***Base de datos***

Ing. Yolanda Moya

Nombre: Jhonathan Chiliquinga Curso: 2”A”

***Realizar un modelo lógico del ejercicio propuesto.***

La empresa desea realizar un control de sus ventas y montajes, para lo cual se tiene en cuenta:

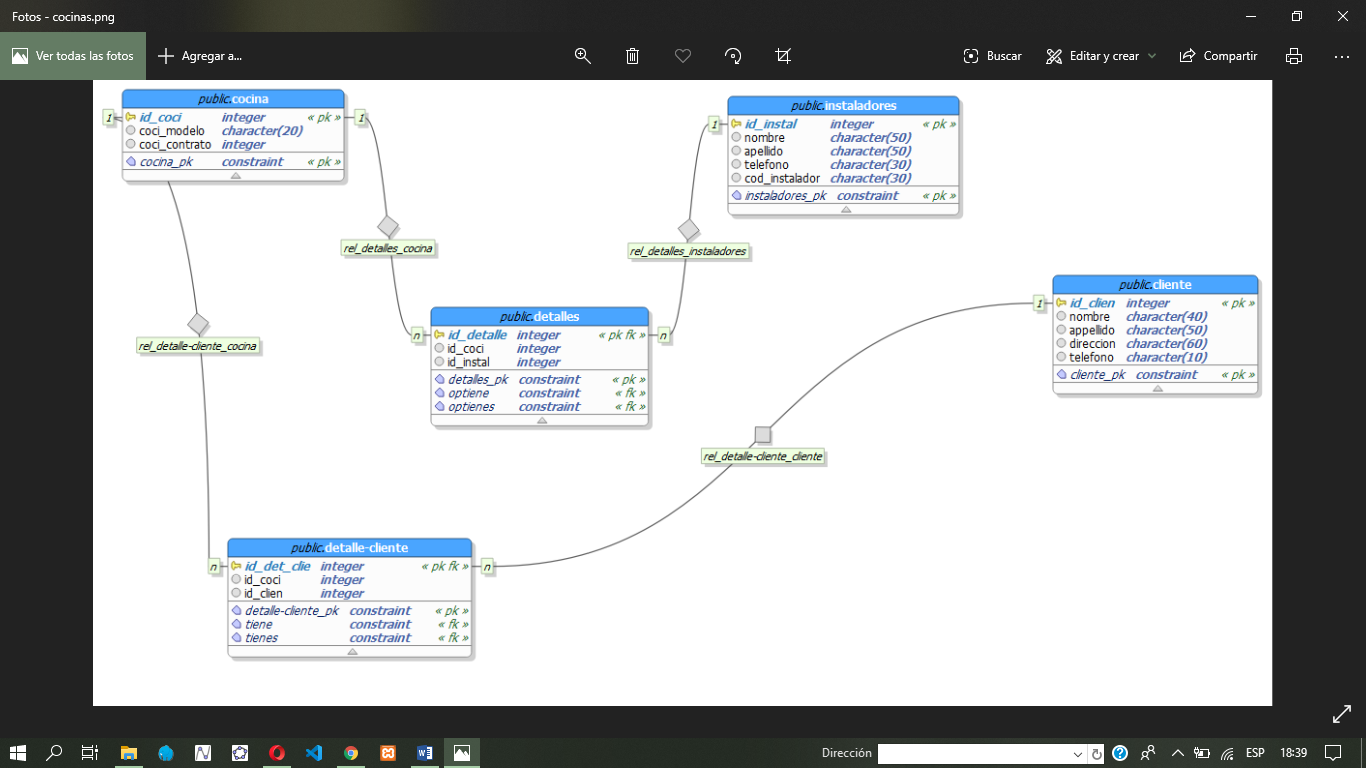
* De cada modelo de cocina nos interesa el número de referencia del modelo.
* De un montador nos interesa su cédula, nombre, dirección, teléfono de contacto y el número de cocinas que ha montado de cada modelo.
* Cada modelo cocina lo debe montar al menos un montador, y el mismo montador puede montar varios modelos, porque no se especializan en ninguno en concreto.
* De un cliente nos interesa su cédula, nombre, dirección y teléfono. Cada modelo de cocina pueden comprarlo uno o varios clientes, y el mismo cliente puede comprar varias modelos de cocinas.

Usando la herramienta:

PgModeler - Postgresql Database Modelar

Con esta herramienta nos permite modelar tablas con sus campos y entidades de cada uno de ellos.

Si tenemos entidades que van de varios a varios tenemos que romper su relación en otras tablas donde se bajan sus a id que es pk de cada tabla y lo transformamos en una fk.



# ***Normalización de bases de datos***

La **normalización de bases de datos** es un proceso que consiste en designar y aplicar una serie de reglas a las relaciones obtenidas tras el paso del [modelo entidad-relación](https://es.wikipedia.org/wiki/Modelo_entidad-relaci%C3%B3n) al [modelo relacional](https://es.wikipedia.org/wiki/Modelo_relacional). Con objeto de minimizar la redundancia de datos, facilitando su gestión posterior.

Las bases de datos relacionales se normalizan para:

* Evitar la redundancia de los datos.
* Evitar problemas de actualización de los datos en las tablas.
* Proteger la integridad de los datos.

En el modelo relacional es frecuente llamar tabla a una relación, aunque para que una tabla bidimensional sea considerada como una relación tiene cumplir con algunas restricciones:

* Cada columna debe tener su nombre único.
* No puede haber dos filas iguales. No se permiten los duplicados.
* Todos los datos en una columna deben ser del mismo tipo.

***Tipos de Normalización***

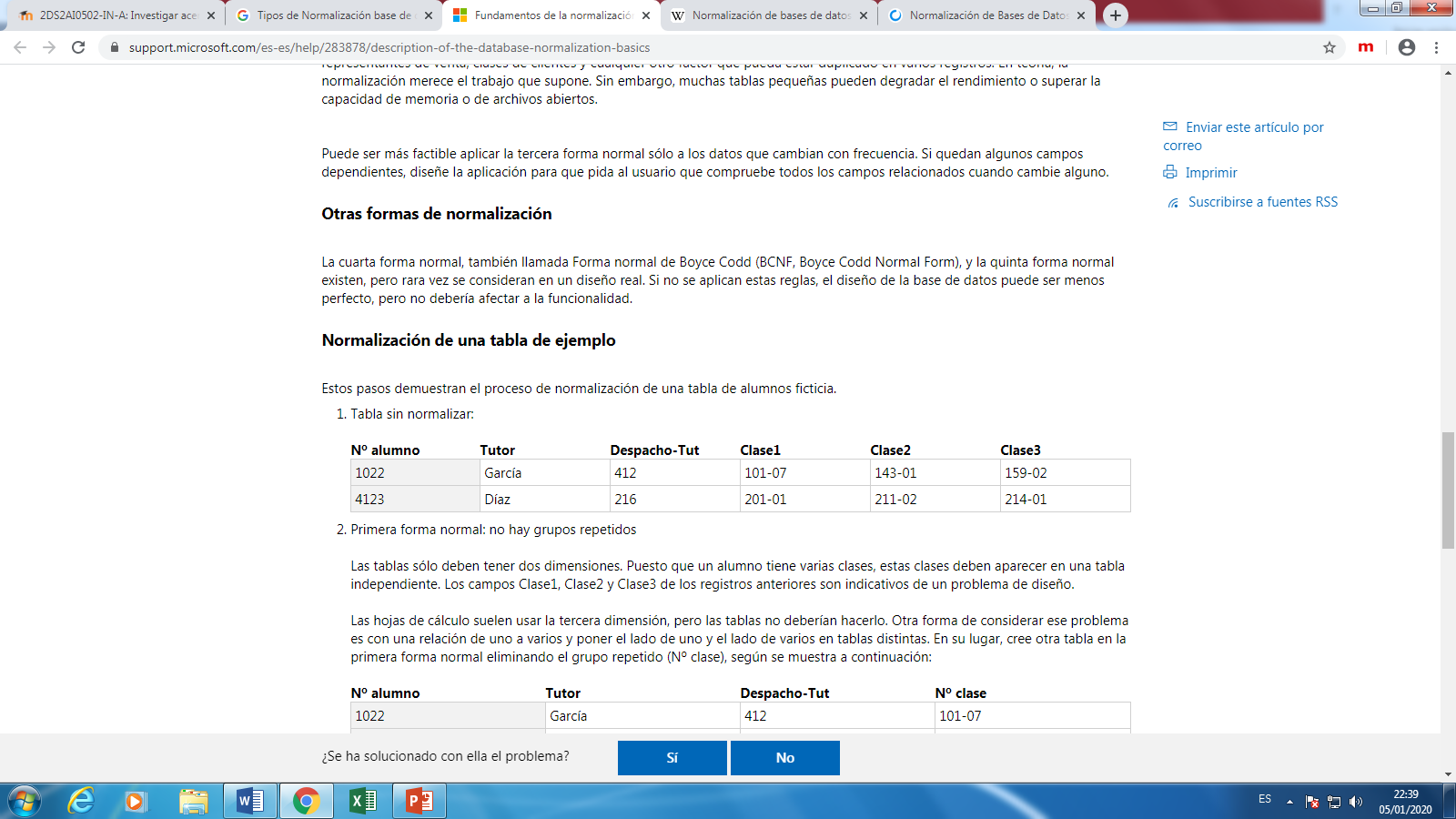
Existen 3 niveles de normalización que deben respetarse para poder decir que nuestra Base de Datos, se encuentra normalizada, es decir, que cumple con los requisitos naturales para funcionar óptimamente y no perjudicar el rendimiento por mala arquitectura.

