In [74]:

```python
import mysql.connector
import sqlalchemy
import numpy as np
import pandas as pd
from pandas import read_csv
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from matplotlib.pyplot import figure
import seaborn as sns
import math
from decimal import Decimal

import statistics as stat
```

- **Ejercicio 1**:

  Crea una base de datos relacional simple utilizando MySQL(https://www.mysql.com/) y conectala a Python

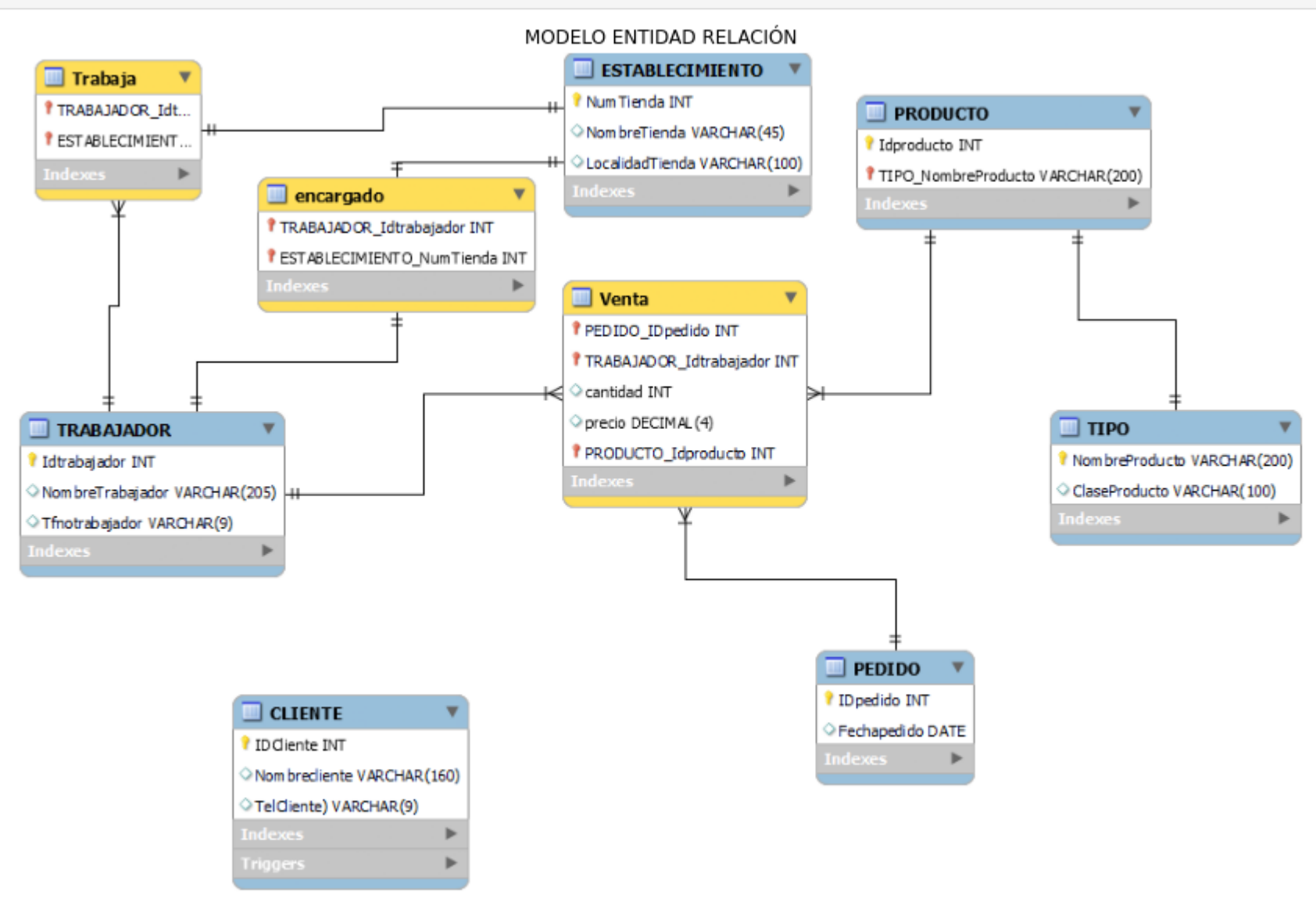
Para realizar el ejercicio he creado una base de datos en Workbench, usando el modelo creado en el ejercicio 13.1

