Práctica 2 Parte 2: Integración con Python

viu Universidad Internacional de Valencia

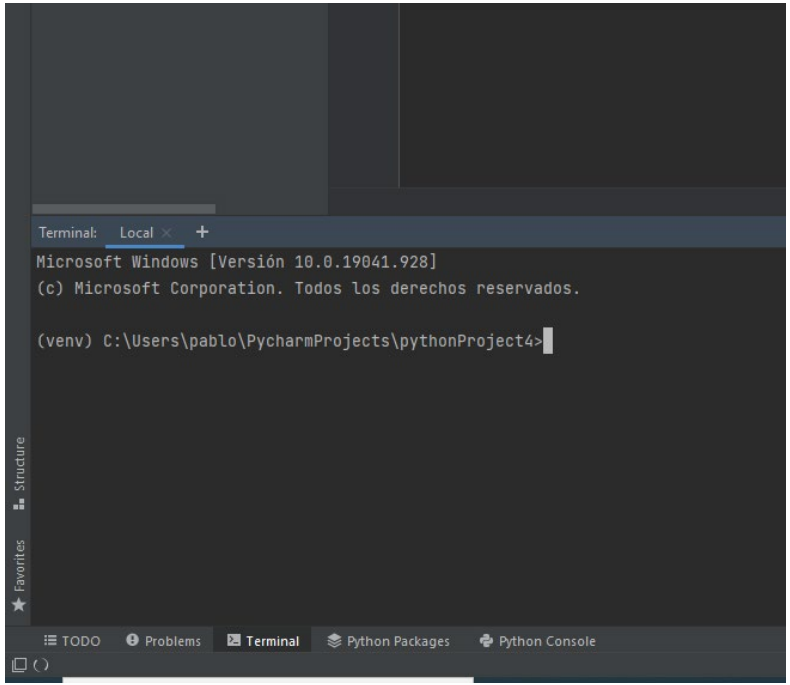
07.11.23

- Los datos que se introducen, se modifican, se borran o se consultan en una BBDD se hacen normalmente a través de aplicaciones cliente.
- En Cassandra esto es aun más relevante pues el propio diseño de las tablas está fuertemente ligado a como la base de datos trabajará con ellas.
- Utilizaremos el lenguaje Python para realizar estos programas y el IDE Pycharm, aunque esto último no es obligatorio.
- En esta práctica veremos una metodología básica de manipulación de datos así como de presentación de la información.

Descargar Pycharm https://www.jetbrains.com/es-es/pycharm/download

1. Pulsar en Terminal como se muestra en la imagen
2. Introducir el siguiente comando:

   pip install cassandra-driver
3. Pulsa en Python Packages y deberías ver el paquete cassandra-driver instalado



- Para importa la librería necesaria:

```
from cassandra.cluster import Cluster
```

- Para conectarnos a cassandra ejecutamos el siguiente código

```
cluster = Cluster()
session = cluster.connect('nombreKeyspace')
```

- En el paréntesis sustituir ‘nombreKeyspace’ por el nombre del Keyspace con el que se quiera trabajar
- Al finalizar la aplicación no nos olvidemos de cerrar la sesión

```
session.shutdown()
```

- Conexión con direcciones ip diferentes a localhost:

```
#cluster = Cluster(['192.168.0.1', '192.168.0.2'], port=..., ssl_context=...)
```

Metodología: Definición Entidades y Relaciones

- Python, como tantos otros, es un lenguaje orientado a objetos.
- Por tanto, nos permite de forma sencilla representar las entidades como clases.
- Lo primero que haremos será definer las clases correspondientes a a las entidades Cliente, Pedido y Producto.
- También crearemos la clase PedPro que define las relaciones entre Pedido y Producto al ser una relación n:m
- En Pedido definimos el atributo IdCliente debido a la relación entre Cliente y Pedido 1:n