Estas 3 reglas de Normalización se les conocen como las 3 formas normales.

La cuarta forma normal, también llamada Forma normal de Boyce Codd (BCNF, Boyce Codd Normal Form), y la quinta forma normal existen, pero rara vez se consideran en un diseño real. Si no se aplican estas reglas, el diseño de la base de datos puede ser menos perfecto, pero no debería afectar a la funcionalidad.

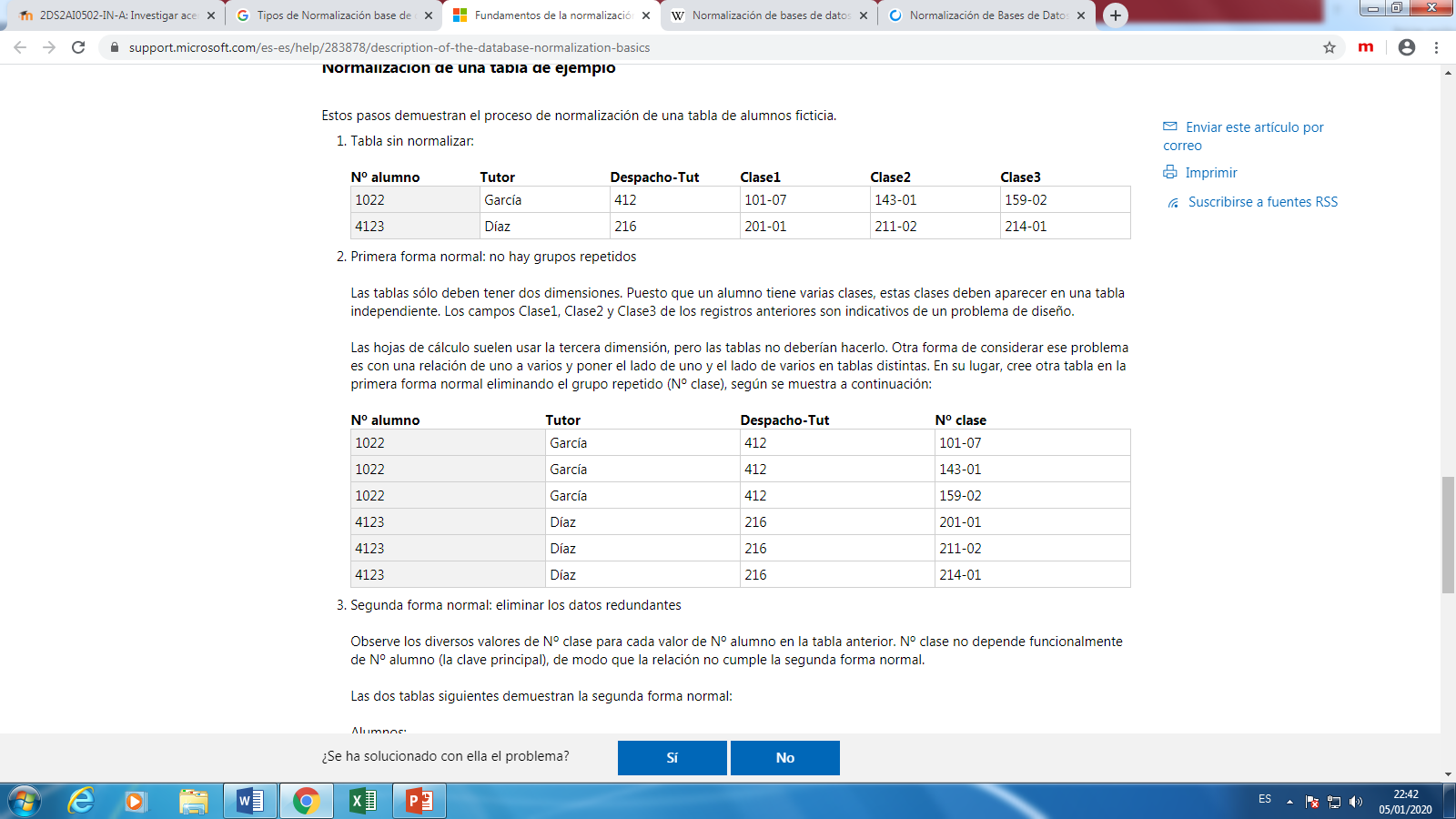
Estos pasos demuestran el proceso de normalización de una tabla de alumnos ficticia.

Tabla sin normalizar:



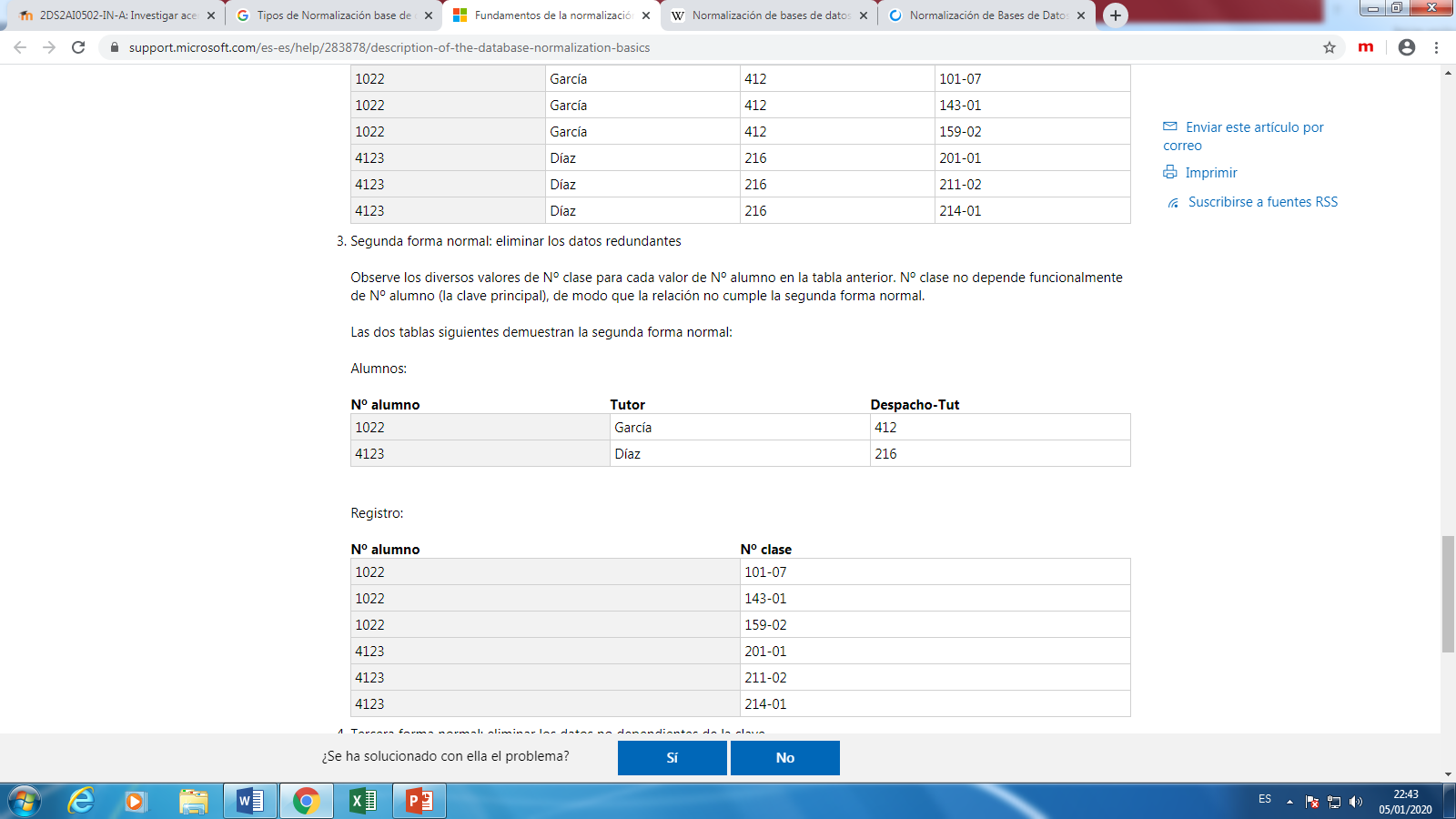
1. Primera forma normal: no hay grupos repetidos

Las tablas sólo deben tener dos dimensiones. Puesto que un alumno tiene varias clases, estas clases deben aparecer en una tabla independiente. Los campos Clase1, Clase2 y Clase3 de los registros anteriores son indicativos de un problema de diseño.  
  
