

# Proyecto Final: Panadería el Volován

Bases de Datos no Estructuradas

Marcela Cruz Larios

Yeudiel Lara Moreno

Pablo David Castillo del Valle

# Introducción

En este proyecto se trabajó con los datos de la panadería *El Volován* situada en Coatepec, Veracruz. Hasta ahora toda la contabilidad del negocio se lleva a cabo de manera manual y la información de ventas, compra de insumos, etc se almacena en registros que son llenados a mano.

[illegible]

**Figura 1:** Formatos usados para registrar la información de contabilidad

El objetivo del proyecto es trasladar el sistema de contabilidad a una aplicación que permita guardar en una base de datos la información que antes sólo se tenía en formatos físicos. Esto permite automatizar tareas y reducir el número de errores humanos, al mismo tiempo que facilita la tarea de análisis de las ventas.

## Objetivo

Desarrollar una aplicación en la cual pueda llevarse registro de las órdenes que los clientes han comprado, así como de los pedidos que han encargado. También, en la misma aplicación se desea llevar el control de los ingresos como de los gastos del negocio, así como el estado del inventario. Los usuarios de esta aplicación deben poder crear órdenes, guardar pedidos para el futuro, llevar seguimiento de los pedidos que se entregan cada día. De igual manera, poder al final del día cerrar caja y generar el reporte diario de gastos, ingresos y ganancias.

## Tecnologías utilizadas

Para esta práctica se utilizaron tres bases de datos distintas: **MongoDB**, **Redis** y **MySQL**. Como lenguaje se optó por **Python** y por lo tanto fue necesario hacer uso de los respectivos drivers de cada una de las bases de datos: **PyMongo**, **redis-py** y **PyMySQL**.

Cada una de estas bases fue desplegada dentro de un contenedor de Docker.

Del mismo modo, utilizaremos Python y la biblioteca **tkinter** para la creación de una UI que le permita a los usuarios interactuar con nuestra base de una manera más amigable.



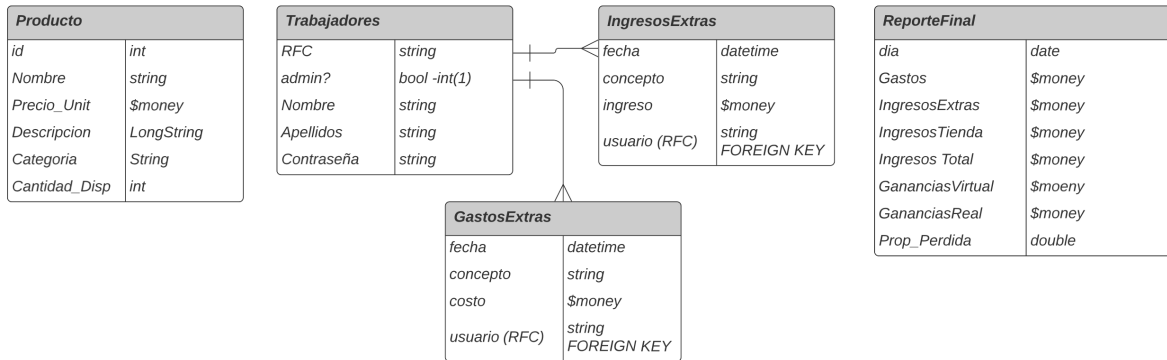
## Diseño de base de datos

Dentro de nuestro sistema se utiliza la siguiente información:

1. Productos y trabajadores. Esta información no es necesario que se modifique constantemente y se suele mantener sin cambios. Estos datos se guardaron en una base de datos relacional, MySQL.
2. Reportes diarios. Cada día se genera un reporte del total de gastos y ventas, esto se almacena en una base relacional: MySQL.
3. Ordenes y Pedidos. Se registra cada orden que se hace y cada pedido (órdenes que se realizan con anticipación y se programan para cierto día de entrega). Debido a que cada orden contiene la información de los productos que se están comprando, así como el número de piezas de cada uno de estos, se optó por usar una base de datos de documentos para esta información, debido a su flexibilidad. Se utilizó MongoDB.
4. Productos e insumos. Esta información representa la disponibilidad que se tiene de cada producto, es decir, el inventario, así mismo de insumos. Debido a que es información que cambia constantemente se decidió tener estos datos de manera temporal en una base de tipo llave valor, en este caso Redis. Al iniciar el día hace una copia en Redis de las tablas Productos e Insumos que están en MySQL y durante el día se modifican según los productos sean vendidos, se preparen más productos o se compren insumos. Al final del día, se se actualizan las tablas de SQL con la versión más actualizada de la información.
5. Gastos e Ingresos extras. Hay productos que no son tan fácilmente facturables y que se compran día con día, de los cuales no es necesario llevar un control tan estricto como con los insumos, además de que se van comprando conforme se van necesitando (por ejemplo esponjas, frutas, verduras, etc.) Esto entra como gasto extra del día porque se va pagando conforme se tiene dinero ese mismo día. Por otro lado, en ingresos extras entra el anticipo de los productos. También, cuando lo que hay en caja no es suficiente para cubrir los gastos del día se inyecta dinero que esta en reserva y con esto se cubren los gastos extras. Esta es información que solo se requiere tener de manera temporal, por lo que se optó por guardarla en Redis y al final del día guardar el resumen en MySQL.

## MySQL

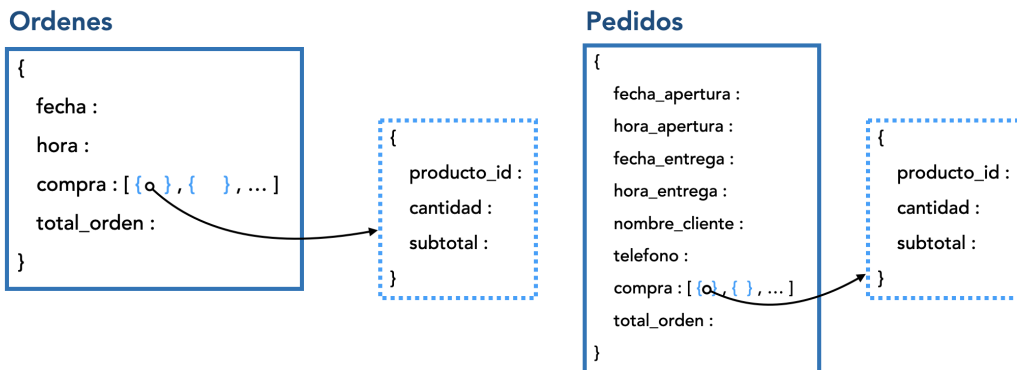
El modelo relacional implementado en MySQL es el siguiente:



En la tabla **Producto** se guardan el catálogo de los productos que se venden en la panadería, así como la información relacionada a cada uno de estos. En **Trabajadores** se tiene información de los empleados, observamos que las tablas **IngresosExtras** y **GastosExtras** referencian a **Trabajadores** pues se guarda quién hizo el registro de ese ingreso o gasto (se puede saber pues el trabajador a cargo de la caja está logueado en el sistema con su RFC). Finalmente, la tabla **ReporteFinal** guarda un resumen de los gastos e ingresos del día, así como ganancias o pérdidas.

## MongoDB

En esta base se guarda la información de las órdenes y los pedidos, se siguió el siguiente modelo con dos colecciones:

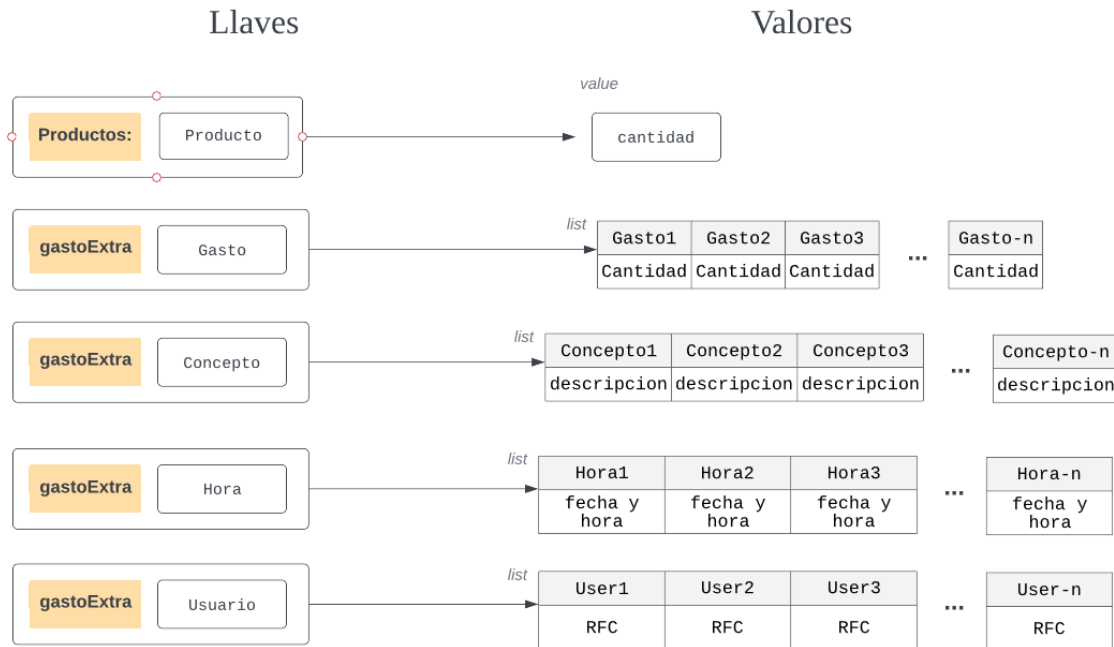


Podemos observar que para las órdenes registramos la fecha y hora y en el atributo **compra** se guarda una lista de documentos, cada uno representando uno de los productos que se adquirió, por cada producto se guarda su identificador, el número de piezas que se compraron y el subtotal. Finalmente, también se guarda el costo total de la orden.

Por otro lado, en pedidos se guarda la hora y fecha de cuando se hace el pedido, así como la hora y fecha del momento en el que se pidió la entrega del pedido. Se registra así mismo el nombre y teléfono del cliente, en **compra** se guarda una lista de documentos, cada uno representando uno de los productos que se adquirió, por cada producto se guarda su identificador, el número de piezas que se compraron y el subtotal. Finalmente, también se guarda el costo total de la orden.

## Redis

El modelo usado para la los datos almacenados en redis consiste en :



Podemos observar una situación importante de nuestro modelo y es que en lugar de utilizar un hash para almacenar los gastos donde hubieran campos hora, concepto, usuario. Se optó por utilizar cuatro listas distintas, esto desde nuestro punto de vista tiene dos ventajas, la primera es que la podemos manejar como una pila, en la que el usuario puede registrar cada uno de los ingresos o gastos extras y cada vez que se registre la información quedará hasta arriba, esto es útil ya que en caso de registrar una transacción errónea se tiene la posibilidad de poder eliminar la última transacción realizada de forma eficiente.

La otra ventaja que le vimos a este modelo, es que realmente de esos datos a lo largo del día no nos interesa consultar las fechas ni los usuarios ni las descripciones de los gastos o ingresos, la información más importante que se llega a consultar en el día a día es la de los gastos extras totales y los ingresos extras totales, por lo tanto saber esa información es mucho más directo de implementar.

También cabe destacar que pasar de listas a dataframe es bastante sencillo.

Por último es importante mencionar que el diagrama mostrado no es nuestro modelo completo, sin embargo es representativo ya que lo último que falta mostrar ahí son las llaves y valores de **ingresoExtra** el cuál tiene exactamente la misma estructura y funcionamiento que **gastoExtra** por lo tanto para tener una mejor presentación se optó por omitir esas llave-valor

## Queries

A continuación, se muestran las consultas que se implementaron para el funcionamiento de la aplicación, así como el código de las mismas. Todas se hicieron desde python a través de los respectivos drivers de las distintas bases de datos usadas.

