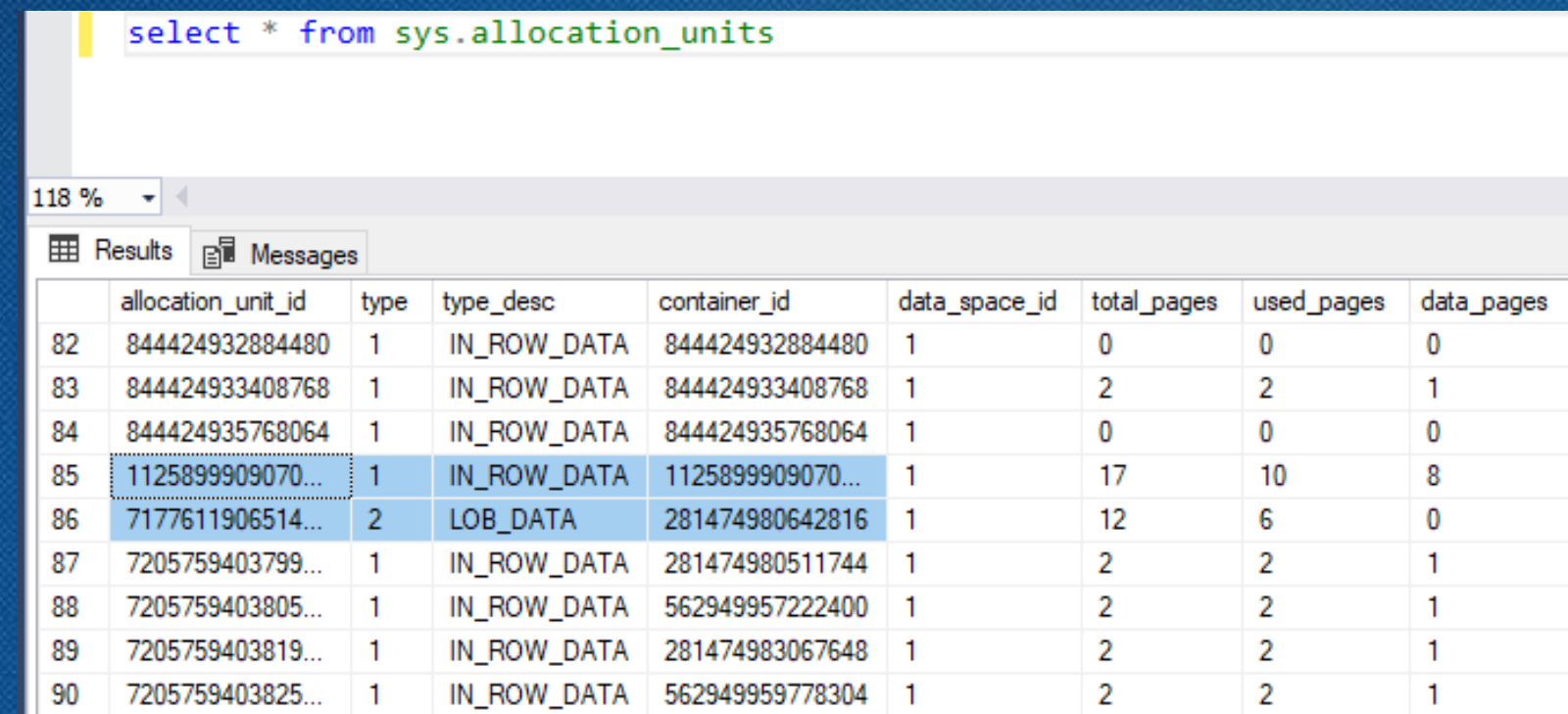
Definiciones de FOREIGN KEY en **CREATE TABLE** para MySQL

```
CREATE TABLE departamentos { dep int NOT NULL,  
departamento varchar(255),  
PRIMARY KEY (dep)  
}
```

```
CREATE TABLE personas  
{  
per int NOT NULL,  
nombre varchar(255),  
apellido1 varchar(255),  
dep int NOT NULL,  
PRIMARY KEY (per),  
FOREIGN KEY (dep) REFERENCES departamentos(dep)  
}
```


TABLAS

Las tablas son objetos de base de datos que contienen todos sus datos. En las tablas, los datos se organizan con arreglo a un formato de filas y columnas, similar al de una hoja de cálculo. Cada fila representa un registro único y cada columna un campo dentro del registro.