Las hojas de cálculo suelen usar la tercera dimensión, pero las tablas no deberían hacerlo. Otra forma de considerar ese problema es con una relación de uno a varios y poner el lado de uno y el lado de varios en tablas distintas. En su lugar, cree otra tabla en la primera forma normal eliminando el grupo repetido (Nº clase), según se muestra a continuación:

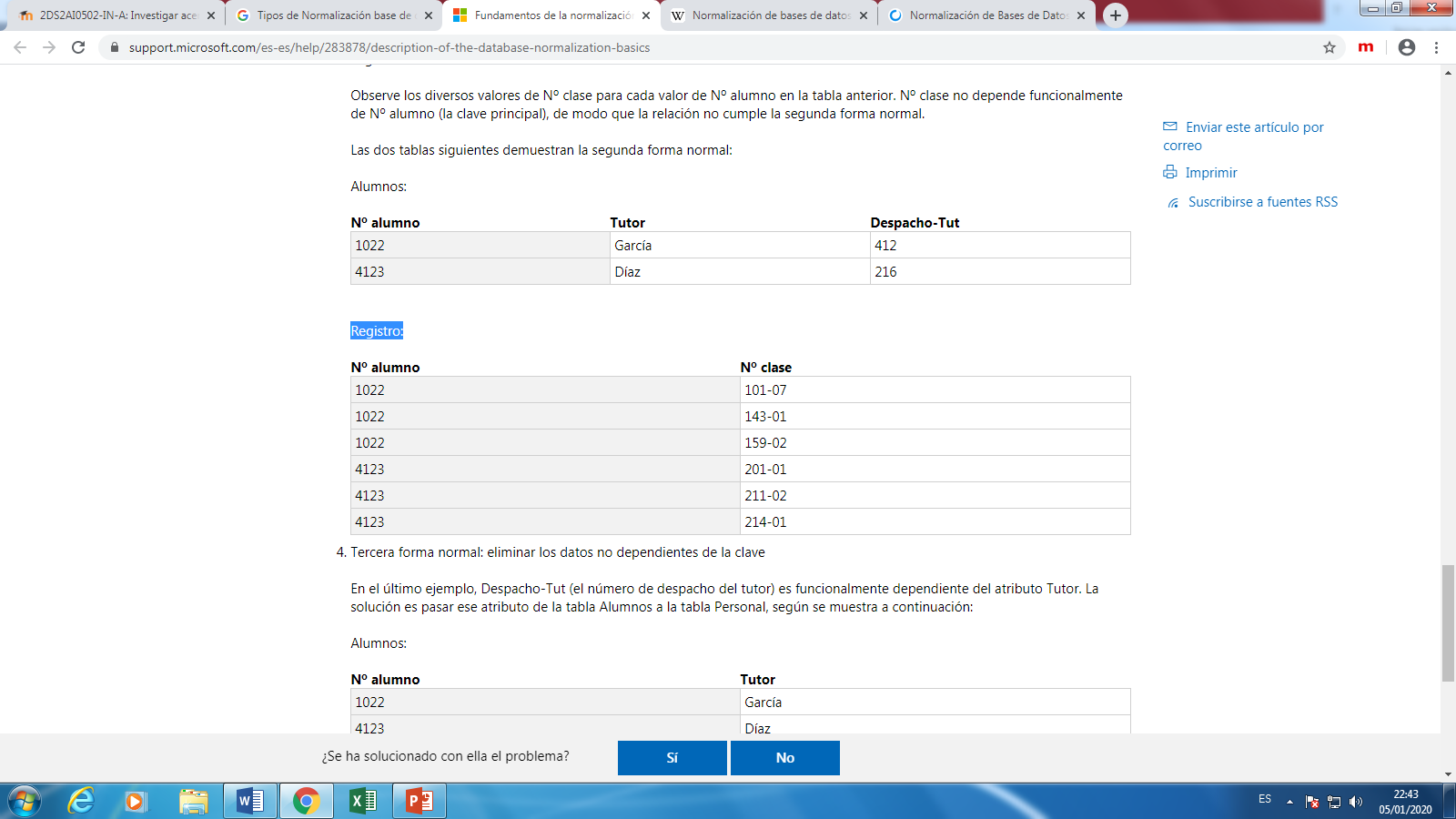


1. Segunda forma normal: eliminar los datos redundantes

Observe los diversos valores de Nº clase para cada valor de Nº alumno en la tabla anterior. Nº clase no depende funcionalmente de Nº alumno (la clave principal), de modo que la relación no cumple la segunda forma normal.  
  
Las dos tablas siguientes demuestran la segunda forma normal:  
  
Alumnos:

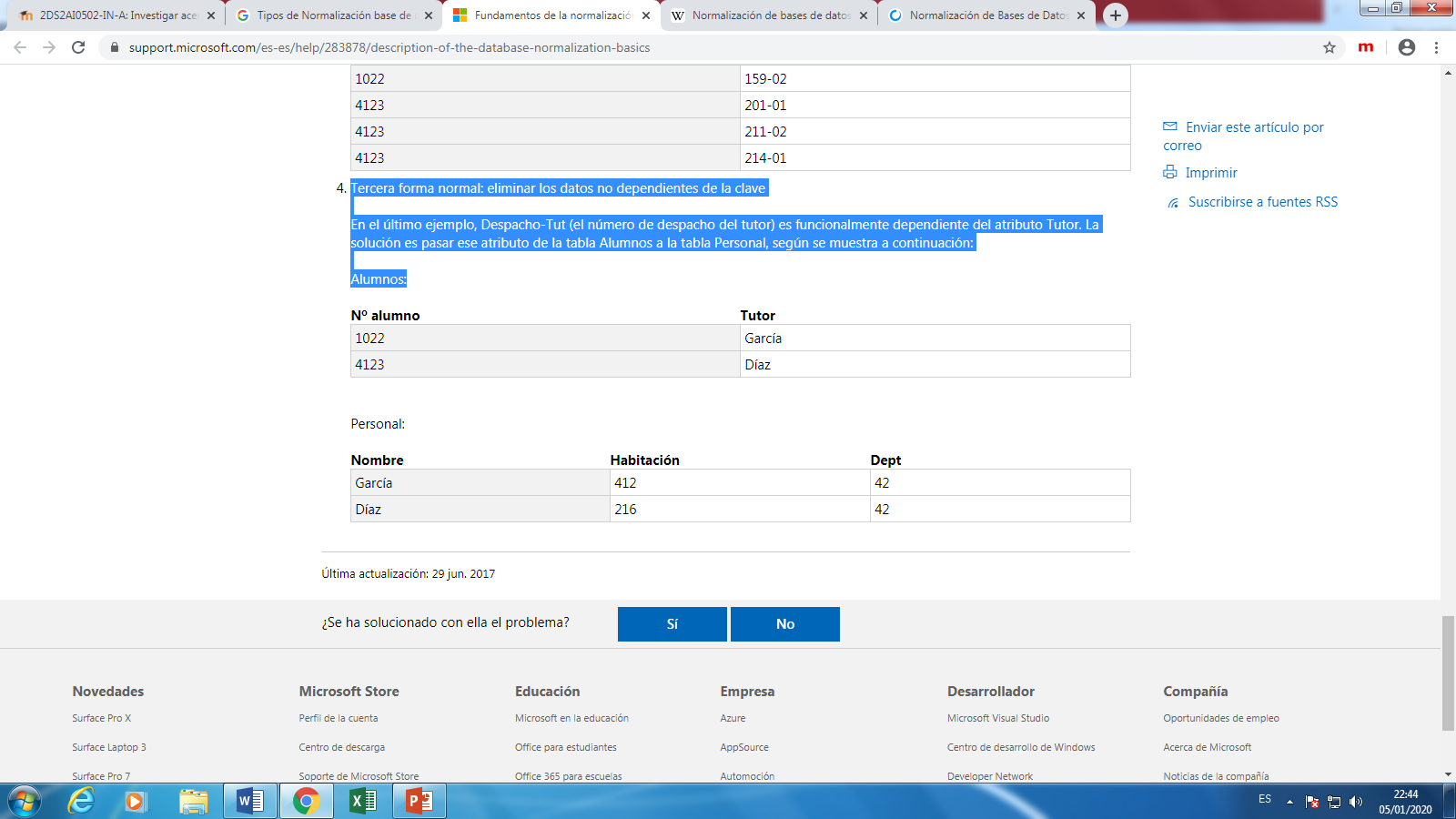


Registro:

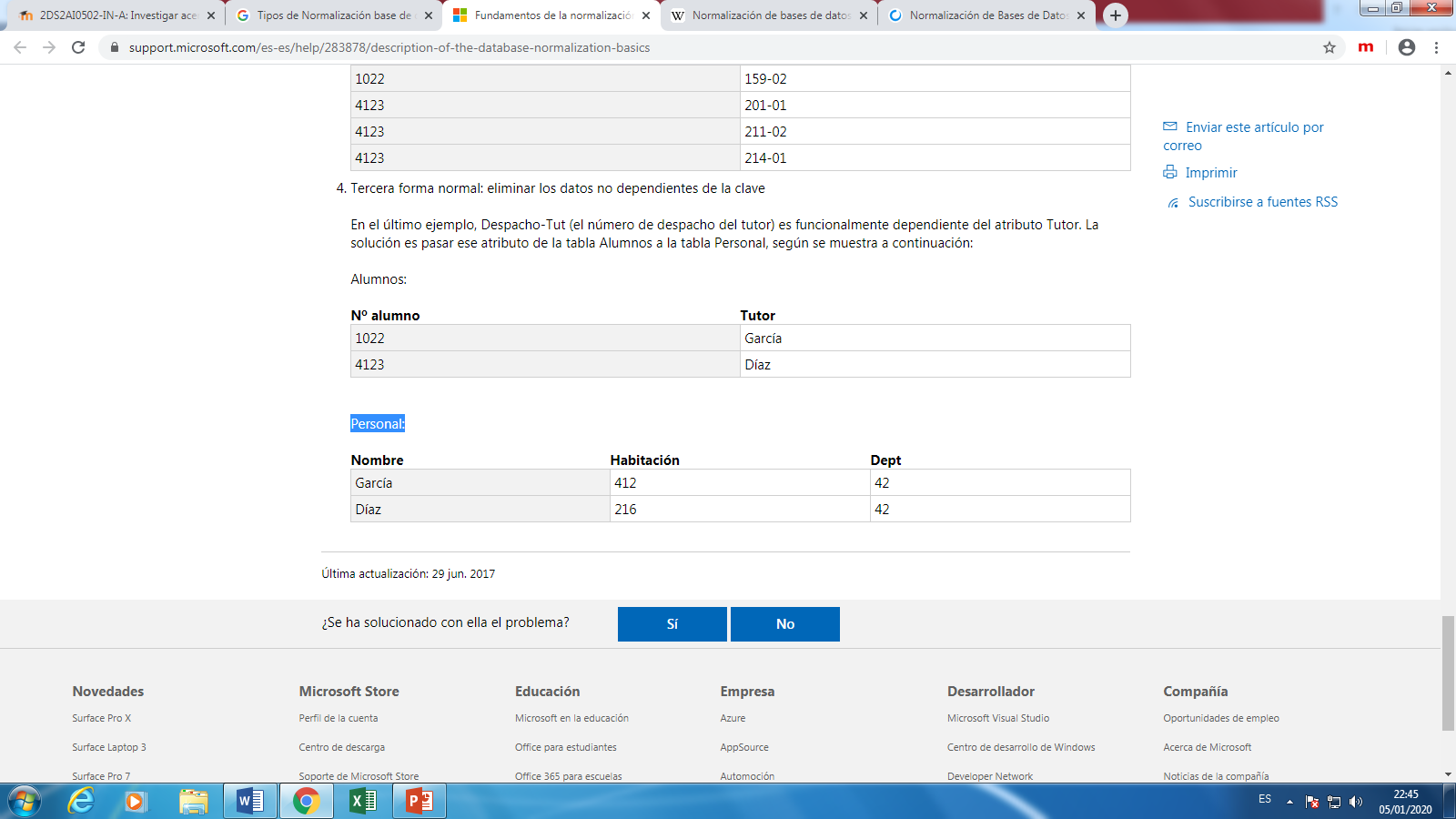


1. Tercera forma normal: eliminar los datos no dependientes de la clave

En el último ejemplo, Despacho-Tut (el número de despacho del tutor) es funcionalmente dependiente del atributo Tutor. La solución es pasar ese atributo de la tabla Alumnos a la tabla Personal, según se muestra a continuación:  
  
Alumnos:



Personal:



## *Investigar la sentencia JOIN*

Los **JOINs** en **SQL** sirven para **combinar filas de dos o más tablas** basándose en un campo común entre ellas, devolviendo por tanto datos de diferentes tablas. Un JOIN se produce cuando dos o más tablas se juntan en una sentencia SQL.

Existen más **tipos de joins en SQL** que los que aquí se explican, como **CROSS JOIN**, O **SELF JOIN**, pero no todos ellos están soportados por todos los **sistemas de bases de datos**. Los más importantes son los siguientes:

1. INNER JOIN: Devuelve **todas las filas** cuando hay al menos **una coincidencia** en **ambas** tablas.
2. LEFT JOIN: Devuelve todas las filas de la tabla de la **izquierda**, y las filas coincidentes de la tabla de la **derecha**.
3. RIGHT JOIN: Devuelve todas las filas de la tabla de la **derecha**, y las filas coincidentes de la tabla de la **izquierda**.
4. OUTER JOIN: Devuelve **todas las filas** de las dos tablas, la izquierda y la derecha. También se llama FULL OUTER JOIN.

### ***INNER JOIN***

**INNER JOIN** selecciona todas las filas de las dos columnas siempre y cuando haya **una coincidencia entre las columnas en ambas tablas**. Es el tipo de JOIN más común.

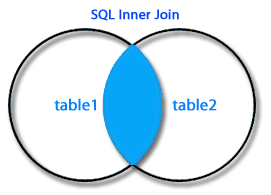
SELECT nombreColumna(s)

FROM tabla1

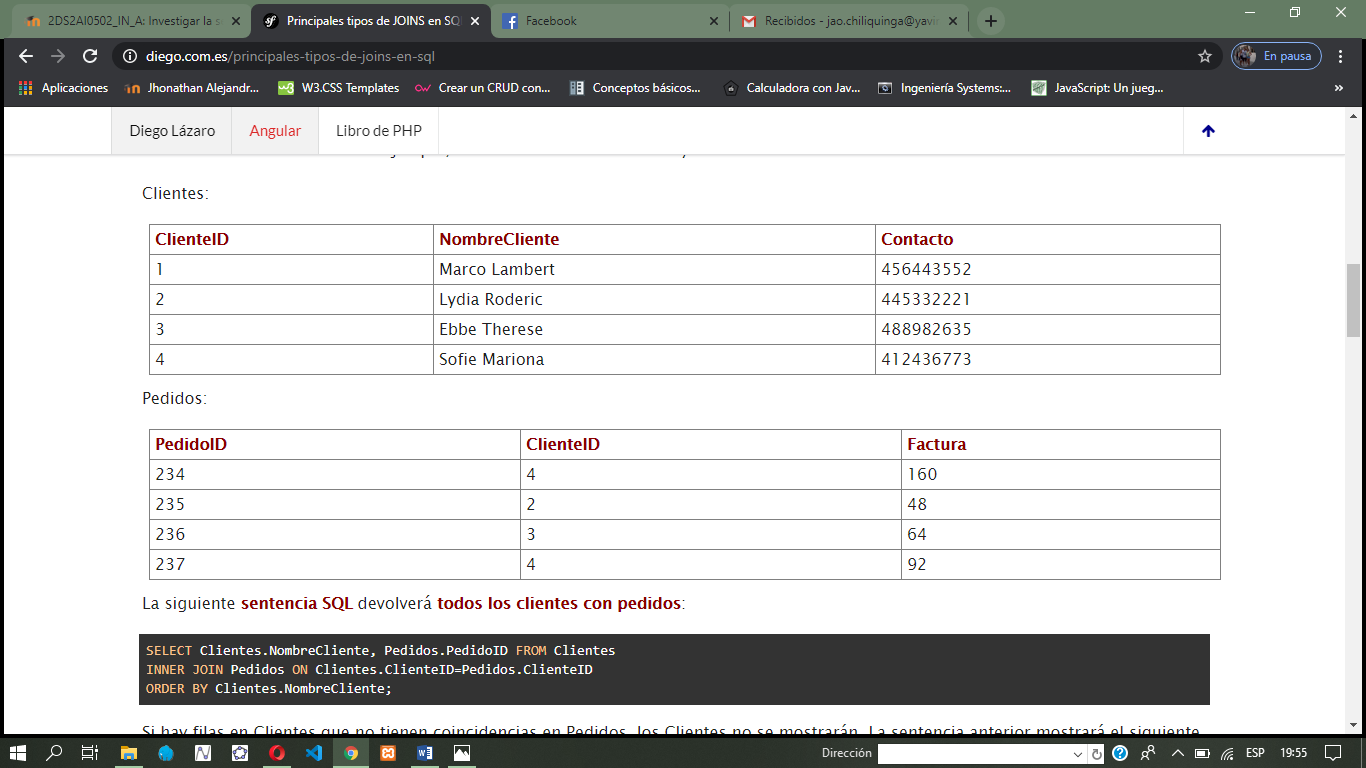
INNER JOIN tabla2

ON tabla1.nombreColumna=table2.nombreColumna;