- Primero he creado el modelo relaciomal con las tablas, columnas, y claves foráneas.
- Luego mediante Forward Engineer he creado las base de datos del modelo relacional.
- Por último hemos introducido datos que cumplieran las condiciones del modelo relacional

Abajo mostramos el esquema relacional

In [5]:

```python
img = mpimg.imread("modeloer.png")
plt.figure(figsize = (15,15))
plt.imshow(img)
plt.title('MODELO ENTIDAD RELACIÓN')
plt.axis('off')
plt.show()
```



Descripción de la base de datos:

Tenemos nueve tablas, de las cuales, cinco no tienen clave foránea y cuatro sí(como puede verse en el esquema relacional) . Estas tablas han sido creadas siguiendo el ejercicio 13.1, tras un proceso de normalización, en que la cuarta forma normal no se pudo completar.

Tablas sin claves foráneas.: La tabla clientes contiene el número de ientificador del cliente, su nombre y apellidos, y su teléfono. La tabla trabajador contiene unas columnas iguales a cliente. número de identificador del trabajador, nombre, teléfono. La tabla establecimiento contiene el número de identificador del local, el nombre y la localidad. la tabla tipo contienen el nombre dle producto y su clase. La tabla pedido contienen el número de pedido y la fecha en que fue hecho.

De tablas con claves foráneas tenemos de dos tipos: dos que salieron de forma natural en el proceso de normalización(aunque salieron dos más al intentar llegar a la cuarta forma normal que no hemos considerado para completar este ejercicio por practicidad. Estas dos tablas son Producto, que nos apareción al llegar a la tercera forma normal, y Ventas. Las dos "entidades" que hemos omitido son Compras y Ventas por local, que serían exactamente igual que Ventas pero cambiando la columna IDTrabajador por IDCliente(Compras) o NumTienda(Ventas por Local).

Las "otras" dos tablas que las uso de modo auxiliar, creadas en Workbench, son Trabaja y Encargado. La primera es una interrelación 1-N en que indica en que establecimiento trabaja cada trabajador. Mientras que Encargado es una tabla simple que nos indica que trabajador es encargado de cada establecimeinto.

El motivo por el que he creado estas tablas auxiliares es porque las Relationshpis que permite workbench obligan a una clave foránea en una de las tablas, rompiendo la normalización. Además estaba introduciendo como clave foránea en cada tabla, la clave primaria de la otra tabla, llegando a un punto en que no podía rellenar una tabla entera ya que no tenía la clave foránea definida. Así que como estrategía he creado estas dos tablas a modo de "Interrelación". Puede que el modelo relacional esté mal planteado.

In [6]:

```python
# Nos conectamos a la base datos creada

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Cubano0$",
  db="negocio"
)
```

In [7]:

```python
cur=mydb.cursor() # y creamos el cursor
```

1. Vamos a hacer una exploracion de la base de datos

- primero miramos cuantas tablas tenemos

In [8]:

```python
cur.execute( "Show tables;" )
res=cur.fetchall()
for i in res:
    print(i)
```

```
('cliente',)
('encargado',)
('establecimiento',)
('pedido',)
('producto',)
('tipo',)
('trabaja',)
('trabajador',)
('venta',)
```

In [9]:

```python
len(res)
```

Out[9]:

```
9
```

- Miramos las relaciones entre tablas, así como como sus llaves foráneas, sólo mostraré aquellas que tienen llaves foráneas

In [10]:

```python
cur2=mydb.cursor()
cur2.execute( "DESCRIBE encargado" )
res2=cur2.fetchall()
for j in res2:
    print(j)
```

```
('TRABAJADOR_Idtrabajador', b'int', 'NO', 'PRI', None, '')
('ESTABLECIMIENTO_NumTienda', b'int', 'NO', 'PRI', None, '')
```

Podemos ver que esta tabla relaciona al encargado por tienda con dos llaves foráneas, de la tabla establecimeinto y de trabajador. Relación 1-N

In [11]:

```python
cur2.execute( "DESCRIBE producto" )
res2=cur2.fetchall()
for j in res2:
    print(j)
```

```
('Idproducto', b'int', 'NO', 'PRI', None, '')
('TIPO_NombreProducto', b'varchar(200)', 'NO', 'PRI', None, '')
```

Podemos ver que Producto tiene una clave foránea de la tabla TIPO, relación 1-1

In [12]:

```python
cur2.execute( "DESCRIBE trabaja" )
res2=cur2.fetchall()
for j in res2:
    print(j)
```

```
('TRABAJADOR_Idtrabajador', b'int', 'NO', 'PRI', None, '')
('ESTABLECIMIENTO_NumTienda', b'int', 'NO', 'PRI', None, '')
```