## MySQL

1. Tener la capacidad de añadir nuevos productos al catálogo, a partir del nombre, la descripción del producto, la categoría, el precio y la cantidad

```
def nuevo_producto(nombre: str, precio_unit: float, descripcion: str, categoria: str, cantidad_disp: int):  
  
    sql = "INSERT INTO productos (nombre, precio_unit, descripcion, categoria, cantidad_disp) VALUES (%s, %s, %s, %s, %s)"  
    val = [nombre, precio_unit, descripcion, categoria, cantidad_disp]  
    mycursor.execute(sql, val)  
    mydb.commit()  
  
    return
```

En esta query simplemente estamos añadiendo un nuevo registro a la tabla `productos`.

2. Dado un producto obtener la cantidad de productos disponibles

```
def get_all_productos():  
    result = pd.read_sql("SELECT nombre, precio_unit, cantidad_disp FROM productos", mydb)  
    return result
```

Debido a que se guarda en la tabla productos la cantidad disponible de cada uno de estos, basta con hacer un `SELECT` sobre esta tabla.

3. En cualquier momento tener la posibilidad de guardar los gastos e ingresos de un día en la base de datos, así como el reporte del día

```
def guardar_gasto(fecha: datetime, concepto: str, costo: float, usuario: str):  
    sql = "INSERT INTO hist_gastos (fecha, concepto, costo, usuario) VALUES (%s, %s, %s, %s)"  
    val = [fecha, concepto, costo, usuario]  
    mycursor.execute(sql, val)  
    mydb.commit()  
    return  
  
def guardar_ingreso(fecha: datetime, concepto: str, ingreso: float, usuario: str):  
    sql = "INSERT INTO hist_ingresos_extras (fecha, concepto, ingreso, usuario) VALUES (%s, %s, %s, %s)"  
    val = [fecha, concepto, ingreso, usuario]  
    mycursor.execute(sql, val)  
    mydb.commit()  
    return
```

Nuevamente, la instrucción se resuelve con un sencillo `INSERT INTO`.

## MongoDB

1. Dado un día, cuánto dinero se obtuvo en órdenes de ese día.

```
def venta_ordenes_por_dia(fecha:datetime):
    fecha = fecha.strftime('%Y-%m-%d')
    total = db.Ordenes.aggregate([{"$match": {"fecha": fecha}},
                                  {"$group": {"_id": "$fecha",
                                               "total_dia": {"$sum": '$total_orden'}}},
                                  {"$project": {"_id": 0, "total_dia": 1}}])

    res = []
    for t in total:
        res.append(t)
    return res[0]
```

Para poder responder esta pregunta, filtramos para quedarnos únicamente con los documentos (de la colección `Ordenes`) cuya fecha coincida con la especificada. Luego, agrupamos respecto a esta fecha y sumamos los valores del atributo `total_orden`. De esta manera recuperamos el monto total de las órdenes hechas tal día.

## 2. Dada una fecha, qué pedidos se entregan ese día.

```
def consultar_entregas(fecha_entrega:datetime = datetime.today(), format:bool=True):
    fecha_entrega = fecha_entrega.strftime('%Y-%m-%d')
    resultado = db.Pedidos.find({"fecha_entrega": fecha_entrega}).sort('hora_entrega')

    if format:
        df = pd.DataFrame()
        for p in resultado:
            l = []
            for prod in p['compra']:
                l.append('{} : {}'.format(prod['producto_id'], prod['cantidad']))
            columna = []
            columna.append('Hora: {}'.format(p['hora_entrega']))
            columna.append('Tel: {}'.format(p['telefono']))
            total = "${:,}".format(p['total_orden'])
            columna.append('Total: {}'.format(total))
            columna = columna + l
            df = pd.concat([df, pd.DataFrame(columna, columns = [p['nombre_cliente']], axis=1)])
        df = df.fillna('')
        return df

    pedidos = []
    for p in resultado:
        pedidos.append(p)
    return pedidos
```

Para esta consulta simplemente buscamos en la colección `Pedidos` todos los documentos con `fecha_entrega` igual a la fecha especificada. El resto de esta función se encarga de formatear el resultado para obtener en un dataframe legible la información obtenida de la consulta.