Se ve más claro utilizando una **imagen**:



Vamos a verlo también con un ejemplo, mediante las tablas **Clientes** y **Pedidos**:



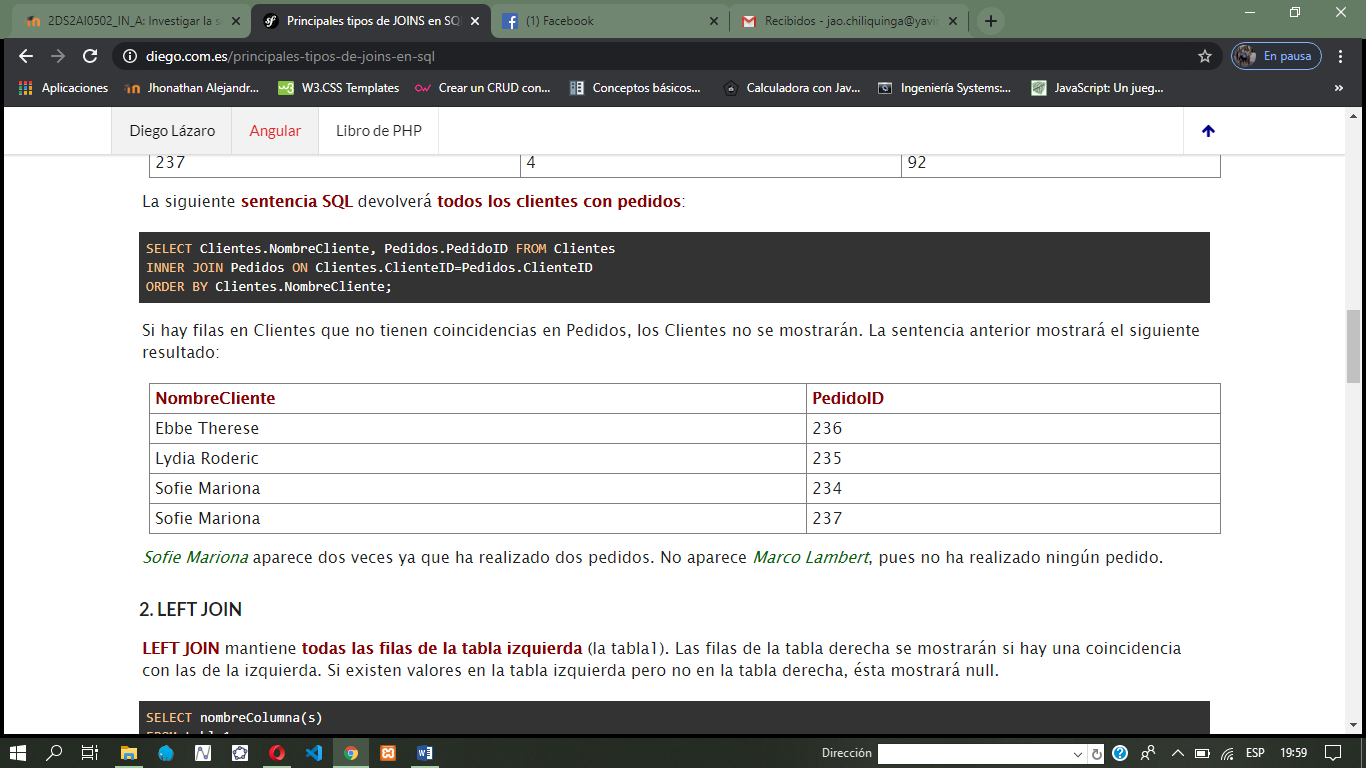
La siguiente **sentencia SQL** devolverá **todos los clientes con pedidos**:

SELECT Clientes.NombreCliente, Pedidos.PedidoID FROM Clientes

INNER JOIN Pedidos ON Clientes.ClienteID=Pedidos.ClienteID

ORDER BY Clientes.NombreCliente;

Si hay filas en Clientes que no tienen coincidencias en Pedidos, los Clientes no se mostrarán. La sentencia anterior mostrará el siguiente resultado:



### ***LEFT JOIN***

**LEFT JOIN** mantiene **todas las filas de la tabla izquierda** (la tabla1). Las filas de la tabla derecha se mostrarán si hay una coincidencia con las de la izquierda. Si existen valores en la tabla izquierda pero no en la tabla derecha, ésta mostrará null.

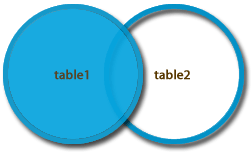
SELECT nombreColumna(s)

FROM tabla1

LEFT JOIN tabla2

ON tabla1.nombreColumna=tabla2.nombreColumna;

La **representación de LEFT JOIN en una imagen** es:



Tomando de nuevo las tablas de **Productos** y **Pedidos**, ahora queremos mostrar **todos los clientes**, **y cualquier pedido** que pudieran haber encargado:

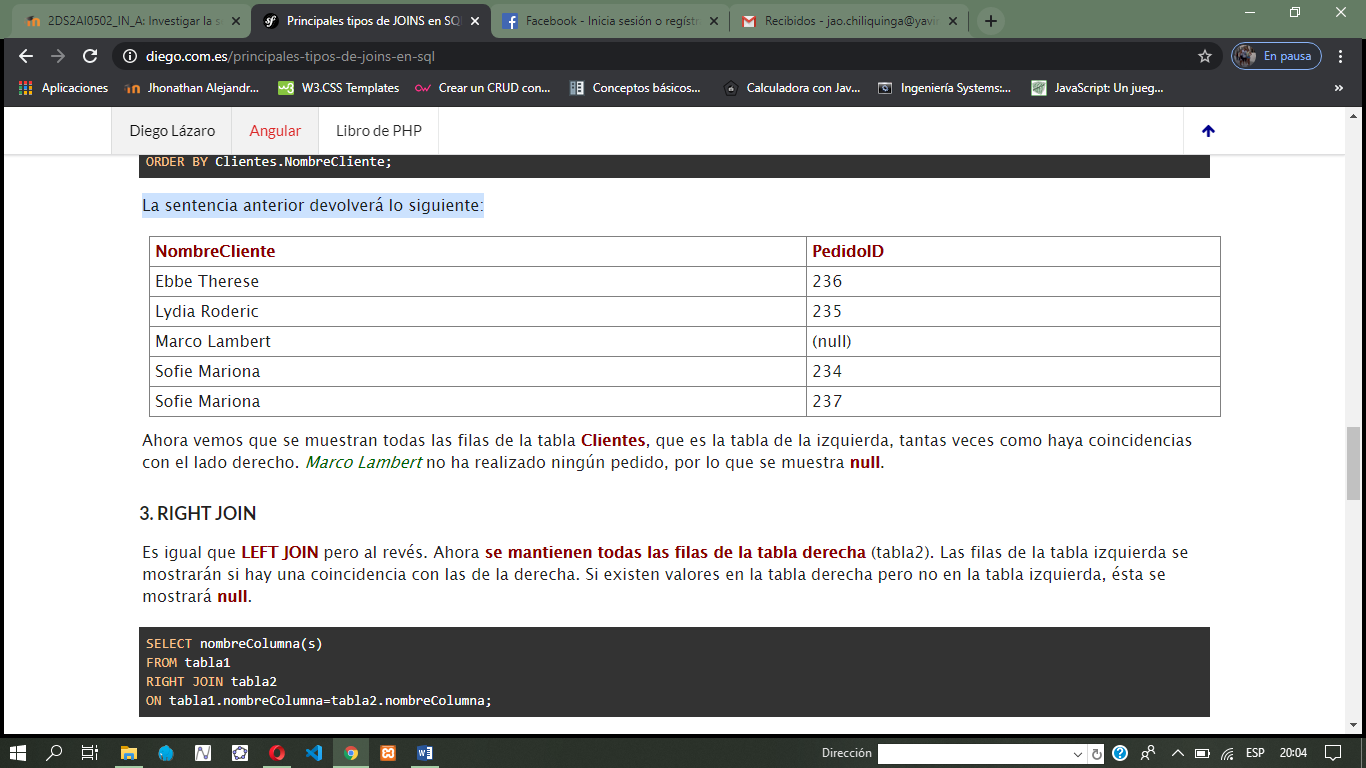
SELECT Clientes.NombreCliente, Pedidos.PedidoID

FROM Clientes LEFT JOIN Pedidos

ON Clientes.ClienteID=Pedidos.ClienteID

ORDER BY Clientes.NombreCliente;

La sentencia anterior devolverá lo siguiente:



### ***RIGHT JOIN***

Es igual que **LEFT JOIN** pero al revés. Ahora **se mantienen todas las filas de la tabla derecha** (tabla2). Las filas de la tabla izquierda se mostrarán si hay una coincidencia con las de la derecha. Si existen valores en la tabla derecha pero no en la tabla izquierda, ésta se mostrará **null**.

