Entrega final. Desarrollo de Base de Datos.

PLATAFORMA DE INTERMEDIACIÓN ENTRE COMERCIOS GASTRONÓMICOS Y USUARIOS.

Por Germán G. Weckeser.

Capítulo 1.

1.- Temática.

La temática elegida para este proyecto es el modelo de negocios utilizado en las aplicaciones digitales conocidas popularmente como *"apps de delivery"*. En el caso concreto se tomó a la empresa Rappi.

1.- Objetivo.

El presente trabajo tendrá como principal objetivo el desarrollo de una base de datos que pueda ser utilizada en una plataforma digital que se dedica a la intermediación entre usuarios gastronómicos y los diversos establecimientos culinarios. Todo ello, facilitando su materialización permitiendo que una tercera parte (Rappi tendero) lleve de forma eficiente y rápida el pedido hecho por el cliente.

Los datos que serán almacenados en la base de datos que se construirá como trabajo final tendrá como objetivo almacenar datos cuantitativos y cualitativos. Todo ello, a los efectos de poder determinar estrategias de carácter empresarial.

2.- Hipótesis.

El objetivo del análisis que se llevará a cabo una vez finalizada la creación de la base de datos será analizar la tendencia de los consumidores, ya que el objetivo será poseer datos cualitativos tal como se señaló en el punto anterior. Todo ello, con un enfoque en los consumidores, su edad, género y zona donde estos residen.

Asimismo, se intentará analizar si existe o no una preferencia respecto a un producto en particular, como así también si es posible determinar un índice de cancelaciones de órdenes respecto a las órdenes que son efectivamente entregadas. En adición a ello, lograr entender cuál es el tiempo promedio en que se recepta la orden y esta es entregada.

Capítulo 2.

1.- Objetivo.

El objetivo principal es la creación de una base de datos que pueda ser utilizada por la organización con un esquema de negocio similar a la aplicación Rappi. Asimismo, como objetivo general del proyecto se busca hacer consultas a la base de datos que nos permitan tomar mejores decisiones a la hora de lanzar campañas publicitarias, de descuentos o tomar decisiones estratégicas para robustecer la presencia de la

empresa en ciertas zonas específicas donde la demanda de productos sea más elevada.

2.- Segmento a quien estaría dirigido el producto final del proyecto y objetivos específicos de este.

Sin perjuicio de que el producto final sería de utilización para la organización mencionada, la BD que se desarrolle se encuentra dirigida a los sectores estratégicos del negocio. Estos tomarán como base los datos vertidos en la BD para luego tomar decisiones estratégicas para el negocio. Tales como pueden ser:

1. Reclutar nuevos aliados gastronómicos (Restaurantes) acordes a las principales demandas del público.
2. Determinar cuáles son las áreas donde existe una mayor demanda. Buscando generar estímulos para que nuevos repartidores utilicen la plataforma para que funcione como nexo entre los usuarios, el repartidor y el consumidor final.
3. Poseer conocimiento de los usuarios a los efectos de determinar cuales son sus preferencias y apuntar adecuadamente los recursos de marketing, fidelización de la empresa. Entre otras campañas o futuros lanzamientos de ofertas.

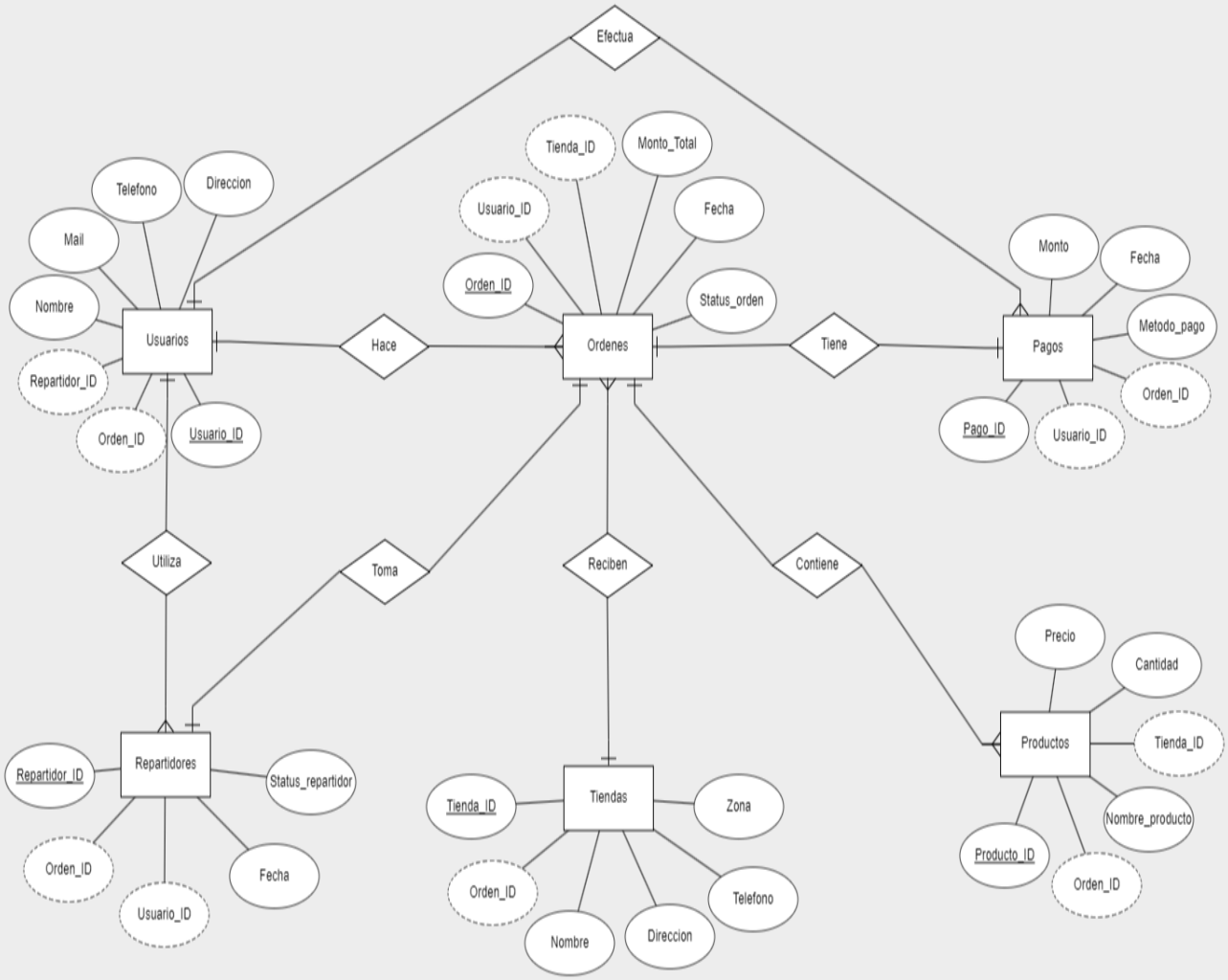
Capítulo 3.