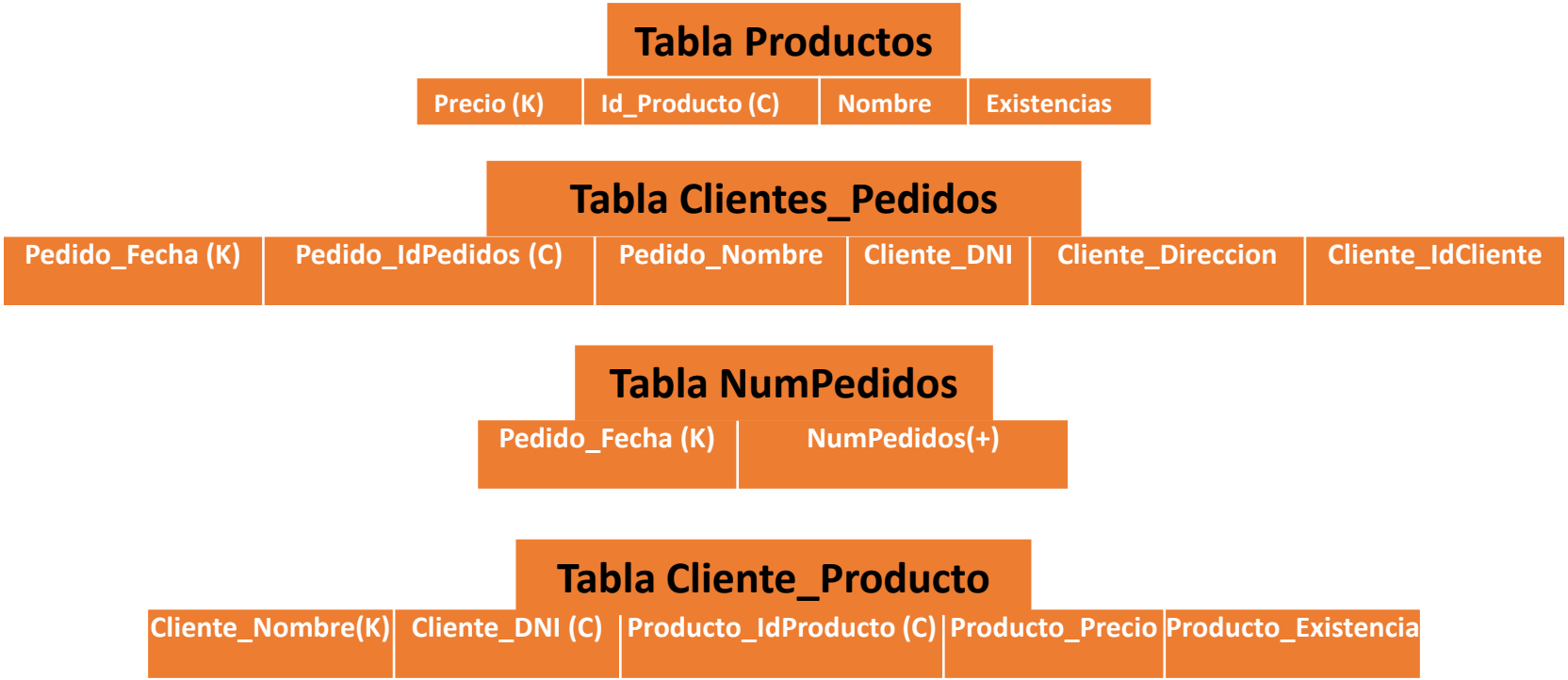
```
class Cliente:
    def __init__(self, IdCliente, Nombre, DNI, Direccion):
        self.IdCliente = IdCliente
        self.Nombre = Nombre
        self.DNI = DNI
        self.Direccion = Direccion
class Pedido:
    def __init__(self, IdPedido, Fecha, IdCliente):
        self.IdPedido = IdPedido
        self.Fecha = Fecha
        self.IdCliente = IdCliente
class PedPro:
    def __init__ (self, IdPedido, IdProducto, Unidades):
        self.IdPedido = IdPedido
        self.IdProducto = IdProducto
        self.Unidades = Unidades
class Producto
    def __init__(self, IdProducto,Nombre,Precio, Existencias):
        self.IdProducto = IdProducto
        self.Nombre = Nombre
        self.Precio =Precio
        self.Existencias = Existencias
```

- En Python, a diferencia de otros lenguajes como Java no es posible declarar varios constructores para la misma clase.
- Si necesitamos crear varios constructores necesitamos declarar una clase nueva.

```
class Cliente:
    def __init__(self, IdCliente, Nombre, DNI, Direccion):
        self.IdCliente = IdCliente
        self.Nombre = Nombre
        self.DNI = DNI
        self.Direccion = Direccion
class Pedido:
    def __init__(self, IdPedido, Fecha, IdCliente):
        self.IdPedido = IdPedido
        self.Fecha = Fecha
        self.IdCliente = IdCliente
class PedPro:
    def __init__ (self, IdPedido, IdProducto, Unidades):
        self.IdPedido = IdPedido
        self.IdProducto = IdProducto
        self.Unidades = Unidades
class Producto
    def __init__(self, IdProducto,Nombre,Precio, Existencias):
        self.IdProducto = IdProducto
        self.Nombre = Nombre
        self.Precio =Precio
        self.Existencias = Existencias
```

