



Curso SQL – Comisión 45230

## Gestión de clínica odontológica pequeña

Proyecto final

Leandro Benavides



## Gestión de clínica odontológica pequeña

Consiste en una clínica odontológica de 5 consultorios y equipos de imágenes. Debido a que los odontólogos no trabajan solo en un lugar ni todo el día, esos 5 consultorios pueden alojar a muchos más odontólogos.

También contempla que los socios gerentes de la clínica son odontólogos que trabajan en el centro.

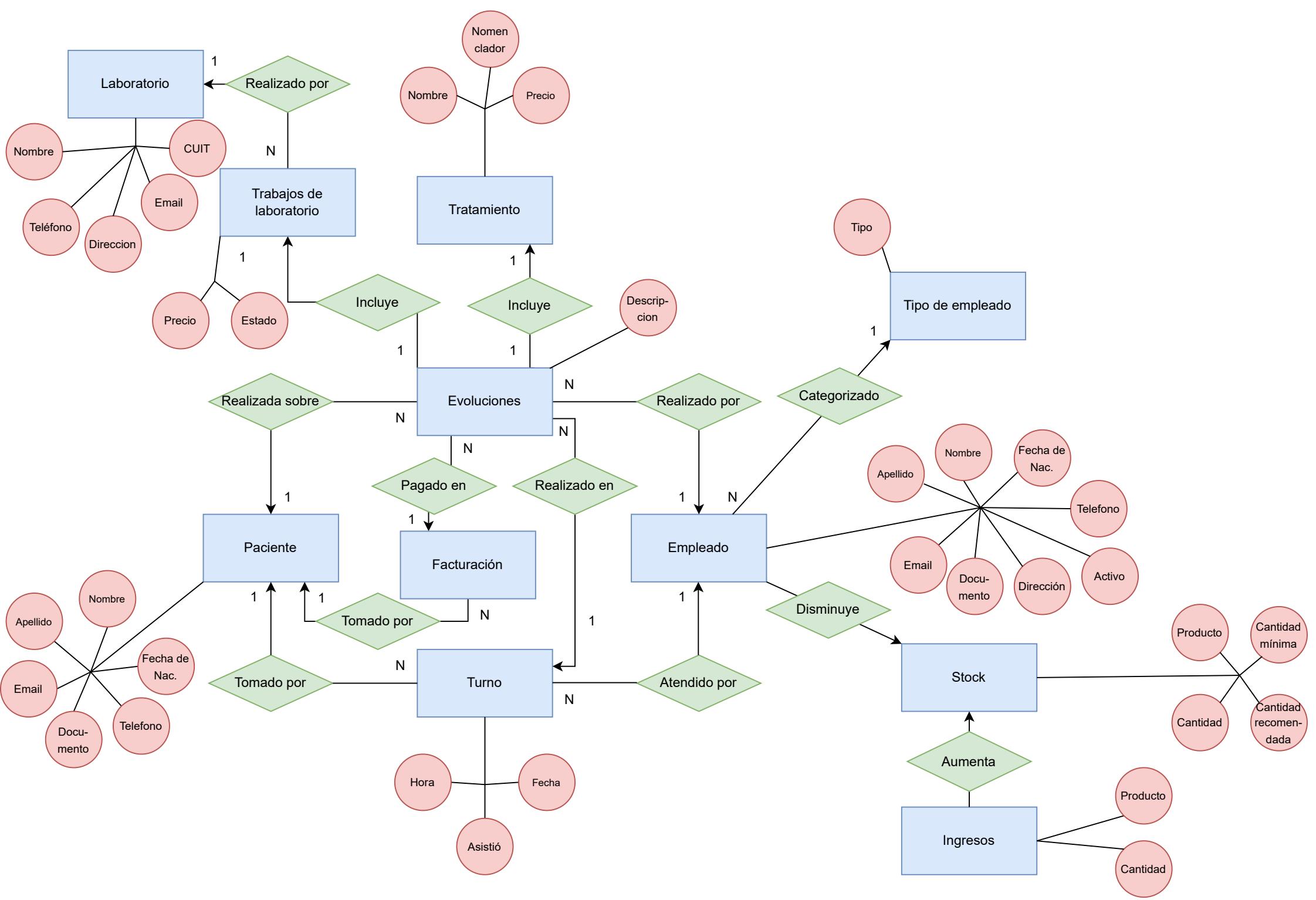
Lo anterior surge de una clínica real que esta actualmente funcionando, donde mi pareja es una socia gerente. Están utilizando actualmente un sistema de este estilo pero tienen dificultades para modificarlo a nuevas problemáticas y necesidades, por lo que me puse a la tarea de generarlo nuevamente.

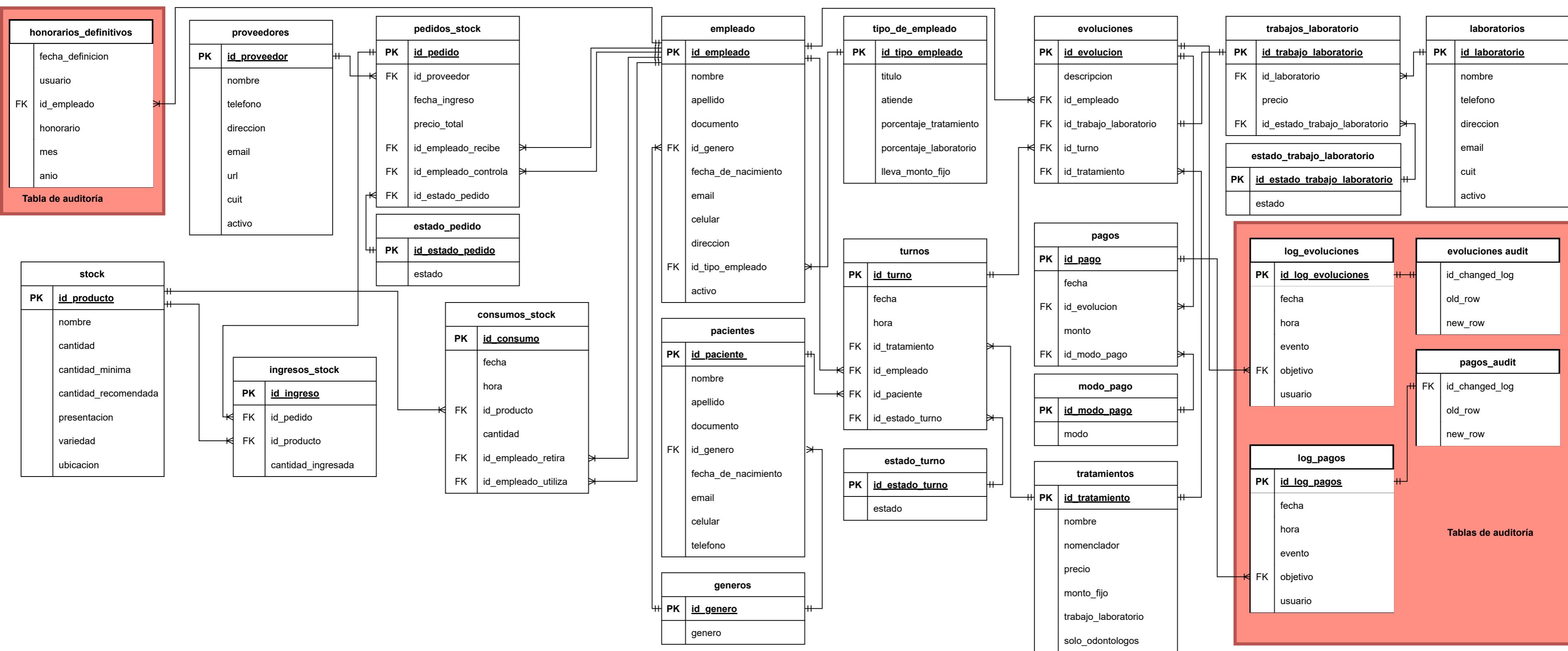


## Gestión de clínica odontológica pequeña

Incluye funcionalidades distintas de la clínica, por ejemplo:

- Sistema de turnos
- Historias clínicas
- Trabajos de laboratorios
- Facturación y pagos de clientes
- Facturación mensual de odontólogos integrada
- Stock automático por consumo e ingreso
- Pedidos a proveedores





# Gestión de clínica odontológica pequeña – Tablas y Triggers

**CODERHOUSE**

Primero incluyo una serie de tablas sencillas, que definen estados de procesos. También incluyo un tabla de géneros por si fuera necesario en algún momento modificar el sistema para contemplar distintas identidades de género. Todos los campos que no son PRIMARY KEY corresponden a campos VARCHAR(20)



Ninguna de estas tablas tienen TRIGGERS y corresponden a la configuración de la base de datos, el usuario no debe interaccionar con estas tablas salvo en referencias hechas en otras tablas.

La ventaja de presentarlo así es que permite ver más claramente los procesos del centro, como el proceso de hacer un pedido o otorgar un turno.

Estas tablas sólo deberían ser modificadas (haciendo UPDATE o INSERT) mediante los administradores por una decisión de modificar el funcionamiento del centro.

# Gestión de clínica odontológica pequeña – Tablas y Triggers

**CODERHOUSE**

Continuando con la configuración del funcionamiento del centro odontológico tenemos la tabla **tipo\_de\_empleado**.

Cada fila de esta tabla describe un tipo de empleado posible en el centro. Las columnas describen como se comporta ese tipo de empleado en cuanto a la facturación variable de la clínica y sus tareas en ella.

tipo_de_empleado		
INT	PK	<u>id_tipo_empleado</u>
VARCHAR		titulo
TINYINT		atiende
TINYINT		porcentaje_tratamiento
TINYINT		porcentaje_laboratorio
TINYINT		lleva_monto_fijo

INT  
VARCHAR  
TINYINT  
TINYINT  
TINYINT  
TINYINT

PK  
id\_tipo\_empleado

titulo  
atiende  
porcentaje\_tratamiento  
porcentaje\_laboratorio  
lleva\_monto\_fijo

Si el empleado realiza tratamientos odontológicos  
Porcentaje del pago del cliente que se lleva el empleado  
Porcentaje costo de laboratorio que debe afrontar el empleado  
Si el empleado cobra los montos fijos correspondiente a ciertos tratamientos

## TRIGGERS:

- BEFORE INSERT/UPDATE
  - validar\_datos\_tipo\_empleado\_insert
  - validar\_datos\_tipo\_empleado\_update

Ambos validan que los porcentajes estén entre 0 y 100. Tire un error si no se cumple. La positividad de los porcentajes se aseguran con un CONSTRAINT de UNSIGNED para esos campos.

```
INSERT INTO tipo_de_empleado (titulo, atiende, porcentaje_tratamiento, porcentaje_laboratorio, lleva_monto_fijo) VALUES
('Socio', 1, 70, 50, 0),
('Odontólogo', 1, 40, 50, 0),
('Repcionista', 0, 0, 0, 1),
('Asistente', 0, 0, 0, 1);
```

# Gestión de clínica odontológica pequeña – Tablas y Triggers

**CODERHOUSE**

La siguiente tabla es la tabla **tratamientos**. Esta tabla reúne los tratamientos que se realizan en la clínica e incluye todo lo que se puede cobrar en el centro. Incluye los precios actuales de los tratamientos.

tratamientos		
INT	PK	<u>id_tratamiento</u>
VARCHAR		nombre
VARCHAR		nomenclador
DECIMAL		precio
DECIMAL		monto_fijo
TINYINT		trabajo_laboratorio
TINYINT		solo_odontologos

**TRIGGERS:**  
No tiene TRIGGERS esta tabla

Código único identificador. Acá los inventé

Valor fijo que se le paga al que realiza, si le corresponde

Booleano. Define si este tratamiento requiere un trabajo de laboratorio

Booleano. Define si este tratamiento lo pueden realizar solamente odontólogos.  
Por ejemplo, los tratamientos radiológicos los pueden realizar todos los empleados entrenados

```
INSERT INTO tratamientos (nombre, nomenclador, precio, monto_fijo, trabajo_laboratorio, solo_odontologos) VALUES
('Caries', 'CI.1012.01', 3500.00, 0.00, 0, 1),
('Radiografía panorámica', 'CI.1034.01', 1650.25, 350.00, 0, 0),
('Implante', 'CI.1102.01', 26750.00, 0.00, 0, 1),
('Corona sobre implante', 'CI.1102.02', 47855.00, 0.00, 1, 1),
:::
```

# Gestión de clínica odontológica pequeña – Tablas y Triggers

**CODERHOUSE**

Pasando ahora a las tablas que tienen más interacción con el usuario, tenemos la tabla ***laboratorios***. Reúne a los laboratorios con lo que trabaja o trabajó el centro para realizar mecánica dental.

laboratorios		
INT	PK	<u>id_laboratorio</u>
VARCHAR		nombre
VARCHAR		telefono
VARCHAR		direccion
VARCHAR		email
VARCHAR		cuit
TINYINT		activo

Booleano. Si se está trabajando actualmente con este laboratorio

## TRIGGERS:

- BEFORE INSERT/UPDATE
  - validar\_datos\_laboratorios\_insert
  - validar\_datos\_laboratorios\_update

Ambos son equivalentes. Validan la estructura del email utilizando la función VALIDAR\_EMAIL y limpian los strings de teléfono y cuit para remover caracteres innecesarios (función LIMPIAR\_STRING).

Si el email es inválido, tiran un error.

```
INSERT INTO laboratorios (nombre, telefono, direccion, email, cuit, activo) VALUES
('Dental Lab', '47233265', 'Av. Rivadavia 2154, Balvanera', NULL, '30215469857', 1),
('Super Lab', '42569874', 'Av. Maipu 1656, Vicente Lopez, Pcia. de Buenos Aires', 'superlab@gmail.com', '30256547891', 0),
('Fym Dental Lab', '43254785', 'Av. Rivadavia 1936, Balvanera', 'fymdental@gmail.com', '30487963525', 1),
```

:

# Gestión de clínica odontológica pequeña – Tablas y Triggers

**CODERHOUSE**

Ahora presento la tabla **pacientes**. Esta reúne toda la información personal de las personas que se atienden o se atendieron en el centro alguna vez.

pacientes		
	PK	<u>id_paciente</u>
INT		nombre
VARCHAR		apellido
VARCHAR		documento
DECIMAL		
INT	FK	id_genero
		Referencia tabla <b>generos</b> (N-a-1)
DATE		fecha_de_nacimiento
VARCHAR		email
VARCHAR		celular
VARCHAR		telefono

## TRIGGERS:

- BEFORE INSERT/UPDATE
  - validar\_datos\_paciente\_insert y validar\_datos\_paciente\_update
  - Ambos son equivalentes y hacen 3 cosas generales:
    - Chequea que hayan ingresado al menos un número de contacto. Tira un error si no es así.
    - Valida la estructura del email con función VALIDAR\_EMAIL. Tira un error si no es válido.
    - Limpia los strings de los teléfonos y el documento con la función LIMPIAR\_STRING.

```
INSERT INTO pacientes (nombre, apellido, documento, id_genero, fecha_de_nacimiento, email, celular, telefono) VALUES
('Juan', 'Straciarella', 34521478, 2, '1990-02-25', 'jstraciarella@gmail.com', '1549769856', '41527014'),
('Estela', 'Marquez', 16529689, 1, '1965-07-10', 'estela.marquez26@gmail.com', '1554789561', '50249014'),
('Juliana Ester', 'Porto', 23457898, 1, '1973-10-07', 'julianap163@gmail.com', '1543689501', '47926520'),
```

:

# Gestión de clínica odontológica pequeña – Tablas y Triggers

**CODERHOUSE**

La tabla **empleados** reúne los datos de todos los empleados que trabajan o trabajaron en el centro.

empleado		
	INT PK	<u>id_empleado</u>
VARCHAR		nombre
VARCHAR		apellido
DECIMAL		documento
INT	FK	id_genero
		Referencia tabla <b>generos</b> (N-a-1)
DATE		fecha_de_nacimiento
VARCHAR		email
VARCHAR		celular
VARCHAR		direccion
INT	FK	id_tipo_empleado
		Referencia tabla <b>tipo_de_empleado</b> (N-a-1)
TINYINT		activo
		Booleano. Indica si el empleado trabaja actualmente

## TRIGGERS:

- BEFORE INSERT/UPDATE
  - validar\_datos\_empleado\_insert y validar\_datos\_empleado\_update
  - Ambos sin equivalentes y hacen 2 cosas generales:
    - Valida la estructura del email con función VALIDAR\_EMAIL.  
Tira un error si no es válido.
    - Limpia los strings del celular y el documento con la función LIMPIAR\_STRING.

```
INSERT INTO empleados (nombre, apellido, documento, id_genero, fecha_de_nacimiento, email, celular, direccion, id_tipo_empleado, activo) VALUES
('Sebastián', 'Galarza', 34987014, 2, '1990-02-16', 'sgalarza@gmail.com', '1543580123', 'Cochabamba 498, CABA', 1, 1),
('Adriana', 'Lillian', 32421598, 1, '1989-12-14', 'adrilili@gmail.com', '1543659878', 'Cochabamba 498, CABA', 1, 1),
('Ximena', 'Ponce', 37259846, 1, '1993-09-26', 'ximelaloca@gmail.com', '1535980147', 'Av Santa Fe 3256 6A, CABA', 2, 1),
```

:

# Gestión de clínica odontológica pequeña – Tablas y Triggers

**CODERHOUSE**

La tabla ***honorarios\_definitivos*** es la primera tabla de auditoría que presento. Reúne los honorarios variables que se les pagan a los empleados cada período mes-año. La terna (id\_empleado, mes, anio) forma la PRIMARY KEY de la tabla. Esta tabla sólo se puede modificar por el administrador.

honorarios_definitivos		
DATE		fecha_definicion
VARCHAR		usuario
INT	FK	id_empleado
DECIMAL		honorario
INT		mes
INT		anio

Fecha de inserción en la tabla

Referencia tabla *id\_empleado* (N-a-1)

Honorario para el período mes-año

## TRIGGERS:

### • BEFORE INSERT

- insertar\_usuario\_honorarios: Este trigger sólo inserta el valor de la función USER() en el campo usuario, ya que no se puede hacer como DEFAULT

Esta tabla se puebla automáticamente. La idea es que al final de un mes trabajado, un usuario (normalmente sólo un socio) corre el procedimiento HONORARIOS con el período mes-año que acaba de terminar. Por ejemplo:

```
CALL HONORARIOS(9, 2022, 1, 0);
```

El cuarto parámetro determina si los honorarios que se muestran son los definitivos o no. Puede llamar cuantas veces quiera con un 0 en ese campo.

Cuando decide que está todo bien y decide cerrar el mes, llama al procedimiento igual, pero con un 1 en el último campo. Eso hace que inserte los honorarios para los empleados en la tabla ***honorarios\_definitivos***.

Más adelante se explica mejor el procedimiento, sólo es para mostrar el comportamiento de la tabla.

En el archivo .sql se insertan datos manualmente para simular meses cerrados, pero en la realidad no deberían insertarse manualmente.

Para testear, dejé el mes de septiembre (09) con pagos insertados pero sin cerrar para poder probar el procedimiento.

# Gestión de clínica odontológica pequeña – Tablas y Triggers

CODERHOUSE

La tabla **turnos** contiene la información del sistema de turnos de la clínica. Incluye tanto los turnos pasados como futuros.

turnos		
	PK	id_turno
INT	DATE	fecha
TIME		hora
INT	INT	id_tratamiento
		Referencia tabla <a href="#">tratamientos</a> (N-a-1)
INT	INT	id_empleado
		Referencia tabla <a href="#">empleados</a> (N-a-1)
INT	INT	id_paciente
		Referencia tabla <a href="#">pacientes</a> (N-a-1)
INT	INT	id_estado_turno
		Referencia tabla <a href="#">estado_turno</a> (N-a-1)

## TRIGGERS:

- BEFORE INSERT/UPDATE
  - validar\_datos\_turnos\_insert y validar\_datos\_turnos\_update
  - Ambos sin equivalentes y hacen 3 cosas generales:
    - Valida que se inserte o un id\_empleado o un id\_tratamiento, pero no ambos.
    - Chequeo que el id\_empleado corresponda a un odontólogo activo (con atiende = 1 y activo = 1).
    - Chequeo si el id\_tratamiento ingresado corresponde a uno que puede realizar cualquier empleado (solo\_odontologos = 0).

Hay dos tipos de turnos: un turno común con un odontólogo y un turno para un tratamiento sin odontólogo (una radiografía panorámica por ejemplo). Se distingue entre ambos por las columnas id\_empleado e id\_tratamiento. Si se llena la columna id\_empleado, es un turno del primer tipo y si se llena id\_tratamiento, es del segundo tipo.

```
INSERT INTO turnos (fecha, hora, id_estado_turno, id_paciente, id_empleado, id_tratamiento) VALUES
-- Odontólogo id 1
('2022-02-12', '10:00', 2, 19, 1, NULL), ('2022-02-12', '12:00', 2, 6, 1, NULL), ('2022-02-12', '14:00', 1, 14, 1, NULL), ('2022-02-12', '16:00', 2, 2, 1, NULL),
('2022-04-25', '10:30', 2, 25, 1, NULL), ('2022-04-25', '11:30', 2, 19, 1, NULL), ('2022-04-25', '13:30', 2, 10, 1, NULL), ('2022-04-25', '15:00', 2, 2, 1, NULL),
('2022-06-17', '10:30', 1, 13, 1, NULL), ('2022-06-17', '12:00', 2, 16, 1, NULL), ('2022-06-17', '14:00', 2, 22, 1, NULL), ('2022-06-17', '15:30', 2, 25, 1, NULL),
:;
```

# Gestión de clínica odontológica pequeña – Tablas y Triggers

CODERHOUSE

La tabla **evoluciones** es la mas importante. Una evolución es una acción que se realiza sobre un paciente, como por ejemplo arreglar una caries o tomar una radiografía. En un turno, un odontólogo puede realizar varias evoluciones sobre un paciente, cada una será una fila de esta tabla.

evoluciones	
INT	PK id_evolucion
VARCHAR	descripcion
INT	FK id_empleado
	Referencia tabla <b>empleados</b> (N-a-1)
INT	FK id_trabajo_laboratorio
	Referencia tabla <b>trabajos_laboratorio</b> (1-a-1)
INT	FK id_turno
	Referencia tabla <b>turnos</b> (N-a-1)
INT	FK id_tratamiento
	Referencia tabla <b>tratamientos</b> (N-a-1)

## TRIGGERS:

- BEFORE INSERT
  - new\_lab\_work: Si al tratamiento ingresado le corresponde un trabajo de laboratorio, inserta una fila vacía en esa tabla y actualiza la columna id\_trabajo\_laboratorio en esta.
- AFTER INSERT/UPDATE: Dos triggers distintos que pueblan tablas de auditoría (siguiente diapositiva).

La columna id\_empleado parece redundante ya que la columna id\_turno ya se refiere a id\_empleado. Sin embargo, puede suceder que a un paciente lo atienda un odontólogo distinto al que se le asignó, por algún imprevisto.

La columna id\_trabajo\_laboratorio se llena automáticamente. Cuando el id\_tratamiento ingresado corresponde a un tratamiento con trabajo de laboratorio (en tabla tratamientos), se inserta una fila en la tabla trabajos\_laboratorio (se presenta más adelante) y se coloca el id de esa fila generada en la columna id\_trabajo\_laboratorio.

```
INSERT INTO evoluciones (id_tratamiento, id_turno, id_empleado, descripcion) VALUES
(7, 1, 1, 'Se cementó un perno sobre la pieza 23'),
(4, 2, 1, 'Se colocó y atornilló un implante de porcelana sobre el implante colocado en la pieza 17'),
(5, 2, 1, 'Se realizó un tratamiento de conducto debido a un nervio comprometido por infección en la pieza 20'),
:
```

# Gestión de clínica odontológica pequeña – Tablas y Triggers

**CODERHOUSE**

Las tablas ***log\_evoluciones*** y ***evoluciones\_audit*** son dos tablas de auditoría para la tabla evoluciones

log_evoluciones	
INT	PK <u><b><i>id_log_evoluciones</i></b></u>
DATE	fecha
TIME	hora
VARCHAR	evento
INT	FK objetivo
VARCHAR	usuario

UPDATE o INSERT  
Referencia tabla ***evoluciones*** (N-a-1)

Referencia tabla ***log\_evoluciones*** (1-a-1)

evoluciones audit	
	id_changed_log
	old_row
	new_row

INT  
VARCHAR  
VARCHAR

Esta tabla se puebla con dos TRIGGERS de la tabla evoluciones:

AFTER INSERT - generar\_log\_evolucion\_insert  
AFTER UPDATE - generar\_log\_evolucion\_update

Incluye la información de quién realizó la acción, cuando y a qué línea

Esta tabla registra los cambios generados en una fila por un UPDATE, guardando los datos viejos y nuevos. Se puebla con uno de los TRIGGERS anteriores:

AFTER UPDATE - generar\_log\_evolucion\_update

Incluye la información de quién a qué log se refiere, los datos viejos y los nuevos, ambos en forma de strings concatenados.

# Gestión de clínica odontológica pequeña – Tablas y Triggers

**CODERHOUSE**

La tabla trabajos\_laboratorio detalla la información de los trabajos que se mandan a hacer a laboratorios internos o externos a la clínica. Incluye información del precio y del estado del proceso

trabajos_laboratorio	
INT	PK <u><a href="#">id_trabajo_laboratorio</a></u>
INT	FK <u><a href="#">id_laboratorio</a></u>
DECIMAL	precio
INT	FK <u><a href="#">id_estado_trabajo_laboratorio</a></u>

Referencia tabla *laboratorios* (N-a-1)

Referencia tabla *estado\_trabajo\_laboratorio* (N-a-1)

## TRIGGERS:

### • BEFORE UPDATE

- Si la modificación que estamos realizando involucra asignar un laboratorio al trabajo, modifica automáticamente el estado de 'Iniciado' a 'Asignado'

Al agregar una fila a la tabla evoluciones con un tratamiento al que le corresponde un trabajo de laboratorio, se agrega una fila vacía con estado 'Iniciado'.

A medida que el trabajo avanza, se modifica manualmente algunos campos.

# Gestión de clínica odontológica pequeña – Tablas y Triggers

**CODERHOUSE**

La tabla **pagos** es la otra tabla más importante, ya que la facturación y los honorarios surge de esta tabla. Se permite realizar varios pagos para la misma evolución

pagos	
INT	PK <b>id_pago</b>
DATE	fecha
INT	FK id_evolucion
DECIMAL	monto
INT	FK id_modo_pago

Referencia tabla **evoluciones** (N-a-1)

Referencia tabla **modo\_pago** (N-a-1)

## TRIGGERS:

- AFTER INSERT/UPDATE: Dos triggers distintos que pueblan tablas de auditoría (siguiente diapositiva).

```
INSERT INTO pagos (fecha, id_evolucion, monto, id_modo_pago) VALUES
('2022-02-12', 1, 11500.0, 5), ('2022-02-12', 2, 47855.0, 3), ('2022-02-12', 3, 16500.0, 1), ('2022-02-12', 4, 26750.0, 2),
('2022-02-12', 6, 3500.0, 1), ('2022-04-25', 7, 105000.0, 4), ('2022-04-25', 8, 3500.0, 1), ('2022-04-25', 9, 16500.0, 5), (
('2022-04-25', 12, 11500.0, 3), ('2022-06-17', 13, 11500.0, 2), ('2022-06-17', 14, 6500.0, 3), ('2022-06-17', 15, 6500.0, 4)
```

:

# Gestión de clínica odontológica pequeña – Tablas y Triggers



Las tablas *log\_pagos* y *pagos\_audit* son dos tablas de auditoría para la tabla pagos. Funcionan igual que las tablas de auditoría de la tabla evoluciones.

log_pagos		
INT	PK	<u>id_log_pagos</u>
DATE		fecha
TIME		hora
VARCHAR		evento
INT	FK	objetivo
VARCHAR		usuario

UPDATE o INSERT

Referencia tabla *log\_pagos* (1-a-1)

Referencia tabla *pagos* (N-a-1)

pagos_audit		
INT	FK	<u>id_changed_log</u>
VARCHAR		old_row
VARCHAR		new_row

Esta tabla se puebla con dos TRIGGERS de la tabla pagos:

AFTER INSERT - generar\_log\_pago\_insert  
AFTER UPDATE - generar\_log\_pago\_update

Incluye la información de quién realizó la acción, cuando y a qué línea

Esta tabla registra los cambios generados en una fila por un UPDATE, guardando los datos viejos y nuevos. Se puebla con uno de los TRIGGERS anteriores:

AFTER UPDATE - generar\_log\_pago\_update

Incluye la información de quién a qué log se refiere, los datos viejos y los nuevos, ambos en forma de strings concatenados.

# Gestión de clínica odontológica pequeña – Tablas y Triggers

**CODERHOUSE**

La tabla **stock** lleva cuenta de todos los materiales consumibles de la clínica. Fuera de el ingreso inicial, se modifica automáticamente.

stock	
INT	PK <u>id_producto</u>
VARCHAR	nombre
INT	cantidad
INT	cantidad_minima
INT	cantidad_recomendada
VARCHAR	presentacion
VARCHAR	variedad
VARCHAR	ubicacion

## TRIGGERS:

- BEFORE INSERT/UPDATE
  - cantidades\_stock\_insert y cantidades\_stock\_update
  - Ambos son equivalentes, chequean que la cantidad mínima sea menor que la cantidad recomendada, para que tenga sentido.
  - Las cantidades están definidas como UNSIGNED, lo que asegura su positividad.

```
INSERT INTO stock (nombre, cantidad, cantidad_minima, cantidad_recomendada, presentacion, variedad, ubicacion) VALUES
('Guantes de latex', 6, 3, 19, 'Caja/s', 'Talle XS', 'Estantería e4'), ('Guantes de latex', 18, 5, 19, 'Caja/s', 'Talle S', 'Estantería c2'),
('Guantes de latex', 4, 4, 20, 'Caja/s', 'Talle M', 'Estantería e1'), ('Guantes de latex', 11, 5, 16, 'Caja/s', 'Talle L', 'Estantería g1'),
('Guantes de latex', 12, 8, 8, 'Caja/s', 'Talle XL', 'Estantería d1'), ('Barbijos quirúrgicos', 13, 6, 10, 'Caja/s', NULL, 'Estantería f3'),
:;
```

# Gestión de clínica odontológica pequeña – Tablas y Triggers



La tabla **proveedores** reúne la información de los proveedores de materiales consumibles

proveedores	
INT	PK <u>id_proveedor</u>
VARCHAR	nombre
VARCHAR	telefono
VARCHAR	direccion
VARCHAR	email
VARCHAR	url
VARCHAR	cuit
TINYINT	activo

Si se trabaja actualmente con este proveedor

## TRIGGERS:

- BEFORE INSERT/UPDATE
  - validar\_datos\_proveedores\_insert
  - validar\_datos\_proveedores\_update

Ambos son iguales. Validan la estructura del email utilizando la función VALIDAR\_EMAIL y limpian los strings de teléfono y cuit para remover caracteres innecesarios (función LIMPIAR\_STRING).

Si el email es inválido, tiran un error.

```
INSERT INTO proveedores (nombre, telefono, direccion, email, url, cuit, activo) VALUES
    ('Greenberg', '47653548', 'Av. Santa Fe 3256, CABA', 'ventas@greenberg.com.ar', 'www.greenbang.com.ar', 30524125630, 1),
    ('Insumos Dental Total', '41258964', 'Riobamba 1415, 2B, CABA', 'ventas@dentaltotal.com.ar', NULL, 30356589591, 1),
    ('Viscay Implementos', '50265896', 'Fray Justo Sarmiento 252, Vicente Lopez', 'info@viscay.com.ar', NULL, 30154858965, 0),
```

:

# Gestión de clínica odontológica pequeña – Tablas y Triggers

**CODERHOUSE**

La tabla **consumos\_stock** se encarga de recopilar la utilización de materiales consumibles. Cada vez que se ingresa el consumo de un producto, se da de baja automáticamente de la tabla stock. Contempla que el producto lo retira un empleado pero lo usa otro

consumos_stock		
	PK	<u>id_consumo</u>
INT		fecha
DATE		hora
TIME		
INT	FK	id_producto
INT		cantidad
INT	FK	id_empleado_retira
INT	FK	id_empleado_utiliza

Referencia a **productos** (N-a-1)

Referencia a **empleados** (N-a-1). El que retira del stock.

Referencia a **empleados** (N-a-1). El que lo utiliza.

## TRIGGERS:

- BEFORE INSERT/UPDATE
  - consumo\_stock\_insert: Chequea que haya suficiente stock para la cantidad consumida. Si no lo hay, tira un error. Luego, hace un UPDATE de la tabla stock para el producto consumido
  - consumo\_stock\_update: Funciona muy similar a la versión para INSERT pero el chequeo de stock y el UPDATE de stock es levemente diferente. A la cantidad actual, le suma la cantidad consumida vieja (OLD.cantidad) y le resta la nueva (NEW.cantidad). El chequeo de stock sigue esta regla.

```
INSERT INTO consumos_stock (fecha, hora, id_producto, cantidad, id_empleado_retira, id_empleado_utiliza) VALUES
('2022-04-05', '19:44', 2, 1, 13, 4), ('2022-04-09', '16:42', 7, 1, 11, 4), ('2022-04-26', '14:51', 19, 1, 8, 3),
('2022-04-09', '11:02', 11, 1, 4, 2), ('2022-04-09', '15:39', 2, 1, 14, 6), ('2022-04-01', '14:30', 4, 1, 5, 6),
('2022-04-13', '13:40', 7, 1, 9, 2), ('2022-04-28', '14:28', 13, 1, 7, 6), ('2022-04-28', '15:54', 12, 1, 2, 4),
```

:

# Gestión de clínica odontológica pequeña – Tablas y Triggers

**CODERHOUSE**

La tabla **pedidos\_stock** contiene la información de los pedidos a proveedores de productos consumibles de stock. Contempla que un empleado reciba el pedido y que otro (o el mismo) controle su contenido.

pedidos_stock	
INT	PK <u><a href="#">id_pedido</a></u>
INT	FK id_proveedor
DATE	fecha_ingreso
DECIMAL	precio_total
INT	FK id_empleado_recibe
INT	FK id_empleado_controla
INT	FK id_estado_pedido

Referencia a **proveedores** (N-a-1)

Referencia a **empleados** (N-a-1). El que recibe el pedido.

Referencia a **empleados** (N-a-1). El que controla sus contenidos.

Referencia a **estado\_pedido** (N-a-1).

## TRIGGERS:

- BEFORE INSERT/UPDATE
  - `avance_de_pedido_insert`
  - `avance_de_pedido_update`

Ambos son equivalentes. Chequean una correlación entre el estado del pedido y las columnas llenas. Si el pedido pasa a ser 'Recibido', la columna `id_empleado_recibe` debe estar llena y si pasa a 'Chequeado', ambas columnas `id_empleado_recibe` e `id_empleado_controla` deben estar llenas.

```
INSERT INTO pedidos_stock (id_proveedor, fecha_ingreso, precio_total, id_estado_pedido, id_empleado_recibe, id_empleado_controla) VALUES  
(1, '2022-04-01', 105000.00, 3, 9, 9),  
(4, '2022-04-01', 45000.00, 3, 13, 10),  
(2, '2022-07-02', 85000.00, 3, 12, 12),  
⋮
```

# Gestión de clínica odontológica pequeña – Tablas y Triggers

**CODERHOUSE**

La última tabla presentada es la tabla **ingresos\_stock**. Esta tabla está directamente ligada con la tabla pedidos\_stock. Acá, cada línea es el ingreso al stock de un producto, cada pedido conteniendo potencialmente varios productos.

ingresos_stock	
INT	PK <u>id_ingreso</u>
INT	FK id_pedido
INT	FK id_producto
INT	cantidad_ingresada

Referencia a **pedidos\_stock** (N-a-1)

Referencia a **productos**(N-a-1).

## TRIGGERS:

- AFTER INSERT/UPDATE
  - **ingreso\_stock\_insert**: Hace un UPDATE de la tabla stock, sumando la cantidad ingresada de este producto.
  - **ingreso\_stock\_update**: Similar a la variante INSERT, este trigger le resta la cantidad ingresada vieja (OLD.cantidad\_ingresada) y le suma la nueva (NEW.cantidad\_ingresada).

La positividad de la cantidad está asegurada por su condición de UNSIGNED.

```
INSERT INTO ingresos_stock (id_pedido, id_producto, cantidad_ingresada) VALUES  
(1, 17, 16), (1, 18, 13), (1, 9, 10), (1, 10, 18), (1, 11, 10),  
(1, 22, 20), (2, 3, 20), (2, 4, 16), (2, 5, 8);  
:
```

El primer procedimiento es **obtener\_porcentajes**. Es el primero de una serie de procedimientos cortos utilizados para calcular honorarios.

```
CREATE PROCEDURE obtener_porcentajes (IN id_odontologo INT, OUT porcentaje_tratamiento INT, OUT porcentaje_laboratorio INT)
BEGIN
    SELECT te.porcentaje_tratamiento, te.porcentaje_laboratorio INTO porcentaje_tratamiento, porcentaje_laboratorio
    FROM empleados e
    JOIN tipo_de_empleado te ON e.id_tipo_empleado = te.id_tipo_empleado
    WHERE e.id_empleado = id_odontologo;
END$$
```

Con un input de el id de un odontólogo de la tabla id\_empleados, genera dos parámetros de output, los porcentajes que le corresponden a ese empleado según su tipo.

Estos porcentajes se utilizan para calcular honorarios.

Este procedimiento es llamado por la función *honorarios\_mensuales*

El procedimiento **obtener\_facturación\_mensual** se utiliza para obtener la suma de todos los pagos que se realizaron a un odontólogo en un período dado.

```
CREATE PROCEDURE obtener_facturacion_mensual (IN id_odontologo INT, IN mes INT, IN anio INT, OUT facturacion_mensual DECIMAL(10,2))
BEGIN
    -- Utilizo una vista generada más abajo
    SELECT SUM(facturacion) INTO facturacion_mensual
    FROM facturacion_odontologo
    WHERE id_empleado = id_odontologo
    AND MONTH(fecha) = mes
    AND YEAR(fecha) = anio;
END$$
```

Con el **id\_empleado** del odontólogo y el período de interés expresando en **INT**, el procedimiento aprovecha la vista *facturación\_mensual* para obtener el valor sin repetir código.

Genera un parámetro de output con la facturación mensual del odontólogo en el período indicado.

Este procedimiento también es llamado por la función *honorarios\_mensuales*.

# Gestión de clínica odontológica pequeña – Stored Procedures



El procedimiento **obtener\_costo\_laboratorio** se utiliza para obtener la suma de todos los gastos generados por un odontólogo en trabajos de laboratorio

```
CREATE PROCEDURE obtener_costo_laboratorio (IN id_odontologo INT, IN mes INT, IN anio INT, OUT laboratorios_mensual DECIMAL(10,2))
BEGIN
    DECLARE check_null INT;

    -- Esta primera query me permite chequear si alguno de los precios que voy a sumar es NULL
    SELECT SUM(ISNULL(tl.precio)) INTO check_null
    FROM evoluciones ev
    JOIN turnos tu ON ev.id_turno = tu.id_turno
    JOIN trabajos_laboratorio tl ON ev.id_trabajo_laboratorio = tl.id_trabajo_laboratorio
    WHERE ev.id_empleado = id_odontologo
    AND MONTH(tu.fecha) = mes
    AND YEAR(tu.fecha) = anio;

    -- Si algun precio es NULL, tira un error para que lo ingresen. Así evita olvidos de ingresar valores de trabajos de laboratorio
    IF check_null <> 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Al menos uno de los precios de los trabajos del mes no tiene un valor asignado';
    END IF;

    SELECT SUM(tl.precio) INTO laboratorios_mensual
    FROM evoluciones ev
    JOIN turnos tu ON ev.id_turno = tu.id_turno
    JOIN trabajos_laboratorio tl ON ev.id_trabajo_laboratorio = tl.id_trabajo_laboratorio
    WHERE ev.id_empleado = id_odontologo
    AND MONTH(tu.fecha) = mes
    AND YEAR(tu.fecha) = anio;
END$$
```

Con el id\_empleado del odontólogo y el período de interés expresando en INT, el procedimiento suma todos los costos de laboratorio de las evoluciones del período indicado

Genera un parámetro de output con el costo de laboratorio del odontólogo en el período indicado.

Este procedimiento también es llamado por la función *honorarios\_mensuales*.

Incluye un chequeo de que todos los trabajos tengan asignado un precio, para evitar olvidos.

Similar al procedimiento obtener\_porcentajes, **obtener\_estado\_monto\_fijo** se utiliza para chequear si a un dado empleado le corresponde cobrar los montos fijos de los tratamientos. Como regla general, los montos fijos los pueden cobrar los no odontólogos, como incentivo a realizar tareas extras que alivien a los odontólogos y puedan ser realizadas por todos.

```
CREATE PROCEDURE obtener_estado_monto_fijo (IN id_asistente INT, OUT cobra_monto_fijo TINYINT)
BEGIN
    SELECT te.lleva_monto_fijo INTO cobra_monto_fijo
    FROM empleados em
    JOIN tipo_de_empleado te ON em.id_tipo_empleado = te.id_tipo_empleado
    WHERE em.id_empleado = id_asistente;
END$$
```

Este procedimiento es llamado por la función *adicional\_montos\_fijos*.

# Gestión de clínica odontológica pequeña – Stored Procedures



El procedimiento **obtener\_adicional** se utiliza para sumar todos los montos\_fijos a cobrar por un empleado en un período determinado

```
CREATE PROCEDURE obtener_adicional (IN id_asistente INT, IN mes INT, IN anio INT, OUT adicional DECIMAL (10,2))
BEGIN
    SELECT SUM(montos.monto_fijo) INTO adicional
    FROM (
        SELECT tr.monto_fijo
        FROM pagos p
        JOIN evoluciones ev ON p.id_evolucion = ev.id_evolucion
        JOIN tratamientos tr ON ev.id_tratamiento = tr.id_tratamiento
        WHERE ev.id_empleado = id_asistente
        AND MONTH(p.fecha) = mes
        AND YEAR(p.fecha) = anio
        GROUP BY ev.id_evolucion -- Por si hay más de un pago por evolucion, que no cuente doble
    ) AS montos;
END$$
```

Suma sobre todos los pagos realizados que están asignados a un empleado los montos fijos de los tratamientos indicados en el período indicado. Se toma el cuidado de agrupar por evolución en la subconsulta para evitar contar dos veces, en el caso de que haya más de un pago por evolución.

Este procedimiento es llamado por la función *adicional\_montos\_fijos*.

# Gestión de clínica odontológica pequeña – Stored Procedures



El procedimiento **honorarios** se utiliza para determinar los honorarios variables a cobrar por los empleados en un período mes-año. Este procedimiento se pensó para usar cuando se cierra el mes, es decir, para obtener una lista de honorarios a pagar.

```
CREATE PROCEDURE honorarios (IN mes INT, IN anio INT, IN atiende TINYINT, IN definitivo TINYINT)
BEGIN
    -- Dependiendo de si quiero honorarios de odontólogos o de otros empleados, debo cambiar la función que llama más adelante
    IF atiende = 1 THEN
        SET @function_call = 'honorario_mensual';
        SET @nombre = 'honorarios';
    ELSEIF atiende = 0 THEN
        SET @function_call = 'adicional_montos_fijos';
        SET @nombre = 'montos_fijos';
    ELSE
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Valor inválido para "atiende". Debe ser 1 o 0 para odontólogos u otros, respectivamente.';
    END IF;

    -- Esta query es la que el usuario ve
    SET @clause = CONCAT('SELECT e.nombre, e.apellido, ', @function_call, '(e.id_empleado, ', mes, ', ', anio, ') as "", @nombre, "" ');
    SET @clause = CONCAT(@clause, ',FROM empleados e JOIN tipo_de_empleado te ON e.id_tipo_empleado=te.id_tipo_empleado WHERE e.activo = 1 AND te.atiende = ', atiende, ',');
    PREPARE runSQL FROM @clause;
    EXECUTE runSQL;
    DEALLOCATE PREPARE runSQL;

    -- Si determinó que los honorarios son definitivos, inserta en la table honorarios_definitivos
    -- Utiliza una query similar a la anterior como subconsulta
    IF definitivo = 1 THEN
        SET @insert_clause = CONCAT('INSERT INTO honorarios_definitivos (id_empleado, mes, anio, honorario)');
        SET @insert_clause = CONCAT(@insert_clause, 'SELECT e.id_empleado, ', mes, ', ', anio, ', ', @function_call, '(e.id_empleado, ', mes, ', ', anio, ') as "", @nombre, "" ');
        SET @insert_clause = CONCAT(@insert_clause, ',FROM empleados e JOIN tipo_de_empleado te ON e.id_tipo_empleado=te.id_tipo_empleado WHERE e.activo=1 AND te.atiende=', atiende, ',');
        PREPARE runSQLInsert FROM @insert_clause;
        EXECUTE runSQLInsert;
        DEALLOCATE PREPARE runSQLInsert;
    END IF;
END$$
```

Además de los inputs de mes y año, tiene un input booleano ‘atiende’ que se refiere a si quiero calcular honorarios de odontólogos u otros empleados. Esto cambia la función llamada, de *honorarios\_mensuales* a *adicional\_montos\_fijos*.

El último input, ‘definitivo’ es el que indica si se cierra el mes. Mientras sea 0, el procedimiento muestra la tabla de honorarios, pero al ingresar 1, el procedimiento hace un INSERT para cada empleado de la tabla en la tabla de auditoría *honorarios\_definitivos*.

Con eso, se considera cerrado el mes, y no se puede volver a correr el procedimiento con definitivo = 1 ya que viola la PRIMARY KEY de la tabla destino.

## Gestión de clínica odontológica pequeña – Stored Procedures



El procedimiento ***lista\_emails*** se utiliza para obtener una lista de mails de los pacientes, principalmente para marketing.

```
CREATE PROCEDURE lista_emails (IN order_column VARCHAR(50), IN direction VARCHAR(4))
BEGIN
    IF order_column <> '' THEN
        SET @order_clause = CONCAT('ORDER BY ', order_column, ' ', direction);
    ELSE
        SET @order_clause = '';
    END IF;

    SET @clause = CONCAT('SELECT p.nombre, p.apellido, p.documento, g.genero, p.fecha_de_nacimiento, p.email');
    SET @clause = CONCAT(@clause, ' FROM pacientes p JOIN generos g ON p.id_genero = g.id_genero', @order_clause);
    PREPARE runSQL FROM @clause;
    EXECUTE runSQL;
    DEALLOCATE PREPARE runSQL;
END$$
```

Por medio de inputs, permite elegir la columna por la cual se ordena y en que sentido, 'ASC' o 'DESC'.

# Gestión de clínica odontológica pequeña – Stored Procedures



```
CREATE PROCEDURE aumentar_precios (IN porcentaje_aumento INT, aplicar_a_monto_fijo TINYINT)
BEGIN
    DECLARE i, m, n, row_id INT;
    DECLARE old_price, old_fixed DECIMAL(10,2);

    IF aplicar_a_monto_fijo NOT IN (0, 1) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Variable aplicar_a_monto_fijo puede ser sólo 0 o 1';
    END IF;

    SET i = 1;
    SELECT COUNT(*) INTO n FROM tratamientos;

    -- Hago un loop sobre todos los tratamientos aumentando el precio
    -- No es necesario que los id_tratamientos sean consecutivos
    WHILE i <= n DO
        SET m = i - 1;
        SELECT id_tratamiento, precio, monto_fijo INTO row_id, old_price, old_fixed FROM tratamientos LIMIT m, 1;

        IF aplicar_a_monto_fijo = 0 THEN
            UPDATE tratamientos
            SET precio = old_price * (1 + (porcentaje_aumento / 100))
            WHERE id_tratamiento = row_id;
        ELSEIF aplicar_a_monto_fijo = 1 THEN
            UPDATE tratamientos
            SET precio = old_price * (1 + (porcentaje_aumento / 100)),
                monto_fijo = old_fixed * (1 + (porcentaje_aumento / 100))
            WHERE id_tratamiento = row_id;
        END IF;

        SET i = i + 1;
    END WHILE;
END$$
```

El último procedimiento es **aumentar\_precios** se utiliza para cambiar a la vez todos los precios de los tratamientos un porcentaje determinado.

El input ‘porcentaje\_aumento’ es autoexplicativo. Si quiero aumentar 10%, ingreso 10.

El input ‘aplicar\_a\_monto\_fijo’ es un booleano. Indica si se debe aplicar el aumento a los montos\_fijos de los tratamientos también.

Hace un loop por todos los tratamientos, haciendo UPDATES de las columnas precio y monto\_fijo (si corresponde).

# Gestión de clínica odontológica pequeña – Funciones



```
CREATE FUNCTION honorario_mensual(id_odontologo INT, mes INT, anio INT)
RETURNS DECIMAL(10,2)
READS SQL DATA
BEGIN
DECLARE porcentaje_tratamiento, porcentaje_laboratorio INT;
DECLARE facturacion_mensual, laboratorios_mensual, honorarios DECIMAL(10,2);

-- Obtengo los porcentajes correspondientes al odontólogo
CALL obtener_porcentajes(id_odontologo, porcentaje_tratamiento, porcentaje_laboratorio);

-- Si el empleado no recibe porcentaje de tratamientos, ya devuelvo 0.00
-- Esto evita ingresar empleado que no atienden sin chequear más
IF porcentaje_tratamiento = 0 THEN
    RETURN 0.00;
END IF;

-- Obtengo la facturación del mes de el odontólogo
CALL obtener_facturacion_mensual(id_odontologo, mes, anio, facturacion_mensual);

-- Obtengo los costos de laboratorio de los tratamientos del odontólogo
CALL obtener_costo_laboratorio(id_odontologo, mes, anio, laboratorios_mensual);

-- Realizo la cuenta de honorarios. Le sumo el porcentaje que le corresponde de los pagos realizados en el mes
-- Le resto el porcentaje que le corresponde del costo de laboratorio
SET honorarios = (facturacion_mensual * porcentaje_tratamiento - laboratorios_mensual * porcentaje_laboratorio) / 100;

-- Por si todos los valores son NULL, quiero devolver siempre números
IF honorarios IS NULL THEN
    RETURN 0.00;
ELSE
    RETURN honorarios;
END IF;
END$$
```

La función ***honorario\_mensual*** es la que realiza la cuenta para generar el honorario de un odontólogo en un período mes-año.

Dentro de ella, llama a varios procedimientos, encargados de la lectura de las tablas para obtener datos.

Luego simplemente calcula el honorario:

- Suma el porcentaje de la facturación que le corresponde
- Resta el porcentaje del costo de laboratorio

Para evitar dar resultados de NULL en el caso de no haber pagos para el período, chequea si el honorario es NULL y devuelve 0,00 en ese caso.

Esta función puede ser utilizada por el usuario directamente. También es utilizada por el procedimiento *honorarios* para generar una tabla de honorarios para el mes.

# Gestión de clínica odontológica pequeña – Funciones



```
CREATE FUNCTION adicional_montos_fijos(id_asistente INT, mes INT, anio INT)
RETURNS DECIMAL(10,2)
READS SQL DATA
BEGIN
DECLARE cobra_monto_fijo TINYINT;
DECLARE adicional DECIMAL(10,2);

-- Obtengo el estado del empleado, si cobra monto fijo
CALL obtener_estado_monto_fijo(id_asistente, cobra_monto_fijo);

-- Si ingresé un odontólogo ya devuelvo 0.00, sin necesidad de chequear el estado
IF cobra_monto_fijo = 0 THEN
    RETURN 0.00;
END IF;

-- Calculo los adicionales que le corresponden el período indicado
CALL obtener_adicional(id_asistente, mes, anio, adicional);

-- Por si todos los adicionales son NULL, quiero retornar siempre números
IF adicional IS NULL THEN
    RETURN 0.00;
ELSE
    RETURN adicional;
END IF;
END$$
```

La función **adicional\_montos\_fijos** es la que realiza la cuenta para generar el adicional el sueldo de un empleado que cobra montos fijos

Dentro de ella, llama a varios procedimientos, encargados de la lectura de las tablas para obtener datos.

Para evitar dar resultados de NULL en el caso de no haber pagos para el período, chequea si el adicional es NULL y devuelve 0,00 en ese caso.

Esta función puede ser utilizada por el usuario directamente. También es utilizada por el procedimiento *honorarios* para generar una tabla de honorarios para el mes.

# Gestión de clínica odontológica pequeña – Funciones



```
CREATE FUNCTION limpiar_string(input VARCHAR(50))
RETURNS VARCHAR(50)
DETERMINISTIC
BEGIN
    DECLARE cleaned_string VARCHAR(50);
    DECLARE c VARCHAR(1);
    DECLARE i, n INT;

    SET cleaned_string = '';
    SET n = CHAR_LENGTH(input);
    SET i = 0;

    -- Permite que empiece con un + ya que puede ser un teléfono internacional
    -- Se usa para otras cosas que no son teléfonos, pero las chances que se dé algo así son bajas
    IF LOCATE('+', input) IN (1, 2) THEN
        SET cleaned_string = '+';
    END IF;

    -- Hago un loop sobre los caracteres, chequeando si son números. Si lo son, los agrego al string limpio
    WHILE i <= n DO
        SET c = RIGHT(LEFT(input, i),1);

        IF LOCATE(c, '0123456789') != 0 THEN
            SET cleaned_string = CONCAT(cleaned_string, c);
        END IF;

        SET i = i + 1;
    END WHILE;

    RETURN cleaned_string;
END$$
```

La función **limpiar\_string** se utiliza para generar datos sin caracteres innecesarios. Se utiliza para limpiar números de teléfono, documento y CUIT que generalmente contienen guiones, puntos y demás.

Como input se tiene el string a limpiar.

La función hace un loop sobre todos los caracteres del string, chequeando si el carácter es un número. Si lo es, lo agrega a un nuevo string limpio.

Hace una sola salvedad. Si el primer carácter es un signo +, lo deja, ya que corresponde a un número de teléfono internacional. El signo + en otra posición es removido.

Esto puede traer problemas de que no corresponda realmente a un teléfono, pero las chances son bajas, especialmente considerando que los que ingresan los datos son empleados y no pacientes directamente.

La función **validar\_email** se utiliza evaluar la estructura de una dirección de email y determinar si conforma a las reglas actuales. Lo hace mediante una expresión regular que no entiendo totalmente. Devuelve un booleano, indicando si la dirección es válida o no.

```
CREATE FUNCTION validar_email (input VARCHAR(50))
RETURNS TINYINT
NO SQL
BEGIN
    -- El REGEXP la saque de https://stackoverflow.com/questions/12759596/validate-email-addresses-in-mysql
    -- No se bien como funciona, pero chequeandolo parece andar bien con los formatos válidos de email del presente
    IF input REGEXP '^[a-zA-Z0-9.!#$%&'^*+/-=?_{}~]*@[a-zA-Z0-9._-]*@[a-zA-Z0-9._-]*?[a-zA-Z][a-zA-Z]{2,63}$' THEN
        RETURN 1;
    END IF;
    RETURN 0;
END$$
```

*magia*

# Gestión de clínica odontológica pequeña – Vistas

**CODERHOUSE**

Finalmente, presento las vistas generadas para esta base de datos. Primero, tenemos la vista **historia\_clinica**.

```
CREATE OR REPLACE VIEW historia_clinica AS (
  SELECT t.fecha, t.hora, p.documento, e.descripcion, em.apellido AS 'odontologo', tl.precio AS 'costo_laboratorio', lb.nombre AS 'laboratorio'
  FROM evoluciones e
  JOIN turnos t ON e.id_turno = t.id_turno
  JOIN empleados em ON em.id_empleado = e.id_empleado
  JOIN pacientes p ON t.id_paciente = p.id_paciente
  LEFT JOIN trabajos_laboratorio tl ON e.id_trabajo_laboratorio = tl.id_trabajo_laboratorio
  LEFT JOIN laboratorios lb ON tl.id_laboratorio = lb.id_laboratorio
  ORDER BY fecha, hora
);
```

```
SELECT * FROM historia_clinica WHERE documento = '18895062';
```

fecha	hora	documento	descripcion	odontologo	costo_laboratorio	laboratorio
2022-02-12	10:00:00	18895062	Se cementó un perno sobre la pieza 23	Galarza	NULL	NULL
2022-04-25	11:30:00	18895062	Se cementó un perno sobre la pieza 26	Galarza	NULL	NULL
2022-06-17	12:00:00	18895062	Se colocó y atornilló un implante de porcelana sobre el implante colocado en la pieza 3	Rodón	15000.00	Fym Dental Lab

```
SELECT * FROM historia_clinica WHERE documento = '34352612' AND fecha BETWEEN '2022-03-01' AND '2022-07-01';
```

fecha	hora	documento	descripcion	odontologo	costo_laboratorio	laboratorio
2022-04-25	10:30:00	34352612	Se colocó un implante de titanio luego de una cir...	Benitez	NULL	NULL
2022-04-25	16:30:00	34352612	Se realizó una radiografía panorámica de la boc...	Brea	NULL	NULL
2022-06-17	10:30:00	34352612	Se colocó y atornilló un implante de porcelana s...	Wilson	5000.00	NULL
2022-06-17	10:30:00	34352612	Se realizó una radiografía panorámica de la boc...	Wilson	NULL	NULL
2022-06-17	14:00:00	34352612	Se le preparó una placa miorrelajante a la pacie...	Galarza	10000.00	NULL
2022-06-17	14:00:00	34352612	Se colocó un implante de titanio luego de una cir...	Galarza	NULL	NULL
2022-06-17	14:00:00	34352612	Se realizó una limpieza completa con ultrasonido...	Lillian	NULL	NULL
2022-06-17	14:00:00	34352612	Se colocó un implante de titanio luego de una cir...	Lillian	NULL	NULL
2022-06-17	14:00:00	34352612	Se removió una caries en la pieza 24	Lillian	NULL	NULL

Esta vista detalla todas la información de los tratamientos que se realizó un paciente en el centro.

Es muy importante para el funcionamiento del centro.

Como los datos fueron generados al azar se da que un paciente tiene 3 turnos en el mismo día con 3 odontólogos distintos, dos de ellos a la misma hora. NO está implementado el control de que esto no pueda suceder

# Gestión de clínica odontológica pequeña – Vistas



La vista **agenda** permite obtener información de los turnos. Se utiliza para obtener los turnos de un día, semana o mes en particular, para mostrarlo en un front-end

```
CREATE OR REPLACE VIEW agenda AS (
    SELECT tu.fecha, tu.hora, e.apellido AS 'odontologo', CONCAT(p.nombre, ' ', p.apellido) AS 'paciente', p.documento, p.celular AS 'contacto'
    FROM turnos tu
    JOIN pacientes p ON tu.id_paciente = p.id_paciente
    JOIN empleados e ON tu.id_empleado = e.id_empleado
    WHERE tu.id_estado_turno = 3 -- Los turnos que todavía no sucedieron
    AND tu.id_tratamiento IS NULL -- Ignoro los turnos que corresponden a estudios radiológicos
    ORDER BY fecha, hora
);
```

Toma únicamente los turnos que no sucedieron e ignora los turnos radiológicos (tienen su propia vista)

```
SELECT * FROM agenda WHERE fecha = '2022-12-13' ORDER BY odontologo;
```

fecha	hora	odontologo	paciente	documento	contacto
2022-12-13	10:00:00	Benitez	Yvonne Lyons	34352612	1535280599
2022-12-13	12:00:00	Benitez	Solomon Nguyen	18895062	1553730945
2022-12-13	13:00:00	Benitez	Mingo Gomez	29568965	1554129021
2022-12-13	17:30:00	Benitez	Veda Kaufman	12929457	1531582363
2022-12-13	10:00:00	Galarza	Pedro Julian Milan	17845025	1543569014
2022-12-13	12:00:00	Galarza	Jana Duran	11158527	1555647381
2022-12-13	13:00:00	Galarza	Solomon Nguyen	18895062	1553730945
2022-12-13	17:30:00	Galarza	Usnavi Gonzalez	31642589	1549863025
2022-12-13	10:00:00	Lillian	Juliana Ester Porto	23457898	1543689501
2022-12-13	12:00:00	Lillian	Melisa Sertain	40158965	1537894014
2022-12-13	13:00:00	Lillian	Pamela Campos	23330615	1551002884
2022-12-13	17:30:00	Lillian	Juan Straciattella	34521478	1549769856
2022-12-13	10:00:00	Ponce	Melisa Sertain	40158965	1537894014
2022-12-13	12:00:00	Ponce	Esteban Julio Nuñ...	45165896	1544398855
2022-12-13	13:00:00	Ponce	Jana Duran	11158527	1555647381
2022-12-13	17:30:00	Ponce	Cameron Mayo	29190432	1538523683
2022-12-13	10:00:00	Rodón	Solomon Nruuen	18895062	1553730945

La agenda de la clínica en un día en particular

```
SELECT * FROM agenda WHERE WEEK(fecha) = 50 ORDER BY odontologo;
```

fecha	hora	odontologo	paciente	documento	contacto
2022-12-13	10:00:00	Benitez	Yvonne Lyons	34352612	1535280599
2022-12-13	12:00:00	Benitez	Solomon Nguyen	18895062	1553730945
2022-12-13	13:00:00	Benitez	Mingo Gomez	29568965	1554129021
2022-12-13	17:30:00	Benitez	Veda Kaufman	12929457	1531582363
2022-12-13	10:00:00	Galarza	Pedro Julian Milan	17845025	1543569014
2022-12-13	12:00:00	Galarza	Jana Duran	11158527	1555647381
2022-12-13	13:00:00	Galarza	Solomon Nguyen	18895062	1553730945
2022-12-13	17:30:00	Galarza	Usnavi Gonzalez	31642589	1549863025
2022-12-13	10:00:00	Lillian	Juliana Ester Porto	23457898	1543689501
2022-12-13	12:00:00	Lillian	Melisa Sertain	40158965	1537894014
2022-12-13	13:00:00	Lillian	Pamela Campos	23330615	1551002884
2022-12-13	17:30:00	Lillian	Juan Straciattella	34521478	1549769856
2022-12-13	10:00:00	Ponce	Melisa Sertain	40158965	1537894014
2022-12-13	12:00:00	Ponce	Esteban Julio Nuñ...	45165896	1544398855
2022-12-13	13:00:00	Ponce	Jana Duran	11158527	1555647381
2022-12-13	17:30:00	Ponce	Cameron Mayo	29190432	1538523683
2022-12-13	10:00:00	Rodón	Solomon Nruuen	18895062	1553730945

Esta consulta permite obtener los turnos de una semana en particular, pero como solamente se ingresaron para un día de algunos meses, parece que no funciona, pero sí.

## Gestión de clínica odontológica pequeña – Vistas



Similar a la anterior, la vista **agenda\_radiologica** incluye todos los turnos futuros asignados a un estudio radiológico.

```
CREATE OR REPLACE VIEW agenda_radiologica AS (
    SELECT tu.fecha, tu.hora, CONCAT(p.nombre, ' ', p.apellido) AS 'paciente', p.documento, p.celular AS 'contacto', tr.nombre AS 'tratamiento'
    FROM turnos tu
    JOIN pacientes p ON tu.id_paciente = p.id_paciente
    JOIN tratamientos tr ON tu.id_tratamiento = tr.id_tratamiento
    WHERE id_estado_turno = 3 -- Los turnos que todavía no sucedieron
    AND tu.id_empleado IS NULL -- Sólo los turnos para radiológicos
    ORDER BY fecha, hora
);
```

Toma únicamente los turnos que no sucedieron e ignora los turnos con odontólogos.

```
SELECT * FROM agenda_radiologica WHERE WEEK(fecha) = 50;
```

fecha	hora	paciente	documento	contacto	tratamiento
2022-12-13	10:30:00	Cruz Cash	44218202	1516314639	Radiografía panorámica
2022-12-13	16:00:00	Colt Young	26981742	1572127341	Radiografía panorámica

# Gestión de clínica odontológica pequeña – Vistas

CODERHOUSE

```
CREATE OR REPLACE VIEW performance AS (
    SELECT em.apellido AS 'odontólogo', tr.nombre AS 'tratamiento', COUNT(ev.id_tratamiento) AS 'cantidad', tu.fecha
    FROM evoluciones ev
    JOIN empleados em ON ev.id_empleado = em.id_empleado
    JOIN tratamientos tr ON ev.id_tratamiento = tr.id_tratamiento
    JOIN turnos tu ON ev.id_turno = tu.id_turno
    WHERE id_tipo_empleado IN (
        SELECT id_tipo_empleado
        FROM tipo_de_empleado
        WHERE atiende = 1 -- Sólo me fijo en la performance de odontólogos
    )
    GROUP BY em.apellido, tr.nombre, tu.fecha
);
```

```
SELECT tratamiento, SUM(cantidad) AS 'cantidad'
FROM performance
WHERE odontólogo = 'Galarza'
AND YEAR(fecha) = '2022' GROUP BY tratamiento;
```

tratamiento	cantidad
Perno	5
Corona sobre implante	2
Tratamiento de conducto	4
Implante	3
Caries	3
Alineadores	2
Placa miorrelajante	2
Radiografía panorámica	1

Ahora para vistas que permiten generar informes, la vista **performance** me permite evaluar el desempeño de los odontólogos.



# Gestión de clínica odontológica pequeña – Vistas

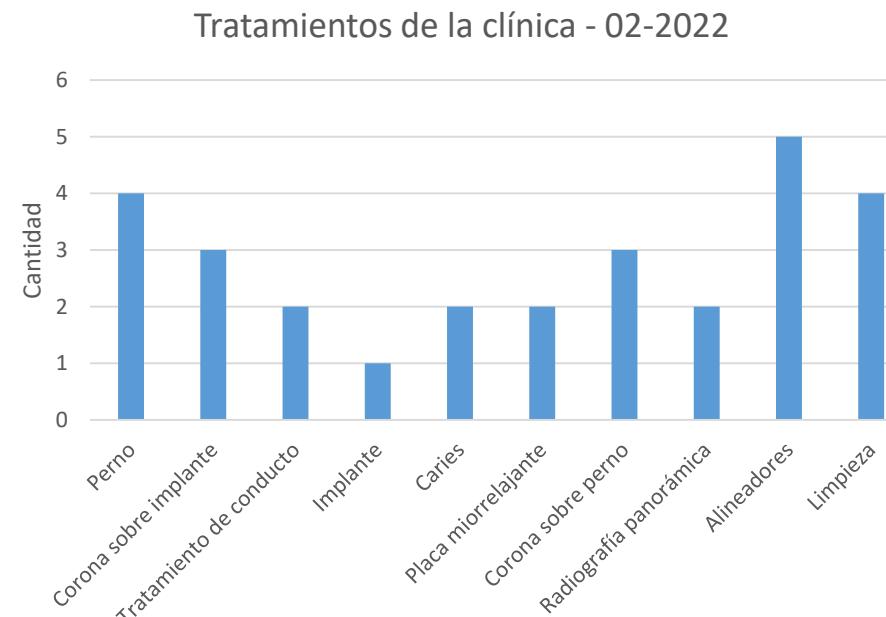
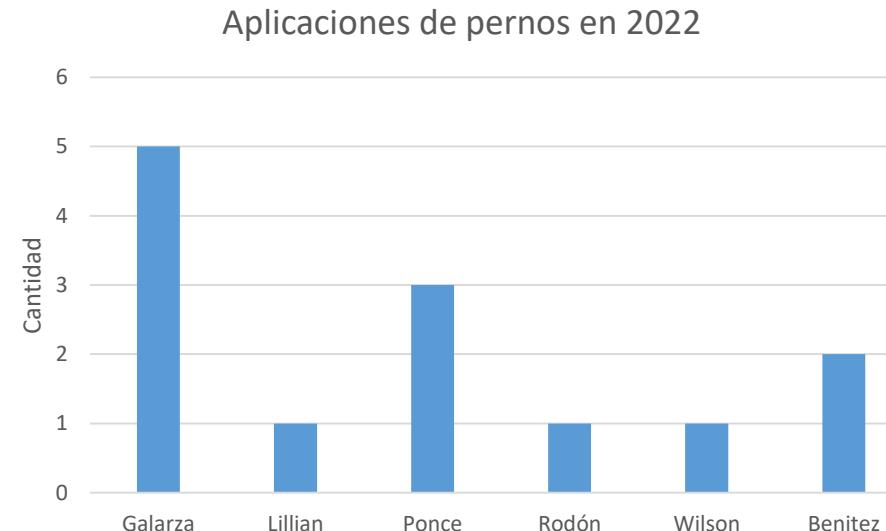
CODERHOUSE

```
SELECT odontologo, SUM(cantidad) AS 'cantidad'  
FROM performance  
WHERE tratamiento = 'Perno'  
AND YEAR(fecha) = '2022' GROUP BY odontologo;
```

odontologo	cantidad
Galarza	5
Lillian	1
Ponce	3
Rodón	1
Wilson	1
Benitez	2

```
SELECT tratamiento, SUM(cantidad) AS 'cantidad'  
FROM performance  
WHERE MONTH(fecha) = 2 AND YEAR(fecha) = '2022'  
GROUP BY tratamiento;
```

tratamiento	cantidad
Perno	4
Corona sobre implante	3
Tratamiento de conducto	2
Implante	1
Caries	2
Placa miorrelajante	2
Corona sobre perno	3
Radiografia panorámica	2
Alineadores	5
Limpieza	4



Esta vista me permite ver todo acerca de los tratamientos que se están haciendo en la clínica

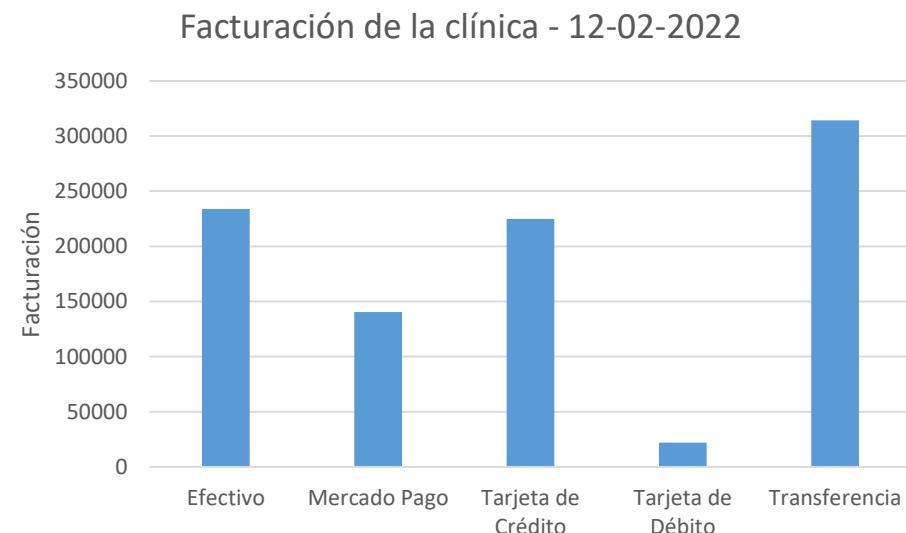
# Gestión de clínica odontológica pequeña – Vistas

La vista **facturación\_clinica** me permite evaluar el estado de la facturación de la clínica entera.

```
CREATE OR REPLACE VIEW facturacion_clinica AS (
  SELECT p.fecha, mp.modo, tr.nombre AS 'tratamiento', SUM(p.monto) as 'facturacion'
  FROM pagos p
  JOIN modo_pago mp ON p.id_modo_pago = mp.id_modo_pago
  JOIN evoluciones ev ON p.id_evolucion = ev.id_evolucion
  JOIN tratamientos tr ON ev.id_tratamiento = tr.id_tratamiento
  GROUP BY p.fecha, mp.modo, tr.nombre
  ORDER BY p.fecha
);
```

```
SELECT modo, SUM(facturacion) as 'facturacion'
FROM facturacion_clinica
WHERE fecha = '2022-02-12'
GROUP BY modo;
```

modo	facturacion
Efectivo	233759.00
Mercado Pago	140365.00
Tarjeta de Crédito	224805.00
Tarjeta de Débito	22100.25
Transferencia	314381.12

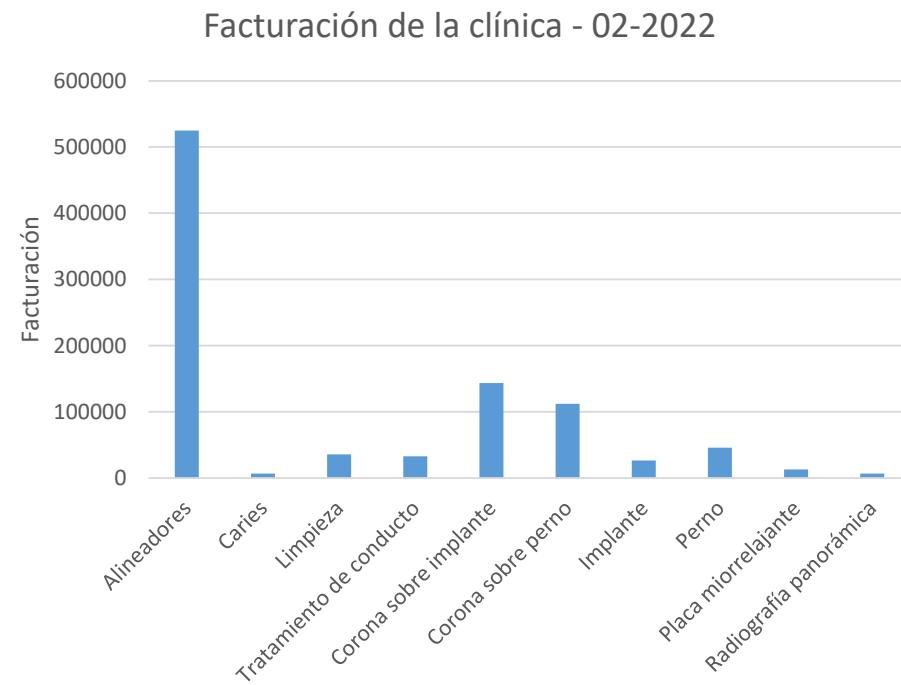


# Gestión de clínica odontológica pequeña – Vistas

**CODERHOUSE**

```
SELECT tratamiento, SUM(facturacion) AS 'facturacion'  
FROM facturacion_clinica  
WHERE MONTH(fecha) = 2  
GROUP BY tratamiento;
```

tratamiento	facturacion
Alineadores	525000.00
Caries	7000.00
Limpieza	35800.00
Tratamiento de conducto	33000.00
Corona sobre implante	143565.00
Corona sobre perno	112350.00
Implante	26750.00
Perno	46000.00
Placa morrelajante	13000.00
Radiografía panorámica	6601.00



# Gestión de clínica odontológica pequeña – Vistas

**CODERHOUSE**

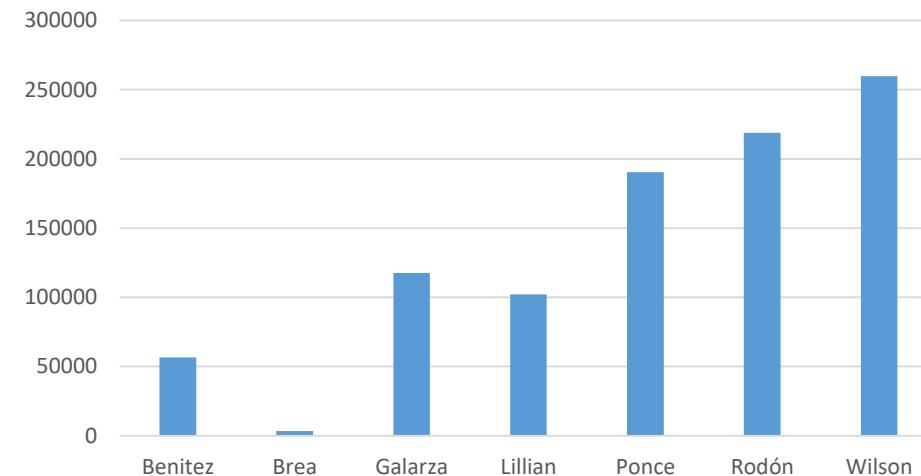
La vista **facturación odontólogo** me permite evaluar el estado de la facturación de los odontólogos individuales.

```
CREATE OR REPLACE VIEW facturacion_odontologo AS (
    SELECT p.fecha, em.apellido AS 'profesional', em.id_empleado, tr.nombre AS 'tratamiento', SUM(p.monto) as 'facturacion'
    FROM pagos p
    JOIN evoluciones ev ON p.id_evolucion = ev.id_evolucion
    JOIN tratamientos tr ON ev.id_tratamiento = tr.id_tratamiento
    JOIN empleados em ON ev.id_empleado = em.id_empleado
    GROUP BY p.fecha, em.apellido, tr.nombre
    ORDER BY p.fecha
);
```

```
SELECT profesional, SUM(facturacion) AS 'facturacion'
FROM facturacion_odontologo
WHERE MONTH(fecha) = 2 AND YEAR(fecha) = 2022
GROUP BY profesional;
```

profesional	facturacion
Benitez	56600.25
Brea	3300.50
Galarza	117605.00
Lillian	102210.00
Ponce	190500.25
Rodón	218950.00
Wilson	259900.00

Facturación de odontólogos - 02-2022



# Gestión de clínica odontológica pequeña – Vistas

CODERHOUSE

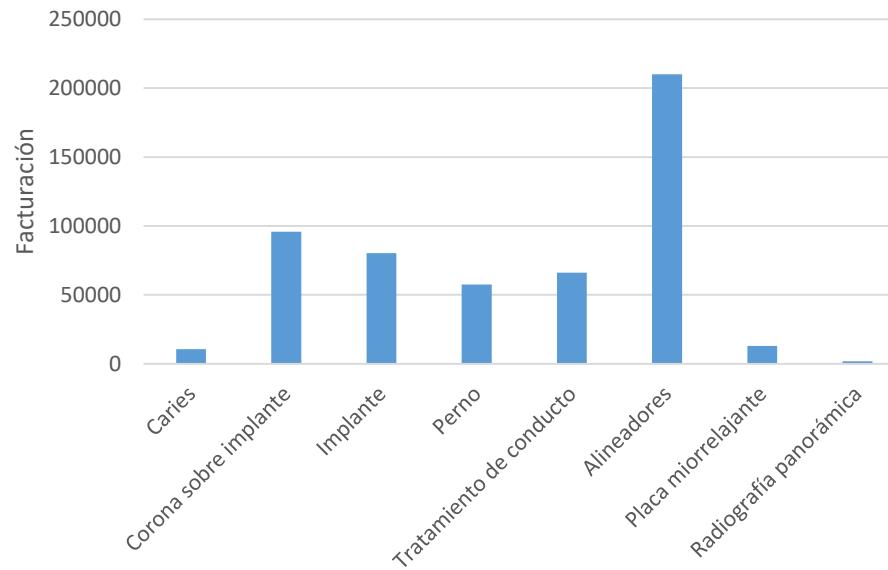
```
SELECT tratamiento, SUM(facturacion) AS 'facturacion'  
FROM facturacion_odontologo  
WHERE profesional = 'Galarza'  
GROUP BY tratamiento;
```

tratamiento	facturacion
Caries	10500.00
Corona sobre implante	95710.00
Implante	80250.00
Perno	57500.00
Tratamiento de conducto	66000.00
Alineadores	210000.00
Placa miorrelajante	13000.00
Radiografía panorámica	1650.25

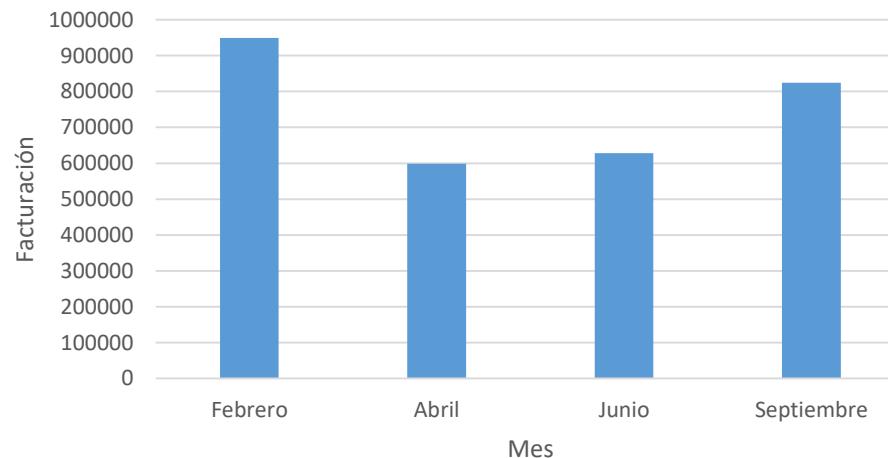
```
SELECT MONTH(fecha), SUM(facturacion) AS 'facturacion'  
FROM facturacion_odontologo  
GROUP BY MONTH(fecha);
```

MONTH(fecha)	facturacion
2	949066.00
4	598860.50
6	628366.25
9	824086.00

Facturación por tratamiento - Galarza 2022



Facturación por mes



Esto se puede hacer con la vista anterior también

## Gestión de clínica odontológica pequeña – Vistas



Por último, la vista **bajo\_stock** me permite ver una lista de productos consumibles que es necesario pedir, ya que hay menos de la cantidad mínima determinada

```
CREATE OR REPLACE VIEW bajo_stock AS (
    SELECT st.nombre, st.variedad, CONCAT(st.cantidad, ' ', st.presentacion) AS 'stock_actual', CONCAT(st.cantidad_recomendada, ' ', st.presentacion) AS 'pedido_recomendado'
    FROM stock st
    WHERE cantidad <= cantidad_minima
    ORDER BY st.nombre
);
```

```
SELECT * FROM bajo_stock;
```

nombre	variedad	stock_actual	pedido_recomendado
Barbijos quirúrgicos	NULL	3 Caja/s	10 Caja/s
Fresa p/ torno	Cónica tamaño 3	0 Unidad/es	8 Unidad/es
Fresa p/ torno	Cilíndrica tamaño 3	0 Unidad/es	20 Unidad/es
Guantes de latex	Talle XL	7 Caja/s	8 Caja/s
Lima p/ conducto	Tamaño 2	0 Unidad/es	10 Unidad/es
Lima p/ conducto	Tamaño 3	1 Unidad/es	10 Unidad/es
Lima p/ conducto	Tamaño 5	3 Unidad/es	10 Unidad/es
Papel absorbente p/ conducto	NULL	7 Unidad/es	8 Unidad/es

Un avance de esta vista sería vincular cada producto con proveedores posibles, de manera de generar uno o más pedidos automáticamente para cubrir todas las necesidades. Queda para el próximo curso

Tabla	tipo_de_empleado								
<b>Descripción</b> Contiene los tipos de empleados en el centro con campos que permiten generar una facturación									
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_tipo_empleado	INT			TRUE	TRUE		TRUE	
	título	VARCHAR	50		TRUE	TRUE			
	atiende	TINYINT			TRUE		1		Booleano. Si el tipo de empleado realiza tratamientos odontológicos
	porcentaje_tratamiento	TINYINT		FALSE	TRUE		40		Porcentaje de las ganancias que se lleva el empleado. Trigger BEFORE controla que esté entre 0 y 100
	porcentaje_laboratorio	TINYINT		FALSE	TRUE		50		Porcentaje del costo de laboratorio que se deduce del pago del paciente. Trigger BEFORE controla que esté entre 0 y 100
	lleva_monto_fijo	TINYINT			TRUE		0		Booleano. Indica si se lleva el monto fijo estipulado en tabla tratamientos, si tuviera
Tabla	tratamientos								
<b>Descripción</b> Describe los tratamientos que se realizan en la clínica									
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_tratamiento	INT			TRUE	TRUE		TRUE	
	nombre	VARCHAR	50		TRUE				
	nomenclador	VARCHAR	10		TRUE	TRUE			Códigos de tratamiento según la sociedad argentina de odontología (inventados por ahora)
	precio	DECIMAL	(10, 2)		TRUE				
	monto_fijo	DECIMAL	(10, 2)		TRUE		0		Monto fijo que se lleva el empleado que lo realiza, si le corresponde
	trabajo_laboratorio	TINYINT			TRUE		0		Booleano. Determina si el tratamiento incluye un trabajo de laboratorio
	solo_odontologos	TINYINT			TRUE		1		Booleano. Determina si el tratamiento lo puede realizar sólo odontólogos
Tabla	laboratorios								
<b>Descripción</b> Incluye los laboratorios con los que trabaja la clínica									
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_laboratorio	INT			TRUE	TRUE		TRUE	
	nombre	VARCHAR	50		TRUE				
	telefono	VARCHAR	20		TRUE				Trigger BEFORE para limpiar input
	direccion	VARCHAR	255						
	email	VARCHAR	50						Trigger BEFORE para validar estructura de email
	cuit	VARCHAR	11			TRUE			Trigger BEFORE para limpiar input
	activo	TINYINT			TRUE		1		Booleano. Si actualmente se está trabajando con este laboratorio
Tabla	modo_pago								
<b>Descripción</b> Incluye los modos de pago disponibles									
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_modo_pago	INT			TRUE	TRUE		TRUE	
	modo	VARCHAR	50		TRUE	TRUE			
Tabla	generos								
<b>Descripción</b> Tabla de generos asignables a personas									
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_genero	INT			TRUE	TRUE		TRUE	
	genero	VARCHAR	50		TRUE	TRUE			
Tabla	estado_turno								
<b>Descripción</b> Describe las opciones para el estado de un turno asignado									
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_estado_turno	INT			TRUE	TRUE		TRUE	
	estado	VARCHAR	20		TRUE	TRUE			
Tabla	estado_trabajo_laboratorio								
<b>Descripción</b> Describe las opciones para el estado de un trabajo de laboratorio									
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_estado_trabajo_laboratorio	INT			TRUE	TRUE		TRUE	
	estado	VARCHAR	20		TRUE	TRUE			
Tabla	pacientes								
<b>Descripción</b> Almacena los pacientes con su información personal y de contacto									
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_paciente	INT			TRUE	TRUE		TRUE	
	nombre	VARCHAR	50		TRUE				
	apellido	VARCHAR	50		TRUE				
	documento	DECIMAL	(10, 0)		TRUE	TRUE			Trigger BEFORE para limpiar input
FK	id_genero	INT			TRUE				
	fecha_de_nacimiento	DATE			TRUE				
	email	VARCHAR	50						Trigger BEFORE para validar estructura de email
	celular	VARCHAR	20						Trigger BEFORE para limpiar input y otro para chequear que provea un teléfono o un celular
	telefono	VARCHAR	20						Trigger BEFORE para limpiar input y otro para chequear que provea un teléfono o un celular
Tabla	empleados								
<b>Descripción</b> Almacena los empleados del centro odontológico con sus datos									
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_empleado	INT			TRUE	TRUE		TRUE	
	nombre	VARCHAR	50		TRUE				
	apellido	VARCHAR	50		TRUE				
	documento	DECIMAL	(10, 0)		TRUE	TRUE			Trigger BEFORE para limpiar input
FK	id_genero	INT			TRUE				
	fecha_de_nacimiento	DATE			TRUE				
	email	VARCHAR	50			TRUE			Trigger BEFORE para validar estructura de email
	celular	VARCHAR	20						Trigger BEFORE para limpiar input
	direccion	VARCHAR	255						
FK	id_tipo_empleado	INT			TRUE				
	activo	TINYINT			TRUE		1		Booleano. Si el empleado está trabajando actualmente
Tabla	honorarios_definitivos								
<b>Descripción</b> Tabla de auditoría sobre los honorarios que cobran los odontólogos. Se puebla cuando se "cierra el mes" y si hay errores se deben corregir a mano.									
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
	fecha_definicion	DATE			TRUE	TRUE	CURDATE()		
	usuario	VARCHAR	50		TRUE				Se ingresa sólo con un trigger BEFORE por ser una función volátil
PK	id_empleado	INT			TRUE				Junto a mes y año, forman una PRIMARY KEY
	honorario	DECIMAL	(10, 2)		TRUE		2		Junto a año e id_empleado, forman una PRIMARY KEY
	mes	INT			TRUE				Junto a id_empleado y mes, forman una PRIMARY KEY
	año	INT			TRUE				
Tabla	turnos								
<b>Descripción</b> Incluye todos los turnos que se reservan en la clínica									
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_turno	INT			TRUE	TRUE		TRUE	
	fecha	DATE			TRUE				
	hora	TIME			TRUE				
FK	id_estado_turno	INT			TRUE		2		
FK	id_paciente	INT			TRUE				
FK	id_empleado	INT			TRUE				Lleva valor si es un turno con un odontólogo. Trigger BEFORE chequea que sea un odontólogo y que sea exclusivo con la comuna id_tratamiento
FK	id_tratamiento	INT			TRUE				Lleva valor si es un turno para un estudio. Trigger BEFORE chequea que sea un tratamiento adecuado y que sea exclusivo con la comuna id_empleado
Tabla	trabajos_laboratorio								
<b>Descripción</b> Incluye todos los trabajos que se mandan a hacer a un laboratorio externo o interno que se cobra por separado. Se crea automáticamente una fila vacía al hacer un INSERT de una evolución con un id_tratamiento que lo necesita									
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_trabajo_laboratorio	INT			TRUE	TRUE		TRUE	
FK	id_laboratorio	INT			TRUE				
	presto	DECIMAL	(10, 2)						Puede ser NULL al ingresar el trabajo, pero al momento de generar las facturaciones debe tener un valor
FK	id_estado_trabajo_laboratorio	INT			TRUE		1		Estado actual del trabajo de laboratorio. Si tiene un estado de iniciado y se le asigna un valor a id_laboratorio, se cambia el estado a Asignado con trigger
Tabla	evoluciones								
<b>Descripción</b> Cada fila es una evolución en la historia clínica de un paciente. Representa un único tratamiento descrito por la tabla tratamientos. Permite obtener historias clínicas									
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_evolucion	INT			TRUE	TRUE		TRUE	
FK	id_tratamiento	INT			TRUE				
	descripcion	VARCHAR	1000						
FK	id_turno	INT			TRUE				
FK	id_empleado	INT			TRUE				
FK	id_trabajo_laboratorio	INT			TRUE				Si el id_tratamiento corresponde un trabajo de laboratorio, se crea automáticamente una fila vacía en esa tabla y se llena esta columna con esa fila nueva
Tabla	pagos								
<b>Descripción</b> Pagos que realizan los clientes									
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_pago	INT			TRUE	TRUE		TRUE	

	fecha	DATE		TRUE	CURDATE()				
PK	id_evolucion	INT		TRUE		Puede haber más de un pago por evolución			
	monto	DECIMAL (10, 2)		TRUE	1				
FK	id_modo_pago	INT		TRUE					
Tabla	<b>log_pagos</b>								
Descripción	Tabla de auditoría de la tabla pagos. Se llena cada vez que se modifica la tabla pagos								
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT		
PK	id_log_pagos	INT			TRUE	TRUE	TRUE		
	fecha	DATE			TRUE	CURDATE()			
	hora	TIME			TRUE	CURTIME()			
	evento	VARCHAR	20		TRUE		Evento realizado sobre la tabla pagos (INSERT, UPDATE)		
FK	objetivo	INT			TRUE		id de la fila objetivo del evento		
	usuario	VARCHAR	50		TRUE				
Tabla	<b>pagos_audit</b>								
Descripción	Tabla de auditoría secundaria de la tabla pagos. Almacena los datos de las filas que sufren un UPDATE								
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
FK	id_changed_log	INT			TRUE	TRUE		TRUE	id de la fila de log_pagos que registró este UPDATE
	old_row	VARCHAR	3000		TRUE				string generado de la fila original de la tabla pagos
	new_row	VARCHAR	3000		TRUE				string generado de la nueva fila modificada en la tabla pagos
Tabla	<b>log_evoluciones</b>								
Descripción	Tabla de auditoría de la tabla evoluciones. Se llena cada vez que se modifica la tabla evoluciones								
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_log_evoluciones	INT			TRUE	TRUE		TRUE	
	fecha	DATE			TRUE	CURDATE()			
	hora	TIME			TRUE	CURTIME()			
	evento	VARCHAR	20		TRUE				Evento realizado sobre la tabla pagos (INSERT, UPDATE)
FK	objetivo	INT			TRUE				id de la fila objetivo del evento
	usuario	VARCHAR	50		TRUE				
Tabla	<b>evoluciones_audit</b>								
Descripción	Tabla de auditoría secundaria de la tabla evoluciones. Almacena los datos de las filas que sufren un UPDATE								
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
FK	id_changed_log	INT			TRUE	TRUE		TRUE	id de la fila de log_evoluciones que registró este UPDATE
	old_row	VARCHAR	3000		TRUE				string generado de la fila original de la tabla evoluciones
	new_row	VARCHAR	3000		TRUE				string generado de la nueva fila modificada en la tabla evoluciones
Tabla	<b>stock</b>								
Descripción	Tabla que lleva cuenta de los materiales consumibles y sus cantidades. Fuera de el ingreso inicial, se modifica automáticamente con los INSERT en las tablas consumos_stock e ingresos_stock								
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_producto	INT			TRUE	TRUE		TRUE	
	nombre	VARCHAR	100		TRUE				
	cantidad	INT			FALSE	TRUE			
	cantidad_minima	INT			FALSE	TRUE	1		Cantidad mínima para no ser considerado de bajo stock. Trigger BEFORE chequea que no sea mayor que cantidad_recomendada
	cantidad_recomendada	INT			FALSE				Cantidad recomendada para realizar un pedido cuando está con bajo stock. Trigger BEFORE chequea que no sea menor que cantidad_minima
	presentación	VARCHAR	50		TRUE				
	variedad	VARCHAR	50						Variabilidad sobre un mismo producto, por ejemplo el tamaño de guantes de latex
	ubicación	VARCHAR	50						Ubicación en un depósito
Tabla	<b>proveedores</b>								
Descripción	Tabla que junta los proveedores de los productos de stock								
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_proveedor	INT			TRUE	TRUE		TRUE	
	nombre	VARCHAR	50		TRUE				
	telefono	VARCHAR	50		TRUE				Trigger BEFORE para limpiar el input a sólo numéricos
	dirección	VARCHAR	255						
	email	VARCHAR	50						Trigger BEFORE para validar la estructura del email
	url	VARCHAR	100						
	cult	VARCHAR	11		TRUE	TRUE	1		Trigger BEFORE para limpiar el input a sólo numéricos
	activo	TINYINT			TRUE		1		Booleano. Si se están realizando pedidos a este proveedor
Tabla	<b>consumos_stock</b>								
Descripción	Tabla que lleva cuenta de las reducciones del stock de consumibles. Modifica la tabla de stock								
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_consumo	INT			TRUE	TRUE		TRUE	
	fecha	DATE			TRUE	CURDATE()			
	hora	TIME			TRUE	CURTIME()			
FK	id_producto	INT			TRUE				
	cantidad	INT			FALSE	TRUE			Trigger BEFORE chequea que haya suficiente stock de este producto para este consumo
FK	id_empleado_retira	INT			TRUE				id del empleado que lo retira del depósito
FK	id_empleado_utiliza	INT			TRUE				id del empleado que lo utilizará en un consultorio
Tabla	<b>estado_de_pedido</b>								
Descripción	Tabla que detalla los estados en los que pueden estar los pedidos de consumibles								
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_estado_pedido	INT			TRUE	TRUE		TRUE	
	estado	VARCHAR	20		TRUE	TRUE			
Tabla	<b>pedidos_stock</b>								
Descripción	Tabla que detalla los pedidos que se realizan a los proveedores								
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_pedido	INT			TRUE	TRUE		TRUE	
FK	id_proveedor	INT			TRUE				
	fecha_ingreso	DATE			TRUE	CURDATE()			
	precio_total	DECIMAL (10, 2)							
FK	id_estado_pedido	INT							
FK	id_empleado_recibe	INT							Trigger BEFORE UPDATE chequea que se llene esta columna si pasa a estado 'Recibido'
FK	id_empleado_controla	INT							Trigger BEFORE UPDATE chequea que se llene esta columna y la anterior si pasa a estado 'Chequeado'
Tabla	<b>ingresos_stock</b>								
Descripción	Tabla que detalla los ingresos de productos al stock, correspondientes a los pedidos de la tabla pedidos_stock. Modifica la tabla de stock								
INDEX	COLUMN	TYPE	LEN	SIGNED	NOT NULL	UNIQUE	DEFAULT	AUTO_INCREMENT	NOTES
PK	id_ingreso	INT			TRUE	TRUE		TRUE	
FK	id_pedido	INT			TRUE				
FK	id_producto	INT			TRUE				
	cantidad_ingresada	INT			FALSE	TRUE			