## 3. Dada una fecha, cuánto dinero se cobro en anticipos de pedidos realizados ese día.

```
def pago_anticipos_por_dia(fecha_apertura:datetime = datetime.today()):
    fecha_apertura = fecha_apertura.strftime('%Y-%m-%d')
    total = db.Pedidos.aggregate([{"$match": {"fecha_apertura": fecha_apertura}},
                                  {"$group": {"_id": "$fecha_apertura",
                                               "total_anticipos_dia": {"$sum": '$total_orden'}}},
                                  {"$project": {"_id": 0, "total_anticipos_dia": 1}}])

    res = []
    for t in total:
        res.append(t)
    return res[0]['total_anticipos_dia']*0.5
```

Buscamos en la colección `Pedidos` los documentos cuya fecha de apertura (fecha en que se realizó el pedido) coincida con la especificada. Luego, sumamos sobre el monto total de los pedidos y devolvemos esto multiplicado por 0.5 pues el anticipo es del 50 %.

4. Dada una fecha, cuánto dinero se cobrará en pago del precio restante de los pedidos que se entregan ese día.

```
def pago_entregas_por_dia(fecha_entrega:datetime = datetime.today()):
    fecha_entrega = fecha_entrega.strftime('%Y-%m-%d')
    total = db.Pedidos.aggregate([{"$match": {"fecha_entrega": fecha_entrega}},
                                  {"$group": {"_id": "fecha_entrega",
                                               "total_pago_entregas" : {'$sum': '$total_orden'} }},
                                  {"$project": {"_id":0, "total_pago_entregas" : 1 }]])

    res = []
    for t in total:
        res.append(t)
    return res[0]['total_pago_entregas']*0.5
```

De manera similar a la consulta anterior, se seleccionan solo los pedidos cuya fecha de entrega coincida con la especificada y luego se suma sobre los montos totales de estos pedidos. Finalmente, se regresa la cantidad obtenida multiplicada por 0.5.

5. Dada una fecha, cuántas piezas de cada producto se han encargado en pedidos que se entregan ese día.

```
def productos_para_entregas(fecha_entrega:datetime = datetime.today()):
    fecha_entrega = fecha_entrega.strftime('%Y-%m-%d')
    resultado = db.Pedidos.aggregate([{"$match": {"fecha_entrega": fecha_entrega}},
                                       {"$unwind": "$compra" },
                                       {"$group": {"_id": "$compra.producto_id",
                                                    "total_piezas" : {'$sum': '$compra.cantidad'}},
                                       {"$project": {"_id":1, "total_piezas" : 1 }]])

    pedidos = []
    for p in resultado:
        pedidos.append(p)
    return pd.DataFrame(pedidos).sort_values('total_piezas', axis=0, ascending=False)
```

Para esta consulta primero se seleccionan los documentos (en la colección `pedidos`) cuya fecha de entrega coincida con la fecha especificada. Luego, usando `$unwind` separamos los productos de cada orden, pues estos se encuentran dentro de un arreglo. Finalmente, agrupamos respecto al id de cada producto y sumamos sobre las cantidades.

## Redis

En el caso de Redis, no hubo tantas consultas como tal ya que la usamos como base de datos para hacer caching, ya que consideramos que era la mejor manera de aprovechar su velocidad.

1. Al final del día poder consultar los productos restantes en la panadería

```
def CerrarDia(self, restantes):
    productos=[]
    cantidad=[]
    for index, row in restantes.iterrows():
        productos.append(row[0])
        cantidad.append(int(self.get_sobranes(row[0])))
    df=pd.DataFrame()
    df['Producto']=productos
    df['Cantidad']=cantidad
    return(df)
```