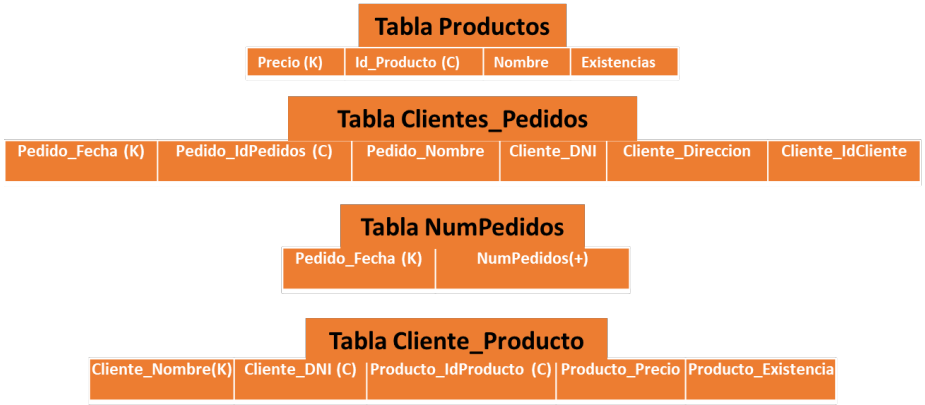
- El modelo de Cassandra es desnormalizado, es decir, la información puede repetirse en varias tablas.
- La estructura del esquema Cassandra, que está diseñado en base a las consultas, no tiene que coincidir específicamente con las operaciones de modificación de datos.
- Por esa razón en nuestros sistemas vamos a considerar la inserción de información siguiendo el modelo conceptual aunque teniendo en cuenta las tablas de nuestro esquema. Por ejemplo, no habrá una manera de inserter solo un cliente pues no tenemos ninguna tabla que lo necesite. En el caso de haberla en el futuro, se desarrollaría el código fuente pertinente.
- Tendremos que distinguir entre insertar instancias de entidades, relaciones entre instancias y varias de las anteriores a la vez.

Inserción de entidades



Que tipo de inserciones habría que considerar

- Las tablas almacenan información de las siguientes entidades o relaciones por orden:
  - Entidad Productos (tabla Productos)
  - Relación Cliente-Pedidos (tabla Clientes_Pedidos)
  - Entidad Pedidos (tabla NumPedidos)
  - Relación Cliente-Pedidos-Productos (table_cliente_producto)
- Por lógica de negocio los pedidos siempre se insertarán en una relación y siempre tendrán un cliente y productos asociados.
- Por tanto consideraremos las inserciones de productos por separado y la relación Cliente-Pedidos-Productos




```
insertStatementNumPedidos =
session.prepare(
        "UPDATE numpedidos SET numpedidos
= numpedidos + 1 WHERE pedido_fecha = ?")
```