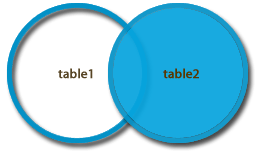
SELECT nombreColumna(s)

FROM tabla1

RIGHT JOIN tabla2

ON tabla1.nombreColumna=tabla2.nombreColumna;

La **imagen que representa a RIGHT JOIN** es:



De nuevo tomamos el ejemplo de **Clientes** y **Pedidos**, y vamos a hacer el mismo ejemplo anterior, pero cambiado LEFT por RIGHT:

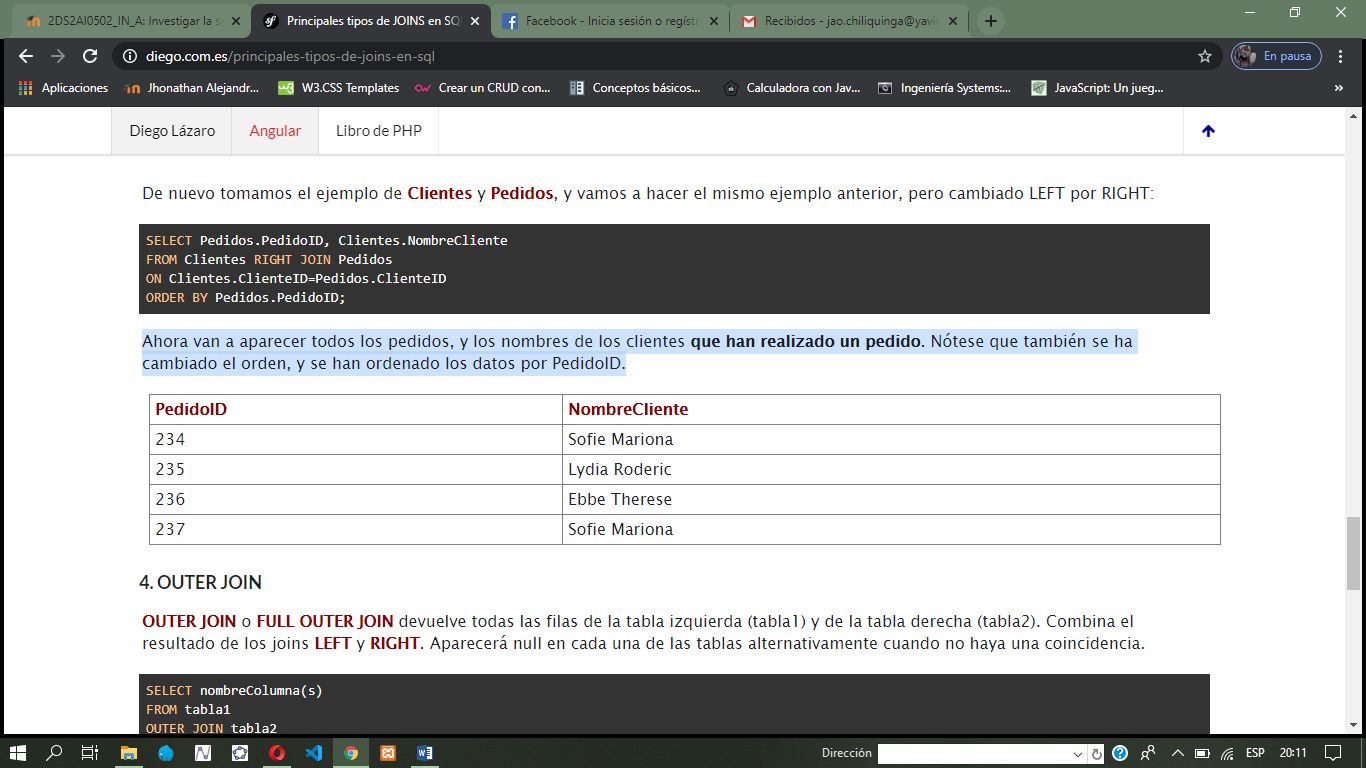
SELECT Pedidos.PedidoID, Clientes.NombreCliente

FROM Clientes RIGHT JOIN Pedidos

ON Clientes.ClienteID=Pedidos.ClienteID

ORDER BY Pedidos.PedidoID;

Ahora van a aparecer todos los pedidos, y los nombres de los clientes **que han realizado un pedido**. Nótese que también se ha cambiado el orden, y se han ordenado los datos por PedidoID.



### ***OUTER JOIN***

**OUTER JOIN** o **FULL OUTER JOIN** devuelve todas las filas de la tabla izquierda (tabla1) y de la tabla derecha (tabla2). Combina el resultado de los joins **LEFT** y **RIGHT**. Aparecerá null en cada una de las tablas alternativamente cuando no haya una coincidencia.

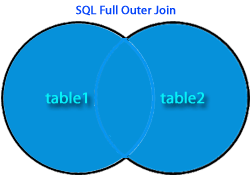
SELECT nombreColumna(s)

FROM tabla1

OUTER JOIN tabla2

ON tabla1.nombreColumna=tabla2.nombreColumna;

La **imagen que representa el OUTER JOIN** es la siguiente:



Vamos a obtener **todas las filas** de las tablas **Clientes** y **Pedidos**:

SELECT Clientes.NombreCliente, Pedidos.PedidoID

FROM Clientes OUTER JOIN Pedidos

ON Clientes.ClienteID=Pedidos.ClienteID

ORDER BY Clientes.NombreCliente;

La sentencia devolverá **todos los Clientes** y **todos los Pedidos**, si un cliente no tiene pedidos mostrará **null** en **PedidoID**, y si un pedido no tuviera un cliente mostraría **null** en **NombreCliente** (en este ejemplo no sería lógico que un Pedido no tuviera un cliente).

La sintaxis de **OUTER JOIN o FULL OUTER JOIN no existen en MySQL**, pero se puede conseguir el mismo resultado de diferentes formas, esta es una:

SELECT Clientes.NombreCliente, Pedidos.PedidoID

FROM Clientes

LEFT JOIN Pedidos ON Clientes.ClienteID=Pedidos.ClienteID

UNION

SELECT Clientes.NombreCliente, Pedidos.PedidoID

FROM Clientes

RIGHT JOIN Pedidos ON Clientes.ClienteID=Pedidos.ClienteID;

## *Ejercicio en clase: Curso de formación (solo DDL)*

Subir un archivo de texto con las instrucciones para crear las tablas del ejercicio planteado en clase.

1\_create database cursos,

2\_create table empleado,

3\_ create table empleado(id\_Empleado serial,

Nombre character(30),

Apellido character(20),

Telefono varchar(15),

Sueldo numeric,

constraint "pkEmpleado" primary key(id\_Empleado));

select \* from empleado

4\_como subir un archivo en exel

COPY PUBLIC.empleado from 'G:\archivos\empleado.csv' DELIMITER ',' CSV HEADER;

5\_create table cursos

create table Cursos(

id\_Curso serial,

Nombre character(30),

Num\_hora numeric,

Costo\_curso numeric,

constraint "pkCursos" primary key(id\_Curso)

)

select \* from cursos

insert into cursos (nombre,num\_hora, costo\_curso) values ('marketin',100,300),

('cosina',200,500),

('informatica',50,200)

6\_ create table Ciudad(

id\_Ciudad serial,

Nombre character(30),

constraint "pkCiudad" primary key(id\_Ciudad)

)

select \* from ciudad

insert into ciudad (nombre) values ('quito'),('ambato'),('loja'),('guayaquil')

7\_ create table Edicion(

id\_edicion serial,

fecha\_ini character(20),

fecha\_fin character(20),

id\_ciudad integer,

constraint "pkIdEdicion" primary key(id\_edicion),

constraint "fkIdCiudad" foreign key (id\_Ciudad) references Ciudad(id\_ciudad)

);

select \* from Edicion

insert into Edicion(fecha\_ini,fecha\_fin,id\_ciudad) values('1/20/2020','2/20/2020',2),

('1/30/2020','2/20/2020',3),('3/20/2020','4/30/2020',4)

8\_ create table Curso\_edicion(

idcurso\_edicion serial,

id\_Edicion integer,

id\_Curso integer,

id\_empleadoProf integer,

fecha\_inicio character(20),

fecha\_fin character(20),

constraint "pkIdCurso\_edicion" primary key (idcurso\_edicion),

constraint "fkIdEdicion" foreign key (id\_Edicion) references Edicion (id\_Edicion),

constraint "fkCursos" foreign key (id\_Curso) references Cursos (id\_Curso),

constraint "fkid\_EmpleadoProf" foreign key (id\_EmpleadoProf) references Empleado(id\_Empleado)

);

select \* from curso\_edicion

drop table curso\_edicion

COPY PUBLIC.curso\_edicion from 'G:\archivos\cursos.csv' DELIMITER ',' CSV HEADER;