The screenshot shows a SQL query window with the command `select * from sys.allocation_units`. Below the query, the results are displayed in a table grid. The table has 9 columns: `allocation_unit_id`, `type`, `type_desc`, `container_id`, `data_space_id`, `total_pages`, `used_pages`, and `data_pages`. The results are numbered 82 through 90. Row 85 is highlighted with a blue background.

	allocation_unit_id	type	type_desc	container_id	data_space_id	total_pages	used_pages	data_pages
82	844424932884480	1	IN_ROW_DATA	844424932884480	1	0	0	0
83	844424933408768	1	IN_ROW_DATA	844424933408768	1	2	2	1
84	844424935768064	1	IN_ROW_DATA	844424935768064	1	0	0	0
85	1125899909070...	1	IN_ROW_DATA	1125899909070...	1	17	10	8
86	7177611906514...	2	LOB_DATA	281474980642816	1	12	6	0
87	7205759403799...	1	IN_ROW_DATA	281474980511744	1	2	2	1
88	7205759403805...	1	IN_ROW_DATA	562949957222400	1	2	2	1
89	7205759403819...	1	IN_ROW_DATA	281474983067648	1	2	2	1
90	7205759403825...	1	IN_ROW_DATA	562949959778304	1	2	2	1

IDENTITY

Una columna de **IDENTITY** es una columna numérica en una tabla que se completa automáticamente con un valor entero cada vez que se inserta una fila.

Las columnas de identidad a menudo se definen como columnas de enteros, pero también se pueden declarar como bigint, smallint, tinyint o numéricas o decimales, siempre que la escala sea 0.

Los valores generados automáticamente para cada fila insertada se basan en la semilla y una propiedad de incremento de la columna de identidad. La siguiente sintaxis se utiliza al definir una columna de identidad:

```
IDENTITY [ (seed, increment)]
```

La seed es el primer valor cargado en la tabla, y el increment se agrega al valor de identidad anterior cargado para crear el siguiente valor subsiguiente. Tanto los valores iniciales como los incrementales deben proporcionarse juntos si desea anular los valores predeterminados. Si no se proporcionan valores de inicialización e incremento, los valores predeterminados para inicialización e incremento son ambos 1.

WHERE

La cláusula **WHERE** de SQL se utiliza para especificar una condición al recuperar un conjunto de datos de una tabla o de un conjunto de tablas. Si se cumple la condición dada, la consulta devuelve los valores relacionados con la condición que se especifique en la cláusula **WHERE**. Debe usar la cláusula **WHERE** para filtrar los registros y obtener solo los registros necesarios.

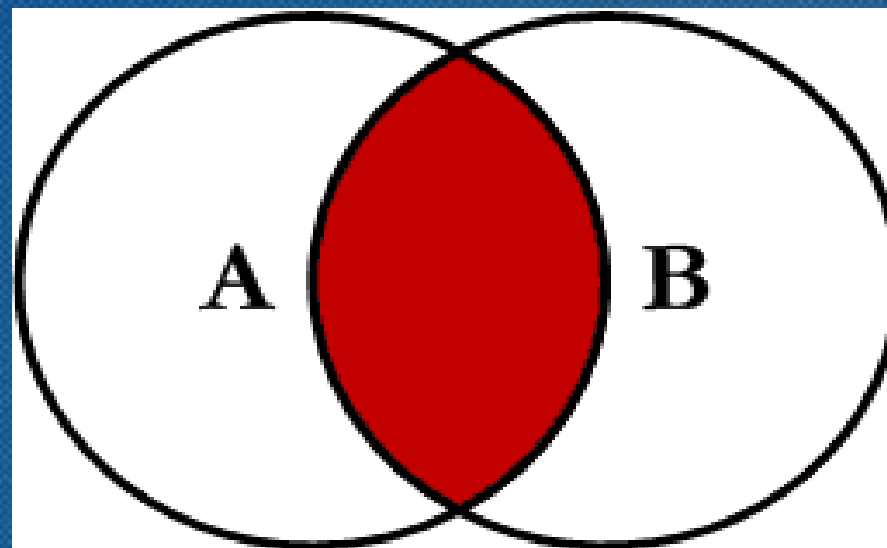
La cláusula **WHERE** no solo se usa en la instrucción **SELECT**, sino que también se usa en la instrucción **UPDATE** y **DELETE**.