Esta tabla tiene dos foráneas, en ella se relaciona la cantidad de trabajadores por local, con una relación 1-N

In [13]:

```python
cur2.execute( "DESCRIBE venta" )
res2=cur2.fetchall()
for j in res2:
    print(j)
```

```
('PEDIDO_IDpedido', b'int', 'NO', 'PRI', None, '')
('TRABAJADOR_Idtrabajador', b'int', 'NO', 'PRI', None, '')
('cantidad', b'int', 'YES', '', None, '')
('precio', b'int', 'YES', '', None, '')
('PRODUCTO_Idproducto', b'int', 'NO', 'PRI', None, '')
```

Es una relación M-N-P , con tres llaves foráneas de TRABAJADOR, PEDIDO, PRODUCTO

- Exploro un poco una de las 5 tablas sin clave foránea, la tabla de CLIENTE

In [14]:

```python
cur.execute( " select * from cliente limit 5; ")# invocamos la tablas, y mostramos la cabecera
res= cur.fetchall()
for k in res:
    print (k)
```

```
(0, 'GENERICO', '900000000')
(1, 'FRANCISCO GARCIA', '932145643')
(2, 'SALVADOR FERNANDEZ', '666555333')
(3, 'LORENA MARTINEZ', '643654678')
(4, 'ALBERTO LORA', '622334455')
```

Buscamos a un cliente en concreto llamado NDI

In [15]:

```python
cur.execute( """ select * from cliente
            where Nombrecliente like "%NDI%"; """)
res= cur.fetchall()
print(res)
```

```
[(12, 'FORMOSE NDI', '627272727')]
```

Podemos ver que el último registro está en minúsculas. Vamos a convertirlo en mayúsculas para mantener la coherencia.

In [20]:

```python
cur.execute( """ select * from cliente
            where Idcliente =15; """)
res= cur.fetchall()
print(res)# miramos la fila en la que se encuentra
```

```
[(15, 'Manuel Marin', '911335588')]
```

In [21]:

```python
cur.execute( """ update cliente set Nombrecliente = 'MANUEL MARIN'where IDCliente =15;""")
# lo sustituimos.
mydb.commit()

# y miramos que esté bien registrado
cur.execute( """ select * from cliente
            where Idcliente =15; """)
res= cur.fetchall()
print(res)
```

```
[(15, 'MANUEL MARIN', '911335588')]
```

- Vamos a explorar un poco la tabla de ventas, que es a la que más operaciones se le puede sacar. Ya que las otras, tras normalizaciones, básicamente son del tipo (número de identificación, nombre, algún dato más) y para buscar resultados no son muy óptimas

In [62]:

```python
# miramos la cabecera de la tabla
cur.execute("""select*from venta
            limit 5""")
res=cur.fetchall()

for i in res:
    print (i)
```

```
(1, 2, 1, 72, 1)
(2, 10, 1, 20, 9)
(3, 3, 1, 68, 1)
(4, 7, 2, 130, 1)
(5, 10, 1, 40, 2)
```

In [25]:

```python
#revisamos sus columnas
cur.execute("""SHOW COLUMNS FROM venta""")
res=cur.fetchall()

for i in res:
    print (i)
```

```
('PEDIDO_IDpedido', b'int', 'NO', 'PRI', None, '')
('TRABAJADOR_Idtrabajador', b'int', 'NO', 'PRI', None, '')
('cantidad', b'int', 'YES', '', None, '')
('precio', b'int', 'YES', '', None, '')
('PRODUCTO_Idproducto', b'int', 'NO', 'PRI', None, '')
```

In [26]:

```python
#miramos el número de filas.
cur.execute("""select*from venta
            """)
res=cur.fetchall()
print (len(res))
```

```
24
```

-Miramos las ventas de los trabajadores

In [33]:

```python
cur.execute("""select COUNT(*) AS number,TRABAJADOR_Idtrabajador
            from venta
            group by TRABAJADOR_Idtrabajador
            HAVING number > 2;""")
res=cur.fetchall()

for i in res:
    print (i)
```

```
(3, 2)
(4, 5)
(3, 13)
```

Podemos ver que la primera columna es el número de ventas, y la segunda, el número de empleado

También podemos ver quien es el empleado que más vende

In [55]:

```python
cur.execute("""select TRABAJADOR_Idtrabajador, COUNT(*) AS number
            from venta
            group by TRABAJADOR_Idtrabajador
            order by number DESC
            limit 1
            ;""")
res=cur.fetchall()
print(res)
```

```
[(5, 4)]
```

Vamos a mirar el total de ventas de cada de trabajador y de cada establecimiento, para ello iremos haciendo una serie de Left Joins seguidos.