## 2. En cualquier momento del día consultar los ingresos extras que se han tenido

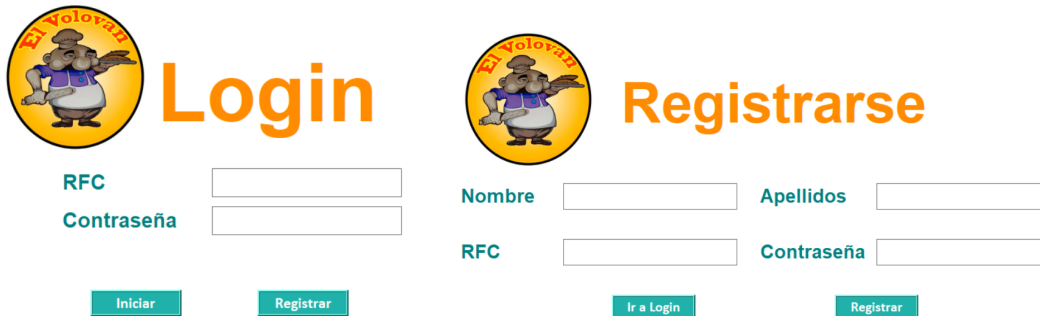
```
def dfGastosExtra(self):
    tiempos=[]
    for i in self.db.lrange('gastoExtra:Hora:', 0, -1):
        tiempos.append(datetime.strptime(i, "%Y-%m-%d %H:%M:%S"))
    gastdict= {'Fecha': tiempos, 'Concepto':self.db.lrange('gastoExtra:Concepto:', 0, -1), \
        'Costo': self.db.lrange('gastoExtra:Gasto:', 0, -1), 'Usuario': self.db.lrange('gastoExtra:Usuario:', 0, -1)}
    gastodf=pd.DataFrame.from_dict(data=gastdict)
    return(gastodf)
```


## 3. En cualquier momento de día poder consultar los gastos extras que se han tenido

```
def dfGastosExtra(self):
    tiempos=[]
    for i in self.db.lrange('gastoExtra:Hora:', 0, -1):
        tiempos.append(datetime.strptime(i, "%Y-%m-%d %H:%M:%S"))
    gastdict= {'Fecha': tiempos, 'Concepto':self.db.lrange('gastoExtra:Concepto:', 0, -1), \
        'Costo': self.db.lrange('gastoExtra:Gasto:', 0, -1), 'Usuario': self.db.lrange('gastoExtra:Usuario:', 0, -1)}
    gastodf=pd.DataFrame.from_dict(data=gastdict)
    return(gastodf)
```

## Interfaz

Para el apartado de interfaz gráfica como es recurrente, tendremos primero el inicio de sesión, con la opción de iniciar sesión o registrarse.






# Login

RFC

Contraseña

Iniciar
Registrar



# Registrarse

Nombre

Apellidos

RFC

Contraseña

Ir a Login
Registrar

Posteriormente se muestran dos menús, en caso de no ser administrador, se nos mandará directo a caja, en caso de serlo se nos mostraran las opciones de administrador.





## Conclusiones

En ciertos aspectos, este proyecto nos resultó más sencillo que lo realizado en las prácticas, como lo fue la parte de lograr conectarnos desde Python a las bases en cada contenedor, o implementar consultas usando las API's de los drivers para Python. Notamos que teníamos un mejor entendimiento de cómo modelar nuestros datos de acuerdo al manejo y a la estructura de los datos que planeábamos usar.

Por otro lado, la parte de integrar la funcionalidad de 3 bases de datos distintas al mismo tiempo fue lo que representó un reto en esta ocasión, así como identificar las sutilezas en el manejo de nuestros datos que implicaban una ventaja al usar una base en lugar de otra. Fue necesario pensar en distintos posibles panoramas de qué bases usar y al final nos decidimos por la opción que implicaba más practicidad en la implementación, pero al mismo tiempo que aprovechara las capacidades de cada tipo de base.

En cuanto a la interfaz gráfica, el hecho de haberla implementado para las últimas 3 prácticas también nos generó mucho más confianza al momento de decidir implementarla para este proyecto final, y como en cada práctica, decidimos ir más allá y retornar a nosotros mismos al buscar añadir funcionalidades que no habíamos usado antes. En definitiva sabíamos que nuestra aplicación tendría una interfaz gráfica, que a pesar de ser sencilla, es muy útil no solo en un caso de uso real de la aplicación, sino también al momento de testear esta.

Finalmente, fue satisfactorio y nos motivó mucho el poder aplicar nuestros conocimientos adquiridos a un caso de la vida real. Pues sabemos que nuestra aplicación podrá ser beneficiosa para el dueño y los trabajadores de la panadería *El Volován*.