INNER JOIN

Cláusula **INNER JOIN**

Lo más usual, lo primero que se suele aprender, es el uso de **INNER JOIN**, o generalmente abreviado como **JOIN**.

Esta cláusula busca coincidencias entre 2 tablas, en función a una columna que tienen en común. De tal modo que sólo la intersección se mostrará en los resultados.

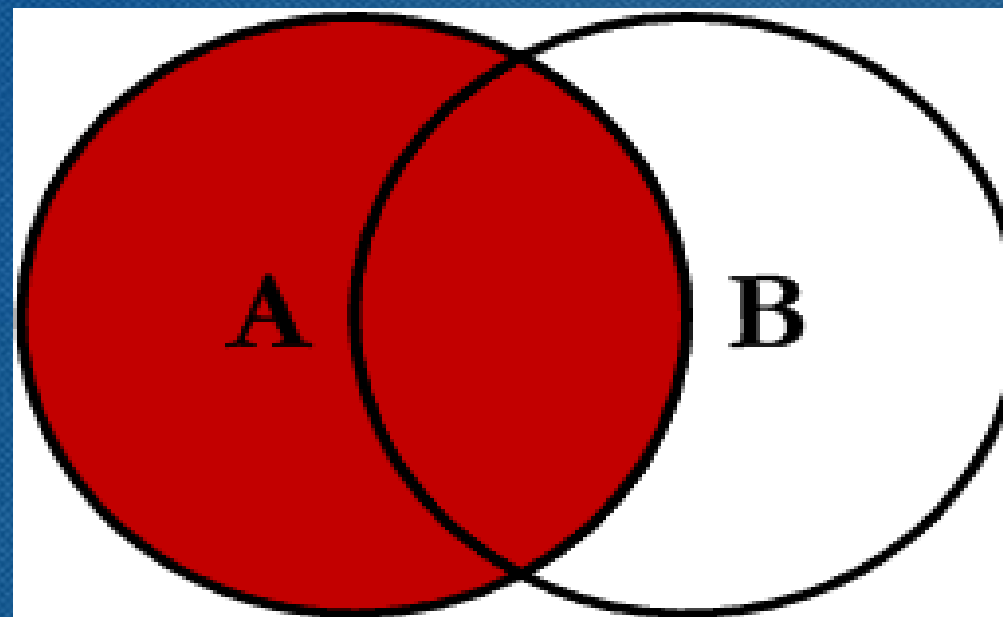


LEFT JOIN

Cláusula **LEFT JOIN**

A diferencia de un **INNER JOIN**, donde se busca una intersección respetada por ambas tablas, con **LEFT JOIN** damos prioridad a la tabla de la izquierda, y buscamos en la tabla derecha.

Si no existe ninguna coincidencia para alguna de las filas de la tabla de la izquierda, de igual forma todos los resultados de la primera tabla se muestran.