Función insertar Producto

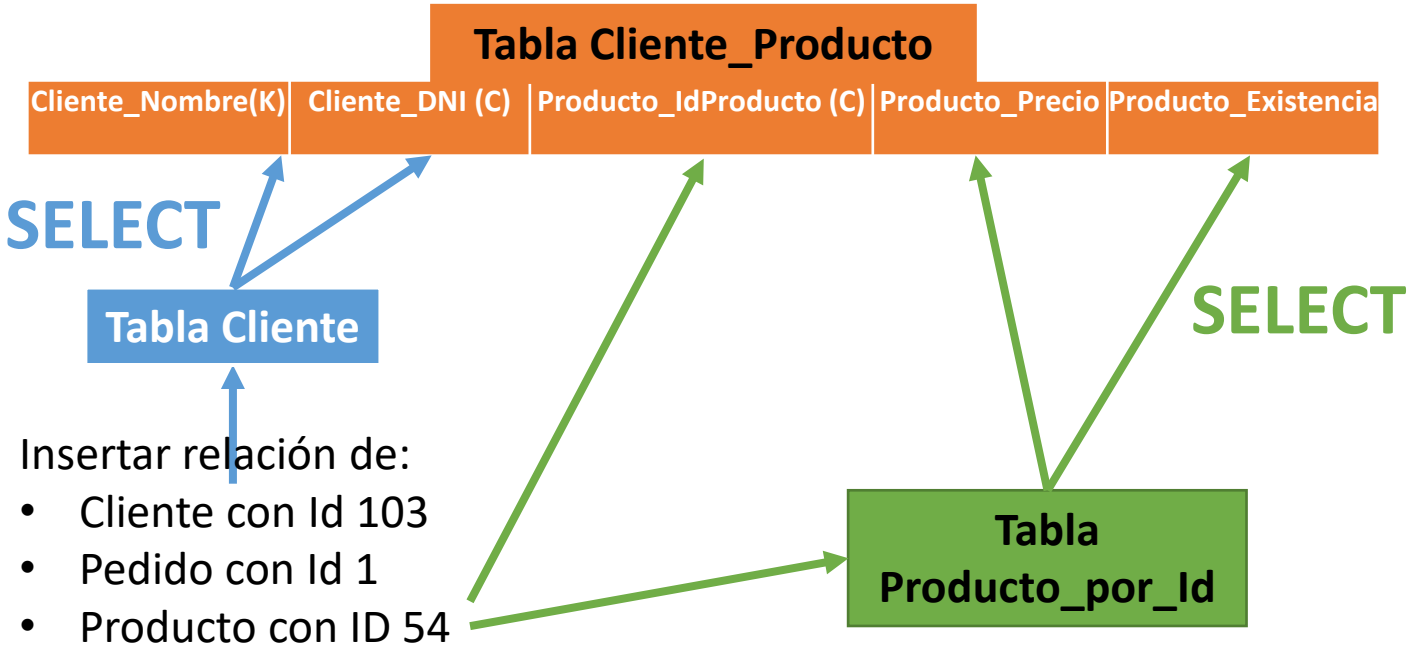
```
def insertProducto ():
    #Pedimos al usuario del programa los datos del producto
    nombre = input ("Dame nombre del producto")
    precio = float (input ("Dame precio del producto"))
    existencias = int (input ("Dame existencias del producto"))
    id = int (input("Dame ID")) #Esto puede ser sustituido por uuid automatico o extraer MAX (id)
    p = Producto (id, nombre, precio, existencias)
    insertStatement = session.prepare ("INSERT INTO productos (precio, idproducto, existencias, nombre) VALUES (?, ?, ?, ?)")
    session.execute (insertStatement, [p.Precio, p.IdProducto, p.Existencias, p.Nombre])
    insertStatement2 = session.prepare("INSERT INTO productos_por_id (id, precio,  existencias, nombre) VALUES (?, ?, ?, ?)")
    session.execute(insertStatement2, [p.IdProducto, p.Precio, p.Existencias, p.Nombre])
```

Tablas soporte

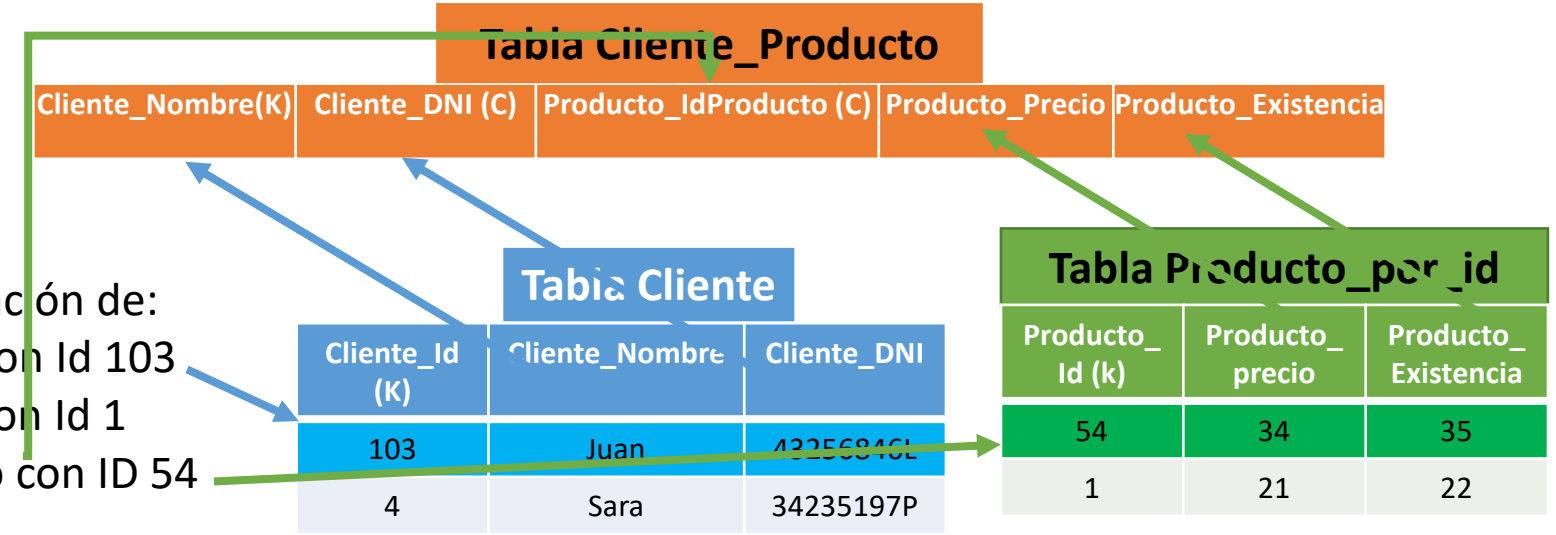
- Objetivo es buscar valores de una instancia que ya están almacenados en la BBDD.
- Estas tablas están pensadas para utilizarlas como soporte en la realización de una operación. Servirán para buscar un valor ya almacenado en la base de datos sin tener que indexar columnas no clave.
- Pueden ser para:
    1. Buscar valores de atributos no clave de una entidad: La partition key será la clave primaria de la entidad.
    2. Buscar clave primaria de una entidad que tenga asociado un valor concreto de un atributo no clave: La partition key será el atributo no clave y la clave primaria de la entidad será clustering key.
    3. Obtener relaciones en base a la clave primaria de una de las entidades relacionadas: La partition key será la clave primaria de la entidad, mientras que la clustering key sería la clave primaria de la otra entidad de la relación