1.- Posibles entidades que encontrarán incluidas en la BD desarrollado.

Las posibles entidades que formarían parte de la BD serían las siguientes:

1. Usuarios.
2. Repartidores.
3. Tiendas.
4. Productos.
5. Pagos.
6. Órdenes.

2.- Diagrama entidad relación (DER) a utilizar.



2.1.- Explicación DER.

En este apartado se procederá a desgranar cada una de las entidades aquí presentadas y se explicará cada uno de sus atributos.

2.1.1.- Órdenes.

Esta entidad se describe cada una de las órdenes que se hacen mediante la aplicación. El código de identificación denominado como "Orden_ID" es el correspondiente a una venta efectuada en una fecha específica.

Tipo de clave	Campo	Tipo de campo	Detalle
PK	Orden_ID	Int	Cada orden tiene un número asignado.

FK	Usuario_ID	Int.	Cada usuario posee un número de identificación. Permite la relación con la entidad "Usuarios".
-	Monto_total	Int.	Monto total de la orden efectuada.
FK	Tienda_ID	Int.	Cada tienda posee un número de identificación. Permite la relación con la entidad "Tienda".
-	Fecha	Date	Fecha en que se hizo la orden.
-	Status_orden	Varchar	Estado de la orden.

2.1.2.- Usuarios.

La entidad usuarios posee datos de los usuarios que utilizan la aplicación y efectúan pedidos mediante la aplicación.

Tipo de clave	Campo	Tipo de campo	Detalle
PK	Usuario_ID	Int.	Cada usuario posee un número asignado.
FK	Orden_ID	Int.	Numero que identifica la orden y funciona para relacionarla con la entidad"Ordenes".
FK	Repartidor_ID	Int.	Número que identifica al repartidor asignado y la relaciona con la entidad "Repartidores".

-	Nombre	Varchar	Nombre y apellido del usuario.
-	Mail	Varchar	Correo electrónico del usuario
-	Teléfono	Int.	Número de contacto.
-	Dirección	Varchar	Dirección con altura del domicilio.

2.1.3.- Repartidores.

Esta entidad contiene información sobre los repartidores que será asociado a un usuario y órdenes. Cada uno de ellos tiene asignado ID, mediante por el cual, se identifica a cada uno de ellos.

Tipo de clave	Campo	Tipo de campo	Detalle
PK	Repartidor_ID	Int.	Código numérico para identificar a cada repartidor.
FK	Orden_ID	Int.	Número que identifica la orden y funciona para relacionarla con la entidad “Ordenes”.
FK	Usuario_ID	Int.	Cada usuario posee un número de identificación. Permite la relación con la entidad “Usuarios”.
-	Fecha	Date	Fecha en que el repartidor toma la orden.
-	Status_repartidor	Varchar	Activo o inactivo dependiendo cantidad de órdenes

			tomadas en el mes.

2.1.4.- Tiendas.

La entidad contiene información de cada una de las tiendas o “aliados” que reciben las órdenes de los usuarios y luego son entregados por los repartidores en los domicilios indicados.

Tipo de clave	Campo	Tipo de campo	Detalle
PK	Tienda_ID	Int.	Número que identifica de forma única a cada una de las tiendas.
FK	Orden_ID	Int.	Número que identifica cada una de las órdenes. Permite la relación con la entidad “Ordenes”.
-	Nombre	Varchar	Nombre de cada una de las tiendas.
-	Dirección	Varchar	Dirección de la tienda con su altura.
-	Teléfono	Int.	Número de contacto de la tienda.
-	Zona	Varchar	Zona donde se encuentra la tienda.

2.1.5.- Productos.

Esta entidad contiene cada uno de los productos que se encuentran catalogados en la aplicación y que son susceptibles de ser comprados por los usuarios. Dichos productos deben ser ofrecidos por las tiendas que se encuentran en la aplicación.

