ENTREGA FINAL

**ComidappBD**

Introducción:

Las necesidades básicas de las personas han sido un pilar importante para que la tecnología pueda sacar provecho de sus funcionalidades, Comidapp es un software orientado a esos comercios que no pueden tener su medio tecnológico para poder progresar como empresa, este modelo de datos esta basado en un software donde esos comercios de comida callejeros puedan generar ingresos mediante las ventas web, esta idea salió de un proyecto que estoy realizando por fuera de coder y me gustaría sacarle provecho en esta plataforma.

Objetivo:

El objetivo principal de este proyecto es poder diseñar el modelo de datos que permita a los usuarios y clientes de la app poder registrarse y promocionar sus locales mediante esta aplicación, donde se podría respaldar, recuperar y hacer un análisis del rendimiento del mismo local, esto consiste en un administrador del sitio web que confirme o rechace solicitudes de los clientes que quieran unirse a este servicio, el local debería cumplir requisitos mínimos para tener el ingreso y principalmente que este en regla.

Situación problemática:

En el avance tecnológico ya se inventaron ideas y soluciones para problemas de necesidad básica, en este caso ya existe muchos softwares que permiten tener la comida al alcance de tu celular pero por lo menos en Uruguay están dejados de lados los comercios de comida callejera, habiendo muchos comercios que ya están fijos en lugares y habilitados por la Intendencia de Montevideo no pueden contar con este servicio por requisitos que muchos no cumplen, este espacio les permitiría a los comercios poder expandirse al área tecnológico. Esta base de datos esta pensada para este sistema en especial, los requisitos que necesito y lo que creo que seria mas seguro y sencillo para administrar los datos

Modelo Entidad Relación (MER.)

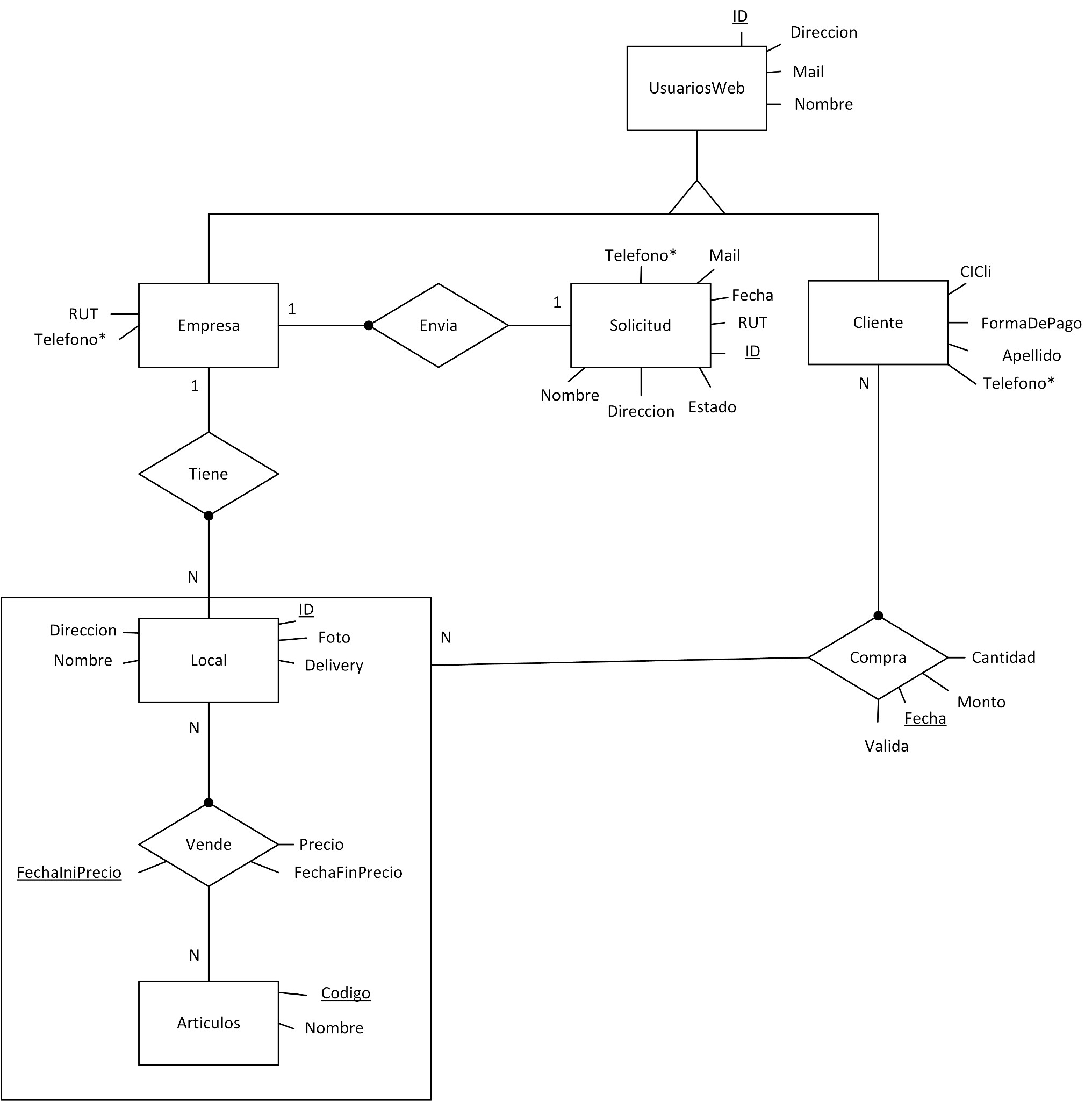
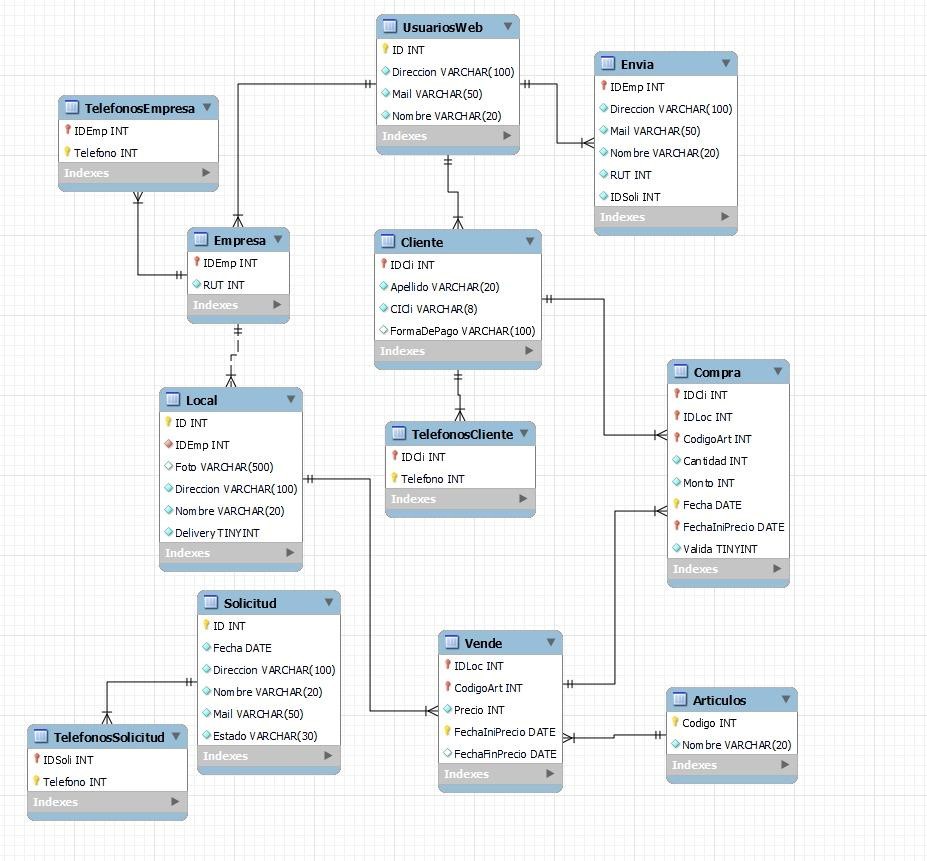


DIAGRAMA ENTIDAD RELACIÓN:



**RELACIONES DE LAS TABLAS**

UsuariosWeb:

* **PRIMARY KEY**: ID

Cliente:

* **PRIMARY KEY**: IDCli
* **FOREIGN KEY**: IDCli refiere a UsuariosWeb.ID

Solicitud:

* **PRIMARY KEY**: ID

Empresa:

* **PRIMARY KEY**: IDEmp
* **FOREIGN KEY**: IDEmp refiere a UsuariosWeb.ID

TelefonosCliente:

* **PRIMARY KEYS**: IDCli, Telefono
* **FOREIGN KEY**: IDCli refiere a Cliente.IDCli

TelefonosSolicitud:

* **PRIMARY KEYS**: IDSoli, Telefono
* **FOREIGN KEY**: IDSoli refiere a Solicitud.ID

TelefonosEmpresa:

* **PRIMARY KEYS**: IDEmp, Telefono
* **FOREIGN KEY**: IDEmp refiere a Empresa.IDEmp

Local:

* **PRIMARY KEY**: ID
* **FOREIGN KEY**: IDEmp refiere a Empresa.IDEmp

Artículos:

* **PRIMARY KEY**: Codigo

Envía:

* **PRIMARY KEY**: IDEmp
* **FOREIGN KEYS**: IDEmp refiere a UsuariosWeb.ID, IDSoli refiere a Solicitud.ID

Vende:

* **PRIMARY KEYS**: IDLoc, CodigoArt, FechaIniPrecio
* **FOREIGN KEYS**: IDLoc refiere a Local.ID, CodigoArt refiere a Articulos.Codigo

Compra:

* **PRIMARY KEYS**: IDCli, IDLoc, CodigoArt, Fecha, FechaIniPrecio
* **FOREIGN KEYS**: IDCli refiere a Cliente.IDCli, IDLoc, CodigoArt, FechaIniPrecio refieren a Vende.IDLoc, Vende.CodigoArt, Vende.FechaIniPrecio

Vistas

**SUCURSAL QUE VENDE MAS**

Esta vista esta echa para localizar cual es la sucursal que mas vende, usamos un **join** para vincular el local con las ventas. Este dato nos ayuda a tener una idea de cual es la sucursal que mejor le va en el rubro de la comida callejera dentro de nuestra app

**CREATE VIEW** SucursalMasVende **AS**

**SELECT**

L.ID **AS** IDLocal,

L.Nombre **AS** NombreSucursal,

**COUNT**(V.IDLoc) **AS** TotalVentas

**FROM**

Local L

**JOIN**

Vende V **ON** L.ID = V.IDLoc

**GROUP BY**

L.ID, L.Nombre

**ORDER BY**

TotalVentas **DESC**

**LIMIT** 1;

**ARTICULO MAS VENDIDO**

En esta vista tenemos los artículos mas vendidos dentro de la aplicación donde vinculamos la cantidad de articulo vendido, esto nos ayuda a tener una idea de cual es el articulo que mas se vende, esto nos ayuda a sacar ofertas o saber los gustos de nuestros clientes

**CREATE VIEW** ArticulosMasVendidos **AS**

**SELECT**

A.Nombre **AS** NombreArticulo,

**COUNT**(V.CodigoArt) **AS** CantidadVendida

**FROM**

Vende V

**JOIN**

Articulos A **ON** V.CodigoArt = A.Codigo

**GROUP BY**

A.Nombre

**ORDER BY**

CantidadVendida **DESC**;

Funciones

Necesitamos una función que sume cuantos clientes tenemos y cuantas empresas, esto nos ayuda a tener un control dentro de nuestro sistema, sacamos los datos de las tablas de empresa y clientes.

**CREATE FUNCTION** TotalEmpresasYClientes()

**RETURNS** VARCHAR(100)

**DETERMINISTIC**

**BEGIN**

**DECLARE** totalEmpresas **INT**;

**DECLARE** totalClientes **INT**;

**DECLARE** result **VARCHAR**(100);

(CONTANDO NUMERO DE CLIENTES Y EMPRESAS)

**SELECT** **COUNT**(\*) **INTO** totalEmpresas **FROM** Empresa;

**SELECT** **COUNT**(\*) **INTO** totalClientes **FROM** Cliente;

(PARA QUE EL RESULTADO SALGA EN CADENA MOSTRANDO UNA RETURN AL LADO DEL OTRO)

**SET** result = **CONCAT**('Total de Empresas: ', totalEmpresas, ', Total de Clientes: ', totalClientes);

**RETURN** result;

**END** //

Esta otra función salió para le necesidad de saber una estadística de si los clientes pagan con tarjeta o con efectivo para sacar promociones para la gente usa tarjetas o efectivo y tener un promedio, se cuenta de las tablas de clientes las formas de pago.

**CREATE FUNCTION** FormasDePagosCuenta()

**RETURNS VARCHAR**(100)

**DETERMINISTIC**

**BEGIN**

**DECLARE** tarjeta\_count **INT**;

**DECLARE** efectivo\_count **INT**;

**DECLARE** result **VARCHAR**(100);

**SELECT** **COUNT**(\*) **INTO** tarjeta\_count

**FROM** Cliente

**WHERE** FormaDePago = 'Tarjeta';

**SELECT** **COUNT**(\*) **INTO** efectivo\_count

**FROM** Cliente

**WHERE** FormaDePago = 'Efectivo';

**SET** result = CONCAT('Clientes que pagan con tarjeta: ', tarjeta\_count, '. Clientes que pagan con efectivo: ', efectivo\_count);

**RETURN** result;

**END** //

Stored Procedures

**Actualizar estado de solicitud**

Este SP se creo con el fin de poder actualizar el estado de una solicitud para corregir errores si una función del backend no llega a funciona o falla algún servicio.

**DELIMITER //**

**CREATE PROCEDURE** ActualizarEstadoSolicitud(

**IN** p\_IDSoli **INT**,

**IN** p\_NuevoEstado **VARCHAR**(30)

)

**BEGIN**

**UPDATE** Solicitud

**SET** Estado = p\_NuevoEstado

**WHERE** ID = p\_IDSoli;

**END //**

**DELIMITER** **;**

**Actualizar estado de delivery**

Bueno por la elección que decidí crear este SP fue para que el administrador de la base pueda modificar el estado de servicio de delivery con el id del local.

**DELIMITER //**

**CREATE PROCEDURE** ActualizarDelivery(

**IN** p\_IDLoc **INT**,

**IN** p\_TieneDelivery **BOOLEAN**

)

**BEGIN**

**UPDATE** Local

**SET** Delivery = p\_TieneDelivery

**WHERE** ID = p\_IDLoc;

**END** **//**

**DELIMITER ;**

**TRIGGERS**

Este trigger lo vamos a utilizar para checkear si el local tiene delivery cuando hacemos la compra, ni bien esta función esta elaborada con un desarrollo aparte, también me gustaría que la base de datos pueda definir si el local tiene o no delivery, también que el administrador pueda ver si agrega o no el servicio cuando este local cuente con el mismo.

**DELIMITER //**

**CREATE TRIGGER** CheckDeliveryAntesDe

**BEFORE INSERT ON** Compra

**FOR EACH** ROW

**BEGIN**

**DECLARE** delivery BOOLEAN;

**SELECT** Delivery INTO delivery

**FROM** Local

**WHERE** **ID** = NEW.IDLoc;

**IF** delivery = FALSE **THEN**

SIGNAL **SQLSTATE** '45000'

**SET MESSAGE\_TEXT** = 'Esta sucursal no ofrece servicio de delivery.';

**END IF;**

**END //**

**DELIMITER ;**

Este trigger nos va ayudar a saber si la dirección donde se hace el envio tiene dirección, en caso de que no tenga o tenga dato nulo no se podría hacer un envio lanzando un error 45000.

**DELIMITER //**

**CREATE TRIGGER** CheckDireccionEnviaAntesDe

**BEFORE INSERT ON** Envia

**FOR EACH ROW**

**BEGIN**

IF NEW.Direccion IS **NULL OR** NEW.Direccion = ' ' **THEN**

SIGNAL **SQLSTATE** '45000'

**SET MESSAGE\_TEXT** = 'No se puede enviar sin una dirección especificada.';

**END IF;**

**END //**

**DELIMITER ;**