- Inserciones: Solo cuando insertemos una relación en la que para una de las entidades relacionadas solo se nos proporcione el valor de su clave primaria
- Actualizaciones y borrados: Al actualizar el valor de una instancia de una entidad o borrar una instancia. En algunas tablas se tendrá como clave primaria un valor que no sea por el que se quiere borrar o actualizar. Es nuestro deber buscar los valores equivalente para realizar el borrado correcto.
    - P. ej: Si queremos borrar los productos que tienen como precio 5€, tendremos que realizar una búsqueda para obtener todos los ids de productos que tengan como precio 5€

Ejemplo tabla soporte tipo 1



Ejemplo tabla soporte tipo 1: Inserción



Insertar relación de:

- Cliente con Id 103
- Pedido con Id 1
- Producto con ID 54

SELECT cliente_nombre, cliente_dni from Cliente WHERE cliente_id = 103;
SELECT producto_precio, producto_existencia from producto_por_id WHERE producto_id = 54;
INSERT INTO Cliente_Producto (cliente_nombre, cliente_dni, producto_id, producto_precio, producto_existencia) VALUES (**Juan, 43256846L**, **54, 34, 35**)

```python
def insertClientePedidosProductosSelectCliente ():
    #Pedimos al usuario del programa los datos del producto
    precio = float (input ("Dame precio del producto"))
    existencias = int (input ("Dame existencias del producto"))
    nombre = input ("Dame nombre del producto")
    idCliente = int(input ("Dame ID del cliente"))
    idProducto = int(input ("Dame ID del Producto"))
    IdPedido = int(input ("Dame ID del Pedido"))
    hoy = date.today()
    cliente = extraerDatosCliente (idCliente)
    if (cliente != None):
        insertStatementClientesPedido = session.prepare(
            "INSERT INTO Clientes_Pedidos (Pedido_Fecha, Pedido_IdPedidos, Pedido_Nombre, Cliente_DNI, Cliente_Direccion, Cliente_IdCliente) VALUES (?, ?, ?, ?, ?, ?)")
        insertStatementClientesProducto = session.prepare(
            "INSERT INTO Cliente_Producto (Cliente_Nombre, Cliente_DNI, Producto_IdProducto, Producto_Precio, Producto_Existencias) VALUES (?, ?, ?, ?, ?)")
        insertStatementNumPedidos = session.prepare(
            "UPDATE \"NumPedidos\" SET NumPedidos = NumPedidos + 1 WHERE Pedido_Fecha = ?")
        session.execute(insertStatementClientesPedido,
                        [hoy, IdPedido, nombre, cliente.DNI, cliente.Direccion, idCliente])
        session.execute(insertStatementClientesProducto,
                        [cliente.Nombre, cliente.Direccion, idProducto, precio, existencias])
        session.execute(insertStatementNumPedidos, [hoy, ])
```

```python
def extraerDatosCliente(idCliente):
    select = session.prepare ("SELECT * FROM clientes_por_id WHERE id = ?") #solo va a devolver una filia pero lo tratamos como si fuesen varias
    filas = session.execute (select, [idCliente,])#Importante, aunque solo haya un valor a reemplazar en el preparedstatemente, hay que poner la ','
    for fila in filas:
        c = Cliente (idCliente, fila.nombre, fila.dni,  fila.direccion) #creamos instancia del cliente
        return c
#Función que ejecuta un SELECT contra la base de datos y extrae la información de los clientes que compraron un producto según su ID
```

Ejemplo tabla soporte tipo 2: Borrado

Borrar clientes llamados 'Juan':

SELECT Cliente_Id where Cliente_nombre = 'Juan'