Tipo de clave	Campo	Tipo de campo	Detalle
PK	Producto_ID	Int.	Número que individualiza a cada producto.
FK	Orden_ID	Int.	Numero que individualiza cada orden. Funciona para relacionar la entidad "Productos" con la entidad "Ordenes".
FK	Tienda_ID	Int.	Número identificatorio de la tienda. Relaciona la entidad con la entidad "Tiendas".
-	Nombre_producto	Varchar	Nombre del producto.
-	Cantidad	Int.	Cantidad de producto por orden.
-	Precio	Int.	Valor de cada unidad.

2.1.6.- Pagos.

En esta entidad se guardan los datos referenciados a los pagos efectuados por los usuarios en las compras efectuadas en los establecimientos gastronómicos.

Tipo de clave	Campo	Tipo de campo	Detalle
PK	Pago_ID	Int.	Es la identificación numérica de cada uno de los pagos efectuados.
FK	Usuario_ID	Int.	Identificación de cada uno de los usuarios y

			relaciona con la entidad “Usuarios”.
FK	Orden_ID	Int.	Identificación numérica de cada una de las órdenes y permite relacionarse con la entidad “Ordenes”.
-	Fecha	Date	Fecha en la que se efectuó el pago.
-	Método_pago	Varchar	Forma en la que se efectuó el pago de la orden.
-	Monto	Int.	Valor final de la compra efectuada.

3.-Modelo a utilizar en la base de datos. (DB)

Ante las interrogantes planteadas en capítulos anteriores y habiendo analizado y descripto el negocio debemos elegir un modelo adecuado para hacer las consultas que consideremos para el proceso que estamos analizando. Además de ello, debemos destacar que existen medidas cuantificables asociadas a estos.

Atento, a ello, como esquema de desarrollo de la base de datos en primer lugar hemos decidido desarrollar en SQL una tabla de hecho. En este caso, es la tabla “órdenes” dado que se identifica como un proceso nuclear del negocio de Rappi. Siendo a siempre vista que las órdenes son una operación vital y cotidiana para la organización, hemos considerado. Por ello, se ha decidido tomar como tabla de hecho este elemento de la base de datos al momento de desarrollarla.

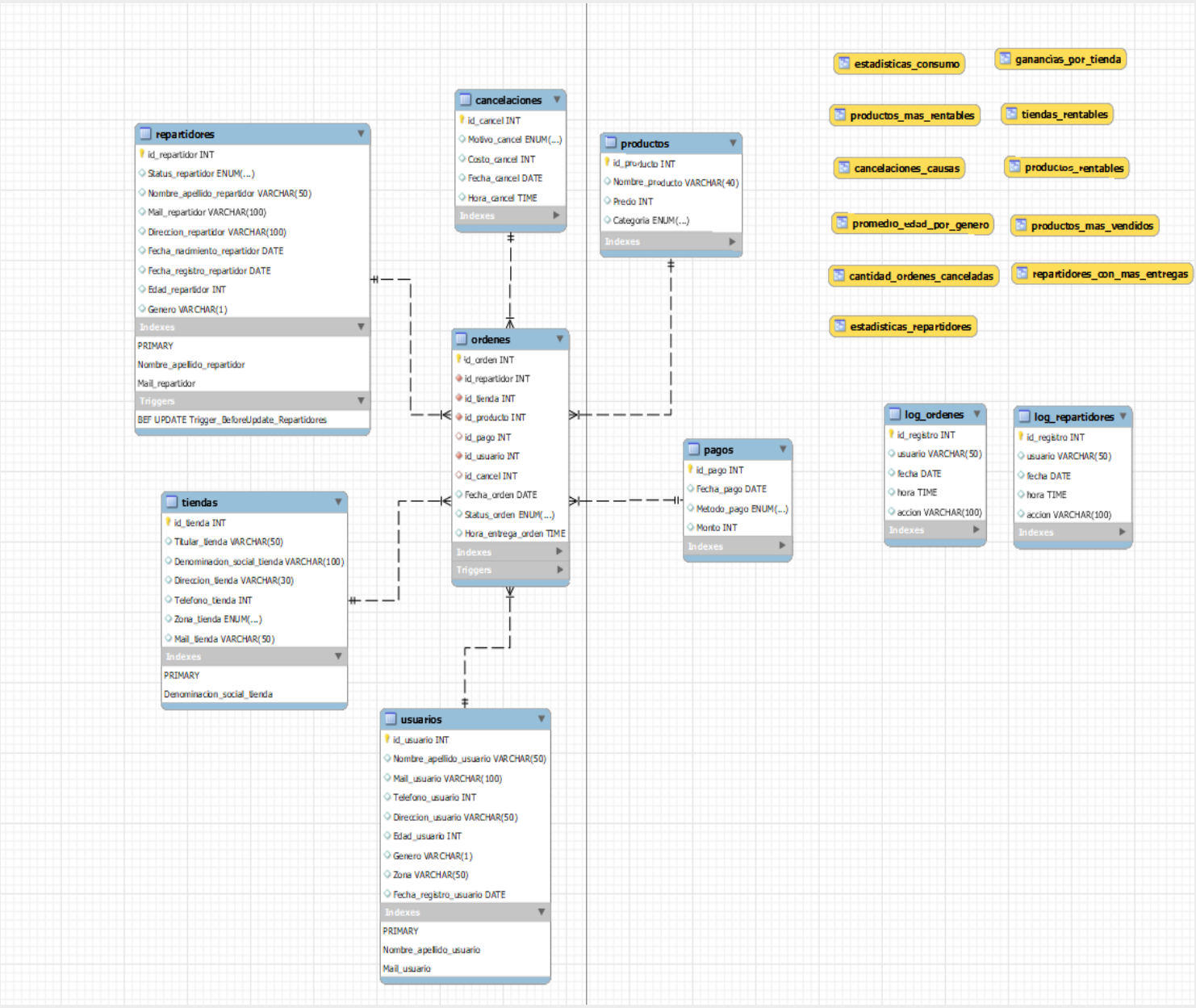
Teniendo en cuenta esto. Se ha considerado necesario crear tablas de dimensión que jugaran como vistas del proceso de ventas de la aplicación. Asimismo, la base de datos poseerá indicadores de un proceso del negocio, los cuales, son cuantificables y nos permitirán

medir nuestro proceso de negocios, tales como podrían ser los importes, las fechas etc.