- Primero de la tabla Venta a Trabajador, que tienen en común la columna IdTrabajador
- Segundo de Trabajador a Trabaja, que siguen teniendo en común la columna IdTrabajador
- tercero de Trabaja a Establecimiento que tienen en común la columna NumTienda
- Luego hacemos un Groupby en Trabajador, y sumamos el total de ventas

In [91]:

```python
y = """
    SELECT  venta.TRABAJADOR_Idtrabajador, sum(venta.precio) as total,
    TRABAJADOR.NombreTrabajador, TRABAJA.ESTABLECIMIENTO_NumTienda,
    ESTABLECIMIENTO.NombreTienda
    FROM VENTA
    LEFT JOIN  trabajador ON venta.TRABAJADOR_Idtrabajador= TRABAJADOR.Idtrabajador
    left join trabaja on TRABAJADOR.Idtrabajador = TRABAJA.TRABAJADOR_Idtrabajador
    left join establecimiento ON TRABAJA.ESTABLECIMIENTO_NumTienda= ESTABLECIMIENTO.NumTienda
    group by TRABAJADOR_Idtrabajador
    ORDER BY total desc
    """
cur.execute(y)
res=cur.fetchall()

for i in res:
    print (i)
```

```
(13, Decimal('470'), 'SARA TENA', 4, 'LOC_ZGZ')
(9, Decimal('330'), 'RUBEN GARCIA', 2, 'LOC_MADRID')
(5, Decimal('235'), 'LORENA LOPEZ', 1, 'LOC_BARCELONA')
(7, Decimal('200'), 'IBAI ROGRIGUEZ', 2, 'LOC_MADRID')
(6, Decimal('170'), 'YOLANDA SANABRIA', 2, 'LOC_MADRID')
(2, Decimal('167'), 'MARIA GARCIA', 1, 'LOC_BARCELONA')
(14, Decimal('130'), 'PIER LUIGI', 4, 'LOC_ZGZ')
(3, Decimal('128'), 'CARLA GOMEZ', 1, 'LOC_BARCELONA')
(8, Decimal('90'), 'LARA FELIX', 2, 'LOC_MADRID')
(1, Decimal('90'), 'JAIRO GONZALEZ', 1, 'LOC_BARCELONA')
(4, Decimal('70'), 'FRANCISO LOPEZ', 1, 'LOC_BARCELONA')
(10, Decimal('60'), 'PABLO GOMEZ', 3, 'LOC_GIJON')
```

In [123...

```python
# vamos a cambiar cambiar el tipo "decimal" por entero,
# primero guardamos los datos decimales en enteros en una lista auxiliar
# segundo convertimos las tuplas en listas
# tercero, insertamos los datos entero en la lista
ent=[]
for j in range(0,len(res)):
    ent.append(int(float(res[j][1]) ))
print (ent)
```

```
[470, 330, 235, 200, 170, 167, 130, 128, 90, 90, 70, 60]
```

In [126...

```python
lista= []
for i in range(0,12):
    lista.append(list(res[i]))
```

In [125...

```python
for i in range (0,len(res)):
    lista[i][1]=ent[i]
for i in lista:
    print (i)
```

```
[13, 470, 'SARA TENA', 4, 'LOC_ZGZ']
[9, 330, 'RUBEN GARCIA', 2, 'LOC_MADRID']
[5, 235, 'LORENA LOPEZ', 1, 'LOC_BARCELONA']
[7, 200, 'IBAI ROGRIGUEZ', 2, 'LOC_MADRID']
[6, 170, 'YOLANDA SANABRIA', 2, 'LOC_MADRID']
[2, 167, 'MARIA GARCIA', 1, 'LOC_BARCELONA']
[14, 130, 'PIER LUIGI', 4, 'LOC_ZGZ']
[3, 128, 'CARLA GOMEZ', 1, 'LOC_BARCELONA']
[8, 90, 'LARA FELIX', 2, 'LOC_MADRID']
[1, 90, 'JAIRO GONZALEZ', 1, 'LOC_BARCELONA']
[4, 70, 'FRANCISO LOPEZ', 1, 'LOC_BARCELONA']
[10, 60, 'PABLO GOMEZ', 3, 'LOC_GIJON']
```

