

# Proyecto final – Curso SQL

Proyecto ‘Travel Agency’.

**Documentación técnica del proyecto: base de datos relacional para “Agencia de viajes – vuelos a bajo costo”**



**Pertenece a: Kary Yessenia Francia Vásquez**

**Comisión: 43420**

**Profesor: Gabriel Almiñana**

**Tutor: José Ignacio López Saez**

INTRODUCCIÓN Y OBJETIVO.....	2
SITUACIÓN PROBLEMÁTICA Y SOLUCIÓN PROPUESTA .....	3
MODELO DE NEGOCIOS .....	4
DIAGRAMA DE ENTIDAD RELACIÓN.....	6
DESCRIPCIÓN DETALLADA DE LAS TABLAS.....	8
DESCRIPCIÓN DETALLADA DE LAS TABLAS BITÁCORA .....	12
VISTAS.....	13
FUNCIONES .....	19
STORED PROCEDURES (SP) .....	22
TRIGGERS .....	28
USUARIOS .....	35
TRANSACCIONES .....	37
TRANSACCIONES Y RESTAURACIÓN.....	39
INFORMES GENERADOS: ANÁLISIS DE DATOS CLAVE .....	40
“POSTERIORES RUMBOS” .....	44
“HERRAMIENTAS Y TECNOLOGÍAS UTILIZADAS” .....	45
SCRIPTS .....	46

## Introducción:

Cada vez son más las personas que buscan estar interconectadas y conocer el mundo que vivimos personalmente, es por ello que la demanda de vuelos a bajo costo se ha visto incrementada en los últimos años. Esto ocasiona que surjan diversas opciones para los clientes potenciales (viajeros) que buscan reservar viajes de forma rápida, sencilla y a un precio accesible.



En este contexto, se presenta el caso de una agencia de viajes online “ficticia” (‘Travel Agency’ S.R.L) que ofrece a sus clientes la opción de hacer reservas de vuelos a bajo costo con un nivel mínimo de servicio y confort. La empresa, tendrá un servicio básico que es accesible para todos los clientes, pero también ofrecerá una variedad de servicios adicionales que los clientes pueden adaptar, según sus preferencias individuales, mejorando su experiencia de viaje.

Es por ello que, la base de datos relacional que se presenta en este proyecto, almacena toda la información necesaria para que los clientes puedan hacer las reservas de vuelos, tales como las fechas de salida y llegada, el destino, la aerolínea, el tipo de asiento, equipaje, etc. Asimismo, la base de datos también almacena la información sobre los clientes, los empleados, las campañas de promociones de la agencia, los comentarios que hacen los clientes sobre su experiencia, promociones, avisos para los clientes sobre la información de su vuelo, etc.

## Objetivo:

El objetivo del proyecto es crear una base de datos para respaldar las operaciones de ‘Travel Agency’, el cual se especializa en ofrecer vuelos de bajo costo.

La plataforma de ‘Travel Agency’, proporciona a los clientes la autonomía de *customizar* su experiencia según sus necesidades y presupuesto. Es por ello que, se implementa un modelo de negocios en el que se ofrece un servicio básico y accesible para todos. Sin embargo, se ofrece también la posibilidad de que el cliente mejore el confort y acceda a servicios adicionales mediante opciones de pago extra.

## Situación problemática:

La situación problemática radica en que la agencia opera sin una base de datos centralizada y eficiente para gestionar la información de sus clientes, vuelos, itinerarios, reservas y transacciones. Esta carencia de una infraestructura sólida de gestión de datos puede dar lugar a una serie de problemas y dificultades que afectan tanto a la operación interna de la agencia como a la satisfacción de los clientes.

Sin una base de datos que almacene la información sobre los vuelos, los asientos disponibles y las reservas realizadas, la agencia podría tener dificultades para gestionar eficientemente las reservas de los clientes.

En consecuencia, la falta de implementación de una base de datos relacional podría llevar a errores en la asignación de asientos, duplicación de reservas y otros problemas de logística. Asimismo, una base de datos ausente limita la capacidad de la agencia para analizar patrones de compra, preferencias de destinos y otros datos relevantes que podrían ayudar a identificar oportunidades comerciales y de tendencias en el mercado. Esta falta de visibilidad podría resultar en una pérdida de oportunidades estratégicas para 'Travel Agency'.

## Solución propuesta:

Para abordar la problemática antes mencionada, 'Travel Agency' debe invertir en la implementación de un sistema de gestión de bases de datos eficiente, que permita centralizar y organizar la información de manera adecuada, ocasionando a su vez una mejora en la personalización de los servicios, la eficiencia en la gestión de vuelos y reservas, el servicio al cliente, las oportunidades comerciales y la seguridad de los datos.



Con una base de datos sólida, la agencia estará mejor equipada para tomar decisiones informadas y brindar una experiencia de usuario más satisfactoria.

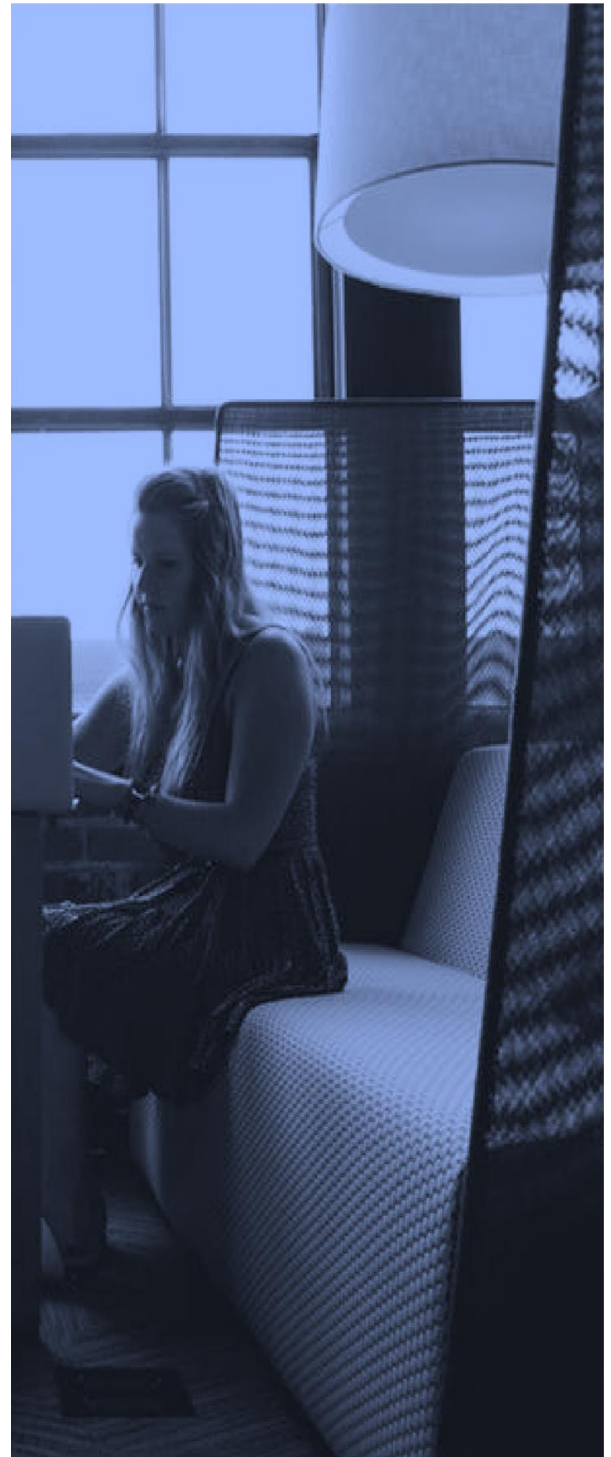
# Modelo de negocios

**“Travel Agency’ logra diferenciarse de sus competidores ofreciendo a los clientes una combinación de precios bajos, una variedad de servicios y una experiencia de usuario sencilla.”**

El modelo de negocios está orientado en proporcionar a los clientes una experiencia de viaje flexible y fácil de *customizar*, ofreciendo vuelos a bajo costo. La plataforma de ‘Travel Agency’ funciona como un agente de viajes online que colabora con cinco aerolíneas para ofrecer tarifas competitivas, ofreciendo como destino ocho países.

Para poder satisfacer las necesidades y preferencias individuales de los clientes, la base de datos permite la personalización modular. Es decir, los clientes pueden agregar servicios adicionales, como asientos con más espacio, mayor capacidad de llevar equipaje y otras comodidades, mediante opciones de pago extra.

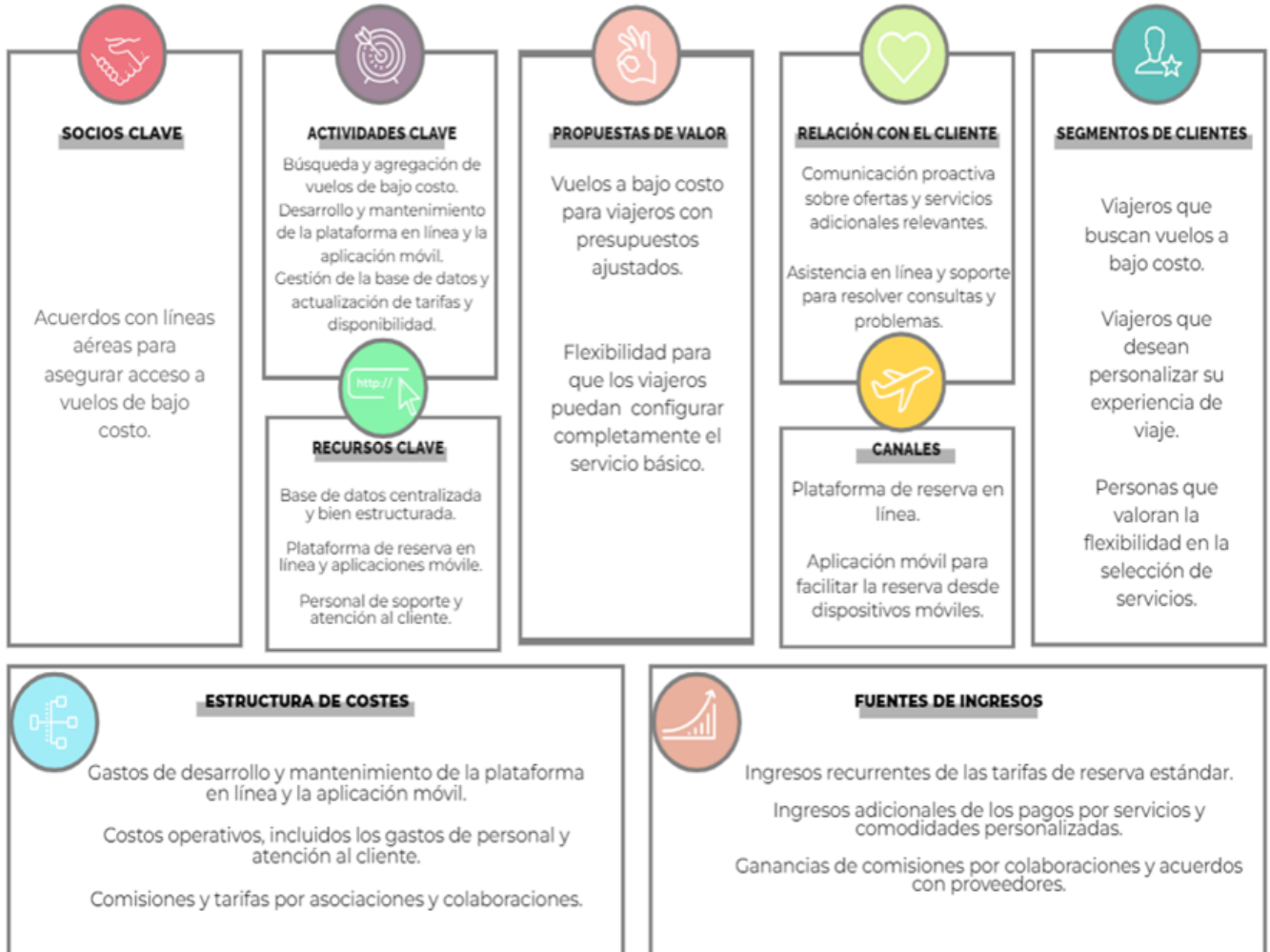
Asimismo, para garantizar la calidad del servicio, se implementa un sistema de comentarios que permite a los clientes compartir sus experiencias y brindar un *feedback* sobre los servicios utilizados. Esta información será valiosa para que la empresa pueda realizar nuevas ofertas y adaptarlas a las necesidades cambiantes de los clientes.



# Modelo de negocios

## CANVAS:

### MODELO DE NEGOCIOS AGENCIA DE VIAJES ONLINE



**Travel agency**  
LOW COST FLIGHTS

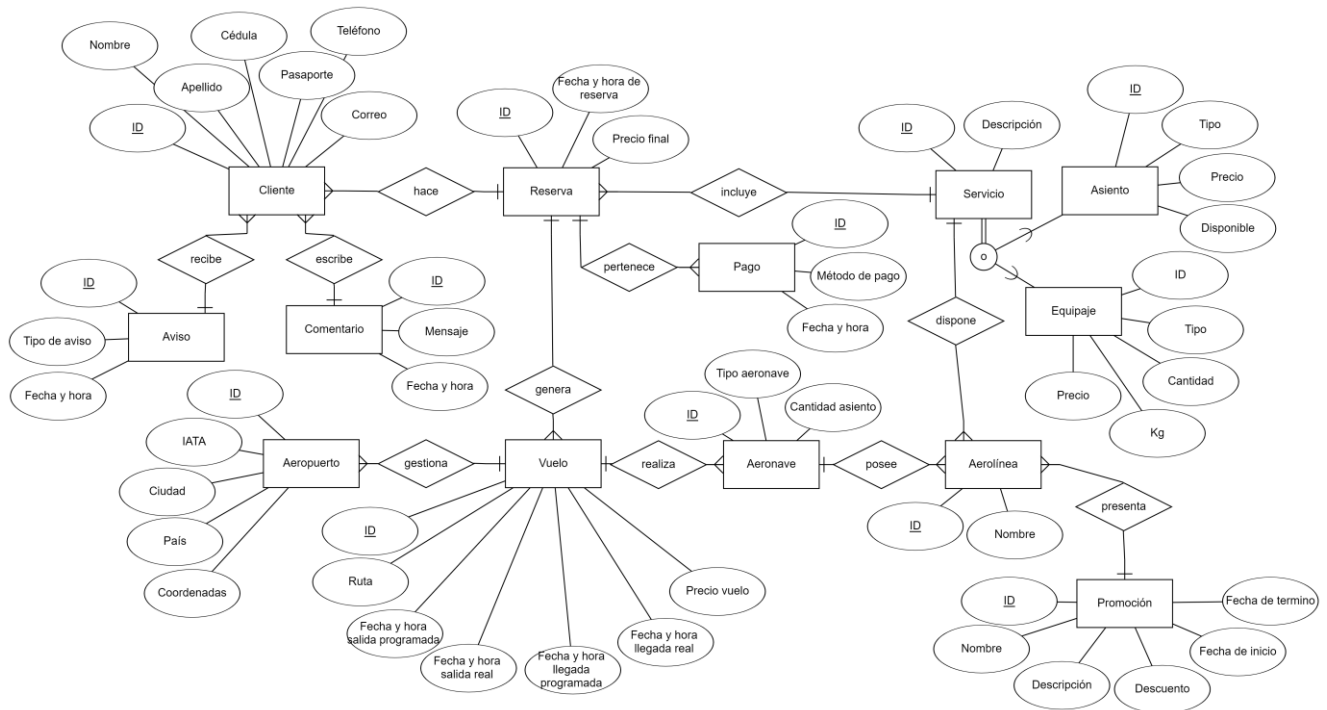
En resumen, se cree que, el modelo de negocios es sostenible porque los consumidores están dispuestos a sacrificar algunas comodidades para ahorrar dinero, y la empresa puede ofrecer un servicio básico que sea asequible para todos los clientes. Además, este modelo de negocio es escalable porque la empresa puede aumentar sus ingresos al aumentar el número de vuelos que vende y al ofrecer más servicios adicionales.



# Diagrama de entidad relación

6

Académicamente se realizó el siguiente diagrama E-R:



**Entidades:** Aerolínea, aeropuerto, asiento, aviso, cliente, comentario, equipaje, escala, aeronave, pago, promoción, reserva, servicio, vuelo.

**Acciones de relacionamiento:** Hace, incluye, genera, realiza, posee, gestiona, dispone, escribe, recibe, encuentra, pertenece, sirve, ajusta, presenta.

## Tipos de relación:

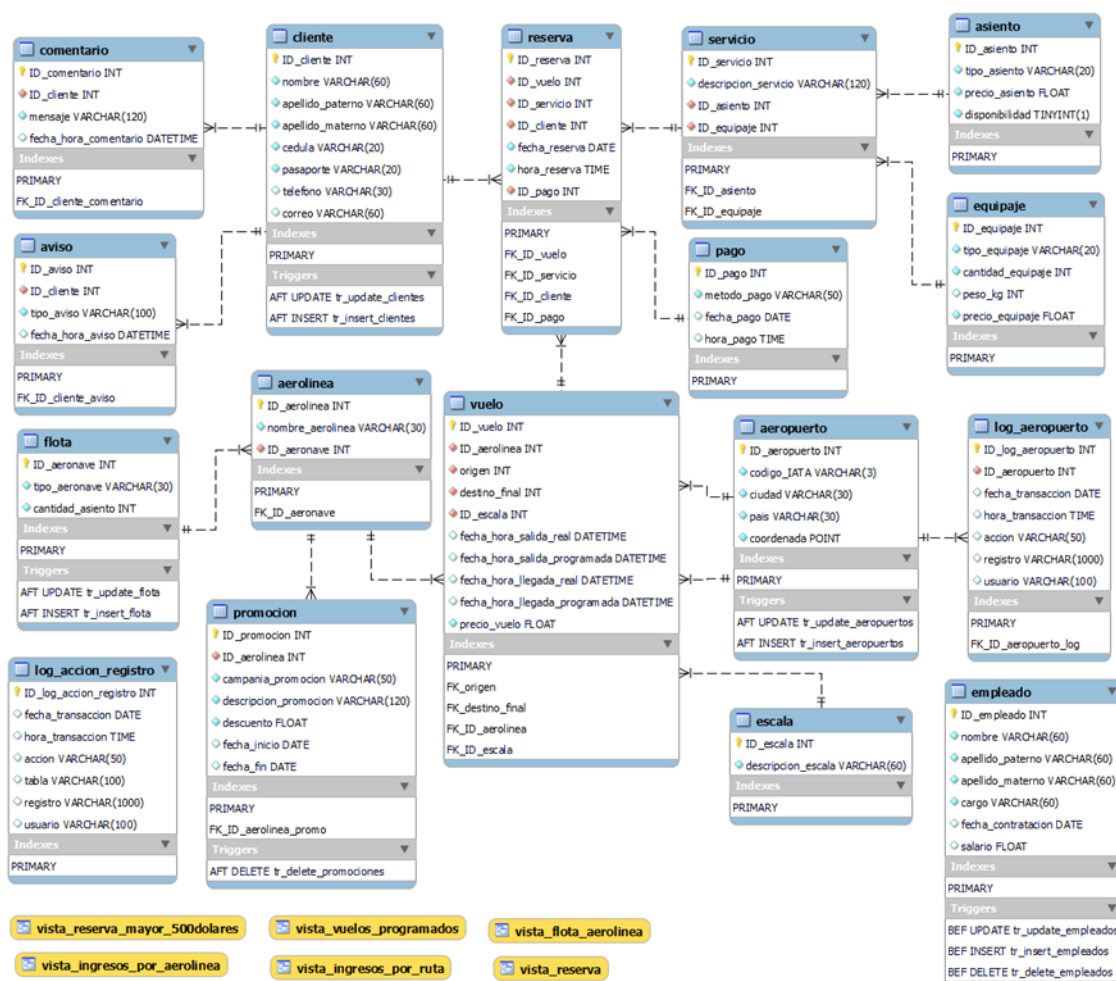
- Un cliente hace muchas reservas. 1: N
- Un servicio se incluye en muchas reservas. 1: N
- Un vuelo genera muchas reservas. 1: N
- Una aeronave realiza muchos vuelos: 1: N
- Una aerolínea posee muchas aeronaves. 1: N
- Un aeropuerto gestiona muchos vuelos. 1: N
- Una aerolínea dispone de varios servicios. 1: N

# Diagrama de entidad relación

7

- Un cliente escribe muchos comentarios: 1: N
- Un cliente recibe varios avisos: 1: N
- Una escala se encuentra en varios vuelos: 1: N
- Un pago pertenece a muchas reservas: 1: N
- Un asiento sirve para varios servicios: 1: N
- Un equipaje se ajusta a varios servicios: 1: N
- Una aerolínea presenta varias promociones. 1: N

Luego de realizar el script para crear la base de datos **Reserva\_Vuelo** y sus respectivas tablas, en MySQL Workbench, se obtuvo el siguiente diagrama E-R:





# Descripción detallada de las tablas

8

A continuación, se muestra el listado de las tablas que conforman la base de datos, con una descripción detallada de su estructura:

➤ **‘Cliente’**: Pasajeros que viajan en las aeronaves de una aerolínea.

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
Cliente	ID_cliente	identificador del cliente	PK		INT
	nombre	nombre del cliente			VARCHAR (60)
	apellido_paterno	apellido paterno del cliente			VARCHAR (60)
	apellido_materno	apellido materno del cliente			VARCHAR (60)
	cedula	número de documento de identidad del cliente			VARCHAR (20)
	pasaporte	número de pasaporte del cliente			VARCHAR (20)
	telefono	teléfono celular del cliente			VARCHAR (30)
	correo	correo electronico del cliente			VARCHAR (60)

➤ **‘Asiento’**: Es la ubicación donde viaja el pasajero dentro de la cabina de pasajeros.

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
Asiento	ID_asiento	identificador del tipo de asiento	PK		INT
	tipo_asiento	nombre del asiento según ubicación en aeronave			VARCHAR (20)
	precio_asiento	precio del asiento en dólares			FLOAT
	disponibilidad	disponibilidad del asiento (libre = TRUE u ocupado = FALSE)			BOOLEAN

➤ **‘Equipaje’**: Son las cosas que lleva el pasajero al viaje, depende del tamaño y la capacidad en kilos.

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
Equipaje	ID_equipaje	identificador del tipo de equipaje	PK		INT
	tipo_equipaje	nombre del equipaje según su peso			VARCHAR (20)
	cantidad_equipaje	número de equipajes			INT
	peso_kg	peso de equipajes en kilos			INT
	precio_equipaje	precio del equipaje en dólares			FLOAT

➤ **‘Aeropuerto’**: Instalación que ocupa una superficie extensa de terreno con pistas adecuadas para el aterrizaje y despegue de aeronaves, su carga, descarga y mantenimiento, y para el control del tráfico aéreo.

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
Aeropuerto	ID_aeropuerto	identificador del aeropuerto	PK		INT
	codigo_IATA	código IATA del aeropuerto			VARCHAR (3)
	ciudad	ciudad donde se encuentra ubicado el aeropuerto			VARCHAR (30)
	pais	país donde se encuentra ubicado el aeropuerto			VARCHAR (30)
	coordenada	latitud y longitud del centro del aeropuerto			POINT

# Descripción detallada de las tablas

9

- **‘Flota’**: Aeronaves pertenecientes a las aerolíneas.

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
Flota	ID_aeronave	identificador de la aeronave	PK		INT
	tipo_aeronave	modelo y marca de la aeronave			VARCHAR (30)
	cantidad_asiento	número de asiento que tiene una aeronave			INT

- **‘Empleado’**: Personal contratado por la empresa para desempeñar diversas funciones dentro de la agencia.

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
Empleado	ID_empleado	identificador del empleado	PK		INT
	nombre	nombre del empleado de la agencia			VARCHAR (60)
	apellido_paterno	apellido paterno del empleado de la agencia			VARCHAR (60)
	apellido_materno	apellido materno del empleado de la agencia			VARCHAR (60)
	cargo	cargo que ocupa el empleado en la agencia			VARCHAR (60)
	fecha_contratacion	fecha de contrato del empleado			DATE
	salario	salario que percibe el empleado en dólares mensualmente			FLOAT

- **‘Pago’**: Método de pago por el cual el cliente entrega a la agencia una cantidad de dinero por la reserva que desea comprar.

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
Pago	ID_pago	identificador del pago	PK		INT
	metodo_pago	tipo de pago efectuado para la compra del boleto aéreo			VARCHAR (50)
	fecha_pago	fecha en la que se efectua el pago			DATE
	hora_pago	hora en la que se efectua el pago			TIME

- **‘Escala’**: Número de paradas intermedias programadas en el itinerario de un viaje aéreo entre el punto de origen y el destino final.

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
Escala	ID_escalas	identificador de la cantidad de escalas	PK		INT
	descripcion_escalas	describe cuantas escalas tiene el vuelo			VARCHAR (60)

- **‘Servicio’**: Incluye el básico que es selección aleatoria de asiento y un equipaje de mano; y la opción de configurarla con servicios complementarios basados en mejor ubicación de asientos y mayor capacidad de equipaje.

# Descripción detallada de las tablas

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
Servicio	ID_servicio	identificador del servicio	PK		INT
	descripcion_servicio	describe el tipo de servicio y sus adicionales			VARCHAR (120)
	ID_asiento	identificador del tipo de asiento		FK	INT
	ID equipaje	identificador del tipo de equipaje		FK	INT

➤ 'Aerolinea': Compañías que ofrecen diferentes servicios de vuelo.

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
Aerolinea	ID_aerolinea	identificador de la aerolinea	PK		INT
	nombre_aerolinea	nombre de la compañía aérea			VARCHAR (30)
	ID_aeronave	identificador de la aeronave		FK	INT

➤ 'Vuelo': Es el tramo de ruta de viaje que realizan los pasajeros.

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
Vuelo	ID_vuelo	identificador del vuelo	PK		INT
	ID_aerolinea	identificador de la aerolinea		FK	INT
	origen	identificador del aeropuerto de donde despegue la aeronave		FK	INT
	destino_final	identificador del aeropuerto final en donde aterriza la aeronave		FK	INT
	ID_escala	identificador de la cantidad de escalas		FK	INT
	fecha_hora_salida_programada	fecha y hora de despegue programada en la ficha de vuelo			DATETIME
	fecha_hora_llegada_programada	fecha y hora de aterrizaje programado en la ficha de vuelo			DATETIME
	fecha_hora_salida_real	fecha y hora de despegue real del aeropuerto de origen			DATETIME
	fecha_hora_llegada_real	fecha y hora de aterrizaje real al destino final			DATETIME
	precio_vuelo	precio del vuelo			FLOAT

➤ 'Reserva': Órdenes que generan los clientes al momento de comprar un billete aéreo y sus servicios.

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
Reserva	ID_reserva	identificador de la reserva de vuelo	PK		INT
	ID_vuelo	identificador del número de vuelo		FK	INT
	ID_servicio	identificador del servicio		FK	INT
	ID_cliente	identificador del cliente		FK	INT
	fecha_reserva	fecha en la que se realiza la reserva de compra			DATE
	hora_reserva	hora en la que se realiza la reserva de compra			TIME
	ID_pago	identificador del pago		FK	INT

# Descripción detallada de las tablas

11

- **‘Comentario’:** Opinión que emite el cliente a la agencia.

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
Comentario	ID_comentario	identificador de la opinión del cliente	PK		INT
	ID_cliente	identificador del cliente		FK	INT
	mensaje	comentario que realiza el cliente			VARCHAR (120)
	fecha_hora_comentario	fecha y hora en la que el cliente realiza el comentario			DATETIME

- **‘Promocion’:** Campaña publicitaria que realiza la agencia para determinados itinerarios de una compañía aérea durante un tiempo limitado mediante una oferta atractiva.

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
Promocion	ID_promocion	identificador de la promoción	PK		INT
	ID_aerolinea	identificador de la aerolinea		FK	INT
	campania_promocion	nombre de la campaña de promoción			VARCHAR (50)
	descripcion_promocion	describe que incluye la promoción			VARCHAR (120)
	descuento	porcentaje de descuento realizado sobre la tarifa normal			FLOAT
	fecha_inicio	fecha de inicio de la promoción			DATE
	fecha_fin	fecha de termino de la promoción			DATE

- **‘Aviso’:** Notificaciones que recibe el cliente por parte de la agencia para comunicarle acciones y actualizaciones sobre su reserva de manera oficial.

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
Aviso	ID_aviso	identificador del aviso	PK		INT
	ID_cliente	identificador del cliente		FK	INT
	tipo_aviso	mensaje de notificación que recibe el cliente			VARCHAR(100)
	fecha_hora_aviso	fecha y hora en la que recibe la notificación el cliente			DATETIME

- *Log\_aeropuerto*: Bitácora que registra los movimientos realizados en la tabla aeropuertos, estos movimientos solo pueden ser de actualización o inserción de registro.

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
log_aeropuerto	ID_log_aeropuerto	identificador de la bitacora de movimientos de la tabla aeropuertos	PK		INT
	ID_aeropuerto	identificador del aeropuerto		FK	INT
	fecha_transaccion	fecha en la que se realiza una acción en la tabla aeropuertos			DATE
	hora_transaccion	hora en la que se realiza una acción en la tabla aeropuertos			TIME
	accion	movimiento que se realiza en la tabla aeropuertos: inserción y actualización			VARCHAR (50)
	registro	Información del registro resultante de la acción realizada			VARCHAR (1000)
	usuario	usuario quien realiza la acción en la tabla aeropuertos			VARCHAR (100)

- *'Log\_acción\_registro'*: Bitácora matriz que registra todos los movimientos realizados en cualquiera de las tablas de la base de datos, estos movimientos pueden ser de actualización, inserción o eliminación de registro.

Tabla	Campos	Descripción	Clave primaria (PK)	Clave foránea (FK)	Tipos de datos
log_accion_registro	ID_log_accion_registro	identificador de la bitacora de movimientos en varias tablas	PK		INT
	fecha_transaccion	fecha en la que se realiza una acción en una tabla			DATE
	hora_transaccion	hora en la que se realiza una acción en una tabla			TIME
	accion	movimiento que se realiza en una tabla: inserción, actualización o eliminación			VARCHAR (50)
	tabla	nombre de la tabla en donde se realiza una accion			VARCHAR (100)
	registro	Información del registro resultante de la acción realizada			VARCHAR (1000)
	usuario	usuario quien realiza la acción en una tabla			VARCHAR (100)

Las vistas que conforman la base de datos se crean para fines de análisis, tomar decisiones de negocio o para rastrear el historial. Estas vistas se detallan a continuación:

- *'vista\_reserva'*: El objetivo de esta vista es obtener una descripción general de todas las reservas realizadas por los clientes.

Esta vista combina información de las tablas cliente, reserva, vuelo, aeropuerto, escala, aerolínea, servicio, asiento, equipaje y pago; devolviendo la siguiente información: nombre, apellido, correo electrónico de la tabla cliente; nombre de la aerolínea de la tabla aerolínea; los alias ae1 y ae2 se utilizan para las tablas aeropuerto (origen) y aeropuerto (destino final), respectivamente (estos alias permiten acceder a los códigos IATA de los aeropuertos); la descripción de la escala de la tabla escala; la fecha y hora de salida y llegada real; la descripción del servicio (equipaje) de la tabla servicio; el tipo de asiento y su precio de la tabla asiento; el precio del equipaje la tabla equipaje; el precio del vuelo de la tabla vuelo (los precios obtenidos se suman y recibe el alias de precio final; el método de pago de la tabla pago; y por último la vista ordena los resultados por la fecha de reserva en orden descendente, lo que significa que las reservas más recientes aparecerán primero en los resultados.

```
CREATE OR REPLACE view vista_reserva
AS
SELECT c.nombre AS nombre,
Concat(c.apellido_paterno, ' ', c.apellido_materno) AS apellidos,
c.correo,
aerolinea.nombre_aerolinea AS aerolinea,
ae1.codigo_iata AS origen,
ae2.codigo_iata AS 'destino final',
esc.descripcion_escalas AS escalas,
fecha_hora_salida_real AS
'fecha y hora salida',
fecha_hora_llegada_real AS
'fecha y hora llegada',
s.descripcion_servicio AS equipaje,
a.tipo_asiento AS asiento,
r.fecha_reserva AS 'fecha reserva',
( a.precio_asiento + e.precio_equipaje
+ v.precio_vuelo ) AS precio_final,
p.metodo_pago AS pago
FROM reserva r
INNER JOIN cliente c
```



```

        ON r.id_cliente = c.id_cliente
INNER JOIN vuelo

        ON r.id_vuelo = v.id_vuelo
INNER JOIN aeropuerto ael
        ON ael.id_aeropuerto = v.origen
INNER JOIN aeropuerto ae2
        ON ae2.id_aeropuerto = v.destino_final
INNER JOIN escala esc
        ON esc.id_escalas = v.id_escalas
INNER JOIN aerolinea
        ON aerolinea.id_aerolinea = v.id_aerolinea
INNER JOIN servicio s
        ON r.id_servicio = s.id_servicio
INNER JOIN asiento a
        ON s.id_asiento = a.id_asiento
INNER JOIN equipaje e
        ON s.id_equipaje = e.id_equipaje
INNER JOIN pago p
        ON r.id_pago = p.id_pago
ORDER BY r.fecha_reserva DESC;

```

- ‘*vista\_vuelos\_programados*’: El objetivo de esta vista es obtener una descripción general de todos los vuelos programados.

La vista combina información de las tablas vuelo, aerolinea, flota, aeropuerto y escala; devolviendo la siguiente información: El nombre de la aerolinea, el tipo de aeronave, el código IATA del aeropuerto de origen, el código IATA del aeropuerto de destino, la cantidad de escalas correspondientes al itinerario de vuelo, la fecha y hora de salida programada, la fecha y hora de llegada programada.

```

CREATE OR REPLACE VIEW vista_vuelos_programados
AS
SELECT aer.nombre_aerolinea          AS aerolinea,
       f.tipo_aeronave              AS aeronave,
       ael.codigo_iata              AS origen,
       ae2.codigo_iata              AS 'destino final',
       esc.descripcion_escalas      AS escalas,
       v.fecha_hora_salida_programada AS 'Salida programada',
       v.fecha_hora_llegada_programada AS 'Llegada Programada'
FROM   vuelo v
INNER JOIN aerolinea aer
        ON v.id_aerolinea = aer.id_aerolinea
INNER JOIN flota f
        ON aer.id_aeronave = f.id_aeronave
INNER JOIN aeropuerto ael
        ON ael.id_aeropuerto = v.origen

```

```
INNER JOIN aeropuerto ae2
    ON ae2.id_aeropuerto = v.destino_final
INNER JOIN escala esc
    ON esc.id_escalas = v.id_escalas;
```

- 'vista\_ingresos\_por\_ruta': El objetivo de la vista es calcular los ingresos por ruta en función de los precios de los asientos, equipajes y vuelos en las reservas realizadas.

La vista combina información de las tablas vuelo, reserva, servicio, asiento y equipaje; devolviendo la siguiente información: La ruta del vuelo (concatena los códigos IATA de los aeropuertos de origen y destino con un guion medio para formar una representación de la ruta) y los ingresos generados por la ruta (calcula la suma de los precios de asientos, equipajes y vuelos para obtener los ingresos totales por ruta). Finalmente, se agrupan los resultados por la columna "ruta", lo que significa que los cálculos SUM se realizarán para cada ruta única.

Este tipo de vista, por ejemplo, se podría usar para identificar las rutas que generan más ingresos, o para identificar las rutas que podrían generar más ingresos con una mayor promoción.

```
CREATE OR REPLACE VIEW vista_ingresos_por_ruta
AS
SELECT Concat(ae1.codigo_iata, '-', ae2.codigo_iata) AS ruta,
Sum(a.precio_asiento + e.precio_equipaje
    + v.precio_vuelo) AS ingresos
FROM vuelo v
INNER JOIN aeropuerto ae1
    ON ae1.id_aeropuerto = v.origen
INNER JOIN aeropuerto ae2
    ON ae2.id_aeropuerto = v.destino_final
INNER JOIN reserva r
    ON r.id_vuelo = v.id_vuelo
INNER JOIN servicio s
    ON r.id_servicio = s.id_servicio
INNER JOIN asiento a
    ON s.id_asiento = a.id_asiento
INNER JOIN equipaje e
    ON s.id_equipaje = e.id_equipaje
GROUP BY ruta;
```

- *'vista\_flota\_aerolinea'*: El objetivo de esta vista es obtener una descripción general de la flota de aeronaves de cada aerolínea. La vista proporciona una manera conveniente de acceder a la información sobre la flota de aeronaves de diferentes aerolíneas. Cada fila de la vista representará una combinación de aerolínea, tipo de aeronave y cantidad de asientos asociados con esa aeronave en la flota de la aerolínea correspondiente.

Esta vista puede ser útil para fines de planificación o para rastrear el historial de la flota. Por ejemplo, se podría usar para identificar las aerolíneas que tienen la flota más grande, o para identificar las aerolíneas que tienen la flota más moderna.

```
CREATE OR REPLACE VIEW vista_flota_aerolinea
AS
SELECT aer.nombre_aerolinea AS aerolinea,
       f.tipo_aeronave      AS aeronave,
       f.cantidad_asiento  AS 'cantidad de asientos'
FROM   aerolinea aer
       INNER JOIN flota f
              ON aer.id_aeronave = f.id_aeronave;
```

- *'vista\_ingresos\_por\_aerolinea'*: El objetivo de esta vista es obtener una descripción general de los ingresos generados por cada aerolínea. Para ello, se calcula los ingresos totales por aerolínea en función de los precios de los asientos, equipajes y vuelos en las reservas realizadas.

La vista combina información de las tablas reserva, vuelo, servicio, asiento, equipaje y aerolínea; devolviendo la siguiente información: El nombre de la aerolínea y los ingresos generados por la aerolínea. Se agrupan los resultados por la columna aerolinea, lo que significa que los cálculos SUM se realizarán para cada aerolínea única.

Esta vista, por ejemplo, se podría usar para identificar las aerolíneas que generan más ingresos, o para identificar las aerolíneas que podrían generar más ingresos con una mayor promoción.

```
CREATE OR REPLACE VIEW vista_ingresos_por_aerolinea
AS
  SELECT aer.nombre_aerolinea AS aerolinea,
         Sum(a.precio_asiento + e.precio equipaje
             + v.precio_vuelo) AS ingresos
  FROM   vuelo v
        INNER JOIN reserva r
              ON r.id_vuelo = v.id_vuelo
        INNER JOIN servicio s
              ON r.id_servicio = s.id_servicio
        INNER JOIN asiento a
              ON s.id_asiento = a.id_asiento
        INNER JOIN equipaje e
              ON s.id_equipaje = e.id_equipaje
        INNER JOIN aerolinea aer
              ON aer.id_aerolinea = v.id_aerolinea
  GROUP BY aerolinea;
```

- *vista\_ reserva\_mayor\_500dolares*: El objetivo de esta vista es mostrar una información detallada sobre las reservas que tienen un precio total (asientos, equipaje y vuelo) mayor a USD 500.

La vista combina información de las tablas reserva, cliente, vuelo, aeropuerto, escala, aerolinea, servicio, asiento, equipaje, pago; devolviendo la siguiente información: nombre, apellidos y correo del cliente, nombre de la aerolínea, código IATA del aeropuerto de origen y de destino final, la cantidad de escalas del itinerario, descripción del equipaje, tipo de asiento, la fecha de la reserva, el precio total de la reserva y el método de pago que utilizó el cliente para abonar la reserva. Finalmente, con la cláusula *WHERE*, se aplica una condición para seleccionar solo las reservas con un precio total mayor a USD 500; y con la cláusula *ORDER BY* se ordena los resultados en orden descendente según la fecha de la reserva.

Esta vista, por ejemplo, se podría usar para analizar las reservas que cuestan más de USD 500, generar informes sobre las reservas que cuestan más de USD 500 o para comparar los precios de las reservas que cuestan más de USD 500.

```

CREATE OR REPLACE VIEW vista_reserva_mayor_500dolares
AS
SELECT c.nombre AS nombre,
Concat(c.apellido_paterno, ' ', c.apellido_materno) AS apellidos,
c.correo,
aerolinea.nombre_aerolinea AS aerolinea,
ae1.codigo_iata AS origen,
ae2.codigo_iata AS 'destino final',
esc.descripcion_escalas AS escalas,
s.descripcion_servicio AS equipaje,
a.tipo_asiento AS asiento,
r.fecha_reserva AS fecha,
( a.precio_asiento + e.precio_equipaje
+ v.precio_vuelo ) AS precio_final,
p.metodo_pago AS pago
FROM reserva r
INNER JOIN cliente c
ON r.id_cliente = c.id_cliente
INNER JOIN vuelo v
ON r.id_vuelo = v.id_vuelo
INNER JOIN aeropuerto ae1
ON ae1.id_aeropuerto = v.origen
INNER JOIN aeropuerto ae2
ON ae2.id_aeropuerto = v.destino_final
INNER JOIN escala esc
ON esc.id_escalas = v.id_escalas
INNER JOIN aerolinea
ON aerolinea.id_aerolinea = v.id_aerolinea
INNER JOIN servicio s
ON r.id_servicio = s.id_servicio
INNER JOIN asiento a
ON s.id_asiento = a.id_asiento
INNER JOIN equipaje e
ON s.id_equipaje = e.id_equipaje
INNER JOIN pago p
ON r.id_pago = p.id_pago
WHERE ( a.precio_asiento + e.precio_equipaje
+ v.precio_vuelo ) > 500
ORDER BY r.fecha_reserva DESC;

```

A continuación, se detallan las funciones que conforman la base de datos:

- *'cantidad\_destino'*: El objetivo de esta función es obtener la cantidad de vuelos que llegan a una ciudad para fines de análisis o para tomar decisiones de negocios. Por ejemplo, la función se podría usar para identificar las ciudades más populares para volar, o las ciudades que podrían beneficiarse de un mayor número de vuelos.

Esta función, devuelve el número de vuelos al aeropuerto de destino especificado por el parámetro *p\_ciudad*. La función, primero, une las tablas vuelo, reserva y aeropuerto para obtener el número de vuelos al aeropuerto de destino. Luego, con la función *COUNT ()* se cuenta el número de filas en la tabla resultante.

```
DELIMITER //
CREATE FUNCTION
    `cantidad_destino` (p_ciudad VARCHAR (30)) RETURNS INT
    READS SQL DATA
BEGIN
    DECLARE resultado INT; SET resultado =
    (
        SELECT      count(destino_final) AS destino
        FROM        vuelo v
        INNER JOIN   reserva r
        ON          v.id_vuelo = r.id_vuelo
        INNER JOIN   aeropuerto AS aer
        ON          aer.id_aeropuerto=v.destino_final
        WHERE       ciudad = p_ciudad);

    RETURN resultado;

END //
```

- *'Total\_ingreso\_cliente'*: El objetivo de esta función es obtener el ingreso total generado por el cliente a fin de poder tomar decisiones de negocios. Por ejemplo, la función se podría usar para identificar los clientes que generan más ingresos, o para identificar los clientes que podrían beneficiarse de una mayor promoción.



Esta función devuelve el total de ingresos del cliente especificado por el parámetro *p\_id\_cliente*. En la función se declaran dos variables locales (*ingreso\_servicio* e *ingreso\_vuelo*).

En la variable *ingreso\_servicio*, se calcula el ingreso total por servicios (asientos y equipaje) para el cliente especificado. Se utilizan las tablas *servicio*, *asiento*, *equipaje* y *reserva*. Luego, con una función de agregación *SUM()* se suman los precios.

En la variable *ingreso\_vuelo*, se calcula el ingreso total por vuelos para el cliente especificado. Se utilizan las tablas *vuelo*, *reserva*. Luego, utiliza la función de agregación *SUM()* para sumar los precios.

Finalmente, la función suma los precios de los servicios y vuelos para obtener el total de ingresos del cliente.

```
DELIMITER //
CREATE FUNCTION
    `total_ingreso_cliente` (p_id_cliente INT) RETURNS INT
    READS SQL DATA
BEGIN
    DECLARE ingreso_servicio INT; DECLARE ingreso_vuelo INT; SET ingreso_servicio =
    (
        SELECT      sum(a.precio_asiento + e.precio_equipaje)
        FROM        servicio s
        INNER JOIN  asiento a
        ON          s.id_asiento = a.id_asiento
        INNER JOIN  equipaje e
        ON          s.id_equipaje = e.id_equipaje
        INNER JOIN  reserva r
        ON          s.id_servicio = r.id_servicio
        WHERE       r.id_cliente = p_id_cliente); SET ingreso_vuelo =
    (
        SELECT      sum(precio_vuelo)
        FROM        vuelo AS v
        INNER JOIN  reserva AS r
        ON          v.id_vuelo = r.id_vuelo
        WHERE       r.id_cliente = p_id_cliente);

    RETURN ingreso_servicio + ingreso_vuelo;
END //
```

- *'Total\_reserva\_cliente'*: El objetivo de esta función es obtener el número total de reservas realizadas por el cliente a fin de poder tomar decisiones de negocios. Por ejemplo, la función se podría usar para identificar los clientes que realizan más reservas, o para identificar los clientes que podrían beneficiarse de un mayor seguimiento.

Esta función devuelve el número total de reservas del cliente especificado por el parámetro *p\_id\_cliente*. La función primero consulta la tabla *reserva* para encontrar todas las reservas que están relacionadas con el cliente. Luego, con la función *COUNT* () se cuenta el número de filas en los resultados de la consulta y así se obtiene el número total de reservas del cliente.

```
DELIMITER //
CREATE FUNCTION
    total_reserva_cliente (p_id_cliente INT) RETURNS INT
    READS SQL DATA
BEGIN
    DECLARE resultado INT; SET resultado =
    (
        SELECT count(*)
        FROM    reserva
        WHERE   reserva.id_cliente = p_id_cliente);

    RETURN resultado;
END //
```

# Stored Procedures (SP)

Los SP que conforman la base de datos se detallan a continuación:

- *'Destinos\_frecuentados'*: El objetivo de este SP es obtener una lista de ciudades y la cantidad de veces que han sido destino final de vuelos, reservados en el sistema.

El SP realiza una serie de consultas a las siguientes tablas: aeropuerto, vuelo y reserva, utilizando *JOINS* para relacionar la información necesaria y agrupando los resultados por ciudad; devuelve una lista de los destinos más frecuentes, ordenados por el número de vuelos a cada destino, en orden descendente (para ver en la primera fila el destino más frecuentado por los clientes).

Esto podría ser útil para fines de análisis o para tomar decisiones de negocios. Por ejemplo, para identificar las ciudades más frecuentemente visitadas, o para identificar los destinos que podrían beneficiarse de un mayor número de vuelos.

```
DELIMITER //
CREATE PROCEDURE
    destinos_frecuentados()
BEGIN
    SELECT      aer.ciudad,
               count(destino_final) AS destino
    FROM        aeropuerto aer
    INNER JOIN  vuelo v
    ON          aer.id_aeropuerto = v.destino_final
    INNER JOIN  reserva r
    ON          v.id_vuelo = r.id_vuelo
    GROUP BY   ciudad
    ORDER BY   destino DESC;END //
```

- *'Update\_add\_flota'*: El objetivo de este SP es insertar o actualizar un registro con la información sobre las aeronaves de una aerolínea. Esta información puede ser útil para fines de planificación o para tomar decisiones de negocios. Por ejemplo, se podría usar para agregar una nueva aeronave a la flota de una aerolínea, o para actualizar la cantidad de asientos en una aeronave existente.

El SP toma tres parámetros: *p\_id\_aeronave*, *p\_tipo\_aeronave*, *p\_cantidad\_asiento*; y primero verifica si la aeronave ya existe en la tabla. Si existe, el procedimiento actualiza la cantidad de asientos en la aeronave. Si la aeronave no existe, el procedimiento la agrega a la tabla. Además, se realizan algunas validaciones para

# Stored Procedures (SP)

asegurarse de que el campo "p\_tipo\_aeronave" no esté vacío. Si el parámetro está vacío o no válido, el procedimiento genera un mensaje de error.

```
DELIMITER //
CREATE PROCEDURE
    update_add_flota (IN p_id_aeronave      INT,
                    IN p_tipo_aeronave    VARCHAR(30),
                    IN p_cantidad_asiento INT)
BEGIN
    DECLARE exists_aeronave BOOLEAN; SELECT EXISTS
        (
            SELECT id_aeronave
            FROM   flota
            WHERE  id_aeronave = p_id_aeronave)
    INTO exists_aeronave; IF p_tipo_aeronave = '' THEN
        SELECT 'El campo tipo de aeronave no puede estar vacio' AS errormsg; ELSEIF
        exists_aeronave THEN
            UPDATE flota
            SET     cantidad_asiento = p_cantidad_asiento
            WHERE  id_aeronave = p_id_aeronave; ELSEIF
            p_tipo_aeronave = '' THEN
                SELECT 'El campo tipo de aeronave no puede estar vacio'; ELSE
                INSERT INTO flota
                (
                    id_aeronave,
                    tipo_aeronave,
                    cantidad_asiento
                )
                VALUES
                (
                    p_id_aeronave,
                    p_tipo_aeronave,
                    p_cantidad_asiento
                ); END IF; END //
```

- 'Update\_add\_cliente': El objetivo de este SP es insertar o actualizar un registro con la información de los clientes. El SP toma los siguientes parámetros: *p\_id\_cliente*, *p\_nombre*, *p\_apellido\_paterno*, *p\_apellido\_materno*, *p\_cedula*, *p\_pasaporte*, *p\_telefono*, *p\_correo*; y primero verifica si el cliente ya existe en la tabla. Si existe, el procedimiento actualiza la información del cliente. Si el cliente no existe, el procedimiento lo agrega a la tabla. Además, se realizan algunas validaciones para asegurarse de que el campo "p\_cedula" no esté vacío. Si el parámetro está vacío o no válido, el procedimiento genera un mensaje de error.

# Stored Procedures (SP)

```

DELIMITER //
CREATE PROCEDURE
    update_add_cliente (IN p_id_cliente      INT,
                       IN p_nombre          VARCHAR(60),
                       IN p_apellido_paterno VARCHAR(60),
                       IN p_apellido_materno VARCHAR(60),
                       IN p_cedula          VARCHAR(20),
                       IN p_pasaporte       VARCHAR(20),
                       IN p_telefono        VARCHAR(30),
                       IN p_correo          VARCHAR(60))
BEGIN
    DECLARE exists_cliente BOOLEAN; SELECT EXISTS
        (
            SELECT id_cliente
            FROM   cliente
            WHERE  id_cliente = p_id_cliente)
    INTO  exists_cliente; IF p_cedula = '' THEN
        SELECT 'El campo cedula no puede estar vacio' AS errormsg; ELSEIF
        exists_cliente THEN
            UPDATE cliente
            SET
                nombre = p_nombre,
                apellido_paterno = p_apellido_paterno,
                apellido_materno = p_apellido_materno,
                cedula = p_cedula,
                pasaporte = p_pasaporte,
                telefono = p_telefono,
                correo = p_correo
            WHERE id_cliente = p_id_cliente; ELSEIF
            p_cedula = '' THEN
                SELECT 'El campo cedula no puede estar vacio'; ELSE
                INSERT INTO cliente
                    (
                        id_cliente,
                        nombre,
                        apellido_paterno,
                        apellido_materno,
                        cedula,
                        pasaporte,
                        telefono,
                        correo
                    )
                VALUES
                    (
                        p_id_cliente,
                        p_nombre,
                        p_apellido_paterno,
                        p_apellido_materno,
                        p_cedula,
                        p_pasaporte,
                        p_telefono,
                        p_correo
                    ); END IF; END //

```

# Stored Procedures (SP)

- 'Update\_add\_empleado': El objetivo de este SP es insertar o actualizar un registro con la información de los empleados. El SP toma los siguientes parámetros: *p\_id\_empleado*, *p\_nombre*, *p\_apellido\_paterno*, *p\_apellido\_materno*, *p\_cargo*, *p\_fecha\_contratacion*, *p\_salario*; y primero verifica si el empleado ya existe en la tabla. Si existe, el procedimiento actualiza la información del empleado. Si el empleado no existe, el procedimiento lo agrega a la tabla. Además, se realizan algunas validaciones para asegurarse de que el campo "p\_cargo" no esté vacío. Si el parámetro está vacío o no válido, el procedimiento genera un mensaje de error.

```

DELIMITER //
CREATE PROCEDURE
    update_add_empleado (IN p_id_empleado      INT,
                        IN p_nombre            VARCHAR(60),
                        IN p_apellido_paterno  VARCHAR(60),
                        IN p_apellido_materno  VARCHAR(60),
                        IN p_cargo              VARCHAR(60),
                        IN p_fecha_contratacion date,
                        IN p_salario           FLOAT)
BEGIN
    DECLARE exists_empleado BOOLEAN; SELECT EXISTS
    (
        SELECT id_empleado
        FROM    empleado
        WHERE   id_empleado = p_id_empleado
    )
    INTO exists_empleado; IF p_cargo = '' THEN
        SELECT 'El campo cargo no puede estar vacio' AS errmsg; ELSEIF
        exists_empleado THEN
            UPDATE empleado
            SET     nombre = p_nombre,
                  apellido_paterno = p_apellido_paterno,
                  apellido_materno = p_apellido_materno,
                  cargo = p_cargo,
                  fecha_contratacion = p_fecha_contratacion,
                  salario = p_salario
            WHERE  id_empleado = p_id_empleado; ELSEIF
            p_cargo = '' THEN
                SELECT 'El campo cargo no puede estar vacio'; ELSE
                INSERT INTO empleado
                (
                    id_empleado,
                    nombre,
                    apellido_paterno,
                    apellido_materno,

```



# Stored Procedures (SP)

```

        cargo,
        fecha_contratacion,
        salario
    )
VALUES
    (
        p_id_empleado,
        p_nombre,
        p_apellido_paterno,
        p_apellido_materno,
        p_cargo,
        p_fecha_contratacion,
        p_salario
    );END IF;END //

```

- *'Update\_add\_aeropuerto'*: El objetivo de este SP es insertar o actualizar un registro con la información de los aeropuertos. El SP toma los siguientes parámetros: *p\_id\_aeropuerto*, *p\_codigo\_IATA*, *p\_ciudad*, *p\_pais*, *p\_coordenada*; y primero verifica si el aeropuerto ya existe en la tabla. Si existe, el procedimiento actualiza la información del aeropuerto. Si el aeropuerto no existe, el procedimiento lo agrega a la tabla. Además, se realizan algunas validaciones para asegurarse de que el campo "p\_código\_IATA" no esté vacío. Si el parámetro está vacío o no válido, el procedimiento genera un mensaje de error.

```

DELIMITER //
CREATE PROCEDURE
    update_add_aeropuerto (IN p_id_aeropuerto INT,
                          IN p_codigo_iata   VARCHAR(3),
                          IN p_ciudad        VARCHAR(30),
                          IN p_pais          VARCHAR(30),
                          IN p_coordenada point)
BEGIN
    DECLARE exists_aeropuerto boolean;SELECT EXISTS
    (
        SELECT id_aeropuerto
        FROM   aeropuerto
        WHERE  id_aeropuerto = p_id_aeropuerto)
    INTO exists_aeropuerto;IF p_codigo_iata = '' THEN
        SELECT 'El campo codigo IATA no puede estar vacio' AS errmsg;ELSEIF
        exists_aeropuerto THEN
            UPDATE aeropuerto
            SET     codigo_iata = p_codigo_iata,
                  ciudad = p_ciudad,
                  pais = p_pais,

```

# Stored Procedures (SP)

```
        coordenada = p_coordenada

WHERE id_aeropuerto = p_id_aeropuerto;ELSEIF
p_codigo_iata = '' THEN
SELECT 'El campo codigo IATA no puede estar vacio';ELSE
INSERT INTO aeropuerto
(
        id_aeropuerto,
        codigo_iata,
        ciudad,
        pais,
        coordenada
)
VALUES
(
        p_id_aeropuerto,
        p_codigo_iata,
        p_ciudad,
        p_pais,
        p_coordenada
);END IF;END //
```

# Triggers

Los triggers que conforman la base de datos se crean para fines de auditoría o para rastrear el historial de cambios en la tabla. Estos se detallan a continuación:

- *'tr\_insert\_clientes'*: El objetivo de este trigger es registrar todas las inserciones realizadas en la tabla cliente.

El trigger se ejecuta después de cada inserción en la tabla cliente e inserta una nueva fila en la tabla *log\_accion\_registro* con la siguiente información: La fecha y hora de la inserción, la acción realizada (inserción), la tabla en la que se insertó la fila (cliente), la fila insertada (el valor de las columnas *id\_cliente*, nombre, apellido paterno, apellido materno, cedula, pasaporte, teléfono, correo) y el usuario que realizó la inserción.

```
CREATE TRIGGER tr_insert_clientes AFTER INSERT
ON cliente
FOR EACH ROW
    INSERT INTO log_accion_registro
        (fecha_transaccion,
        hora_transaccion,
        accion,
        tabla,
        registro,
        usuario)
VALUES (Curdate(),
        Current_time(),
        'insercion',
        'cliente',
        Concat('Nuevo registro insertado: ', new.id_cliente, new.nombre,
        new.apellido_paterno, new.apellido_materno, new.cedula,
        new.pasaporte,
        new.telefono, new.correo),
        CURRENT_USER());
```

- *'tr\_update\_clientes'*: El objetivo de este trigger es registrar todas las actualizaciones realizadas en la tabla cliente.

El trigger se ejecuta después de cada actualización en la tabla cliente e inserta una nueva fila en la tabla *log\_accion\_registro* con la siguiente información: La fecha y hora de la actualización, la acción realizada (actualización), la tabla en la que se actualizó la fila (cliente), la fila actualizada (el valor de las columnas *id\_cliente*, nombre, apellido

# Triggers

paterno, apellido materno, cedula, pasaporte, teléfono, correo) y el usuario que realizó la actualización.

```
CREATE TRIGGER tr_update_clientes AFTER UPDATE
ON cliente
FOR EACH ROW
    INSERT INTO log_accion_registro
        (fecha_transaccion,
         hora_transaccion,
         accion,
         tabla,
         registro,
         usuario)
VALUES (Curdate(),
       Current_time(),
       'actualizacion',
       'cliente',
       Concat('Nuevo registro actualizado: ', new.id_cliente,
             new.nombre,
             new.apellido_paterno, new.apellido_materno, new.cedula,
             new.pasaporte,
             new.telefono, new.correo),
       CURRENT_USER());
```

- *'tr\_insert\_empleados'*: El objetivo de este trigger es registrar todas las inserciones realizadas en la tabla empleado.

Cada vez que se inserte un nuevo registro en la tabla empleado, el trigger se ejecutará antes capturando la fecha, hora, acción realizada (inserción), el nombre de la tabla en la que se insertó la fila (empleado), el contenido del nuevo registro y el usuario que realiza la inserción. Luego, esa información se almacenará en la tabla *log\_accion\_registro* para mantener un registro de todas las inserciones realizadas en la tabla empleado.

```
CREATE TRIGGER tr_insert_empleados BEFORE INSERT
ON empleado
FOR EACH ROW
    INSERT INTO log_accion_registro
        (fecha_transaccion,
         hora_transaccion,
         accion,
         tabla,
         registro,
         usuario)
```

# Triggers

```
VALUES      (Curdate() ,
             Current_time() ,
             'insercion' ,
             'empleado' ,
             Concat('Nuevo registro insertado: ', new.id_empleado, new.nombre,
                    new.apellido_paterno, new.apellido_materno, new.cargo,
                    new.fecha_contratacion,
                    new.salario) ,
             CURRENT_USER());
```

- *'tr\_update\_empleados'*: El objetivo de este trigger es registrar todas las actualizaciones realizadas en la tabla empleado.

El trigger se ejecuta antes de cada actualización en la tabla empleado e inserta una nueva fila en la tabla *log\_accion\_registro* con la siguiente información: la fecha, hora, acción realizada (actualización), el nombre de la tabla en la que se actualizó la fila (empleado), el contenido del registro actualizado y el usuario que realiza la actualización.

```
CREATE TRIGGER tr_update_empleados BEFORE UPDATE
ON empleado
FOR EACH ROW
    INSERT INTO log_accion_registro
        (fecha_transaccion,
         hora_transaccion,
         accion,
         tabla,
         registro,
         usuario)
VALUES      (Curdate() ,
             Current_time() ,
             'actualizacion' ,
             'empleado' ,
             Concat('Nuevo registro actualizado: ', new.id_empleado,
                    new.nombre,
                    new.apellido_paterno, new.apellido_materno, new.cargo,
                    new.fecha_contratacion,
                    new.salario) ,
             CURRENT_USER());
```

- *'tr\_delete\_empleados'*: El objetivo de este trigger es registrar todas las eliminaciones realizadas en la tabla empleado.

El trigger se ejecuta antes de cada eliminación en la tabla empleado e inserta una nueva fila en la tabla *log\_accion\_registro* con la siguiente información: la fecha, hora,

# Triggers

acción realizada (eliminación), el nombre de la tabla en la que se eliminó la fila (empleado), el contenido del registro eliminado y el usuario que realiza la operación de eliminación.

```
CREATE TRIGGER tr_delete_empleados BEFORE DELETE
ON empleado
FOR EACH ROW
    INSERT INTO log_accion_registro
        (fecha_transaccion,
         hora_transaccion,
         accion,
         tabla,
         registro,
         usuario)
VALUES (Curdate(),
       Current_time(),
       'eliminacion',
       'empleado',
       Concat('Registro eliminado: ', old.id_empleado, old.nombre,
             old.apellido_paterno, old.apellido_materno, old.cargo,
             old.fecha_contratacion,
             old.salario),
       CURRENT_USER());
```

- *'tr\_delete\_promociones'*: El objetivo de este trigger es registrar todas las eliminaciones realizadas en la tabla promocion.

El trigger se ejecuta después de cada eliminación en la tabla promocion e inserta una nueva fila en la tabla log\_accion\_registro con la siguiente información: la fecha, hora, acción realizada (eliminación), el nombre de la tabla afectada (promocion), el contenido del registro eliminado y el usuario que realiza la operación de eliminación.

```
CREATE TRIGGER tr_delete_promociones AFTER DELETE
ON promocion
FOR EACH ROW
    INSERT INTO log_accion_registro
        (fecha_transaccion,
         hora_transaccion,
         accion,
         tabla,
         registro,
         usuario)
VALUES (Curdate(),
       Current_time(),
       'eliminacion',
       'promocion',
       Concat('Registro eliminado: ', old.id_promocion, old.nombre,
             old.apellido_paterno, old.apellido_materno, old.cargo,
             old.fecha_contratacion,
             old.salario),
       CURRENT_USER());
```



# Triggers

```
Concat('Registro eliminado: ', old.id_promocion,
old.id_aerolinea,
old.campania_promocion, old.descripcion_promocion, old.descuento,
old.fecha_inicio, old.fecha_fin),
CURRENT_USER());
```

- *'tr\_insert\_flota'*: El objetivo de este trigger es registrar todas las inserciones realizadas en la tabla flota.

Cada vez que se inserte un nuevo registro en la tabla flota, el trigger se ejecutará después capturando la fecha, hora, acción realizada (inserción), el nombre de la tabla en la que se insertó la fila (flota), el contenido del nuevo registro y el usuario que realiza la inserción. Luego, esa información se almacenará en la tabla *log\_accion\_registro* para mantener un registro de todas las inserciones realizadas en la tabla flota.

```
CREATE TRIGGER tr_insert_flota after INSERT
ON flota
FOR EACH ROW
    INSERT INTO log_accion_registro
        (fecha_transaccion,
        hora_transaccion,
        accion,
        tabla,
        registro,
        usuario)
    VALUES (Curdate(),
        Current_time(),
        'insercion',
        'flota',
        Concat('Nuevo registro insertado: ', new.id_aeronave,
        new.tipo_aeronave,
        new.cantidad_asiento),
        CURRENT_USER());
```

- *'tr\_update\_flota'*: El objetivo de este trigger es registrar todas las actualizaciones realizadas en la tabla flota.

El trigger se ejecuta después de cada actualización en la tabla flota e inserta una nueva fila en la tabla *log\_accion\_registro* con la siguiente información: la fecha, hora, acción realizada (actualización), el nombre de la tabla en la que se actualizó la fila (flota), el contenido del registro actualizado y el usuario que realiza la actualización.

# Triggers

```
CREATE TRIGGER tr_update_flota AFTER UPDATE
ON flota
FOR EACH ROW
    INSERT INTO log_accion_registro
        (fecha_transaccion,
         hora_transaccion,
         accion,
         tabla,
         registro,
         usuario)
VALUES (Curdate(),
       Current_time(),
       'actualizacion',
       'flota',
       Concat('Nuevo registro actualizado: ', new.id_aeronave,
             new.tipo_aeronave,
             new.cantidad_asiento),
       CURRENT_USER());
```

- *'tr\_update\_aeropuertos'*: El objetivo de este trigger es registrar todas las actualizaciones realizadas en la tabla aeropuerto.

El trigger se ejecuta después de cada actualización en la tabla aeropuerto e inserta una nueva fila en la tabla *log\_aeropuerto* con la siguiente información: el identificador del aeropuerto recién actualizado, la fecha y hora de la actualización, la acción realizada (actualización), el registro de la información del aeropuerto actualizado y el usuario que realiza la acción.

```
CREATE TRIGGER tr_update_aeropuertos AFTER UPDATE
ON aeropuerto
FOR EACH ROW
    INSERT INTO log_aeropuerto
        (id_aeropuerto,
         fecha_transaccion,
         hora_transaccion,
         accion,
         registro,
         usuario)
VALUES (new.id_aeropuerto,
       Curdate(),
       Current_time(),
       'actualizacion',
       Concat('Nuevo registro actualizado: ', new.id_aeropuerto,
             new.codigo_iata,
             new.ciudad, new.pais),
       CURRENT_USER());
```

# Triggers

- 'tr\_insert\_aeropuertos': El objetivo de este trigger es registrar todas las inserciones realizadas en la tabla aeropuerto.

El trigger se ejecuta después de cada inserción en la tabla aeropuerto e inserta una nueva fila en la tabla *log\_aeropuerto* con la siguiente información: el identificador del aeropuerto recién insertado, la fecha y hora de la actualización, la acción realizada (inserción), el registro de la información del aeropuerto insertado y el usuario que realiza la acción.

```
CREATE TRIGGER tr_insert_aeropuertos AFTER INSERT
ON aeropuerto
FOR EACH ROW
    INSERT INTO log_aeropuerto
        (id_aeropuerto,
         fecha_transaccion,
         hora_transaccion,
         accion,
         registro,
         usuario)
VALUES (new.id_aeropuerto,
       Curdate(),
       Current_time(),
       'insercion',
       Concat('Nuevo registro insertado: ', new.id_aeropuerto,
             new.codigo_iata,
             new.ciudad, new.pais),
       CURRENT_USER());
```

Además del administrador 'Root', se crean tres usuarios para hacer soporte de lectura, actualización, inserción y eliminación de la base de datos "Reserva\_Vuelo", los cuales se detallan a continuación:

- *'usuariolectura'*: Este usuario solo tendrá permiso de lectura.

En este caso, con el comando CREATE USER se crea un usuario nuevo llamado 'usuariolectura' con la contraseña usuariolectura. Después de ejecutar este comando, el usuario 'usuariolectura' será creado en el servidor de base de datos, pero aún no tendrá ningún privilegio.

Luego, con el comando GRANT se le otorga el privilegio SELECT (lectura) al usuario 'usuariolectura' en todas las tablas de la base de datos "Reserva\_Vuelo". Una vez que se ejecuta este comando, el usuario 'usuariolectura' tendrá permisos para realizar consultas de lectura en todas las tablas de la base de datos "Reserva\_Vuelo".

```
CREATE USER 'usuariolectura'@'localhost' IDENTIFIED BY 'usuariolectura';  
GRANT SELECT ON 'Reserva_Vuelo'.* TO 'usuariolectura'@'localhost';  
SHOW GRANTS FOR 'usuariolectura'@'localhost';
```

- *'usuariomodifica'*: Este usuario solo tendrá permiso de lectura, inserción y actualización de los datos.

En este caso, con el comando CREATE USER se crea un usuario nuevo llamado 'usuariomodifica' con la contraseña usuariomodifica. Después de ejecutar este comando, el usuario 'usuariomodifica' será creado en el servidor de base de datos, pero aún no tendrá ningún privilegio.

Luego, con el comando GRANT se le otorgan los privilegios SELECT (lectura), INSERT (inserción) y UPDATE (actualización) al usuario 'usuariomodifica' en todas las tablas de la base de datos "Reserva\_Vuelo". Una vez que se ejecuta este comando, el usuario 'usuariomodifica' tendrá permisos para realizar consultas de lectura, inserción y actualización en todas las tablas de la base de datos "Reserva\_Vuelo".

Sin embargo, con respecto a las tablas log sólo tendrá permisos de lectura por medidas de seguridad en la auditoría. Por lo tanto, se usa la sentencia REVOKE para quitarle los permisos de inserción y actualización.

```
CREATE USER 'usuariomodifica'@'localhost' IDENTIFIED BY 'usuariomodifica';  
GRANT SELECT, UPDATE, INSERT ON Reserva_Vuelo.*  
TO 'usuariomodifica'@'localhost';  
GRANT SELECT, UPDATE, INSERT ON Reserva_Vuelo.log_accion_registro  
TO 'usuariomodifica'@'localhost';  
REVOKE UPDATE, INSERT ON Reserva_Vuelo.log_accion_registro  
FROM 'usuariomodifica'@'localhost';  
GRANT SELECT, UPDATE, INSERT ON Reserva_Vuelo.log_aeropuerto  
TO 'usuariomodifica'@'localhost';  
REVOKE UPDATE, INSERT ON Reserva_Vuelo.log_aeropuerto  
FROM 'usuariomodifica'@'localhost';  
SHOW GRANTS FOR 'usuariomodifica'@'localhost';
```

- *'usuarioelimina'*: Este usuario solo tendrá permiso de lectura y eliminación de la tabla empleado.

En este caso, con el comando CREATE USER se crea un usuario nuevo llamado 'usuarioelimina' con la contraseña usuarioelimina. Después de ejecutar este comando, el usuario 'usuarioelimina' será creado en el servidor de base de datos, pero aún no tendrá ningún privilegio.

Luego, con el comando GRANT se le otorga los privilegios SELECT (lectura) y DELETE (eliminación) al usuario 'usuarioelimina' en la tabla 'empleado' de la base de datos 'Reserva\_Vuelo'. Una vez que se ejecuta este comando, el usuario 'usuarioelimina' tendrá permisos para realizar consultas de lectura y eliminación en la tabla 'empleado' de la base de datos 'Reserva\_Vuelo'.

```
CREATE USER 'usuarioelimina'@'localhost' IDENTIFIED BY 'usuarioelimina';  
GRANT SELECT, DELETE ON 'Reserva_Vuelo'.empleado TO 'usuarioelimina'@'localhost';  
;  
SHOW GRANTS FOR 'usuarioelimina'@'localhost';
```

# Transacciones

Se ejecutan las siguientes transacciones (TCL):

- En primer lugar, se deshabilita la función autocommit. Para ello, se asigna a la variable un cero (0).

```
SELECT @@autocommit;
SET autocommit = 0;
```

- Luego, se inicia una transacción y se selecciona la primera fila de la tabla 'empleado'. Se elimina la fila con 'ID\_empleado' igual a 1 y finalmente se selecciona nuevamente la primera fila de la tabla actualizada. Cabe destacar que, la transacción permite que todos estos comandos se ejecuten como una unidad indivisible. Si ocurriera algún error antes de confirmar la transacción, se deshazarían todos los cambios realizados hasta ese punto (en este caso, se deshacería la eliminación de la fila con 'ID\_empleado' igual a 1).

Para confirmar y guardar definitivamente los cambios realizados dentro de la transacción, se debe ejecutar el comando COMMIT. Sin embargo, como no se quiere eliminar el registro, se van a deshacer los cambios (sin guardar) utilizando el comando ROLLBACK.

```
START TRANSACTION;

SELECT *
FROM empleado
LIMIT 1;DELETE
FROM empleado
WHERE id_empleado = 1;SELECT *
FROM empleado
LIMIT 1;ROLLBACK;
/*COMMIT;*/
```

- Seguidamente, se inicia otra transacción y se inserta un nuevo registro en la tabla 'empleado'. Se realiza una consulta para verificar que la inserción se haya realizado correctamente. Como se mencionó anteriormente, la transacción permite que todos estos comandos se ejecuten como una unidad indivisible, por lo que, si ocurriera algún error antes de confirmar la transacción, se deshazarían todos los cambios

# Transacciones

realizados hasta ese punto (en este caso, se desharía la inserción del nuevo empleado).

Para confirmar y guardar definitivamente los cambios realizados dentro de la transacción, se ejecuta el comando COMMIT. Si se quisiera deshacer los cambios sin guardar, se puede utilizar el comando ROLLBACK.

```
START TRANSACTION;
SELECT *
FROM empleado;
CALL update_add_empleado (6, 'Kary', 'Francia', 'Vasquez',
'analista de negocios', '2023-02-19', 2500);
/* ROLLBACK; */
COMMIT;
```

- Por último, al igual que en las explicaciones anteriores, se inicia una nueva secuencia de comandos que inicia una transacción. Se agregan nuevos registros a la tabla 'cliente' y se crean dos savepoints (puntos de guardado lote\_1 y lote\_2) para marcar diferentes lotes de inserciones. Los savepoints se utilizan para marcar posiciones específicas en la transacción, lo que permite deshacer cambios hasta ese punto en particular.

Se realiza una consulta SELECT en la tabla 'cliente' y se ordena los resultados en orden descendente según el campo 'ID\_cliente'; mostrando todos los registros de la tabla después de las inserciones realizadas hasta ese punto. Luego, se realiza un ROLLBACK en la transacción hasta el savepoint "lote\_1". Es decir, todos los cambios realizados después de ese punto serán deshechos, incluidas las inserciones realizadas en el "lote\_2". Después del ROLLBACK, se realiza nuevamente una consulta SELECT en la tabla 'clientes' y se ordena los resultados en orden descendente según el campo 'ID\_cliente'; mostrando los registros de la tabla después del "rollback" para que se visualice que las inserciones del "lote\_1" fueron deshechas. Finalmente, para confirmar y guardar definitivamente los cambios realizados dentro de la transacción, se ejecuta el comando COMMIT.

```
START TRANSACTION;

SELECT * FROM cliente;

INSERT INTO cliente (ID_cliente, nombre, apellido_paterno, apellido_materno,
cedula, pasaporte, telefono, correo) VALUES
```

```
(NULL, 'Pedro', 'Mora', 'Aguilar', '45517518', 'A1542D', '+59194425534',  
'p.mora@hotmail.com'),  
  
(NULL, 'Martina', 'Guichon', 'Quintero', '75517519', 'L1672R', '+59199825534',  
'marti_15@gmail.com'),  
  
(NULL, 'Andres', 'Rivera', 'Gutierrez', '60551851', 'K8542D', '+59894425524',  
'andy@hotmail.com'),  
  
(NULL, 'Julieta', 'Cepeda', 'Brito', '43228516', 'K9542D', '+59899628514',  
'cjuli@gmail.com');  
  
SAVEPOINT lote_1;  
  
INSERT INTO cliente (ID_cliente, nombre, apellido_paterno, apellido_materno,  
cedula, pasaporte, telefono, correo) VALUES  
  
(NULL, 'Gianella', 'Neyra', 'Rivero', '35517523', 'L1542R', '+59199995534',  
'giani.neyra@hotmail.com'),  
  
(NULL, 'Laura', 'Spoya', 'Rullian', '71139548', 'K3572R', '+59199724434',  
'spoya@gmail.com'),  
  
(NULL, 'Paolo', 'Guerrero', 'Farfan', '50558851', 'K8542D', '+59199342528',  
'p.guerreo@hotmail.com'),  
  
(NULL, 'Tommy', 'Portocarrero', 'Mosca', '43255588', 'L9542B', '+59899628521',  
'tommy.porto@gmail.com');  
  
SAVEPOINT lote_2;SELECT *  
FROM cliente  
ORDER BY id_cliente DESC;ROLLBACK TO lote_1;SELECT *  
FROM cliente  
ORDER BY id_cliente DESC;COMMIT;
```

## Restauración:

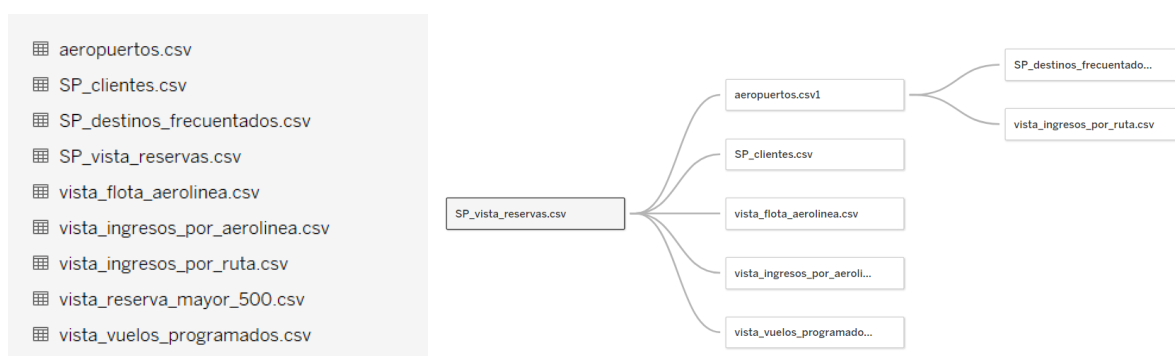
Se realiza un respaldo para almacenar los datos y la estructura de la base de datos 'Reserva\_Vuelo' (tablas y vistas).





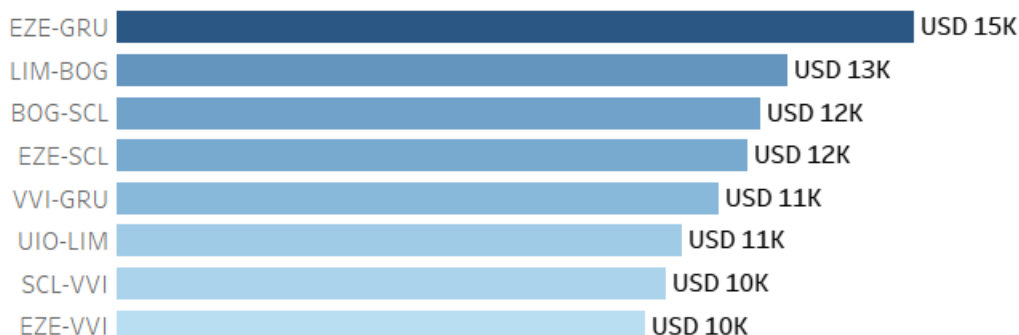
A continuación, se presenta una serie de gráficos, generados mediante el procesamiento y análisis de la información contenida, en la base de datos de 'Travel Agency', durante los meses de enero, febrero y marzo del 2023. Para ello, se utilizan todas las vistas elaboradas previamente, el Stored Procedure (SP) de destinos frecuentados, y una serie de consultas a la tabla clientes y aeropuerto. El software que se emplea como herramienta de visualización es Tableau.

- Primero se realizó la importación de los datos, provenientes de MySQL Workbench, a Tableau.



- Utilizando la vista ingresos por ruta, el SP destinos frecuentados y la tabla de aeropuertos, se observa que:
- La ruta que generó mas ingresos fue desde Buenos Aires (Argentina) hasta San Pablo (Brasil), recaudando aproximadamente USD 15K.

## Top 8 - rutas con mayores ingresos

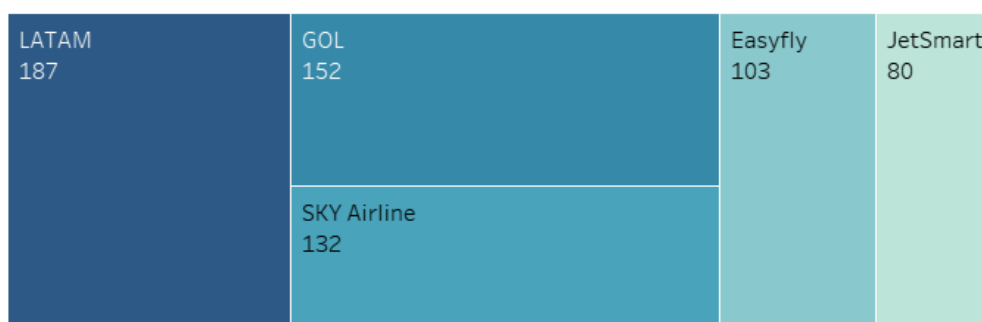


- El destino más popular escogido por los clientes durante el primer trimestre fue Santiago de Chile con un 17%.

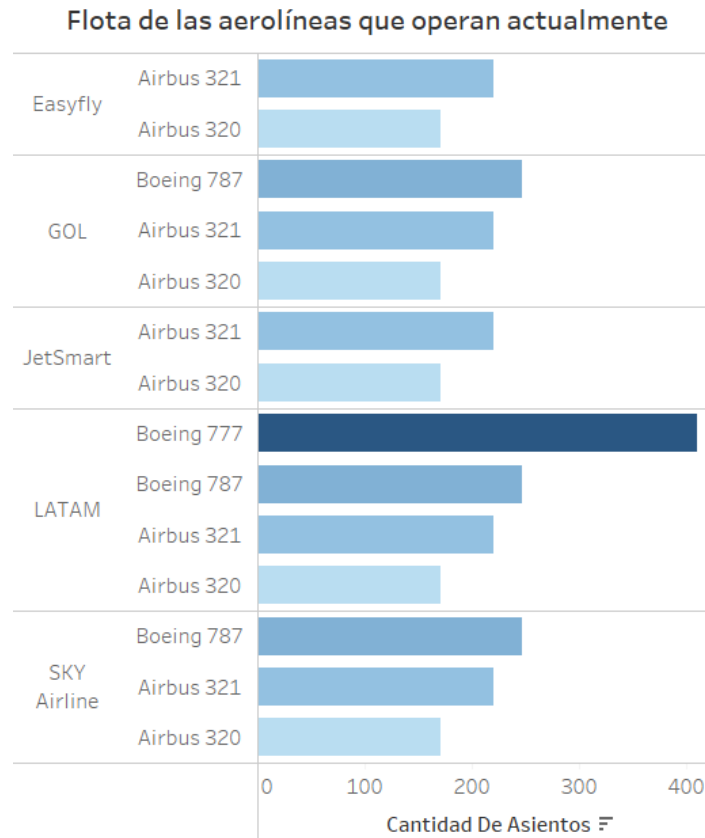


- De los gráficos observados anteriormente, la empresa podría inferir que, para una toma de decisión empresarial, las tres rutas capaces de generar más ingresos con una mayor promoción serían: Lima-Bogotá, Bogotá-Santiago de Chile, Argentina-Santiago de Chile. Sobre todo, siendo que el destino más frecuentado por los clientes es Santiago de Chile.
- Utilizando la vista\_reservas, se identifica que, en función de la cantidad de reservas realizadas en el primer trimestre, LATAM y GOL son las aerolíneas con mayor *marketshare*.

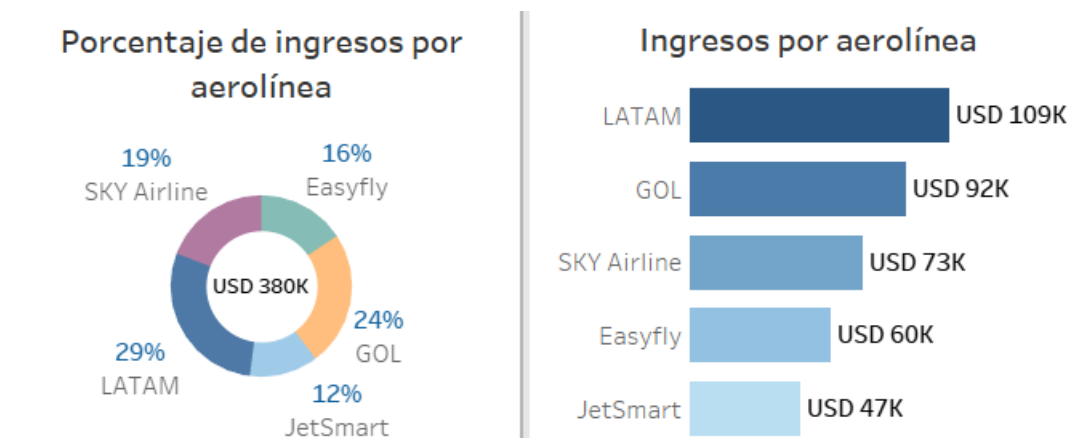
Marketshare



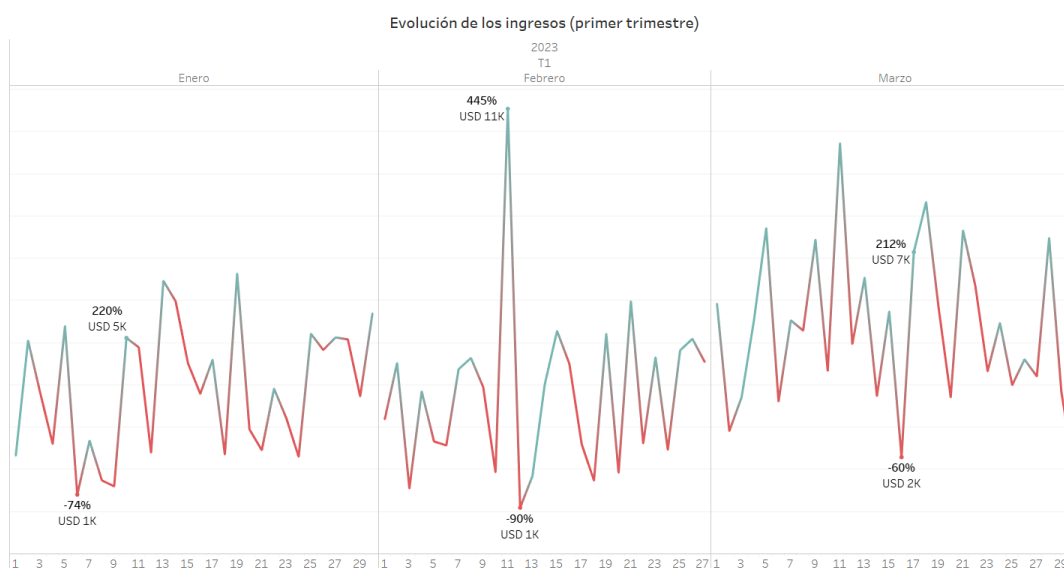
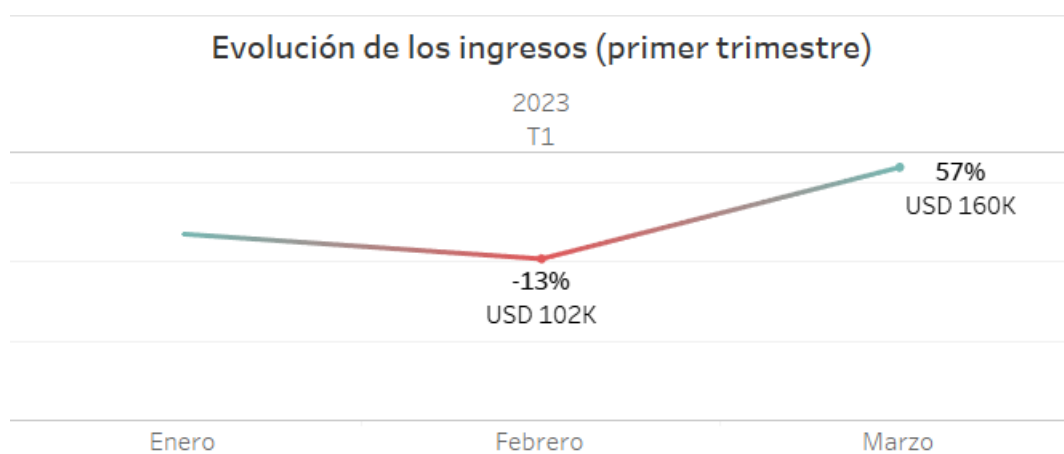
- Utilizando la vista\_flota\_aerolinea, se distingue que la aerolínea que posee la flota más grande es LATAM.



- Utilizando la vista\_ingresos\_por\_aerolinea, se observa que las dos aerolíneas que generaron mayores ingresos fueron LATAM con un 29% (USD 109K) y GOL con 24% (USD 92K).



- Utilizando la vista reserva, se puede observar una evolución de los ingresos, mensualmente y más al detalle diariamente. El mes de marzo, fue el mes que se obtuvo mayores ingresos, alcanzando un porcentaje de diferencia con respecto al mes anterior de 56.6%.



- Finalmente, con las vistas antes mencionadas, se obtuvo los siguientes indicadores para el primer trimestre:

<b>Total ingresos</b> USD 380K	<b>Total reservas</b> 654	<b>Total clientes</b> 91
-----------------------------------	------------------------------	-----------------------------

## “Posteriores rumbos”

Este proyecto tiene como objetivo abordar una problemática real en la industria de agencias de viajes en línea con una plataforma completa y eficiente, capaz de ofrecer no solo pasajes aéreos a bajo costo, sino también una experiencia mejorada para los clientes (buscando satisfacer todas las necesidades de los mismos) y una gestión más efectiva para la empresa en su conjunto.



Como ‘posteriores rumbos’ que debería tomar ‘Travel Agency’ en el futuro, se prevén diversas mejoras y expansiones en su modelo de negocio, tales como:

- La implementación de nuevas funcionalidades centradas en la gestión financiera de la agencia, incluyendo módulos para los impuestos, créditos y otros aspectos contables.
- Desarrollar aún más la plataforma para incluir información sobre:
  - Futuras ubicaciones de la empresa.
  - Depósitos donde se almacenan recursos esenciales.
  - Proveedores de otros servicios u otras empresas y organizaciones comprometidas con la sostenibilidad, como aerolíneas con iniciativas verdes, hoteles eco-amigables y proveedores de servicios turísticos sostenibles.
  - Módulo de campañas que incluyan la colaboración con destinos turísticos que se adhieran a prácticas sostenibles y de bajo impacto ambiental, como los paquetes de viaje a lugares que promuevan la conservación del medio ambiente, el respeto a la cultura local y el desarrollo comunitario. Educando a los clientes sobre cómo minimizar el impacto ambiental durante los viajes, el respeto por la cultura local y su contribución positiva a las comunidades visitadas.
- Diseñar nuevas interfaces que presenten de manera clara y concisa la información relevante para los usuarios.
- Fortalecer el módulo de gestión de usuarios, permitiendo un control más completo y personalizado de las cuentas de los clientes.
- Buscar certificaciones reconocidas internacionalmente que validen las prácticas sostenibles de la agencia, lo que puede aumentar la confianza de los clientes conscientes de la sostenibilidad.

- Adobe Acrobat: para visualizar el informe y la presentación resumida en formato pdf.
- Erdplus.com: para crear el diagrama de entidad relación académicamente.
- Excel: para la normalización de los datos y para la descarga de los reportes exportados de MySQL en formatos CSV.
- ChatGPT: para documentar el código de vistas, funciones, stored procedures, y triggers.
- Instant SQL Formatter: para formatear y reorganizar el código SQL de manera automática y coherente, mejorando la legibilidad y la comprensión del código. (<https://www.dpriver.com/pp/sqlformat.htm>)
- Mockaroo.com: para generar datos ficticios y luego poder insertarlos en la base de datos.
- MySql Workbench: para construir la base de datos y el diagrama EER.
- PowerPoint: Para crear una presentación resumida del proyecto.
- Tableau: para generar las visualizaciones de los reportes generados con MySQL workbench.
- Word: Para redactar el presente informe.
- Looka: para crear el logo de la empresa ficticia.



Dentro del siguiente link:

[https://github.com/KaryFrancia/MySQL\\_Travel\\_Agency](https://github.com/KaryFrancia/MySQL_Travel_Agency)

Se encuentran los scripts del presente proyecto. Los cuales deben ser ejecutados en el siguiente orden:

- 1\_Estructura\_BD\_Reserva\_Vuelo.sql
- 2\_Insercion\_datos.sql
- 3\_Vistas.sql
- 4\_Funciones.sql
- 5\_Stored\_Procedures.sql
- 6\_Triggers.sql
- 7\_Usuarios.sql
- 8\_ConsultasSQL.sql
- 9\_TCL.sql
- 10\_Backup.sql

Además, se incluyen los reportes en formato .csv exportados de MySQL workbench y los archivos del diagrama de entidad relación:

- EER\_Diagram\_Reserva\_Vuelo.mwb

Finalmente, a través del siguiente link, se puede acceder a los Dashboard interactivos realizados en Tableau:

[Análisis datos SQL Reserva Vuelo Francia | Tableau Public](#)

# PROYECTO FINAL SQL

“Agencia de viajes –  
vuelos a bajo costo”



**Travel agency**

LOW COST FLIGHTS



Pertenece a: Kary Yessenia Francia Vásquez

Comisión: 43420

Profesor: Gabriel Almiñana

Tutor: José Ignacio López Saez

**CODERHOUSE**





# CONTENIDO

**Introducción.**

**Objetivos.**

**Modelo de negocios.**

**Base de datos.**

**Vistas.**

**Funciones.**

**Stored Procedures.**

**Triggers.**

**Roadmap de la solución.**

**Informes detallados.**



**“Posteriores rumbos”.**



# INTRODUCCIÓN

Se presenta el caso de una agencia de viajes online “ficticia” (‘Travel Agency’ S.R.L) que ofrece a sus clientes la opción de hacer reservas de vuelos a bajo costo con un nivel mínimo de servicio y confort. Además de un servicio básico, ofrecerá una variedad de servicios adicionales que los clientes pueden adaptar, según sus preferencias individuales, mejorando su experiencia de viaje.

La base de datos relacional, almacena la información necesaria para que los clientes puedan hacer las reservas de vuelos, tales como origen, destino, aerolínea, tipo de asiento, equipaje, etc. También almacena la información de los clientes, empleados, campañas de promociones de la agencia, comentarios que hacen los clientes sobre su experiencia, promociones, avisos para los clientes sobre la información de su vuelo, etc.



# OBJETIVOS



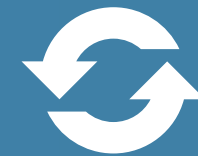
## • CREAR

Crear una base de datos, en MySQL Workbench, para respaldar las operaciones de 'Travel Agency'



## • ANALIZAR

Analizar la información de la base de datos y presentar un informe con métricas clave para el negocio.



## • DISEÑAR

Proponer diseños de mejora en la estructura de negocios de 'Travel Agency', luego de cumplir con los primeros objetivos.

# MODELO DE NEGOCIOS



“‘Travel Agency’ logra diferenciarse de sus competidores ofreciendo a los clientes una combinación de precios bajos, una variedad de servicios y una experiencia de usuario sencilla.”

La plataforma de ‘Travel Agency’ funciona como un agente de viajes online que colabora con cinco aerolíneas para ofrecer tarifas competitivas, ofreciendo como destino ocho países.



JetSMART

GOL  
Linhas aéreas inteligentes

EASYFLY

SKY  
Airline

LATAM  
AIRLINES

# BASE DE DATOS



La base de datos se compone de 17 tablas en total, 6 Vistas, 3 Funciones, 5 Stored Procedures y 10 Triggers.



Los datos almacenados en la base de datos se cargaron de manera manual.



Cuenta con 3 Usuario, además del administrador (cada uno con distintos privilegios a la hora de interactuar con la base de datos).



# VISTAS



## **vista\_reserva**

Proporciona una descripción general de las reservas realizadas por los clientes.



## **vista\_vuelos\_programados**

Proporciona una manera conveniente de acceder a la información de los vuelos programados.



## **vista\_reserva\_mayor\_500dolares**

Proporciona una manera conveniente de acceder a información detallada sobre las reservas que tienen un precio total mayor a USD 500.



## **vista\_flota\_aerolinea**

Proporciona una manera conveniente de acceder a información sobre la flota de aeronaves de diferentes aerolíneas.



## **vista\_ingresos\_por\_aerolinea**

proporciona una manera conveniente de acceder a los ingresos totales generados por aerolínea en función de los precios de los asientos, equipajes y vuelos en las reservas realizadas.



## **vista\_ingresos\_por\_ruta**

Proporciona una manera conveniente de acceder a los ingresos generados por ruta en función de los precios de los asientos, equipajes y vuelos en las reservas realizadas.

# FUNCIONES



## Cantidad\_destino

calcula la cantidad de vuelos con destino a una ciudad específica y devuelve este valor como resultado. Para llamar a esta función se pasa una ciudad como parámetro para obtener la cantidad de vuelos con destino a esa ciudad.



## Total\_ingreso\_cliente

calcula el ingreso total generado por un cliente específico a partir de servicios (asientos y equipaje) y vuelos. Para llamar a esta función se pasa el ID de un cliente como parámetro para obtener el total de ingresos correspondiente.



## Total\_reserva\_cliente

calcula el total de reservas realizadas por un cliente específico. Para llamar a esta función se pasa el ID de un cliente como parámetro para obtener la cantidad total de reservas correspondiente.

# STORED PROCEDURES



## Destinos\_frecuentados

Ejecuta una consulta para determinar los destinos más frecuentados por los clientes en función de la cantidad de veces que cada ciudad de aeropuerto se ha utilizado como destino final en los vuelos y reservas.



## Update\_add\_flota

Realiza operaciones para actualizar la cantidad de asientos de una aeronave existente en la tabla flota, o para agregar una nueva aeronave a la tabla si no existe.



## Update\_add\_cliente

Realiza operaciones para actualizar la información de un cliente existente en la tabla cliente, o para agregar un nuevo cliente a la tabla si no existe.



## Update\_add\_empleado

Realiza operaciones para actualizar la información de un empleado existente en la tabla empleado, o para agregar un nuevo empleado a la tabla si no existe.



## Update\_add\_aeropuerto

Realiza operaciones para actualizar la información de un aeropuerto existente en la tabla aeropuerto, o para agregar un nuevo aeropuerto a la tabla si no existe.



# TRIGGERS



## Tr\_insert\_clientes

Se dispara después de insertar un nuevo registro en la tabla cliente. Registra la acción de inserción, junto con los detalles relevantes del registro insertado en un log de auditoría log\_accion\_registro.



## Tr\_update\_clientes

Se dispara después de actualizar un registro en la tabla cliente. Registra la acción de actualización, junto con los detalles relevantes del registro actualizado en un log de auditoría log\_accion\_registro.



## Tr\_insert\_empleados

Se dispara antes de insertar un nuevo registro en la tabla empleado. Registra la acción de inserción junto con los detalles relevantes del registro en un log de auditoría log\_accion\_registro.



## Tr\_update\_empleados

Se dispara antes de actualizar un registro en la tabla empleado. Registra la acción de actualización junto con los detalles relevantes del registro en un log de auditoría log\_accion\_registro.



## Tr\_delete\_empleados

Se dispara antes de eliminar un registro de la tabla empleado. Registra la acción de eliminación junto con los detalles relevantes del registro en un log de auditoría log\_accion\_registro.

# TRIGGERS



## Tr\_delete\_promociones

Se dispara después de eliminar un registro de la tabla promocion. Registra la acción de eliminación junto con los detalles relevantes del registro en un log de auditoría log\_accion\_registro.



## Tr\_insert\_flota

Se dispara después de insertar un nuevo registro en la tabla flota. Registra la acción de inserción junto con los detalles relevantes del registro en un log de auditoría log\_accion\_registro.



## Tr\_update\_flota

Se dispara después de actualizar un registro en la tabla flota. Registra la acción de actualización junto con los detalles relevantes del registro en un log de auditoría log\_accion\_registro.



## Tr\_update\_aeropuertos

Se dispara después de actualizar un registro en la tabla aeropuerto. Registra la acción de actualización junto con los detalles relevantes del registro en un log específico log\_aeropuerto.



## Tr\_insert\_aeropuertos

Se dispara después de insertar un registro en la tabla aeropuerto. Registra la acción de inserción junto con los detalles relevantes del registro en un log específico log\_aeropuerto.

# ROADMAP DE LA SOLUCIÓN

Creación  
del  
diagrama  
E-R

Creación del  
esquema de la  
base de datos  
(tablas)

Inserción  
manual de los  
datos a la base  
de datos

Creación de  
vistas,  
funciones, SP,  
Triggers

Se realiza el  
backup de las  
vistas y tablas

Generación  
de reportes  
en MySQL

Análisis de la  
información  
en tableau



# INFORMES DETALLADOS

Durante el primer trimestre del año 2023, se obtuvieron los siguientes resultados mediante la generación de gráficos estadísticos y diversos reportes.

Total ingresos

USD 380K



Total reservas

654



Total clientes

91





# INFORMES DETALLADOS

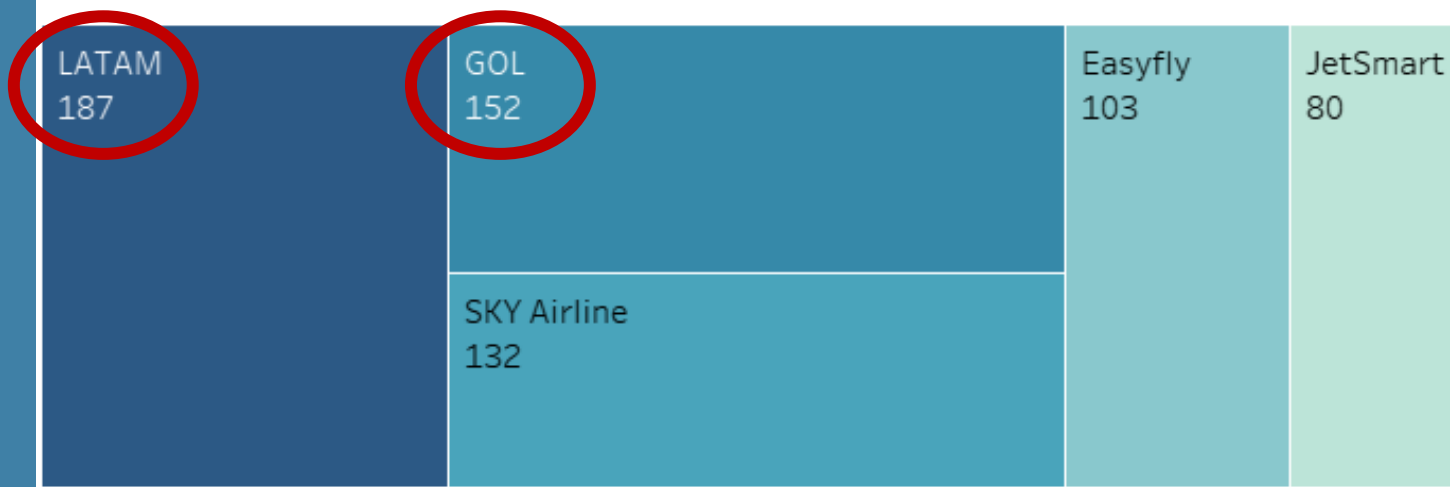
Durante el primer trimestre del año 2023, se obtuvieron los siguientes resultados mediante la generación de gráficos estadísticos y diversos reportes.



# INFORMES DETALLADOS

Durante el primer trimestre del año 2023, se obtuvieron los siguientes resultados mediante la generación de gráficos estadísticos y diversos reportes.

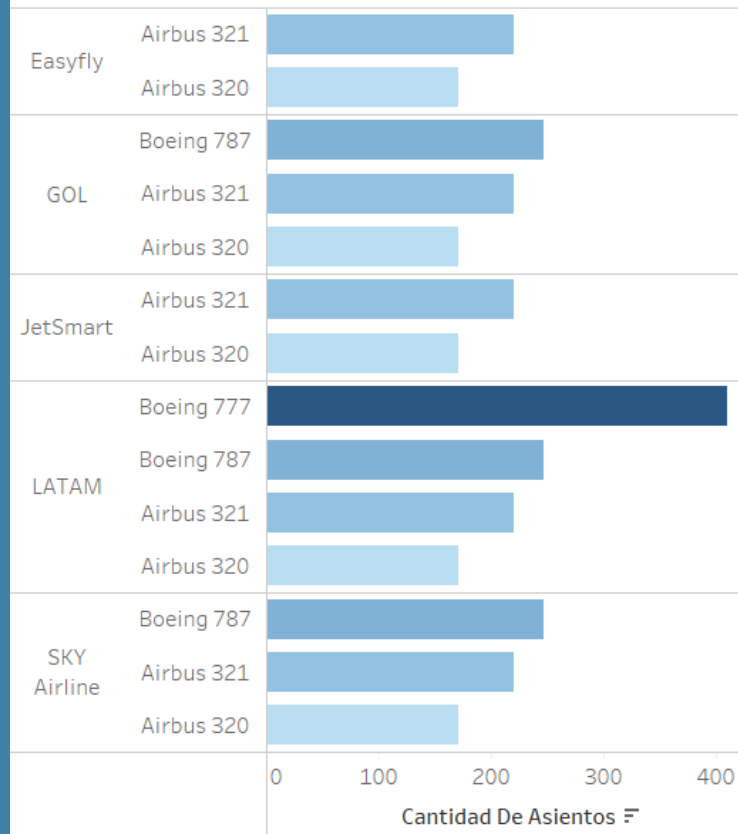
Marketshare



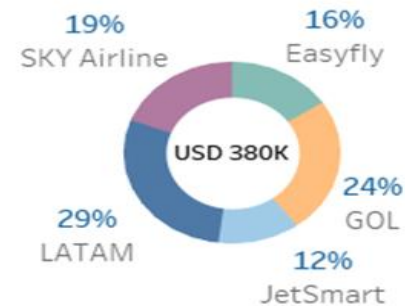
# INFORMES DETALLADOS

Durante el primer trimestre del año 2023, se obtuvieron los siguientes resultados mediante la generación de gráficos estadísticos y diversos reportes.

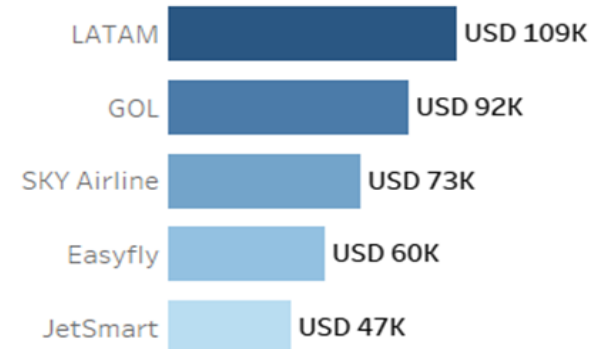
Flota de las aerolíneas que operan actualmente



Porcentaje de ingresos por aerolínea



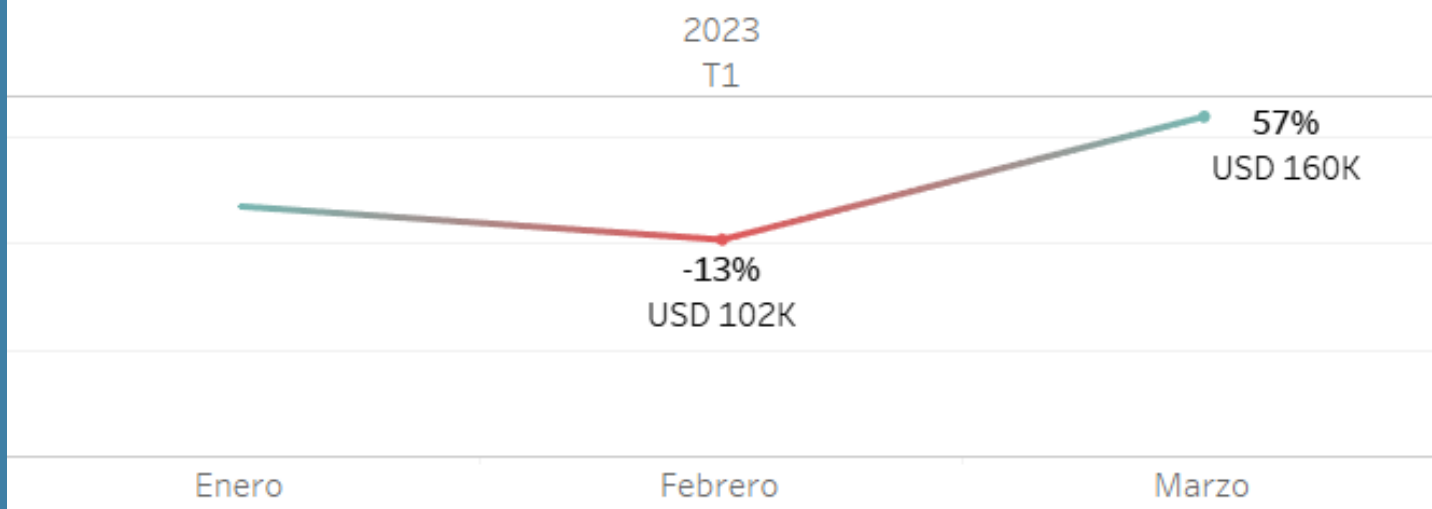
Ingresos por aerolínea



# INFORMES DETALLADOS

Durante el primer trimestre del año 2023, se obtuvieron los siguientes resultados mediante la generación de gráficos estadísticos y diversos reportes.

Evolución de los ingresos (primer trimestre)





# POSTERIORES RUMBOS

Implementar nuevas funcionalidades centradas en la gestión financiera de la agencia, incluyendo módulos para los impuestos, créditos y otros aspectos contables.

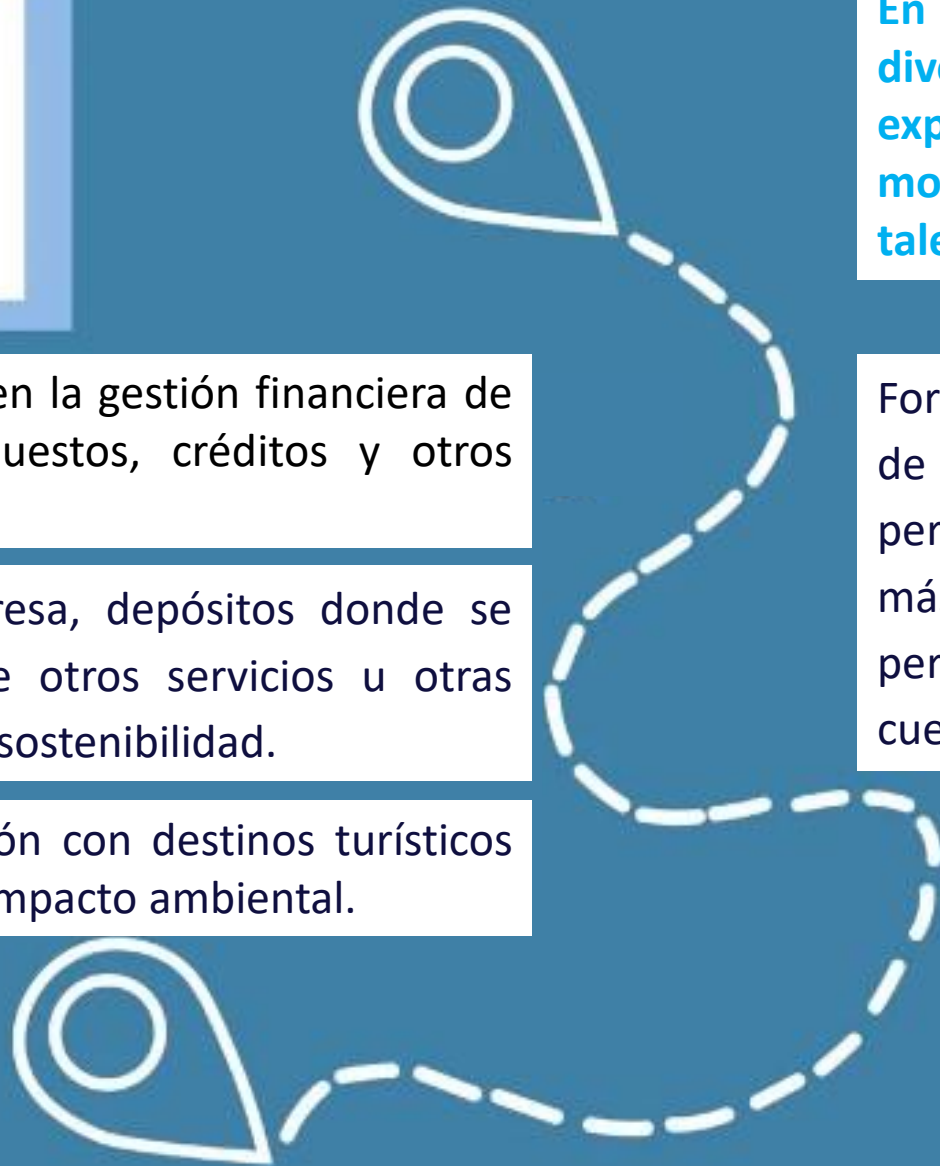
Nuevas tablas: Futuras ubicaciones de la empresa, depósitos donde se almacenan recursos esenciales, proveedores de otros servicios u otras empresas y organizaciones comprometidas con la sostenibilidad.

Incluir en el módulo de campañas la colaboración con destinos turísticos que se adhieran a prácticas sostenibles y de bajo impacto ambiental.

Diseñar nuevas interfaces que presenten de manera clara y concisa la información relevante para los usuarios.

En el futuro, se prevén diversas mejoras y expansiones en su modelo de negocio, tales como:

Fortalecer el módulo de gestión de usuarios, permitiendo un control más completo y personalizado de las cuentas de los clientes.



Para obtener una información más detallada, escanea el código QR y obtén el documento con la descripción funcional técnica completa del proyecto.

# MUCHAS GRACIAS