Habiendo analizado todas estas cuestiones. Se ha considerado que para como esquema para estructurar los datos el más óptimo es el esquema en estrella.

En el centro de este, tendremos una tabla de hechos que será la tabla de “ordenes”. Alrededor de esta tendremos las tablas de dimensión “pagos”, “repartidores”, “productos”, “usuarios”, “tiendas” y “cancelaciones”.

Dado que, cada una de estas tablas dimensionales en definitiva componen un punto de vista de análisis que participa en la descripción del hecho que llamamos “ordenes”. A continuación, se adjunta foto del esquema utilizado.



4.- Tablas de la BD.

4.1.- Tabla órdenes.

CAMPO	TIPO DE CLAVE	TIPO DE CAMPO
id_orden	PK	Int.
id_repartidor	FK	Int.
id_tienda	FK	Int.
id_producto	FK	Int.
id_pago	FK	Int.
id_usuario	FK	Int.
id_cancel	FK	Int.
Fecha_orden	-	Date
Status_orden	-	ENUM ('Entregada','Cancelada')
Hora_entrega_orden	-	Time

4.2.- Tabla Pagos.

La tabla de dimensión “pagos” se relaciona con la tabla de hechos “ordenes” mediante su PK que funciona como FK en esta última. En la presente se contiene la información referente a los pagos efectuados.

CAMPO	TIPO DE CLAVE	TIPO DE CAMPO
Id_pago	PK	Int.
Fecha_pago	-	Date
Metodo_pago	-	ENUM ('Tarjeta de credito', 'Tarjeta de debito', 'Efectivo')
Monto	-	Int.

4.3.- Tabla Repartidores.

La tabla de dimensión “repartidores” se relaciones con la tabla de hechos “ordenes” mediante su PK que funciona como FK de esta última. En la presente tabla tendremos guardada la información referente a los repartidores.

CAMPO	TIPO DE CLAVE	TIPO DE CAMPO
Id_repartidor	PK	INT
Status_repartidor	-	ENUM ('Activo','Inactivo')
Nombre_apellido_repartidor	-	Varchar
Mail_repartidor	-	Varchar
Direccion_repartidor	-	Varchar
Fecha_nacimiento_repartidor	-	Date
Fecha_registro_repartidor	-	Date
Edad_repartidor	-	Int.
Genero	-	varchar

4.4.- Tabla productos.

La tabla de dimensión "productos" se relaciona con la tabla de hechos "ordenes" mediante su PK que funciona como FK en esta última. En la presente se contiene la información de los productos que son ordenados por los usuarios.

CAMPO	TIPO DE CLAVE	TIPO DE CAMPO
Id_prducto	PK	Int.
Nombre_producto	-	Varchar
Cantidad	-	Int.
Precio	-	Int.
Categoria	-	ENUM ('Alimentos y bebidas','Farmacia y cuidado personal','Hogar')

4.5.- Tabla usuarios.

La tabla de dimensión "usuarios" se relaciona con la tabla de hechos "ordenes" mediante su PK que funciona como FK en esta última. En la presente se contiene la información de los usuarios que efectúan órdenes a las diferentes tiendas.

CAMPO	TIPO DE CLAVE	TIPO DE CAMPO
Id_usuario	PK	-
Nombre_apellido_usuario	-	Varchar
Mail_uasuario	-	Varchar
Telefono_usuario	-	Int.
Direccion_usuario	-	Varchar.
Edad_usuario	-	Int
Genero	-	Varchar (1)

Zona	-	Varchar.
Fecha_registro_usuario	-	Date

4.6.- Tabla tiendas.

La tabla de dimensión “tiendas” se relaciona con la tabla de hechos “ordenes” mediante su PK que funciona como FK en esta última. En la presente se contiene la información de las tiendas que preparan las órdenes que son efectuadas por los usuarios para entregarlas a los repartidores y posteriormente que estos lo entreguen al usuario.

CAMPO	TIPO DE CLAVE	TIPO DE CAMPO
Id_teinda	PK	Int.
Titular_tienda	-	Varchar
Denominación_social_tienda	-	Varchar
Direccion_tienda	-	Varchar
Telefono_tienda	-	Int.
Zona_tienda	-	ENUM ('Recoleta','Caballito','Almagro','Colegiales')
Mail_tienda	-	Int.

4.7.- Tabla cancelaciones.

La tabla de dimensión “cancelaciones” se relaciona con la tabla de hechos “ordenes” mediante su PK que funciona como FK en esta última. En la presente se contiene la información de las cancelaciones que pueden ser efectuadas tanto por las tiendas, el usuario o el mismo repartidor.