Tabla Cliente_por nombre

Cliente_Nombre(K)	Cliente_Id (C)
Juan	103
Sara	104
Juan	102

Tabla Cliente_Por_Id

Cliente_Id(K)	Cliente_Nombre
103	Juan
104	Sara
102	Juan

```
DELETE FROM Cliente_por_id WHERE Cliente_Id = 103;
DELETE FROM Cliente_por_id WHERE Cliente_Id = 102;
```

Ejemplo tabla soporte tipo 2: Borrado

Borrar clientes llamados 'Juan':

SELECT Cliente_Id where Cliente_nombre = 'Juan'

Tabla Cliente_por nombre

Cliente_Nombre(K)	Cliente_Id (C)
Juan	103
Sara	104
Juan	102

Tabla Cliente_Por_Id

Cliente_Id(K)	Cliente_Nombre
103	Juan
104	Sara
102	Juan

```
DELETE FROM Cliente_por_id WHERE Cliente_Id = 103;
DELETE FROM Cliente_por_id WHERE Cliente_Id = 102;
```

Ejemplo tabla soporte Tipo 3: Actualización

Tabla Cliente_Producto

Cliente_Nombre(K)	Cliente_DNI (C)	Producto_IdProducto (C)	Producto_Precio	Producto_Existencia

Actualizar precio Producto con Id 1:

- Precio es ahora 20

Tabla Producto_Clientes

Producto_Id (k)	Cliente_Id (C)
1	3
1	4

Tabla Cliente

Cliente_Id	Cliente_Nombre	Cliente_DNI
3	Juan	43256846L
4	Sara	34235197P

SELECT

UPDATE Cliente_Producto SET Producto_Precio = 20 WHERE Cliente_Nombre = 'Juan' AND Cliente_DNI = '43256846L' AND Producto_Id_Producto=1;
UPDATE Cliente_Producto SET Producto_Precio = 20 WHERE Cliente_Nombre = 'Sara' AND Cliente_DNI = '34235197P' AND Producto_Id_Producto=1;

```python
#Función que actualzia el precio de un producto
def actualizarPrecioProducto ():
    precio = float (input ("Dame precio del producto"))
    idProducto = int(input ("Dame ID del Producto"))
    clientes = extraerClientesProducto(idProducto) #tenemos que saber que clientes han comprado ese producto para poder actualizar en la tabla cliente_producto
    updatePrecioClienteProducto = session.prepare ("UPDATE cliente_producto SET producto_precio = ? WHERE cliente_nombre = ? AND cliente_DNI = ? AND producto_idproduct
    for clienteId in clientes: #por cada cliente ejecutamos un UPDATE
        infoCliente = extraerDatosCliente(clienteId) #infoCliente será una variable que almacena toda la información de un cliente
        session.execute(updatePrecioClienteProducto,[precio, infoCliente.Nombre, infoCliente.DNI, idProducto])
```

```python
#Función que ejecuta un SELECT contra la base de datos y extrae la información de un cliente según su ID
def extraerDatosCliente(idCliente):
    select = session.prepare ("SELECT * FROM clientes_por_id WHERE id = ?") #solo va a devolver una filia pero lo tratamos como si fuesen varias
    filas = session.execute (select, [idCliente,])#Importante, aunque solo haya un valor a reemplazar en el preparedstatemente, hay que poner la ','
    for fila in filas:
        c = Cliente (idCliente, fila.nombre, fila.dni,  fila.direccion) #creamos instancia del cliente
        return c
#Función que ejecuta un SELECT contra la base de datos y extrae la información de los clientes que compraron un producto según su ID
def extraerClientesProducto (idProducto):
    select = session.prepare ("SELECT cliente_id FROM productos_cliente WHERE producto_id = ?") #solo va a devolver una filia pero lo tratamos como si fuesen varias
    idsClientes = [] #donde almacenaremos el retorno de la consulta
    filas = session.execute (select, [idProducto,]) #Importante, aunque solo haya un valor a reemplazar en el preparedstatemente, hay que poner la ','
    for fila in filas: #procesando todos los clientes que compraron ese producto
        idsClientes.append(fila.cliente_id) #como solo queremos los Ids, simplemente vamos añadiendo los valores
    return idsClientes
```