9\_ create table Asistente(

idAsitente serial,

idCurso\_edicion integer,

id\_Empleado integer,

constraint "pkIdAsitente" primary key (idAsitente),

constraint "fkIdCurso\_edicion" foreign key (idcurso\_edicion) references Curso\_edicion(idcurso\_edicion),

constraint "fkIdEmpleado" foreign key (id\_Empleado) references Empleado(id\_Empleado)

);

select \* from Asistente

COPY PUBLIC.asistente from 'G:\archivos\asitente.csv' DELIMITER ';' CSV HEADER;

## *Curso de formación (solo DML)*

Creación de una base de datos con archivos planos

1\_create database cursos,

2\_create table empleado,

3\_ create table empleado(id\_Empleado serial,

Nombre character(30),

Apellido character(20),

Telefono varchar(15),

Sueldo numeric,

constraint "pkEmpleado" primary key(id\_Empleado));

select \* from empleado

4\_como subir un archivo en exel

COPY PUBLIC.empleado from 'G:\archivos\empleado.csv' DELIMITER ',' CSV HEADER;

5\_create table cursos

create table Cursos(

id\_Curso serial,

Nombre character(30),

Num\_hora numeric,

Costo\_curso numeric,

constraint "pkCursos" primary key(id\_Curso)

)

select \* from cursos

insert into cursos (nombre,num\_hora, costo\_curso) values ('marketin',100,300),

('cosina',200,500),

('informatica',50,200)

6\_ create table Ciudad(

id\_Ciudad serial,

Nombre character(30),

constraint "pkCiudad" primary key(id\_Ciudad)

)

select \* from ciudad

insert into ciudad (nombre) values ('quito'),('ambato'),('loja'),('guayaquil')

7\_ create table Edicion(

id\_edicion serial,

fecha\_ini character(20),

fecha\_fin character(20),

id\_ciudad integer,

constraint "pkIdEdicion" primary key(id\_edicion),

constraint "fkIdCiudad" foreign key (id\_Ciudad) references Ciudad(id\_ciudad)

);

select \* from Edicion

insert into Edicion(fecha\_ini,fecha\_fin,id\_ciudad) values('1/20/2020','2/20/2020',2),

('1/30/2020','2/20/2020',3),('3/20/2020','4/30/2020',4)

8\_ create table Curso\_edicion(

idcurso\_edicion serial,

id\_Edicion integer,

id\_Curso integer,

id\_empleadoProf integer,

fecha\_inicio character(20),

fecha\_fin character(20),

constraint "pkIdCurso\_edicion" primary key (idcurso\_edicion),

constraint "fkIdEdicion" foreign key (id\_Edicion) references Edicion (id\_Edicion),

constraint "fkCursos" foreign key (id\_Curso) references Cursos (id\_Curso),

constraint "fkid\_EmpleadoProf" foreign key (id\_EmpleadoProf) references Empleado(id\_Empleado)

);

select \* from curso\_edicion

drop table curso\_edicion

COPY PUBLIC.curso\_edicion from 'G:\archivos\cursos.csv' DELIMITER ',' CSV HEADER;

9\_ create table Asistente(

idAsitente serial,

idCurso\_edicion integer,

id\_Empleado integer,

constraint "pkIdAsitente" primary key (idAsitente),

constraint "fkIdCurso\_edicion" foreign key (idcurso\_edicion) references Curso\_edicion(idcurso\_edicion),

constraint "fkIdEmpleado" foreign key (id\_Empleado) references Empleado(id\_Empleado)

);

select \* from Asistente

COPY PUBLIC.asistente from 'G:\archivos\asitente.csv' DELIMITER ';' CSV HEADER;

## *Consultas SQL: Curso de formación*

/\* 1.seleccionar y ordenar por apellido los empleados que asistieron al curso 2 \*/

select em.apellido, st.idasitente, cu.nombre

from empleado as em, asistente as st, cursos as cu

where em.id\_empleado = st.id\_empleado and cu.id\_curso = 2 ORDER BY em.apellido

/\* 2. mostrar el telefono, codigo del empleado cuyo id = 18 o id = 2 \*/

select em.telefono, em.id\_empleado

from empleado as em

where em.id\_empleado = 18

/\* 3.Listar todos los sueldos de mayor a menor \*/

select em.sueldo

from empleado as em order by sueldo desc

/\* 4.Mostrar la ediciones que se realizaron en quito y guayaquil \*/

select ed.id\_edicion, ci.nombre, ced.idcurso\_edicion

from edicion as ed, ciudad as ci, curso\_edicion as ced

where ed.id\_edicion = ced.idcurso\_edicion and ci.id\_ciudad = ced.idcurso\_edicion and ci.id\_ciudad =4

/\*Listar el id empleados con sus salarios \*/

select r.sueldo , r.id\_empleado

from empleado as r

/\* 6.Mostrar apellidos y nombre de los empleados que han sido (tutores) alfabéticamente \*/

select r.apellido , r.nombre, cd.id\_empleadoprof

from empleado as r, curso\_edicion as cd

where r.id\_empleado = cd.id\_empleadoprof

/\* 7.En que ciudad se dicto la edicion 2 \*/

select ci.nombre, ed.id\_edicion

from ciudad as ci, edicion as ed

where ci.id\_ciudad = ed.id\_ciudad and ed.id\_edicion =2

/\* 8.Mostrar de los asistentes registrados en la edicion 3 \*/

select asi.idasitente, edi.id\_edicion

from asistente as asi, edicion as edi

where asi.idasitente = asi.idasitente and edi.id\_edicion =3

***Instrucciones sql***

* **INSERT**: Inserta filas en una tabla. Se corresponde con la “C” de CRUD.
* **SELECT**: muestra información sobre los datos almacenados en la base de datos. Dicha información puede pertenecer a una o varias tablas. Es la “R”.
* **UPDATE**: Actualiza información de una tabla. Es, obviamente, la “U”.
* **DELETE**: Borra filas de una tabla. Se corresponde con la “D”.

### SELECT

Permite seleccionar las columnas que se van a mostrar y en el orden en que lo van a hacer. Simplemente es la instrucción que la base de datos interpreta como que vamos a solicitar información.

### ALL / DISTINCT

ALL es el valor predeterminado, especifica que el conjunto de resultados puede incluir filas duplicadas. Por regla general nunca se utiliza.

DISTINCT especifica que el conjunto de resultados sólo puede incluir filas únicas. Es decir, si al realizar una consulta hay registros exactamente iguales que aparecen más de una vez, éstos se eliminan. Muy útil en muchas ocasiones.

### Nombres de campos

Se debe especificar una lista de nombres de campos de la tabla que nos interesan y que por tanto queremos devolver. Normalmente habrá más de uno, en cuyo caso separamos cada nombre de los demás mediante comas.

Se puede anteponer el nombre de la tabla al nombre de las columnas, utilizando el formato Tabla.Columna. Además de nombres de columnas, en esta lista se pueden poner constantes, expresiones aritméticas, y funciones, para obtener campos calculados de manera dinámica.

Si queremos que nos devuelva todos los campos de la tabla utilizamos el comodín “\*” (asterisco).

Los nombres indicados deben coincidir exactamente con los nombre de los campos de la tabla, pero si queremos que en nuestra tabla lógica de resultados tengan un nombre diferente podemos utilizar:

### AS

Permite renombrar columnas si lo utilizamos en la cláusula SELECT, o renombrar tablas si lo utilizamos en la cláusula FROM. Es opcional. Con ello podremos crear diversos alias de columnas y tablas. Enseguida veremos un ejemplo.

### FROM

Esta cláusula permite indicar las tablas o vistas de las cuales vamos a obtener la información. De momento veremos ejemplos para obtener información de una sola tabla.

Como se ha indicado anteriormente, también se pueden renombrar las tablas usando la instrucción “AS”.

### WHERE

Especifica la **condición de filtro**de las filas devueltas. Se utiliza cuando no se desea que se devuelvan todas las filas de una tabla, sino sólo las que cumplen ciertas condiciones. Lo habitual es utilizar esta cláusula en la mayoría de las consultas.

### Condiciones

Son **expresiones lógicas**a comprobar para la condición de filtro, que tras su resolución devuelven para cada fila TRUE o FALSE, en función de que se cumplan o no. Se puede utilizar cualquier expresión lógica y en ella utilizar **diversos operadores** como:

* **>** (Mayor)
* **>=** (Mayor o igual)
* **<** (Menor)
* **<=** (Menor o igual)
* **=** (Igual)
* **<>** o **!=** (Distinto)
* IS [NOT] NULL (para comprobar si el valor de una columna es o no es nula, es decir, si contiene o no contiene algún valor).
* **LIKE**: para la comparación de un modelo. Para ello utiliza los caracteres comodín especiales: “%” y “\_”. Con el primero indicamos que en su lugar puede ir cualquier cadena de caracteres, y con el segundo que puede ir cualquier carácter individual (un solo carácter). Con la combinación de estos caracteres podremos obtener múltiples patrones de búsqueda. Por ejemplo:
  + El nombre empieza por A: Nombre LIKE ‘A%’
  + El nombre acaba por A: Nombre LIKE ‘%A’
  + El nombre contiene la letra A: Nombre LIKE ‘%A%’
  + El nombre empieza por A y después contiene un solo carácter cualquiera: Nombre LIKE ‘A\_’
  + El nombre empieza una A, después cualquier carácter, luego una E y al final cualquier cadena de caracteres: Nombre LIKE ‘A\_E%’
* **BETWEEN**: para un intervalo de valores. Por ejemplo:
  + Clientes entre el 30 y el 100: CodCliente BETWEEN 30 AND 100
  + Clientes nacidos entre 1970 y 1979: FechaNac BETWEEN ‘19700101’ AND ‘19791231’
* **IN( )**: para especificar una relación de valores concretos. Por ejemplo: Ventas de los Clientes 10, 15, 30 y 75: CodCliente IN(10, 15, 30, 75)

### ORDER BY

Define el orden de las filas del conjunto de resultados. Se especifica el campo o campos (separados por comas) por los cuales queremos ordenar los resultados.

### ASC / DESC

ASC es el valor predeterminado, especifica que la columna indicad en la cláusula ORDER BY se ordenará de forma ascendente, o sea, de menor a mayor. Si por el contrario se especifica DESC se ordenará de forma descendente (de mayor a menor).

Indicaciones para crear una base de datos con la herramienta PGADMIN y sus instrucciones SQL para modelar.

Crear database nombre de la base de datos

Crear table nombre de la tabla (código de la tabla, campos que necesitemos en la tabla);

Insert into nombre de la tabla (campos) values (valores para el campo);

Si las tablas que crear tienen entidades de muchos a muchos se rompe la relación y que baja a la tabla nueva transformándola en forieng key.

Consultas

Select \* from nombre de la tabla

Select \* from nombre de la tabla where compara las tablas donde se encuentren el a id correspondiente

Select los campos que se desee sacara de las base de datos con los alias

From nombres de las tablas que necesitemos con sus alias

Where comparamos los id en todas las tablas con sus alias

***Otra formas de hacer consultas***

Usar SELECT para recuperar filas y columnas

USE AdventureWorks2012;

SELECT \*

FROM Production.Product

ORDER BY Name ASC;

SELECT p.\*

FROM Production.Product AS p

ORDER BY Name ASC;

Este ejemplo devuelve todas las filas (no se donde se especifica la cláusula), y sólo un subconjunto de las columnas ( Name, ProductNumber, ListPrice) de la Product

SELECT Name, ProductNumber, ListPrice AS Price

FROM Production.Product

ORDER BY Name ASC;

Este ejemplo devuelve solo las filas para las Productque tiene una línea de productos Ry que tienen días para fabricar que es menor que 4.

SELECT Name, ProductNumber, ListPrice AS Price

FROM Production.Product

WHERE ProductLine = 'R'

AND DaysToManufacture < 4

ORDER BY Name ASC;

***Uso de SELECT con encabezados de columna y cálculos***

Los siguientes ejemplos devuelven todas las filas de la Producttabla. El primer ejemplo devuelve las ventas totales y los descuentos para cada producto. En el segundo ejemplo, los ingresos totales se calculan para cada producto.

SELECT p.Name AS ProductName,

NonDiscountSales = (OrderQty \* UnitPrice),

Discounts = ((OrderQty \* UnitPrice) \* UnitPriceDiscount)

FROM Production.Product AS p

INNER JOIN Sales.SalesOrderDetail AS sod

ON p.ProductID = sod.ProductID

ORDER BY ProductName DESC;

Esta es la consulta que calcula los ingresos de cada producto en cada pedido de ventas.

SELECT 'Total income is', ((OrderQty \* UnitPrice) \* (1.0 - UnitPriceDiscount)), ' for ',

p.Name AS ProductName

FROM Production.Product AS p

INNER JOIN Sales.SalesOrderDetail AS sod

ON p.ProductID = sod.ProductID

ORDER BY ProductName ASC;

***Uso de DISTINCT con SELECT***

El siguiente ejemplo se utiliza DISTINCT para evitar la recuperación de títulos duplicados.

SELECT DISTINCT JobTitle

FROM HumanResources.Employee

ORDER BY JobTitle;

***Crear tablas con SELECT INTO***

El siguiente primer ejemplo crea una tabla temporal nombrada #Bicyclesen tempdb.

USE tempdb;

IF OBJECT\_ID (N'#Bicycles',N'U') IS NOT NULL

DROP TABLE #Bicycles;

SELECT \*

INTO #Bicycles

FROM AdventureWorks2012.Production.Product

WHERE ProductNumber LIKE 'BK%';

Este segundo ejemplo crea la tabla permanente NewProducts.

USE AdventureWorks2012;

IF OBJECT\_ID('dbo.NewProducts', 'U') IS NOT NULL

DROP TABLE dbo.NewProducts;

ALTER DATABASE AdventureWorks2012 SET RECOVERY BULK\_LOGGED;

SELECT \* INTO dbo.NewProducts

FROM Production.Product

WHERE ListPrice > $25

AND ListPrice < $100;

ALTER DATABASE AdventureWorks2012 SET RECOVERY FULL;

***Uso de subconsultas correlacionadas***

USE AdventureWorks2012;

SELECT DISTINCT Name

FROM Production.Product AS p

WHERE EXISTS

(SELECT \*

FROM Production.ProductModel AS pm

WHERE p.ProductModelID = pm.ProductModelID

AND pm.Name LIKE 'Long-Sleeve Logo Jersey%');

USE AdventureWorks2012;

SELECT DISTINCT Name

FROM Production.Product

WHERE ProductModelID IN

(SELECT ProductModelID

FROM Production.ProductModel AS pm

WHERE p.ProductModelID = pm.ProductModelID

AND Name LIKE 'Long-Sleeve Logo Jersey%');

El siguiente ejemplo usa INy recupera una instancia del nombre y apellido de cada empleado para el cual es el bono en la SalesPersontabla 5000.00y para el cual los números de identificación del empleado coinciden en las tablas Employeey SalesPerson

SELECT DISTINCT p.LastName, p.FirstName

FROM Person.Person AS p

JOIN HumanResources.Employee AS e

ON e.BusinessEntityID = p.BusinessEntityID WHERE 5000.00 IN

(SELECT Bonus

FROM Sales.SalesPerson AS sp

WHERE e.BusinessEntityID = sp.BusinessEntityID);

Una subconsulta correlacionada también se puede usar en la HAVINGcláusula de una consulta externa. Este ejemplo encuentra los modelos de productos para los cuales el precio de lista máximo es más del doble del promedio del modelo

SELECT p1.ProductModelID

FROM Production.Product AS p1

GROUP BY p1.ProductModelID

HAVING MAX(p1.ListPrice) >=

(SELECT AVG(p2.ListPrice) \* 2

FROM Production.Product AS p2

WHERE p1.ProductModelID = p2.ProductModelID);

Este ejemplo utiliza dos subconsultas correlacionadas para encontrar los nombres de los empleados que han vendido un producto en particular.

SELECT DISTINCT pp.LastName, pp.FirstName

FROM Person.Person pp JOIN HumanResources.Employee e

ON e.BusinessEntityID = pp.BusinessEntityID WHERE pp.BusinessEntityID IN

(SELECT SalesPersonID

FROM Sales.SalesOrderHeader

WHERE SalesOrderID IN

(SELECT SalesOrderID

FROM Sales.SalesOrderDetail

WHERE ProductID IN

(SELECT ProductID

FROM Production.Product p

WHERE ProductNumber = 'BK-M68B-42')));

***Uso de GROUP BY***

El siguiente ejemplo encuentra el total de cada pedido de ventas en la base de datos.

SELECT SalesOrderID, SUM(LineTotal) AS SubTotal

FROM Sales.SalesOrderDetail

GROUP BY SalesOrderID

ORDER BY SalesOrderID;

***Usar GROUP BY con múltiples grupos***

El siguiente ejemplo encuentra el precio promedio y la suma de las ventas del año hasta la fecha, agrupadas por ID de producto e ID de oferta especial.

SELECT ProductID, SpecialOfferID, AVG(UnitPrice) AS [Average Price],

SUM(LineTotal) AS SubTotal

FROM Sales.SalesOrderDetail

GROUP BY ProductID, SpecialOfferID

ORDER BY ProductID;

***Uso de GROUP BY y WHERE***

El siguiente ejemplo pone los resultados en grupos después de recuperar solo las filas con precios de lista mayores que $1000.

SELECT ProductModelID, AVG(ListPrice) AS [Average List Price]

FROM Production.Product

WHERE ListPrice > $1000

GROUP BY ProductModelID

ORDER BY ProductModelID;