CAMPO	TIPO DE CLAVE	TIPO DE CAMPO
Id_cancel	PK	Int.
Motivo_cancel	-	ENUM ('Falta disponibilidad repartidor', 'Cancelacion del usuario', 'Mal tiempo', 'Cancelacion de la tienda')
Costo_cancel	-	Int.
Fecha_cancel	-	Date
Hora_cancel	-	Time

Capítulo 4. Objetos de la base de datos “Rappi”.

1.- Vistas.-

1.1.- Vista “Estadísticas repartidores”.

La vista en cuestión se va utilizar para conocer las estadísticas de las repartidores. El objeto de esta es saber la cantidad de repartidores activos e inactivos que se encuentran en la organización.

Esta vista tiene como finalidad comprender si existe una relación entre el rango etario del repartidor y cual es el grupo con mayor inactividad. Todo ello, con la finalidad de poder resolver si es necesario ampliar o no la flota disponible para cumplir con las órdenes.

Código utilizado:

```
CREATE VIEW estadisticas_repartidores AS

SELECT

  (SELECT COUNT(*) FROM Repartidores WHERE Status_repartidor = 'Activo') AS RepartidoresActivos,

  (SELECT COUNT(*) FROM Repartidores WHERE Status_repartidor = 'Inactivo') AS RepartidoresInactivos

FROM

  dual;
```

1.2.- Vista “Tiendas rentables”.

Tal como lo indica su nombre el objeto de la vista es conocer la tienda más rentable de todo el universo disponible de estas que venden productos para que sean entregados por los repartidores. En esta, podremos ver el nombre de la tienda y sus correspondientes ingresos.

Con esta información es posible establecer tiers entre las diversas tiendas y conocer cuales de ellas son las MVP para atraer clientes a la plataformas. De esta forma poder firmar contratos de exclusividad con estas captando su universo de clientes.

Código utilizado:

```
CREATE VIEW tiendas_rentables AS

SELECT

  Tiendas.Denominacion_social_tienda,
```

```
    SUM(Productos.Precio) AS IngresosTotales
FROM
    Tiendas
JOIN Ordenes ON Tiendas.id_tienda = Ordenes.id_tienda
JOIN Productos ON Ordenes.id_producto = Productos.id_producto
GROUP BY Tiendas.Denominacion_social_tienda
ORDER BY IngresosTotales DESC;
```

1.3.- Vista "Repartidores con más entregas".

Esta vista tiene por objetivo saber cual de todos los repartidores han hecho más entregas. De esta forma podemos conocer el comportamiento de los repartidores con mayor cantidad de entregas, pudiendo deducir su mayor disponibilidad que otros repartidores.

Código utilizado:

```
CREATE VIEW repartidores_con_mas_entregas AS
SELECT r.Nombre_apellido_repartidor, COUNT(*) AS Cantidad_Entregas
FROM Ordenes o
JOIN Repartidores r ON o.id_repartidor = r.id_repartidor
WHERE o.Status_orden = 'Entregada'
GROUP BY r.Nombre_apellido_repartidor
ORDER BY Cantidad_Entregas DESC;
```

1.4.- Vista "Productos más vendidos".

La vista en cuestión nos proporciona una estadística de los productos más vendidos por las tiendas. El criterio utilizado fue la mayor cantidad de ventas que estas obtuvieron entendiendo si es un producto que el cliente consume de forma unitaria o hace compras a mayor volumen. De esta forma podemos crear una estrategia de segmentación y estrategia de marketing desde la plataforma para fomentar la venta de estos productos populares. Asimismo, podríamos calcular la periodicidad con que se pedirán estos productos por los consumidores finales.

Código utilizado:

```
CREATE VIEW productos_mas_vendidos AS

SELECT p.Nombre_producto, COUNT(*) AS Cantidad_Vendida

FROM Ordenes o

JOIN Productos p ON o.id_producto = p.id_producto

WHERE o.Status_orden = 'Entregada'

GROUP BY p.Nombre_producto

ORDER BY Cantidad_Vendida DESC;
```

1.5.- Vista "Productos más rentables".

Similar a la vista anterior solo que el criterio que se utilizó cambia. El criterio utilizado es el dinero que ingresó cada producto. De esta forma, podemos hacer un estudio detallado sobre el consumidor conociendo su poder adquisitivo y de esta forma buscar crear un status en este cuando se consuma dicho producto.

Código utilizado:

```
CREATE VIEW productos_mas_rentables AS

SELECT p.Nombre_producto, SUM(p.Precio) AS Ingreso_Total

FROM Ordenes o

JOIN Productos p ON o.id_producto = p.id_producto

GROUP BY p.Nombre_producto

ORDER BY Ingreso_Total DESC;
```

1.6.- Vista "Cantidad ordenes canceladas".

Mediante esta vista podremos sustraer estadísticas de órdenes canceladas. Esto nos permite no solo estimar una cantidad de pérdida o desechos que se producen por las órdenes canceladas. En un futuro sería interesante poder segmentar esta información por periodo de años.

Código utilizado:

```
CREATE VIEW cantidad_ordenes_canceladas AS

SELECT COUNT(*) AS CantidadCanceladas
```

```
FROM Ordenes

WHERE Status_orden = 'Cancelada';
```

1.7. Vista “Cancelaciones causas”.

Esta vista nos proporcionará estadística de motivos de las cancelaciones de las órdenes. Al igual que la otra vista nos permite conocer de forma eficaz la mayor causa de cancelación de las órdenes. De esta forma se podría establecer patrones e intentar ajustar las causas que pueden ser imputables a la organización previniendo la perdida de futuros ingresos.

Código utilizado:

```
CREATE VIEW cancelaciones_causas AS

SELECT COUNT(*) AS CantidadCanceladas, Motivo_cancel

FROM Ordenes

JOIN Cancelaciones ON Ordenes.id_cancel = Cancelaciones.id_cancel

WHERE Ordenes.Status_orden = 'Cancelada'

GROUP BY Motivo_cancel

ORDER BY CantidadCanceladas DESC

LIMIT 1;
```

1.8.- Vista “Ganancias por tienda”.

Con esta vista se puede conocer las ganancias de cada una de las tiendas. Esta información podría ser útil para crear nuevas alianzas con tiendas similares, tanto en su estilo, concepto o productos que estas venden. De esta forma se podría ampliar la red de tiendas que trabajen con Rappi.

Código utilizado:

```
CREATE VIEW ganancias_por_tienda AS

SELECT Tiendas.Denominacion_social_tienda, SUM (Pagos.Monto) AS Ganancias

FROM Ordenes

JOIN Pagos ON Ordenes.id_pago = Pagos.id_pago

JOIN Tiendas ON Ordenes.id_tienda = Tiendas.id_tienda

GROUP BY Tiendas.Denominacion_social_tienda;
```

1.9.- Vista “Promedio por género”.

Con esta vista podemos conocer el promedio de edad por género de los usuarios de la aplicación. Con esta información podríamos crear una ssegmentación de una campaña de marketing orientado a un rango de personas que se encuentren en ese promedio de edad.

Código utilizado:

```
CREATE VIEW promedio_edad_por_genero AS

SELECT Genero, AVG(Edad_usuario) AS PromedioEdad

FROM Usuarios

GROUP BY Genero;
```

1.10.- Vista de “Estadísticas de consumo”.

Con esta vista podemos conseguir una estadística de consumo, encontraremos información de género, gasto promedio y el producto más comprado. Con esta vista podemos obtener una información más amplia en donde tendremos un gasto promedio del consumidor dividido por género y producto consumido.

Código utilizado:

```
CREATE VIEW estadisticas_consumo AS

SELECT

   Usuarios.Genero,

   AVG(Pagos.Monto) AS GastoPromedio,

   SUBSTRING_INDEX(GROUP_CONCAT(DISTINCT
Productos.Nombre_producto ORDER BY Cantidad DESC SEPARATOR ', '), ',', 1)
AS 'Producto mas consumido'

FROM

   Usuarios

JOIN Ordenes ON Usuarios.id_usuario = Ordenes.id_usuario

JOIN Pagos ON Ordenes.id_pago = Pagos.id_pago

JOIN (

   SELECT

      Ordenes.id_usuario,
```

```
    Ordenes.id_producto,
    COUNT(*) AS Cantidad
  FROM
    Ordenes
  GROUP BY Ordenes.id_usuario, Ordenes.id_producto
) AS Subconsulta ON Usuarios.id_usuario = Subconsulta.id_usuario
JOIN Productos ON Subconsulta.id_producto = Productos.id_producto
GROUP BY Usuarios.Genero;
```

1.11.- Vista "Productos rentable".

Esta vista permite visualizar ingresos totales generados por los productos mas vendidos y su correspondiente categoría.

Código utilizado:

```
CREATE VIEW productos_rentables AS
SELECT
  Productos.Categoria,
  Productos.Nombre_producto,
  SUM(Productos.Precio) AS IngresosTotales
FROM
  Productos
JOIN Ordenes ON Productos.id_producto = Ordenes.id_producto
GROUP BY Productos.Categoria, Productos.Nombre_producto
HAVING IngresosTotales = (
  SELECT
    MAX(IngresosTotales)
  FROM
```

```
    (
      SELECT
        Productos.Categoria,
        Productos.Nombre_producto,
        SUM(Productos.Precio) AS IngresosTotales
      FROM
        Productos
      JOIN Ordenes ON Productos.id_producto = Ordenes.id_producto
      GROUP BY Productos.Categoria, Productos.Nombre_producto
    ) AS TotalesPorCategoria
  WHERE TotalesPorCategoria.Categoria = Productos.Categoria
)
ORDER BY Productos.Categoria;
```

2.- Funciones.

2.1.- Función "Total órdenes entregadas".

Esta función tiene como finalidad ingresar el ID de un repartidor para conocer cuales fueron las cantidades de órdenes que este hizo. Asimismo, debemos tener en cuenta que puede devolver valores null dado que debe cumplirse que el repartidor se encuentre activo.

Código utilizado:

```
DELIMITER //
CREATE FUNCTION Total_Ordenes_Entregadas(id_repartidor_param INT)
RETURNS VARCHAR(100)
READS SQL DATA
BEGIN
  DECLARE total_ordenes INT;
  DECLARE nombre_repartidor VARCHAR(50);
```

```
DECLARE result VARCHAR(100);


SELECT COUNT(*), r.Nombre_apellido_repartidor INTO total_ordenes, nombre_repartidor
FROM Ordenes o
JOIN Repartidores r ON o.id_repartidor = r.id_repartidor
WHERE o.id_repartidor = id_repartidor_param AND o.Status_orden = 'Entregada'
GROUP BY r.Nombre_apellido_repartidor;


SET result = CONCAT('Total de órdenes entregadas: ', total_ordenes, ', Nombre del repartidor: ', nombre_repartidor);
RETURN result;
END //
DELIMITER ;
```

2.2.- Función “Promedio edad usuarios”.

Esta función nos permite calcular el promedio de edad de los usuarios segmentado por barrio. Para que funcione es necesario ingresar los barrios circunscriptos en este caso serían **“Recoleta, Almagro, Caballito o Agronomía”.**

Código utilizado:

```
DELIMITER //
CREATE FUNCTION Promedio_Edad_Usuarios(zona_param VARCHAR(50))
RETURNS DECIMAL(5,2)
READS SQL DATA
DETERMINISTIC
BEGIN
DECLARE promedio_edad DECIMAL(5,2);
```

```
    SELECT AVG(Edad_usuario) INTO promedio_edad
    FROM Usuarios
    WHERE Zona = zona_param;

    RETURN promedio_edad;
END //
DELIMITER ;
```

2.3.- Función "Total Tiendas zona".

Esta función permite ingresando la zona determinada conocer cuantas tiendas existen en un área determinada. Esto facilita la zonificación de las tiendas sabiendo y conociendo que clase de consumo tendrán por zonas. Para que funcione deben utilizarse los barrios **"Recoleta, Almagro, Caballito, Colegiales o Agronomía"**

Código utilizado:

```
DELIMITER //
CREATE FUNCTION Total_Tiendas_Zona(zona_param VARCHAR(50))
RETURNS INT
READS SQL DATA
BEGIN
    DECLARE total_tiendas INT;

    SELECT COUNT(*) INTO total_tiendas
    FROM Tiendas
    WHERE Zona_tienda = zona_param;

    RETURN total_tiendas;
END //
DELIMITER ;
```

2.4.- Función “Detalle producto”.

Mediante esta función podemos conocer un producto y cual es la tienda que lo vendió ingresando el ID de la orden.

Código utilizado:

```
DELIMITER //

CREATE FUNCTION Detalles_Producto(id_orden_param INT) RETURNS VARCHAR(200)

READS SQL DATA

BEGIN

  DECLARE producto_info VARCHAR(200);

  DECLARE nombre_producto VARCHAR(40);

  DECLARE nombre_tienda VARCHAR(100);


  SELECT p.Nombre_producto, t.Denominacion_social_tienda INTO nombre_producto, nombre_tienda

  FROM Ordenes o

  INNER JOIN Productos p ON o.id_producto = p.id_producto

  INNER JOIN Tiendas t ON o.id_tienda = t.id_tienda

  WHERE o.id_orden = id_orden_param;


  SET producto_info = CONCAT('Nombre del producto: ', nombre_producto, ', Tienda: ', nombre_tienda);

  RETURN producto_info;

END //

DELIMITER ;
```

3.- Stored Procedures.

3.1.- Stored Procedure “Insert repartidor”.

Mediante este SP se buza simplificar la inserción de un registro dentro de la tabla de repartidores, cada vez que uno se de de alta. Este procedimiento ahorrará tiempo a los efectos de la nueva carga.

Código utilizado:

```
DELIMITER //

CREATE PROCEDURE Insertar_Repartidor(

   IN status_param ENUM('Activo', 'Inactivo'),

   IN nombre_apellido_param VARCHAR(50),

   IN mail_param VARCHAR(100),

   IN direccion_param VARCHAR(100),

   IN fecha_nacimiento_param DATE,

   IN fecha_registro_param DATE,

   IN edad_param INT,

   IN genero_param VARCHAR(1)

)

BEGIN

   INSERT INTO Repartidores (Status_repartidor, Nombre_apellido_repartidor, Mail_repartidor, Direccion_repartidor, Fecha_nacimiento_repartidor, Fecha_registro_repartidor, Edad_repartidor, Genero)

   VALUES (status_param, nombre_apellido_param, mail_param, direccion_param, fecha_nacimiento_param, fecha_registro_param, edad_param, genero_param);

   SELECT LAST_INSERT_ID() AS id_repartidor;

END //

DELIMITER ;
```

CALL Insertar_Repartidor('Activo', 'Felix Ruano', 'felix@example.com', 'Calle 1928', '1991-01-01', '2022-01-01', 32, 'M');

4.2.- Stored procedure “Ordenar tabla”.

Permite indicar el campo de ordenamiento de una tabla y el tipo de orden mediante parametros. Para que funcione debe ponerse en el parametro 'nombreDetabla' (por ejemplo: Ordenes) 'nombreDecampo' (por ejemplo: Status_orden) y ASC o DESC.

Código utilizable:

```
DELIMITER //

CREATE PROCEDURE Ordenar_Tabla(

  IN nombre_tabla_param VARCHAR(100),

  IN nombre_campo_param VARCHAR(100),

  IN tipo_orden_param VARCHAR(10)

)

BEGIN

  SET @ordenamiento = CONCAT(nombre_campo_param, ' ', tipo_orden_param);

  SET @query = CONCAT('SELECT * FROM ', nombre_tabla_param, ' ORDER BY ', @ordenamiento);

  PREPARE stmt FROM @query;

  EXECUTE stmt;

  DEALLOCATE PREPARE stmt;

END //

DELIMITER ;

CALL Ordenar_Tabla('ordenes', 'Status_orden', 'ASC');
```

4.3.- Stored Procedure “Actualizar estado repartidor”.

Si bien su nombre es autoexplicativo, este stored procedure tiene como finalidad la actualización de la tabla “repartidores”, pero específicamente su

estado de “Activo” o “Inactivo”. La idea de este objeto es evitar la tediosa tarea de modificar dicho registro.

Código utilizado:

```
DELIMITER //

CREATE PROCEDURE Actualizar_Estado_Repartidor(

   IN id_repartidor_param INT,

   IN nuevo_estado_param ENUM('Activo', 'Inactivo')

)

BEGIN

   UPDATE Repartidores

   SET Status_repartidor = nuevo_estado_param

   WHERE id_repartidor = id_repartidor_param;


   SELECT ROW_COUNT() AS filas_actualizadas;

END //

DELIMITER ;

CALL Actualizar_Estado_Repartidor(1, 'Activo');
```

4.4.- Stored procedure “Insertar pagos”.

Este objeto tiene por finalidad insertar pagos en la tabla “pagos”, pudiendo hacerse varias filas en una sola vez. Cuestión que guardará cierta utilidad dado que pueden existir pagos que se efectúen en mismo momentos y por los mismo montos pero asignados a distintas órdenes.

Código utilizado:

```
DELIMITER //

CREATE PROCEDURE Insertar_Pagos()

BEGIN

   -- Primera fila

   INSERT INTO Pagos (Fecha_pago, Metodo_pago, Monto)
```

```
    VALUES ('2023-06-07', 'Tarjeta de crédito', 100);

    -- Segunda fila
    INSERT INTO Pagos (Fecha_pago, Metodo_pago, Monto)
    VALUES ('2023-06-08', 'Tarjeta de débito', 150);

    -- Tercera fila
    INSERT INTO Pagos (Fecha_pago, Metodo_pago, Monto)
    VALUES ('2023-06-09', 'Efectivo', 200);
END //
DELIMITER ;
CALL Insertar_Pagos();
```

5.- Triggers.

5.1.- Trigger que afecta a la tabla de "repartidores" (BEFORE).

Controlará la acción de actualización en la tabla Repartidores. Registrará en el Log_Repartidores el usuario que realiza la actualización, la fecha, la hora y la acción ("Actualización"). Antes de que se realice una actualización en la tabla Repartidores, se ejecutará este trigger y se insertará un registro en el Log_Repartidores.

Código utilizado:

```
DELIMITER //
CREATE TRIGGER Trigger_BeforeUpdate_Repartidores
BEFORE UPDATE ON Repartidores
FOR EACH ROW
BEGIN
    INSERT INTO Log_Repartidores (usuario, fecha, hora, accion)
    VALUES (USER(), CURDATE(), CURTIME(), 'Actualización');
```

```
END //

DELIMITER ;
```

5.2.- Trigger que afecta a la tabla "órdenes" (AFTER).

Controlará la acción de inserción en la tabla Ordenes. Registrará en el Log_Ordenes el usuario que realiza la inserción, la fecha, la hora y la acción ("Inserción"). Después de que se realice una inserción en la tabla Ordenes, se ejecutará este trigger y se insertará un registro en el Log_Ordenes.

Código utilizado:

```
DELIMITER //

CREATE TRIGGER Trigger_AfterInsert_Ordenes

AFTER INSERT ON Ordenes

FOR EACH ROW

BEGIN

   INSERT INTO Log_Ordenes (usuario, fecha, hora, accion)

   VALUES (USER(), CURDATE(), CURTIME(), 'Inserción');

END //

DELIMITER ;
```

Capítulo 5. Conclusiones.

A partir de la información creada para este trabajo (la cual es ficticia) procedo a hacer un pequeño análisis de los datos insertados.

En primer lugar, se puede concluir que la principal causa de cancelaciones de órdenes responde a cancelaciones de pare del usuario. Esto nos puede permitir intuir de forma más o menos certera que el tiempo entre el pedido que se hace en la aplicación, la preparación del producto y la entrega en sí es demasiado elevada para que llegue efectivamente al destinatario final.

Por lo tanto, como primera propuesta, debería intentarse buscar una mejora en los tiempos de reacción o más bien utilizar advertencias en la aplicación para que el usuario previo a efectuar el pedido pueda conocer de forma asertiva y eficiente que su pedido tardará un tiempo aproximado. De esta forma, se da la opción al usuario de cancelar la orden antes de que transcurra demasiado tiempo y luego se pierda el producto (dependiendo de cual se

trate) o que la empresa tenga que abonar al aliado la suma del producto por la cancelación.

Como segundo elemento de análisis es interesante analizar cuales son los productos más consumidos, pero agregando el factor del género. Como primera información obtuvimos que los usuarios de género femenino eligen como producto principal producto los huevos y los de género masculino consumen aceite de oliva. Por otra parte, el gasto promedio de los usuarios femeninos es 63,88$ contra los 71,15$ de los usuarios masculinos.

A partir de esta información, podemos inducir que los principales productos consumidos pertenecen a rubros que podríamos definir como *"de uso cotidiano"*. Entonces, a partir de esto, podemos observar que sería conveniente que se invierta para mejorar la plataforma de negocio "*turbo"* que se encuentra apuntada a resolver rápidamente la necesidad del usuario para estos tipos de productos y la necesidad del usuario es más inmediata y se necesita satisfacerla en un periodo corto de tiempo.

Como otro elemento de análisis poseemos la posibilidad de segmentar a los usuarios por género y rango etario. Como información que se posee es que los usuarios de género masculino tienen un promedio de edad de 32,14 años y los de género femenino tienen un promedio de 28,57 años.

En función de ello, puede concluirse que las campañas de publicidad que se implementen en un futuro deben ir apuntadas a usuarios jóvenes ya establecidos laboralmente y que residen tejidos urbanos densos para que los recorridos de entrega sean cortos y no sea demasiado largo el recorrido que debe efectuar el repartidor.