Actualizar valores de la clave primaria de una tabla

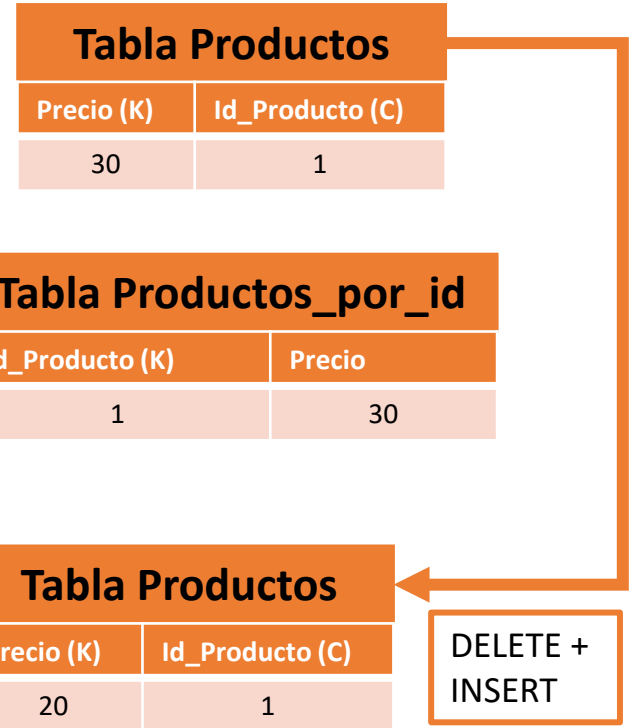
- En un modelo relacional es extraño modificar el valor de una clave primaria de una tabla.
- En Cassandra esto puede ocurrir más a menudo debido a que la clave primaria puede estar compuesta de columnas destinadas a almacenar valores de atributos no clave de una entidad o relación.
- En estos casos no es posible realizar un UPDATE directo, sino que hay que hacer una combinación de DELETE+INSERT
- Primero hay que saber qué valor había almacenado previamente y que debe ser actualizado a través de un SELECT en otra tabla.
- Posteriormente se borra a través de un DELETE con especial cuidado de solo borrar la fila deseada.
- Finalmente se vuelve a insertar la fila con el valor actualizado.

Actualizar precio Producto con Id 1 al valor 20€

SELECT Precio FROM Productos_por_id WHERE ID_Producto = 1;

DELETE FROM Productos WHERE Precio = 30 AND Id_Producto = 1

INSERT INTO Productos (Precio, Id_Producto) VALUES (20, 1);



```python
#Función que actualzia el precio de un producto
def actualizarPrecioProducto ():
    precio = float (input ("Dame precio del producto"))
    idProducto = int(input ("Dame ID del Producto"))
    clientes = extraerClientesProducto(idProducto) #tenemos que saber que clientes han comprado ese producto para poder actualizar en la tabla cliente_producto
    updatePrecioClienteProducto = session.prepare ("UPDATE cliente_producto SET producto_precio = ? WHERE cliente_nombre = ? AND cliente_DNI = ? AND producto_idprod
    for clienteId in clientes: #por cada cliente ejecutamos un UPDATE
        infoCliente = extraerDatosCliente(clienteId) #infoCliente será una variable que almacena toda la información de un cliente
        session.execute(updatePrecioClienteProducto,[precio, infoCliente.Nombre, infoCliente.DNI, idProducto])
    infoProducto = extraerDatosProducto(idProducto)
    if (infoProducto != None): #Comprobar que el idproducto este introducido en la BBDD, si no lo está, no ejecutamos ninguna operación.
        borrarProducto = session.prepare ("DELETE FROM productos WHERE precio = ? AND idproducto = ?")
        session.execute(borrarProducto, [infoProducto.Precio,idProducto])
        insertStatement = session.prepare("INSERT INTO productos (precio, idproducto, existencias, nombre) VALUES (?, ?, ?, ?)")
        session.execute(insertStatement, [precio, idProducto, infoProducto.Existencias, infoProducto.Nombre])
        updateSoporte = session.prepare("UPDATE productos_por_id SET precio = ? WHERE id = ?")
        session.execute(updateSoporte, [precio, idProducto])
        #meter update a la soporte
```

```python
#Función que ejecuta un SELECT contra la base de datos y extrae la información de un producto según su ID
def extraerDatosProducto(idProducto):
    select = session.prepare ("SELECT * FROM productos_por_id WHERE id = ?") #solo va a devolver una fila pero lo tratamos como si fuesen varias
    filas = session.execute (select, [idProducto,])#Importante, aunque solo haya un valor a reemplazar en el preparedstatemente, hay que poner la ','
    for fila in filas:
        p = Producto (idProducto, fila.nombre, fila.precio,  fila.existencias)
        return p
```

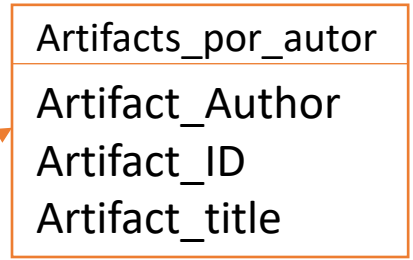
Tablas con claves primarias de atributos de tipo conjunto

Inserciones

- **Como habíamos comentado hay situaciones en las que se requiere buscar por atributos de conjuntos.**
- **En estos casos se implementa como clave una columna que no es un conjunto.**
- **En cuanto haya una inserción de una instancia de la entidad, se deberá insertar una fila por cada valor del conjunto.**

Insertar Nuevo Artifact:

- Artifact_Id=9
- Artifact_Title = Denormalization
- Artifact_Author= {Lucía, Álvaro, Eva}




```
INSERT INTO Artifacts_por_autor (Artifact_Author, Artifact_Id, Artifact_Title) VALUES ('Lucía', 9, 'Denormalization');
INSERT INTO Artifacts_por_autor (Artifact_Author, Artifact_Id, Artifact_Title) VALUES ('Álvaro', 9, 'Denormalization');
INSERT INTO Artifacts_por_autor (Artifact_Author, Artifact_Id, Artifact_Title) VALUES ('Eva', 9, 'Denormalization');
```

- Vamos a añadir el atributo de conjuntos “preferencias” al usuario. Añadiremos la columna gustos a clientes_por_id:

```
ALTER TABLE practica2.clientes_por_id add preferencia
                                    SET<text>;
```

- Posteriormente crearemos otra table que satisfaga la consulta “quiero saber los clientes que tienen como preferencia una determinada sección”.

```
CREATE TABLE practica2.clientes_por_preferencia (preferencia
text, id int, direccion text, dni text, nombre text,
preferencias SET<text>, PRIMARY KEY (preferencia, id)) WITH
CLUSTERING ORDER BY ( id ASC );
```

- Tras la creación de la tabla, incorporamos el código suficiente para almacenar esta información con las funciones ya creadas.

Columns:

Name	Type	Primary Key	Static
preferencia	text	☑	☐
id	int	☑	☐
direccion	text	☐	☐
dni	text	☐	☐
nombre	text	☐	☐
preferencias	set<text>	☐	☐

Código nuevo atributo de conjunto

```python
#Función para pedir datos de un cliente e insertarlos en la BBDD
def insertCliente ():
    #Pedimos al usuario del programa los datos del cliente
    nombre = input ("Dame nombre del cliente")
    direccion = input ("Dame direccion del cliente")
    dni = input ("Dame dni del cliente")
    id = int (input("Dame ID del cliente cliente")) #Esto puede ser sustituido por uuid automatico o extraer MAX (id)
    preferencias = set() #iniciamos la colección (set) que contendra las preferencias a insertar
    preferencia = input ("Introduzca una preferencia, vacio para parar")
    while (preferencia != ""):
        preferencias.add(preferencia)
        preferencia = input("Introduzca una preferencia, vacío para parar")

    c = Cliente (id, nombre, dni, direccion, preferencias)
    insertStatement = session.prepare ("INSERT INTO clientes_por_id (id, nombre, dni, direccion, preferencia) VALUES (?, ?, ?, ?, ?)")
    session.execute (insertStatement, [c.IdCliente, c.Nombre, c.DNI, c.Direccion, c.Preferencias])
    insertStatementPref = session.prepare ("INSERT INTO clientes_por_preferencia (preferencia, id, nombre, dni, direccion, preferencias) VALUES (?, ?, ?, ?, ?, ?)")

    #insertar en preferencias por cliente
    for pref in preferencias:
        session.execute(insertStatementPref, [pref, c.IdCliente, c.Nombre, c.DNI, c.Direccion, c.Preferencias])
```

1. Creación de una tabla que satisfaga la consulta “Quiero saber los pedidos en los que se ha comprado un producto en concreto, incluyendo la fecha de un pedido”
   - Tras la creación de la tabla, incorpore el código suficiente para almacenar esta información con las funciones ya creadas.
2. Añadimos a la entidad producto el atributo de conjuntos “Propiedades”. Posteriormente incluiremos a través de operaciones ALTER TABLE columnas asociadas a ‘Propiedades’ en las tablas Productos y Productos_por_id. Posteriormente crearemos otra tabla que satisfaga la consulta “quiero saber los productos que tienen una propiedad específica”.
   - Tras la creación de la tabla, incorporamos el código suficiente para almacenar esta información con las funciones ya creadas.

Tabla Ejercicio 1

Columna	Clave
Id_Pedido	PK
Id_Producto	CK
Pedido_fecha	

3. Crear una función que borre los pedidos de una fecha concreta
4. Crea una tabla soporte en la que se pueda buscar los ids de un cliente que tengan asociado el mismo valor de una dirección.
5. Utilizando la tabla soporte del ejercicio 4, actualice el valor del nombre en base a una dirección en la tabla clientes_por_id