LEFT JOIN

He aquí una consulta de ejemplo:

```
SELECT
    E.Nombre as 'Empleado',
    D.Nombre as 'Departamento'
FROM Empleados E
LEFT JOIN Departamentos D
ON E.DepartamentoId = D.Id
```

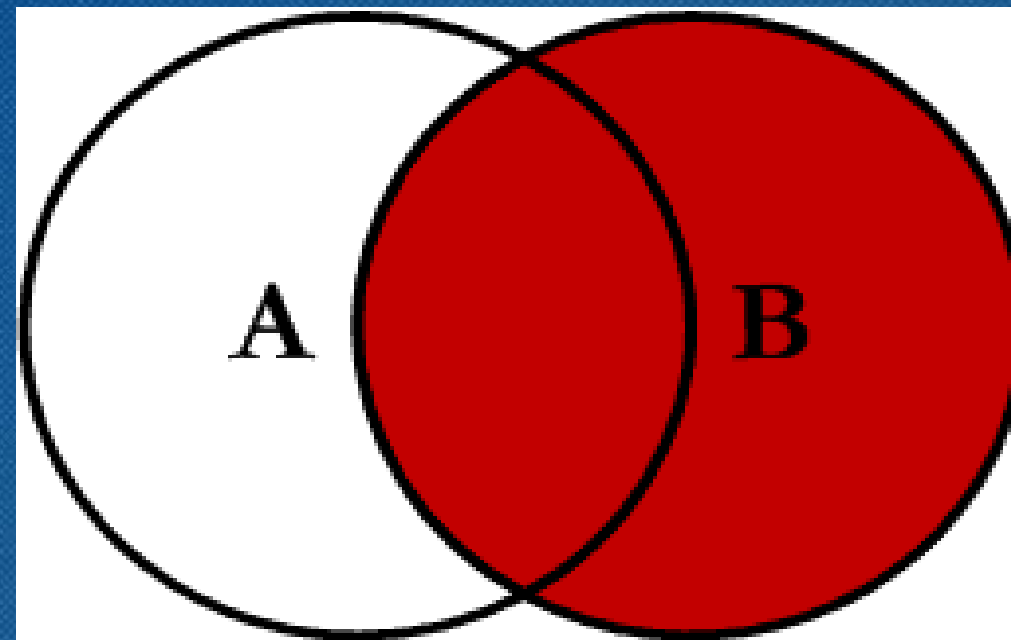
La tabla Empleados es la primera tabla en aparecer en la consulta (en el **FROM**), por lo tanto ésta es la tabla **LEFT** (izquierda), y todas sus filas se mostrarán en los resultados.

La tabla Departamentos es la tabla de la derecha (aparece luego del **LEFT JOIN**). Por lo tanto, si se encuentran coincidencias, se mostrarán los valores correspondientes, pero sino, aparecerá **NULL** en los resultados.

RIGHT JOIN

Cláusula **RIGHT JOIN**

En el caso de **RIGHT JOIN** la situación es muy similar, pero aquí se da prioridad a la tabla de la derecha.



RIGHT JOIN

De tal modo que si usamos la siguiente consulta, estaremos mostrando todas las filas de la tabla de la derecha:

```
SELECT
    E.Nombre as 'Empleado',
    D.Nombre as 'Departamento'
FROM Empleados E
RIGHT JOIN Departamentos D
ON E.DepartamentoId = D.Id
```

La tabla de la izquierda es Empleados, mientras que Departamentos es la tabla de la derecha.

La tabla asociada al **FROM** será siempre la tabla **LEFT**, y la tabla que viene después del **JOIN** será la tabla **RIGHT**.

Entonces el resultado mostrará todos los departamentos al menos 1 vez.

Y si no hay ningún empleado trabajando en un departamento determinado, se mostrará **NULL**. Pero el departamento aparecerá de igual forma.

Referencias :

<https://programacionymas.com/blog/como-funciona-inner-left-right-full-join>

<http://sql.11sql.com/sql-foreign-key.htm>

<https://www.red-gate.com/simple-talk/databases/sql-server/learn/sql-server-identity-column/#.~:text=An%20identity%20column%20is%20a,as%20the%20scale%20is%200.>

<http://sql.11sql.com/sql-primary-key.htm>

<https://learn.microsoft.com/es-es/sql/relational-databases/tables/tables?view=sql-server-ver16>

THANK YOU!