1. Haz algunas cargas simples en DataFrames de consultas a las base de datos.

- Empezamos con el Groupby hecho en la última prueba en que juntábamos 4 tablas mediante Left Joins

In [131...

```python
y = """
    SELECT  venta.TRABAJADOR_Idtrabajador, sum(venta.precio) as total,
    TRABAJADOR.NombreTrabajador, TRABAJA.ESTABLECIMIENTO_NumTienda,
    ESTABLECIMIENTO.NombreTienda
    FROM VENTA
    LEFT JOIN  trabajador ON venta.TRABAJADOR_Idtrabajador= TRABAJADOR.Idtrabajador
    left join trabaja on TRABAJADOR.Idtrabajador = TRABAJA.TRABAJADOR_Idtrabajador
    left join establecimiento ON TRABAJA.ESTABLECIMIENTO_NumTienda= ESTABLECIMIENTO.NumTienda
    group by TRABAJADOR_Idtrabajador
    ORDER BY total desc
    """
df= pd.read_sql(y, con=mydb)
df
```

Out[131...

	TRABAJADOR_Idtrabajador	total	NombreTrabajador	ESTABLECIMIENTO_NumTienda	NombreTienda
0	13	470.0	SARA TENA	4	LOC_ZGZ
1	9	330.0	RUBEN GARCIA	2	LOC_MADRID
2	5	235.0	LORENA LOPEZ	1	LOC_BARCELONA
3	7	200.0	IBAI ROGRIGUEZ	2	LOC_MADRID
4	6	170.0	YOLANDA SANABRIA	2	LOC_MADRID
5	2	167.0	MARIA GARCIA	1	LOC_BARCELONA
6	14	130.0	PIER LUIGI	4	LOC_ZGZ
7	3	128.0	CARLA GOMEZ	1	LOC_BARCELONA
8	8	90.0	LARA FELIX	2	LOC_MADRID
9	1	90.0	JAIRO GONZALEZ	1	LOC_BARCELONA
10	4	70.0	FRANCISO LOPEZ	1	LOC_BARCELONA
11	10	60.0	PABLO GOMEZ	3	LOC_GIJON

*La siguiente consulta será ver que tienda vende más. Básicamente poniendo el agrupamiento en Num_tienda y eliminar las columnas venta.TRABAJADOR_Idtrabajador y TRABAJADOR.NombreTrabajador

In [129...

```python
z = """
    SELECT   sum(venta.precio) as total, TRABAJA.ESTABLECIMIENTO_NumTienda,
    ESTABLECIMIENTO.NombreTienda
    FROM VENTA
    LEFT JOIN  trabajador ON venta.TRABAJADOR_Idtrabajador= TRABAJADOR.Idtrabajador
    left join trabaja on TRABAJADOR.Idtrabajador = TRABAJA.TRABAJADOR_Idtrabajador
    left join establecimiento ON TRABAJA.ESTABLECIMIENTO_NumTienda= ESTABLECIMIENTO.NumTienda
    group by TRABAJA.ESTABLECIMIENTO_NumTienda
    ORDER BY total desc
    """
df1= pd.read_sql(z, con=mydb)
df1
```

Out[129...

	total	ESTABLECIMIENTO_NumTienda	NombreTienda
0	790.0	2	LOC_MADRID
1	690.0	1	LOC_BARCELONA
2	600.0	4	LOC_ZGZ
3	60.0	3	LOC_GIJON

Vemos que la tienda "loc_gijon" tiene muy pocas ventas. Comprobamos en las tablas Venta y Trabaja las ventas por número de identificación de trabajador y local

In [139...

```python
df3= pd.read_sql(""" select * from trabaja
                where ESTABLECIMIENTO_NumTienda=3 """, con=mydb)# filtramos por el número de tienda que es la
df3
```

Out[139...

	TRABAJADOR_Idtrabajador	ESTABLECIMIENTO_NumTienda
0	10	3
1	11	3
2	12	3

In [140...

```python
df2= pd.read_sql(""" select * from venta
                where TRABAJADOR_Idtrabajador= 10 OR TRABAJADOR_Idtrabajador= 11 OR TRABAJADOR_Idtrabajador= 12
                order by TRABAJADOR_Idtrabajador""", con=mydb) # y comprobamos los números de trabajadores , he
# como podríamos haber usado between
df2
```

Out[140...

	PEDIDO_IDpedido	TRABAJADOR_Idtrabajador	cantidad	precio	PRODUCTO_Idproducto
0	2	10	1	20	9
1	5	10	1	40	2

In [ ]: