Proyecto Base de Datos I

Francesco Uccelli

September 2019/November 2019

1 Requisitos

1.1 Introducción

En el presente proyecto desarrolla el proceso de implementación de una base de datos a una empresa real, con el fin de optimizar la recolección, el almacenamiento y el análisis de los datos de la empresa. Se necesitar ‘a corroborar empíricamente el funcionamiento de todo el sistema, por lo que la empresa va a dar uso de este durante un periodo de tiempo antes de la entrega final. Los resultados del proyecto se basarán, en su mayoría, en consultas ejecutadas por el cliente. En este caso en particular al cliente le conviene tener una rápida recolección de datos, por lo que estas mediciones también serán incluidas en el desarrollo del proyecto.
El objetivo de esta primera entrega es presentar los requerimientos extraídos del análisis de la empresa. Se ha estudiado el funcionamiento de esta empresa para desarrollar una base contextual atinada del problema.

1.2 Descripción general del problema/empresa

La empresa ”espíritu” es una hielera con un almacén en Barranco, donde producen hielo con agua purificada por ´osmosis inversa. Esto hace que el hielo demore más en derretirse que un hielo cualquiera. Por esta razón, es muy bueno para cocteles o coolers. La empresa abastece tres tipos de clientes: clientes regulares, clientes ocasionales y clientes al por menor. Los primeros son varios bares y bodegas localizados en Barranco, a los cuales se le entrega una cantidad de bolsas fijas a la semana para llenar su stock, mantener un registro de estas no es tan complicado. Los segundos son clientes ocasionales, que son algunos en general eventos, pero también

empresas con pedidos irregulares. Por último, están los clientes individuales, una persona cualquiera puede contactar al dueño y pedirle una cantidad de bolsas de hielo. Los hielos pueden ser de dos tamaños: grandes o pequeños, el costo de producción es muy difícil de calcular, se usa agua de la red de Lima. Después de todos los procesos de purificación, aproximadamente el 30 por ciento se convierte en hielo.
Todos lo días se hace una ruta de distribución para abastecer a los clientes regulares, se trata también de incluir en la ruta los clientes ocasionales. Por otro lado, a los clientes al por menor se les envía un glovo (servicio de delivery por el cual los clientes pagan). Logo:

1.3 Necesidad/usos de la base de datos

La empresa necesita una base de datos para almacenar todas las ventas, toda la producción y la distribución. Además, necesita una manera fácil de registrar los pedidos y guardarlos, así como también tener fácil acceso a estos. Por otro lado, necesita también poder consultar sus datos para fines contables, como pagar impuestos o establecer márgenes de ganancia. Esta empresa en particular se beneficiará mucho de las consultas ya que uno de los socios dueño se dedica al análisis de data. Esto servirá para tomar decisiones administrativas y direccionar la empresa hacia un crecimiento sostenible.

1.4 ¿Cómo resuelve el problema hoy?

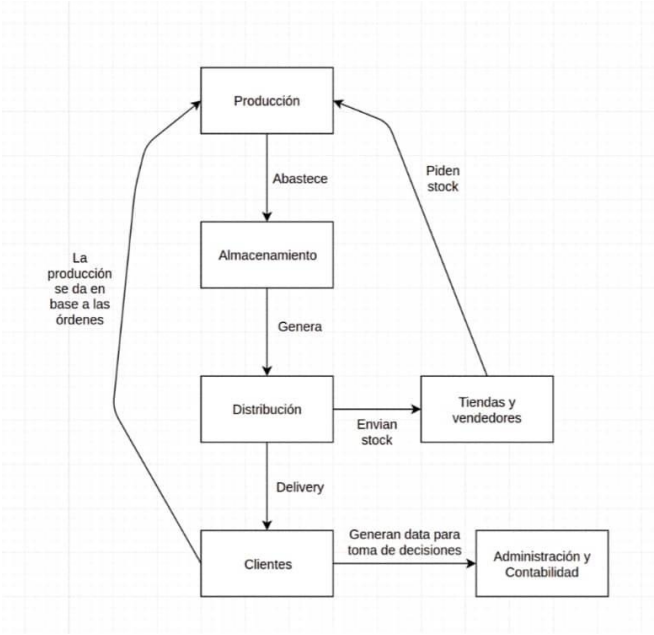
Se seguirá tratando a lo largo del proyecto el sistema como dividido en tres partes esenciales, que son la recolección, el almacenamiento y el análisis de los datos. En primer lugar, la recolección de datos se hace ahora por

WhatsApp. Los clientes se comunican directamente con el dueño para hacerle el pedido y este lo reporta al empleado o se encarga de registrarlo ´el mismo. En segundo lugar, los datos son almacenados por semana en una pizarra en el almacén, donde se apunta diariamente la cantidad de bolsas pedidas y los locales abastecidos. Por último, no hay ningún análisis de los datos más que el de saber las ganancias para pagar impuestos, de los cual se encarga el contador.

1.4.1 ¿Cómo se almacenan/procesa los datos hoy?

Los datos que efectivamente se almacenan, se hacen físicamente en hojas de papel se escriben las ventas de la semana y los clientes. Esto limita mucho la cantidad y las especificaciones que se pueden guardar, cosa que afecta mucho al desarrollo de la empresa.

1.4.2 Flujo de datos



1.5 Descripción detallada del sistema

1.5.1 Objetos de información actuales

Dividiremos los objetos de información en dos periodos: Población de la base de datos con datos pasados y actualización con datos presentes y futuros:
Para la población de la base de datos con información pasada: Ventas semanales: Se ha registrado durante los últimos 2 años aproximadamente, las ventas semanales en cantidad de bolsas. Esto incluye los 7 días de la semana y las ventas diarias.
Ingresos: no se podrán calcular las ganancias ya que es difícil calcular los gastos en general (además no se han registrado) pero si lo ingresos, las ventas por el costo de la venta.
Para la actualización de datos presentes y futuros:
Pedidos: Se deberán registrar los pedidos con cliente, fecha de pedido y fecha esperada de entrega.
Tipo de pago por venta: Es útil para la empresa saber si un cliente pagó con efectivo o tarjeta, si debe pagar aun o, si pidió boleta o factura. Entregas: Se

registrarán todas las entregas y el tipo de cliente: regular, ocasional o al por menor.

1.5.2 Características y funcionalidades esperadas

Por cada parte del sistema se espera una funcionalidad:
Registro: Esta será la parte donde la complejidad pueda quizás escalar un poco más de lo esperado, se espera poder crear una plataforma donde cada empleado de la empresa pueda registrar un pedido y este sea almacenado y pueda ser visto en tiempo real por los otros empleados de la empresa. Algo que facilitará este proceso es que en el momento son solo 3 empleados, pero sería muy útil que sea escalable para facilitar el crecimiento. Desde otro ángulo, antes de este modelado la empresa no tenía forma de contar su producción, ya que las máquinas trabajan 24 horas, normalmente no había forma de contar la producción. A partir de ahora, se cuentan las bolsas que se producen dependiendo de si es hielo pequeño o grande y se suman para tener la cantidad total diaria.

Para el almacenamiento de datos: Se espera simplemente una plataforma digital donde los datos estén almacenados de manera segura y estable, la cantidad de datos no es muy extensa.
Para las consultas: Se espera poder realizar consultas simples a la base de datos, así como generar análisis de los datos pasados cuando sean registrados.

1.5.3 Tipos de usuarios existentes/necesarios

De la base de datos en sí, los únicos usuarios serán los empleados que registren los pedidos y los que se encarguen de hacer las consultas con alguna utilidad. Se evaluará la posibilidad de hacer una plataforma en la que se registren pedidos por los clientes regulares.

1.5.4 Tipos de consulta, actualizaciones

Las consultas se basan básicamente en formas de brindar información para tener un mejor registro de que es lo que ha estado pasando en la empresa. Como, por ejemplo, consultar los ingresos del mes para calcular las ganancias

o saber que cliente ha sido el que más bolsas ha pedido. Por otro lado, cuando se poble la base de datos con datos antiguos, se usarán consultas para generar gráficos y estadísticas que podrán ayudar cuando se trate de buscar inversores.
Finalmente, se probarán consultas aleatorias sin ninguna aparente función para determinar la funcionalidad del sistema de datos establecido. Respecto a las actualizaciones, una vez la base de datos sea poblada los datos no se actualizarán con regularidad, ya que no hay necesidad ni es pertinente con el modelo establecido.

1.5.5 Tamaño estimado de la base de datos

El tamaño de la base de datos real está en el orden de cientos de registros y decenas de inserciones semanales. Se actualizará semanalmente la base de datos y se mantendrá un registro externo diario durante la semana, lo que facilitará el manejo de las órdenes y entregas.

1.6 Objetivos del proyecto

Establecer los requerimientos de almacenamiento y consulta de datos, para agilizar el funcionamiento de la empresa.
Poblar una base de datos con los registros antiguos para así aclarar el crecimiento de la empresa y generar interés de inversionistas.
Desarrollar una plataforma para registrar pedidos por clientes regulares y empleados.

1.7 Referencias del proyecto

Ejemplos de trabajos similares presentados por los profesores Heider Sánchez
Y Teófilo Chambilla

1.8 Eventualidades

1.8.1 Problemas que pudieran encontrase en el proyecto

- La población de la base de datos con los datos antiguos puede tomar tiempo
- La distribución de diferente clientela complicará el registro de los pedidos en tiempo real.
- Toda la información hasta el momento se encuentra registrada en papel.

1.8.2 Limites y alcances del proyecto

El proyecto le va a permitir a la empresa comenzar a registrar sus datos digitalmente, se tratará de hacer un sistema lo más escalable posible, pero mientras la empresa crezca y se diversifique el sistema que se construirá deberá ser reemplazado. La cantidad de datos no debería generar ninguna complicación.

2 Modelo Entidad-Relación

2.1 Reglas semánticas

1. Un cliente tiene nombre por el cual es identificado, y una dirección que Distrito y Calle. Además de un contacto que incluye número de teléfono y cargo del contacto (ejemplo: administrador).
2. Un cliente hace una orden una fecha específica.
3. La orden es identificada por el nombre del cliente y la fecha en la cual quiera recibir el pedido.
4. Se quiere almacenar la cantidad del pedido y el tipo de hielo, así como también la forma en la que genero´ su recibo el cliente.

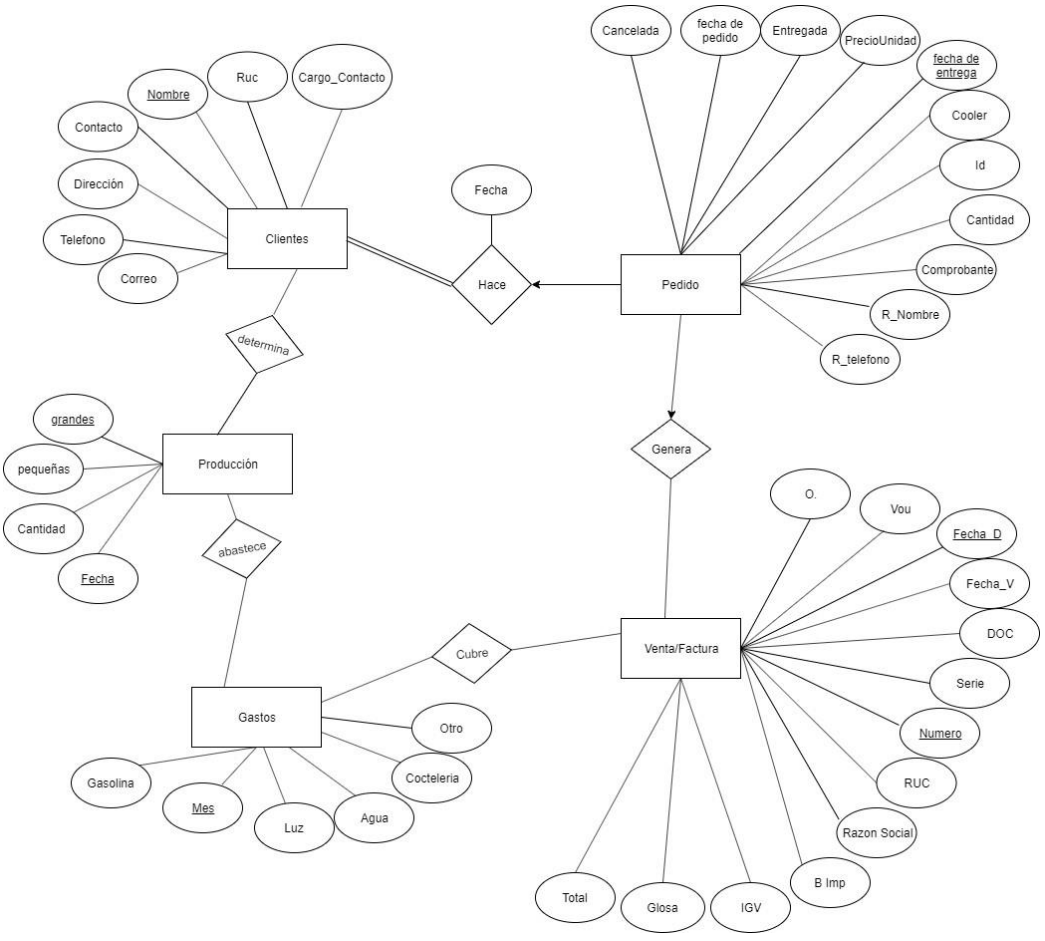
5. Una orden es generada por exactamente un cliente y un cliente puede generar varias ´ordenes.

6. La producción se identifica por el día y tiene tipo y cantidad.

7. Cada pedido genera una sola venta, que es identificada por su número.

8. Las ventas abastecen los gastos de la empresa, que se identifican por mes, y son los gastos de una empresa común.

2.2 Modelo Entidad-Relación



2.3 Especificaciones y consideraciones sobre el modelo

1. Entidad cliente

- Especificaciones: Almacena información sobre los clientes regulares, que son generalmente empresas. Se almacena información sobre el contacto con la empresa
- Consideraciones: No todos los clientes pertenecen a esta tabla, ya que hay clientes ocasionales que piden una sola vez para eventos, por ejemplo. La producción determina si hay capacidad para traer nuevos clientes.

2. Entidad Pedido

    - Especificaciones: Un cliente genera una orden, esta se puede almacenar como entregada o no entregada y cancelada o no cancelada. Eventualmente, se quiere desarrollar una plataforma que muestre los pedidos no entregados y sus características.
    - Consideraciones: Respecto a una orden, se necesita saber el tipo de recibo que genero´ el cliente, si fue boleta o factura, en caso sea factura, se genera una en la tabla de ventas. Una orden puede estar cancelada o no, esto quiere decir que cuando se registra una orden con una fecha, se registra también la fecha de entrega que es en la que se completa la venta y se genera el recibo, este sistema no tiene posibilidad de cancelar pedidos, solo modificarlos.

3. Entidad Venta/factura

    - Especificaciones: Si un pedido emite una factura, se genera una relación con la entidad venta. Esta tiene todos los atributos de una factura estándar, donde se almacena el importe total y el IGV. Además, se tiene una cuenta mensual de ventas que corresponde a Vou.
    - Consideraciones: Si un pedido emite una factura, se genera una relación con la entidad venta. Esta tiene todos los atributos de una factura estándar, donde se almacena el importe total y el IGV. Además, se tiene una cuenta mensual de ventas que corresponde a Vou.

4. Entidad Gastos

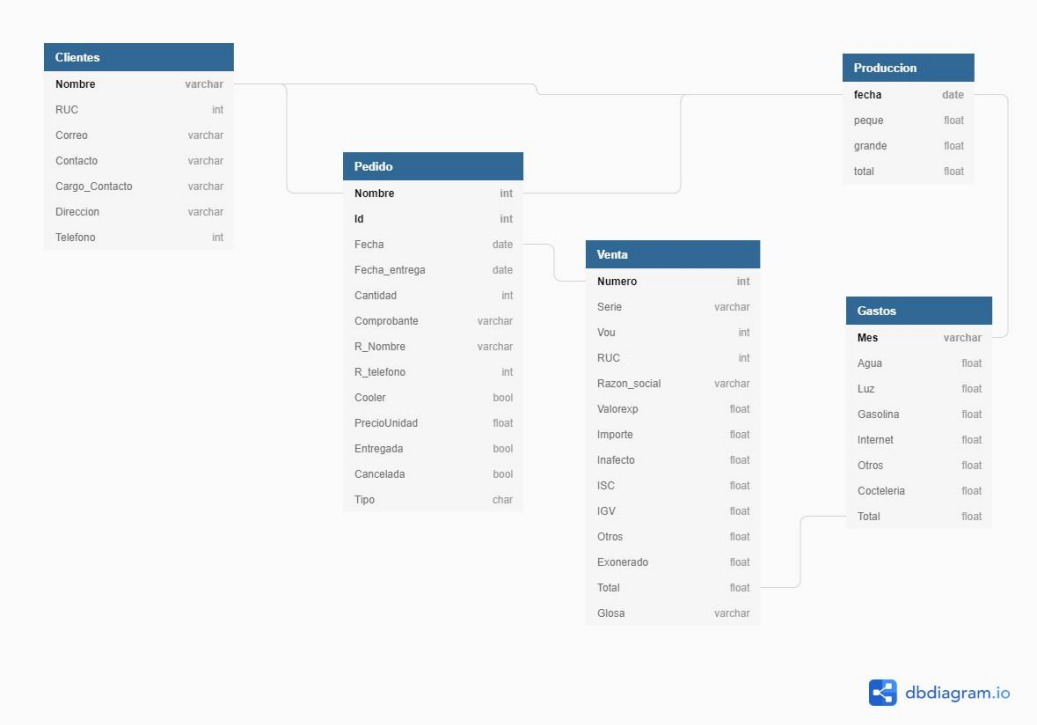
- Especificaciones: Se estableció una entidad para tener cuenta de los gastos mensuales, que incluye" otros" por gatos ocasionales y coctelería por gastos en eventos.
- Consideraciones: Esto ayudara´ a calcular el costo de producción. Los gastos son cubiertos por las ventas.

5. Entidad Producción

- Especificaciones: En la tabla producción se quiere tener un registro diario de la cantidad de bolsas producidas por tipo y en total.
- Consideraciones: Solo se puede aceptar un nuevo pedido o cliente si la producción es suficientemente grande para cubrirlo.

3 Modelo Relacional

3.1 Modelo Relacional



3.2 Especificaciones de transformación

3.2.1 Entidades

3.2.2 Entidades débiles

La única entidad débil corresponde a la tabla pedido, que depende de la tabla cliente. Esto se debe a que no puede existir un pedido sin un cliente asociado a este. Por esta razón la llave foránea de pedido corresponde al nombre del cliente.

3.2.3 Entidades superclase/subclases

Una posible implementación que no corresponde al modelo real podría ser los diferentes tipos de clientes. Ya que en realidad hay diferentes tipos de

clientes: al por mayor, al por menor u ocasionales. La implementación sería como sigue:

- Una clase padre **”Cliente”** con todos los atributos comunes, identificados por un Nombre y o Id. Debe haber una dirección y un número de teléfono.

- Una subclase **”Menor”**, correspondiente a los clientes por menor, identificados por su nombre. Este podría contener sistema de envío y sistema de pedido. Correspondiente por ejemplo a si fue pedido por glovo, teléfono, en persona, etc. Un atributo descuentos, correspondiente a la posibilidad de tener descuentos para un cliente cercano.

- Una segunda subclase **”Mayor”**, correspondiente a los clientes al por mayor, identificados por el nombre de la empresa. Atributos relacionados al contacto en la empresa, cargo nombre y correo. Y RUC para generar la factura fácilmente.

- Una tercera subclase **”Ocasional”** correspondiente a los clientes ocasionales, en su mayoría eventos. Un atributo que determine cuantos días durará el evento, un atributo con un link al evento en Facebook y un atributo correspondiente a la posibilidad de brindar otros servicios (a futuro)

3.2.4 Relaciones binarias

Las relaciones binarias en el modelo determinado corresponden a:

- Un cliente **hace** un pedido. Un cliente hace un pedido y en esta relación se almacena automáticamente una fecha, que registra el día del pedido. Un pedido puede corresponder a exactamente un cliente.

- Un pedido **genera** una factura. Cuando un pedido tiene como comprobante ”factura” se genera una relación con la tabla venta que tiene como etiqueta ”genera”

- Los gastos **abastecen** la producción.

- Las ventas(ingresos) **cubren** los gastos

3.2.5 Relaciones ternarias

- La producción **determina** si se pueden aceptar nuevos pedidos. Esto se debe a que hay una cantidad limitada de bolsas que se pueden producir por día, aproximadamente 120. Entonces si ya hay suficientes pedidos como para usar todo lo producido y almacenado, no se pueden registrar nuevos pedidos y por lo tanto tampoco nuevos clientes.

3.3 Diccionario de datos

1. Tabla Clientes

Clientes				
Nombre del campo	**Tipo de dato**	**Primary Key**	**Foreign Key**	**Descripción**
Nombre	Varchar (65)	x		Se guarda el nombre de la empresa
RUC	Int4			RUC correspondiente a la empresa
Correo	Varchar (65)			Correo correspondiente al contacto
Contacto	Varchar (65)			Nombre del contacto
Cargo_Contacto	Varchar (25)			Puesto del contacto
Dirección	Varchar (165)			Dirección del local
Teléfono	Int4			Teléfono del contacto o local

2. Tabla Producción

Producción				
Nombre del campo	**Tipo de dato**	**Primary Key**	**Foreign Key**	**Descripción**
Fecha	Date	x		Fecha correspondiente a la cantidad de producción
Pequeñas	Int4			Cantidad de bolsas pequeñas
Grandes	Int4			Cantidad de bolsas grandes
Total	Int4			Cantidad de bolsas. Pequeñas + Grandes

3. Tabla Pedido

Pedido				
Nombre del campo	**Tipo de dato**	**Primary Key**	**Foreign Key**	**Descripción**
Nombre	Varchar (65)	x	x	El nombre del cliente que hizo el pedido
Id	Int4	X		Id del pedido generado automáticamente por el software de registro
Fecha	Date			Fecha en la que se hizo el pedido
Fecha_entrega	Date	X		Fecha en la que se entregará el pedido
Cantidad	Int4			Cantidad de bolsas
Comprobante	Varchar (25)			Puede ser boleta o factura
R_Nombre	Varchar (65)			Nombre del encargado de recibir el pedido
R_Teléfono	Int4			Teléfono del encargado de recibir el pedido
Cooler	Varchar(2)			Indica si se ha prestado cooler. SI o NO
PrecioUnidad	Int4			El precio por unidad puede variar según la orden.
Entregada	Varchar(2)			Indica si se ha entregado la orden. SI o NO
Cancelada	Varchar(2)			Indica si se ha cancelado la orden. SI o NO
Tipo	Varchar(1)			Tipo de hielo puede ser pequeño(p) o grande (g)

4. Tabla Venta

Venta				
Nombre del campo	**Tipo de dato**	**Primary Key**	**Foreign Key**	**Descripción**
Numero	Int4	x		El número de venta anual
Serie	Varchar(10)			Serie numérica de factura
Vou	Int4			Numero de venta por mes
RUC	Int4			RUC del cliente
Razón social	Varchar(65)			Nombre formal de la empresa
Valor Exp.	Float			Valor de exporte.
Importe	Float			Importe de las bolsas en el pedido
Inafecto	Float			Inafecto
I.S.C	Float			Impuesto selectivo al consumo
I.G.V	Float			Impuesto general de la venta
Otros	Float			Posibles adiciones al costo
Exonerado	Float			Posibles descuentos
Total	Float			Importe total por venta
Glosa	Varchar(500)			Descripción escrita de la venta

5. Tabla Gastos

Gastos				
Nombre del campo	**Tipo de dato**	**Primary Key**	**Foreign Key**	**Descripción**
Mes	Varchar (65)	x		Mes correspondiente a los gastos
Luz	Float			Gastos del mes en luz
Agua	Float			Gastos del mes en agua
Internet	Float			Gastos del mes en internet
Coctelería	Float			Gastos del mes en coctelería
Gasolina	Float			Gastos del mes en gasolina
Otros	Float			Gastos del mes en otros
Total	Float			Gasto mensual total

4 Implementación de la base de datos

4.1 Creación de Tablas en PostgreSQL

1. Create Producción

```
CREATE TABLE produccion(
fecha date,
pequeñas int4,
grandes int4,
total int4
);
```

2. Create Gastos

```
CREATE TABLE gastos(
mes varchar(20) primary key,
luz real,
agua real,
internet real,
otros real,
gasolina real,
cocteleria real
);
```

3. Create Clientes

```
CREATE TABLE Clientes(
nombre varchar(65) primary key ,
RUC int4,
correo varchar(85) NOT NULL,
contacto varchar(65),
cargo_contacto varchar(25) ,
direccion varchar(165) NOT NULL,
telefono int4
);
```

4. Create Pedidos

```
CREATE TABLE pedido (
    pedidoid serial,
    nombre varchar(65),
    fecha_entrega date,
    fecha date default CURRENT_DATE,
    cantidad int4,
    comprobante varchar(25),
    R_nombre varchar(65),
    R_telefono int4,
    cooler varchar(2),
    preciounidad int4,
    entregada varchar(2),
    cancelada varchar(2),
    tipo varchar(1),
    PRIMARY KEY (nombre,pedidoid,fecha_entrega),
    FOREIGN KEY (nombre) REFERENCES Clientes(nombre)
);
```

5. Create Factura

```
CREATE TABLE factura(
numero int primary key,
serie varchar(10),
vou int4,
RUC int4,
razon_social varchar(65),
valorexp real,
importe real,
inafecto real,
ISC real,
IGV real,
otros real,
exonerado real,
total real,
glosa varchar(500)
);
```

4.2 Carga de datos

Dado que el trabajo se ha desarrollado en un entorno real, la única tabla con suficientes datos como para hacer pruebas es la de ventas. En la que se tiene el registro del presente año en ventas. Esta es cargada al sistema mediante la sentencia ”COPY”.
Por otro lado, para que la experimentación sea correcta. Se generarán diferentes dumps con 100,1000,10000 y 100000 datos para comprobar el funcionamiento de las tablas.

4.3 Simulación de Datos Faltantes

Dado que las tablas no están llenas en el sistema real, se ha procedido a llenarlas con datos aleatorios. Así que no habrá datos faltantes. Para la implementación real, en caso se encuentren datos faltantes, se rellenará la

tabla con un N/A, dado que el sistema es relativamente pequeño, no se darán problemas de espacio.

5 Optimización y Experimentación

5.1 Consultas SQL para el experimento

De las 5 consultas, se van a destinar 4 a hacer la experimentación del sistema. Esto quiere decir que tres consultas van a estar hechas con datos ficticios. Se comprobará el tiempo de las consultas, y se hará un plan de índices para optimizarlas. En cambio, una consulta será hecha a los datos reales, esto nos puede servir para generar alguna conclusión administrativa, y de la misma manera, tratar de optimizarla con índices.

5.1.1 Descripción del tipo de consultas seleccionadas

1. Primera consulta: Para esta consulta pondremos el escenario hipoteticen el que tenemos un nuevo cliente que quiere pedir 30 bolsas de hielo pequeño, por lo que nosotros tendremos que calcular si tenemos la capacidad de distribuirlo.

2. Segunda consulta: En esta consulta vamos a (por fin) descubrir cuánto cuesta hacer una bolsa de hielo. Los costos fijos de producción corresponden al agua y la luz. Entonces: bolsas producidas dividido entre costo mensual.

3. Tercera consulta: Ahora que sabemos cuánto nos cuesta hacer una bolsa de hielo, queremos obtener el margen de ganancia por bolsa de hielo en un mes específico.

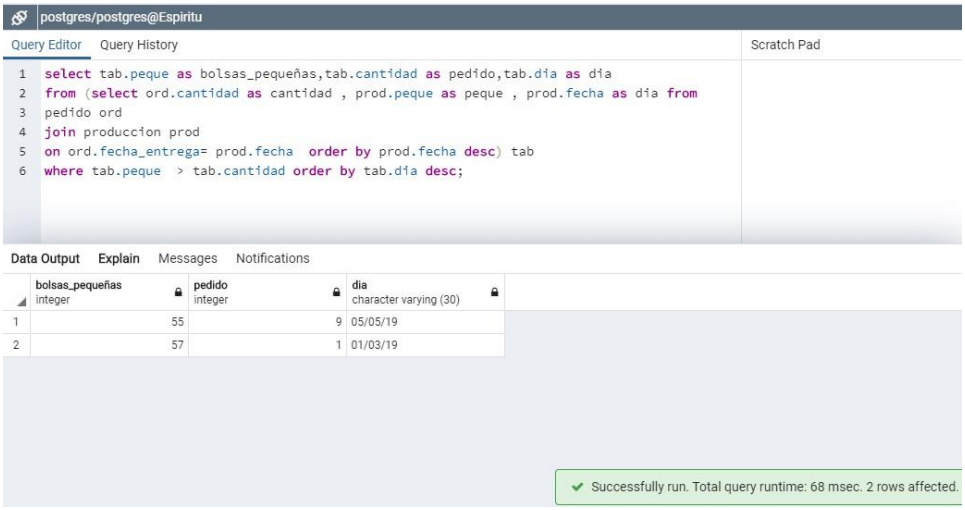
4. Cuarta consulta: Ahora queremos saber que pedidos tenemos que entregar hoy, separado por distritos para hacer una ruta de repartición eficiente

5. Quinta consulta: Para la última consulta queremos conocer el crecimiento de la empresa en los últimos dos años, medido en ventas mensuales. Consultado a los datos reales.

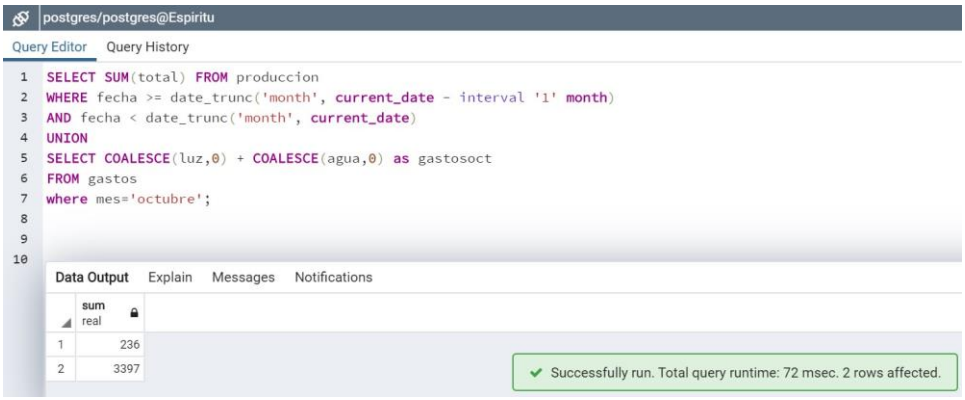
5.1.2 Implementación de consultas en SQL

A continuación, se muestran las consultas implementadas para la base de datos con 100 datos por tabla llenados aleatoriamente. Por esta razón lo números no son necesariamente coherentes con la realidad. Con respecto a la última consulta, la base de datos no esta´ en el sistema postgres, pero nos muestra el crecimiento con respecto al año pasado.

- Primera Consulta



- Segunda Consulta



- Tercera Consulta

```
SELECT sum(coalesce(cantidad,0)*coalesce(preciounidad,0)) as ganancia FROM pedido
WHERE fecha_entrega >= date_trunc('month', current_date - interval '1' month)
AND fecha_entrega < date_trunc('month', current_date) and cancelada ='Y'
UNION
select total from gastos
where mes='octubre';
```

Data Output | Explain | Messages | Notifications

	ganancia real
1	606
2	75205

Successfully run. Total query runtime: 65 msec. 2 rows affected.

- Cuarta Consulta

postgres/postgres@Espiritu

Query Editor | Query History

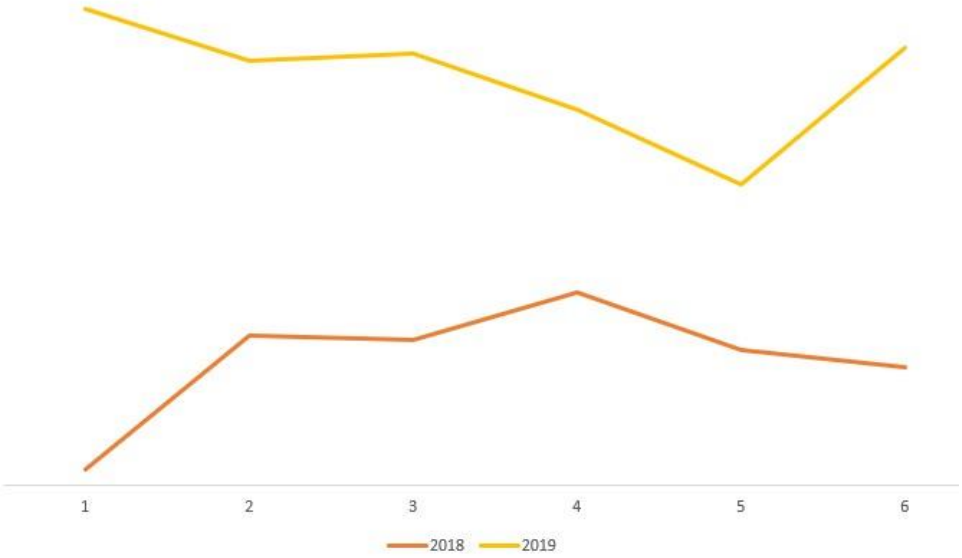
```
1  select cl.direccion as direccion,ped.fecha_entrega entregar ,ped.cantidad bolsas
2  from
3    clientes cl
4    join pedido ped
5    on ped.nombre=cl.nombre
6  WHERE ped.fecha_entrega >= date_trunc('day', current_date - interval '26' day)
7  AND ped.fecha_entrega < date_trunc('day', current_date) and ped.entregada='N'
8  order by ped.fecha_entrega desc;
9
10
11
```

Data Output | Explain | Messages | Notifications

	direccion character varying (165)	entregar date	bolsas integer
1	Ap #804-4291 Eget, Street	2019-10-31	50
2	P.O. Box 937, 8407 Nibh St.	2019-10-31	152
3	P.O. Box 937, 8407 Nibh St.	2019-10-31	87
4	5146 Scelerisque Road	2019-10-31	48
5	5146 Scelerisque Road	2019-10-31	89

Successfully run. Total query runtime: 77 msec. 5 rows affected.

- Quinta consulta



5.2 Metodología del experimento

1. Se crean las tablas coherentemente con el sistema definido en realidad. Incluyendo las dependencias, llaves primarias y llaves foráneas.
2. Se hacen 4 dumps a cada tabla con 100,1000,10000,100000 datos para la experimentación. Aunque esto no tenga sentido para algunas tablas, como por ejemplo la de gastos, que nos guarda el gasto mensual. Esto no ayudara´ a comprobar la eficiencia del modelo definido.
3. Se definen 4 consultas para la experimentación. Definidas en contexto real que podría brindar información necesaria para el desarrollo de la empresa.
4. Se ejecuta EXPLAIN para cada consulta
5. Se ejecuta cada consulta 5 vece y se halla la media de cada una
6. Se definen los índices para la optimización de consultas

7. Se vuelven a ejecutar las consultas comprobando que los resultados sean correctos, se vuelven a medir los tiempos 5 veces y se halla la media.

8. Se almacenan los resultados y se grafican para una comparación más clara

5.3 Optimización de consultas

Luego de haber evaluado las consultas con diferente cantidad de datos, resalta el problema en el que se comienzan a ralentizar las consultas conforme la cantidad de datos aumenta. Para esto se ha analizado el plan generado automáticamente por postgres con la consulta explain y se generarán índices para mejorar el tiempo de las consultas con grande cantidad de datos.

Con respecto a nuestras consultas, tenemos 2 que son con respecto a rangos, por lo que el tiempo conforme sube la cantidad de datos no aumenta demasiado. Por otro lado, tenemos las otras dos consultas que usan de joins. Se especificarán las razones por las cuales se ha implementado cada índice con respecto a cada consulta.

Nombre	Índice	Campos
Idx_clientes_d	Hash	Clientes(dirección)
Idx_clientes	Btree	Clientes(nombre)
Idx_fecha_entrega	Btree	Pedido(fecha_entrega)
Idx_cantidad	Btree	Pedido(cantidad)
Indx_precio	Btree	Pedido(preciounidad)
Indx_g_mes	Hash	Gastos(mes)
Indx_g_total	Btree	Gastos(total)
Indx_peque	Btree	Producción(peque)
Indx_fecha	Btree	Producción(fecha)
Idx_total	Btree	Producción(total)

5.3.1 Planes de índices para Consulta 1

En primer lugar, se analiza el plan de consulta sin índices:

	QUERY PLAN text
1	Subquery Scan on se (cost=17.38..18.14 rows=61 width=12)
2	-> Sort (cost=17.38..17.53 rows=61 width=12)
3	Sort Key: d.fecha DESC
4	-> Hash Join (cost=4.77..15.57 rows=61 width=12)
5	Hash Cond: (b.fecha_entrega = d.fecha)
6	Join Filter: (d.peque > b.cantidad)
7	-> Seq Scan on pedido b (cost=0.00..7.00 rows=300 width=8)
8	-> Hash (cost=3.23..3.23 rows=123 width=8)
9	-> Seq Scan on produccion d (cost=0.00..3.23 rows=123 width=8)

Luego de haber creado los índices:

	QUERY PLAN text
1	Subquery Scan on se (cost=1.39..698516.36 rows=11030888 width=12)
2	-> Merge Join (cost=1.39..588207.48 rows=11030888 width=12)
3	Merge Cond: (d.fecha = b.fecha_entrega)
4	Join Filter: (d.peque > b.cantidad)
5	-> Index Scan Backward using idx_fecha on produccion d (cost=0.42..4834.26 rows=101323 width=8)
6	-> Materialize (cost=0.42..7701.31 rows=106699 width=8)
7	-> Index Scan Backward using idx_fecha_entrega on pedido b (cost=0.42..7434.56 rows=106699 width=8)

Consulta 1: Como se puede ver en los planes de consulta generados por postgre, en esta consulta había dos joins, por lo que ha implementado luego de la indexación en merge join que aumenta en una gran cantidad la eficiencia de la consulta en tablas fáciles de ordenar como esta.

5.3.2 Planes de índices para Consulta 2

En primer lugar, se analiza el plan de consulta sin índices:

	QUERY PLAN text
1	Unique (cost=13.99..14.00 rows=2 width=4)
2	-> Sort (cost=13.99..14.00 rows=2 width=4)
3	Sort Key: (("*SELECT* 1".sum)::real)
4	-> Append (cost=5.77..13.98 rows=2 width=4)
5	-> Subquery Scan on "*SELECT* 1" (cost=5.77..5.79 rows=1 width=4)
6	-> Aggregate (cost=5.77..5.78 rows=1 width=8)
7	-> Seq Scan on produccion (cost=0.00..5.69 rows=32 width=4)
8	Filter: ((fecha >= date_trunc('month'::text, (CURRENT_DATE - '1 mon'::interval month))) AND (fecha < date_trunc('month'::text, (CURRENT_DATE)::timestamp with time zone)))
9	-> Index Scan using gastos_pkey on gastos (cost=0.15..8.17 rows=1 width=4)
10	Index Cond: ((mes)::text = 'octubre'::text)

Luego de haber creado los índices:

QUERY PLAN text
HashAggregate (cost=5133.44..5220.48 rows=8704 width=4)
Group Key: (("*SELECT* 1".sum)::real)
-> Append (cost=3840.70..5111.68 rows=8704 width=4)
-> Subquery Scan on "*SELECT* 1" (cost=3840.70..3840.72 rows=1 width=4)
-> Aggregate (cost=3840.70..3840.71 rows=1 width=8)
-> Seq Scan on produccion (cost=0.00..3587.69 rows=101203 width=4)
Filter: ((fecha >= date_trunc('month'::text, (CURRENT_DATE - '1 mon'::interval month))) AND (fecha < date_trunc('month'::text, (CURRENT_DATE)::timestamp with time zone)))
-> Bitmap Heap Scan on gastos (cost=175.87..1140.41 rows=8703 width=4)
Recheck Cond: ((mes)::text = 'octubre'::text)
-> Bitmap Index Scan on idx_mesg (cost=0.00..173.69 rows=8703 width=0)
Index Cond: ((mes)::text = 'octubre'::text)

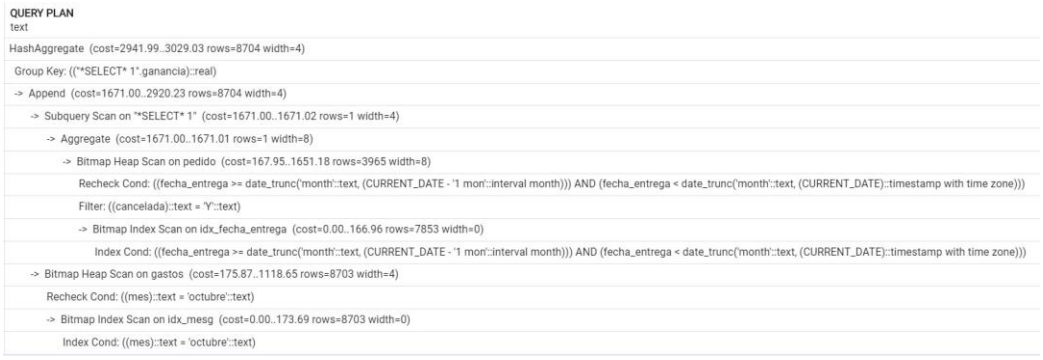
Consulta 2: Como se puede ver en el plan de índices, en la segunda consulta no se hace un join, entonces postgre, cuando se trabaja sin índices simplemente ordena las páginas y las consulta de manera secuencial. En cambio, una vez implementados los índices, se usa un brete para retornar el rango de las condiciones de las querys.

5.3.3 Planes de índices para Consulta 3

En primer lugar, se analiza el plan de consulta sin índices:

	QUERY PLAN text
1	Unique (cost=21.98..21.98 rows=2 width=4)
2	-> Sort (cost=21.98..21.98 rows=2 width=4)
3	Sort Key: (("*SELECT* 1".ganancia)::real)
4	-> Append (cost=13.76..21.96 rows=2 width=4)
5	-> Subquery Scan on "*SELECT* 1" (cost=13.76..13.78 rows=1 width=4)
6	-> Aggregate (cost=13.76..13.77 rows=1 width=8)
7	-> Seq Scan on pedido (cost=0.00..13.75 rows=1 width=8)
8	Filter: (((cancelada)::text = 'Y'::text) AND (fecha_entrega >= date_trunc('month'::text, (CURRENT_DATE - '1 mon'::interval month))) AND (fecha_entrega < date_trunc('month'::text, (CURRENT_DATE)::timestamp with time zone)))
9	-> Index Scan using gastos_pkey on gastos (cost=0.15..8.17 rows=1 width=4)
10	Index Cond: ((mes)::text = 'octubre'::text)

Luego de haber creado los índices:



QUERY PLAN text
HashAggregate (cost=2941.99..3029.03 rows=8704 width=4)
Group Key: (("*SELECT* 1".ganancia)::real)
-> Append (cost=1671.00..2920.23 rows=8704 width=4)
-> Subquery Scan on "*SELECT* 1" (cost=1671.00..1671.02 rows=1 width=4)
-> Aggregate (cost=1671.00..1671.01 rows=1 width=8)
-> Bitmap Heap Scan on pedido (cost=167.95..1651.18 rows=3965 width=8)
Recheck Cond: ((fecha_entrega >= date_trunc('month'::text, (CURRENT_DATE - '1 mon'::interval month))) AND (fecha_entrega < date_trunc('month'::text, (CURRENT_DATE)::timestamp with time zone)))
Filter: ((cancelada)::text = 'Y'::text)
-> Bitmap Index Scan on idx_fecha_entrega (cost=0.00..166.96 rows=7853 width=0)
Index Cond: ((fecha_entrega >= date_trunc('month'::text, (CURRENT_DATE - '1 mon'::interval month))) AND (fecha_entrega < date_trunc('month'::text, (CURRENT_DATE)::timestamp with time zone)))
-> Bitmap Heap Scan on gastos (cost=175.87..1118.65 rows=8703 width=4)
Recheck Cond: ((mes)::text = 'octubre'::text)
-> Bitmap Index Scan on idx_mesg (cost=0.00..173.69 rows=8703 width=0)
Index Cond: ((mes)::text = 'octubre'::text)

Consulta 3: De la misma manera, en la consulta 3 tampoco se utilizan joins. Por lo que postgre lo consulta de la misma manera, ordena y escanea todas las paginas hasta cumplir con la condición dada. Luego de definir los índices, igual que en la consulta anterior, su usa un brete para retornar el rango de la consulta.

5.3.4 Planes de índices para Consulta 4

En primer lugar, se analiza el plan de consulta sin índices:

	QUERY PLAN text
1	Sort (cost=17.16..17.16 rows=1 width=31)
2	Sort Key: ped.fecha_entrega DESC
3	-> Hash Join (cost=13.76..17.15 rows=1 width=31)
4	Hash Cond: ((cl.nombre)::text = (ped.nombre)::text)
5	-> Seq Scan on clientes cl (cost=0.00..3.00 rows=100 width=44)
6	-> Hash (cost=13.75..13.75 rows=1 width=20)
7	-> Seq Scan on pedido ped (cost=0.00..13.75 rows=1 width=20)
8	Filter: (((entregada)::text = 'N'::text) AND (fecha_entrega >= date_trunc('day'::text, (CURRENT_DATE - '26 days'::interval day))) AND (fecha_entrega < date_trunc('day'::text, (CURRENT_DATE)::timestamp with time zone)))

Luego de haber creado los índices:

QUERY PLAN text
Sort (cost=172.48..172.92 rows=177 width=30)
Sort Key: ped.fecha_entrega DESC
-> Hash Join (cost=49.78..165.87 rows=177 width=30)
Hash Cond: ((ped.nombre)::text = (cl.nombre)::text)
-> Bitmap Heap Scan on pedido ped (cost=11.30..126.92 rows=177 width=14)
Recheck Cond: ((fecha_entrega >= date_trunc('day'::text, (CURRENT_DATE - '26 days'::interval day))) AND (fecha_entrega < date_trunc('day'::text, (CURRENT_DATE)::timestamp with time zone)))
Filter: ((entregada)::text = 'N'::text)
-> Bitmap Index Scan on idx_fecha_entrega (cost=0.00..11.26 rows=296 width=0)
Index Cond: ((fecha_entrega >= date_trunc('day'::text, (CURRENT_DATE - '26 days'::interval day))) AND (fecha_entrega < date_trunc('day'::text, (CURRENT_DATE)::timestamp with time zone)))
-> Hash (cost=25.99..25.99 rows=999 width=29)
-> Seq Scan on clientes cl (cost=0.00..25.99 rows=999 width=29)

Consulta 4: En esta consulta si se utiliza un join entre las tablas Cliente y Pedidos, Postgre automáticamente genera un hash join con la condición

dada, que es eficiente, pero podría mejorarse. Bajo este fin, se implementan los índices brete para el rango de la fecha del pedido y un hash para la dirección. Entonces, postgre hace join, escanea usando el brete y retorna todo lo que está asociado con dirección.

5.4 Plataforma de Pruebas

Sistema Operativo	Windows 10
RAM	16GB
CPU	Intel Core i5 - 8250
[HDD] Capacidad	154 GB
[HDD] RPM	7200 RPM
[HDD] Cache	3371 MB
[HDD] Taza trans	550 MB/S
PostgreSQL	12.1

5.5 Medición de tiempos

1. Con 1000 datos:

- Sin índices

Prueba	Consulta 1	Consulta 2	Consulta 3	Consulta 4
1	163	157	134	163
2	160	131	124	157
3	182	125	128	154
4	179	116	141	168
5	149	113	132	156
Media	166.6	128.4	131.8	159.6

- Con índices

Prueba	Consulta 1	Consulta 2	Consulta 3	Consulta 4
1	80	113	121	78
2	90	112	124	71
3	93	101	125	74
4	95	97	111	86
5	93	112	131	87
Media	90.2	107	122.4	79.2

2. Con 10000 datos:

- Sin índices

Prueba	Consulta 1	Consulta 2	Consulta 3	Consulta 4
1	583	155	153	239
2	720	119	125	360
3	706	150	187	191
4	814	133	132	229
5	674	137	153	157
Media	699.4	138.8	150	235.2

- Con índices

Prueba	Consulta 1	Consulta 2	Consulta 3	Consulta 4
1	177	119	142	161
2	181	121	141	168
3	192	132	129	171
4	197	103	125	196
5	182	118	129	178
Media	185.8	118.6	133.2	174.8

3. Con 100000 datos:

- Sin índices

Prueba	Consulta 1	Consulta 2	Consulta 3	Consulta 4
1	5131	163	240	1145
2	5146	149	230	1142
3	5298	152	223	1360
4	5211	170	279	1149
5	5234	219	229	1253
Media	5204	170.6	240.2	1209.8

- Con índices

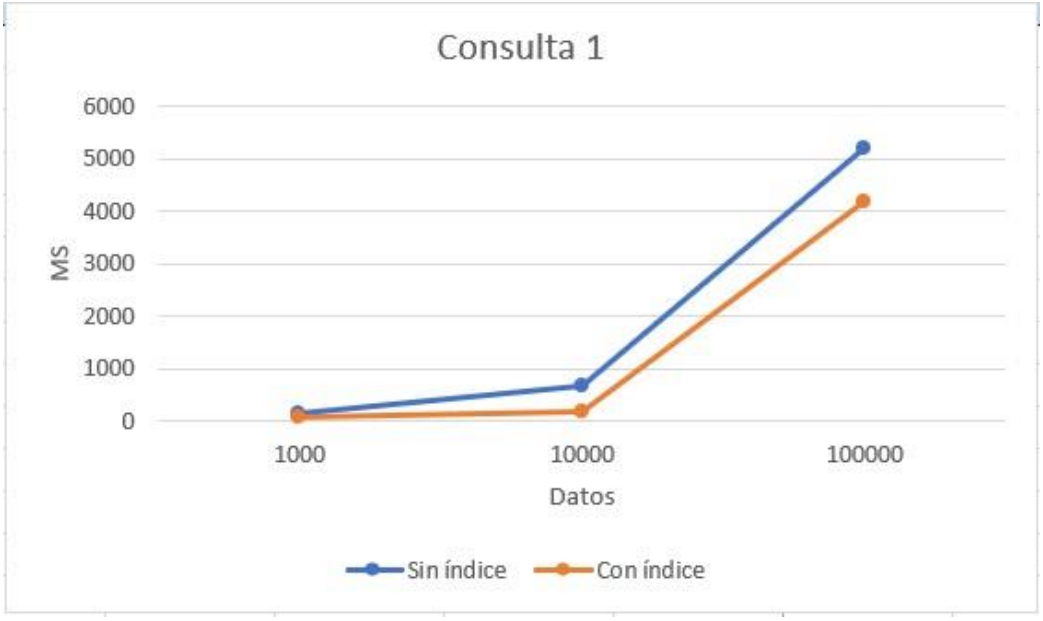
Prueba	Consulta 1	Consulta 2	Consulta 3	Consulta 4
1	4441	151	142	194
2	4015	138	167	230
3	4271	135	144	178
4	4253	143	139	212
5	3987	137	149	248
Media	4193.4	140.8	148.2	212.4

5.6 Resultados

		1000			10000			100000	
Resultados	**SI**	**CI**	**%RE**	**SI**	**CI**	**%RE**	**SI**	**CI**	**%RE**
Consulta1	166	90	0.458	699	185	0.735	5205	4193	0.194
Consulta2	128	107	0.164	138	118	0.153	170	140	0.176
Consulta3	131	122	0.069	150	133	0.113	240	148	0.383
Consulta4	159	79	0.503	235	174	0.26	1210	212	0.825

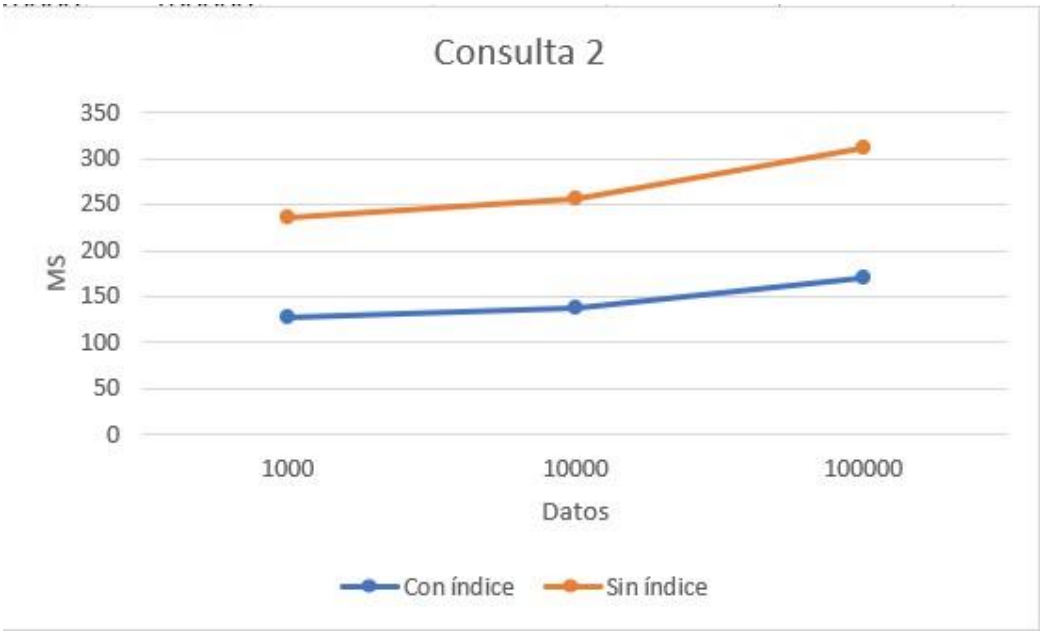
SI= Sin índice, CI= Con índice, RE = reducción porcentual

5.6.1 Consulta 1



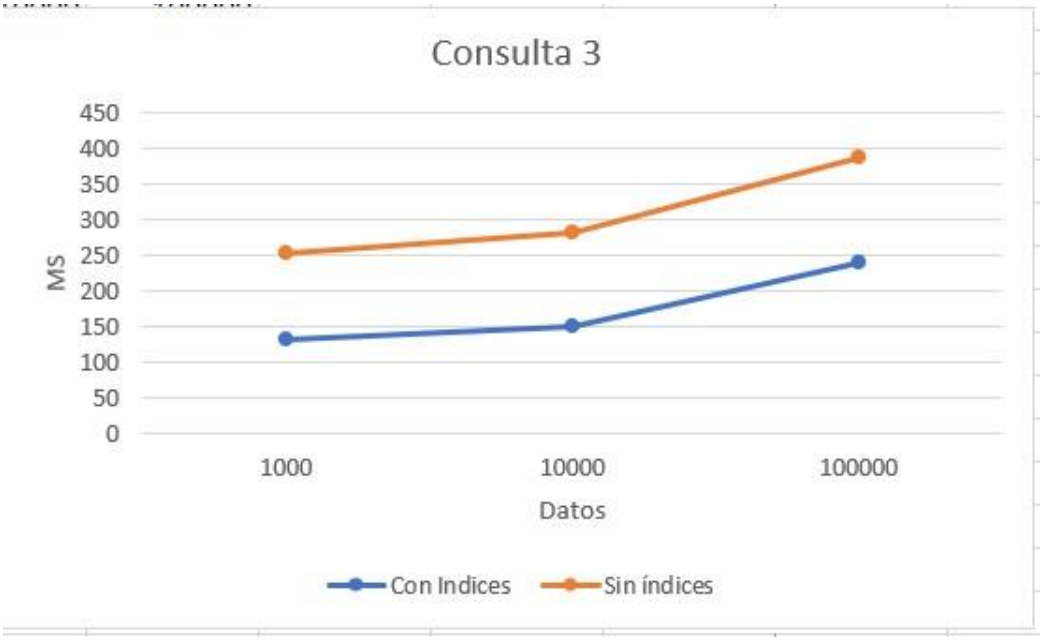
Como se puede ver en la gráfica, el tiempo disminuyó considerablemente con respecto a los índices. Igual el performance de la consulta es bastante lento por lo que se debería plantear la opción de agregar otro tipo de optimización.

5.6.2 Consulta 2



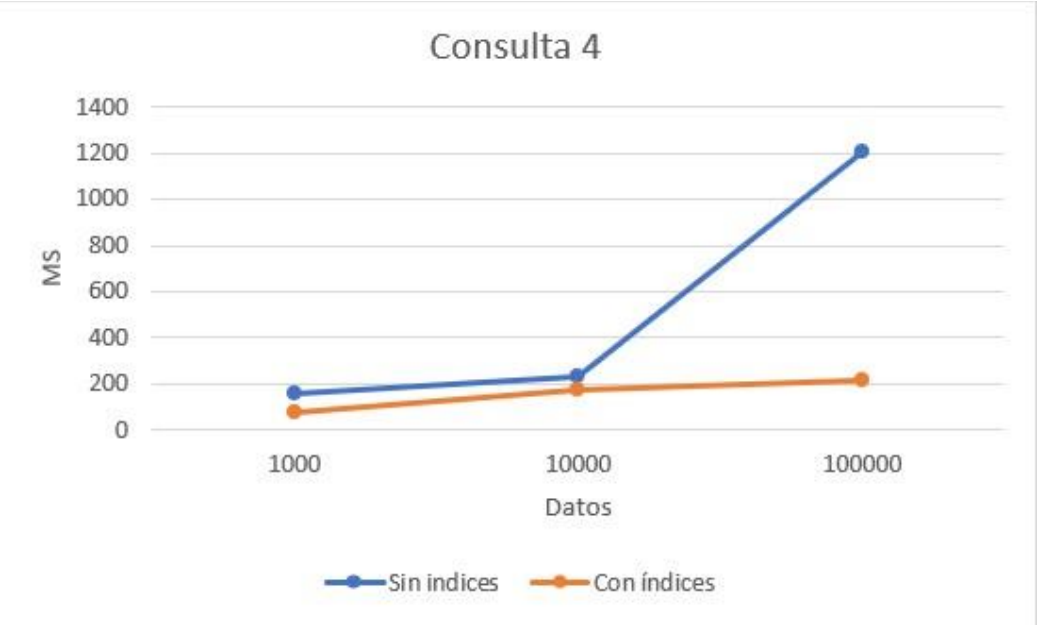
En esta consulta los tiempos no varían demasiado, esto se debe a que los rangos de retorno son bastante pequeños (50 tuplas), definitivamente eso también ayuda ya que, si más tuplas cumplieran con la condición, retorna todo más rápido.

5.6.3 Consulta 3



Al igual que en la consulta 2, los tiempos no varían mucho pero igual ayudan a las comparaciones y son más rápidos de todas formas. Y llega a decrecer el tiempo en 80 porciento cuando se prueba con 100000

5.6.4 Consulta 4



Finalmente, en esta consulta hay una reducción considerable de tiempos, esta´ optimización esta´ muy bien implementada con el join y el brete.

5.7 Análisis y Discusión

Como se puede ver, se ve una tendencia de reducción de tiempo mayor en las consultas 1 y 4, esto se debe a que en estas esta´ presente un join, que cuando ambas tablas están llenas de muchísimos datos demora bastante.

6 Extras

6.1 Vista

Para este contexto sabemos que se va a consultar todo el tiempo los pedidos que se tienen que entregar hoy y sus respectivas direcciones. Si además la orden ya fue cancelada y aún no ha sido entregada, se retorna la dirección y se procede a entregar las bolsas.

```
CREATE VIEW Entregas AS
    SELECT direccion
    FROM pedido,clientes
    WHERE fecha_entrega = current_date
    and cancelada='Y'
    and entregada='N';
```

6.2 Vista Materializada

Hasta el momento, el único comprobante que tenemos almacenado es factura, ya que se generar por medio de la SUNAT entonces definitivamente se guarda un registro. En cambio, cuando se trata de una boleta no necesariamente se almacenan los datos de esta. Para esto creamos una vista materializada que tenga la fecha, la cantidad de bolsas, el precio por bolsa y el IGV correspondiente.

```
CREATE MATERIALIZED VIEW Boleta
AS select fecha,cantidad,preciounidad,IGV
from pedido,factura;
```

6.3 Proceso almacenados

Para complementar la vista creada en la sección anterior, se define un procedimiento almacenado para crear una boleta que se puede llamar simplemente con los atributos en el orden correcto. Este procedimiento inserta los valores en la tabla boleta automáticamente con la fecha del presente día.

```
CREATE PROCEDURE crearboleta(emitida date, cant integer,
                                        precio_unitario real, Impuesto real)
LANGUAGE plpgsql
AS $$
BEGIN
    insert into boleta values(emitida,cant,precio_unitario,Impuesto);
    commit;
END;
$$;
```

6.4 Trigger

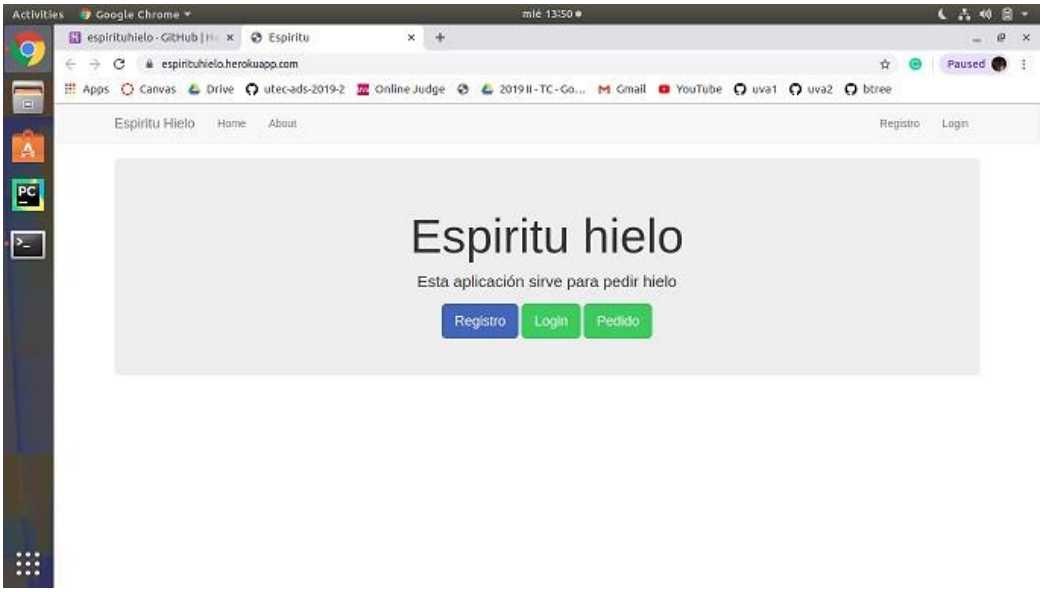
El problema que habilita la viabilidad de este trigger es saber que pedidos se tienen que entregar el presente día. Para esto se crea una tabla de pedidos hoy y se crea una función que inserta en esta tabla los valores almacenados en pedido si la fecha en la que se registra el pedido es la misma para la que se pide. Esto nos sirve para mantener el registro de pedidos hechos por día sin tener que hacer consultas.

```
create table pedidos_hoy(
hoy date,
cantidad integer)

CREATE OR REPLACE FUNCTION func() RETURNS TRIGGER AS $my_table$
   BEGIN
      INSERT INTO pedidos_hoy(book_id, entry_date) VALUES (current_date,new.cantidad);
      RETURN NEW;
   END;
$my_table$ LANGUAGE plpgsql;


CREATE TRIGGER pedidosdehoy AFTER INSERT ON Pedido
 EXECUTE PROCEDURE func();
```

6.5 Front End

Para atacar el problema de una manera más real, se creó una página web en la que se pueden registrar nuevos clientes y registrar pedidos. Consecuentemente, los datos almacenados en esta página web (así como la web también) están en el proveedor de servicios de computación de nube" HEROKU" esto nos permite divulgar mejor la aplicación y tener almacenado

los datos de una manera más segura y eficiente. Se encuentra en: https://espirituhielo.
herokuapp.com/



Se muestra las tablas de la base de datos en la nube, producción, users correspondiente a clientes y pedido.

```
francesco@francesco-HP-Laptop-15-bs1xx:~$ psql postgres://tkmqionogakzdx:a80c41f
6bf582ea300d0d70470ca51e7c5706f5857fa3b6af16c80b67760dadd@ec2-107-20-155-148.com
pute-1.amazonaws.com:5432/dfa2lqmmmvka0j
psql (10.10 (Ubuntu 10.10-0ubuntu0.18.04.1), server 11.6 (Ubuntu 11.6-1.pgdg16.0
4+1))
WARNING: psql major version 10, server major version 11.
         Some psql features might not work.
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 25
6, compression: off)
Type "help" for help.

dfa2lqmmmvka0j=> \dt
              List of relations
 Schema |    Name    | Type  |     Owner
--------+------------+-------+---------------
 public | pedidos    | table | tkmqionogakzdx
 public | produccion | table | tkmqionogakzdx
 public | users      | table | tkmqionogakzdx
(3 rows)
```

6.6 Pregunta Extra

¿Cuál sería la complejidad operacional si escalamos los datos por encima del millón?, realice una comparativa respecto a la cantidad de datos del párrafo anterior. ¿Es sucinte la arquitectura Cliente-Servidor para procesar millones de datos?

Respuesta: La arquitectura cliente-servidor se usa para la comunicación y distribución de tareas normalmente. La complejidad operacional con millones de datos seguirá aumentando mientras aumenten los datos. Si se sabe que la función de esta arquitectura es la comunicación, esta sería muy lenta con muchos datos, por lo que no es necesariamente adecuado. Si lo comparamos con los escenarios anteriores, donde la arquitectura cliente-servidor funciona bien, con millones de datos se perdería la funcionalidad de este sistema por completo. Sería necesario mucho espacio y mucho tiempo para que, con esta arquitectura, el sistema de base de datos continúe funcionando con normalidad.

7 Conclusiones

Conclusiones :

Durante todo el periodo de este trabajo se mantuvo una conversación con el dueño de la empresa para saber sus preferencias. Desde el inicio del proyecto, se generó un problema con respecto a las necesidades del cliente. El cliente, que tiene bajo conocimiento técnico, no sabía que quería y que le convenia implementar. Durante todo el proyecto he ido convirtiendo sus peticiones en implementaciones posibles. Luego de establecer bien los requerimientos, se procedió al diseño del sistema de base de datos. Durante esta parte se mantuvo conversación con el cliente y con los profesores, generándose así, por lo menos 3 borradores del sistema en general. Luego de la aprobación del cliente y los profesores se procedió a la experimentación.
Durante la experimentación, el cliente genero´ un par de consultas comunes y se agregaron las demás. Para demostrar la eficiencia del modelo planteado, se rebalso con datos el sistema y se ejecutaron las consultas. Se aplicaron índices que mejoraron considerablemente el performance. Sin embargo, se debe aclarar que esta no es la única manera de optimizar un sistema. En este contexto, por ejemplo, una optimización semántica o sintáctica pudo haber

resultado más eficiente, se probarán en el futuro definitivamente. Por último, hay que denotar todas las herramientas que se nos ha brindado para hacer este proyecto. Así, se implementaron funciones extra como posible modelo futuro. Se seguir ‘a trabajando con la empresa para lograr un sistema realmente eficiente y que ojalá tenga impacto sobre el crecimiento de esta.

Recomendaciones: Sería ideal tener un sistema lo suficientemente grande como para optimizarlo de una forma práctica y no solo experimental. Aunque esto definitivamente nos ayuda, podrían presentarse casos reales en los que no aplique nuestra optimización.

8 Referencias

1. https://www.postgresql.org/docs/
2. http://www.postgresqltutorial.com/
3. http://www.generatedata.com/
4. https://www.overleaf.com/learn