

Sistema de Aerolíneas



Autor: Ditz Yanina Alejandra

Comisión: 50050

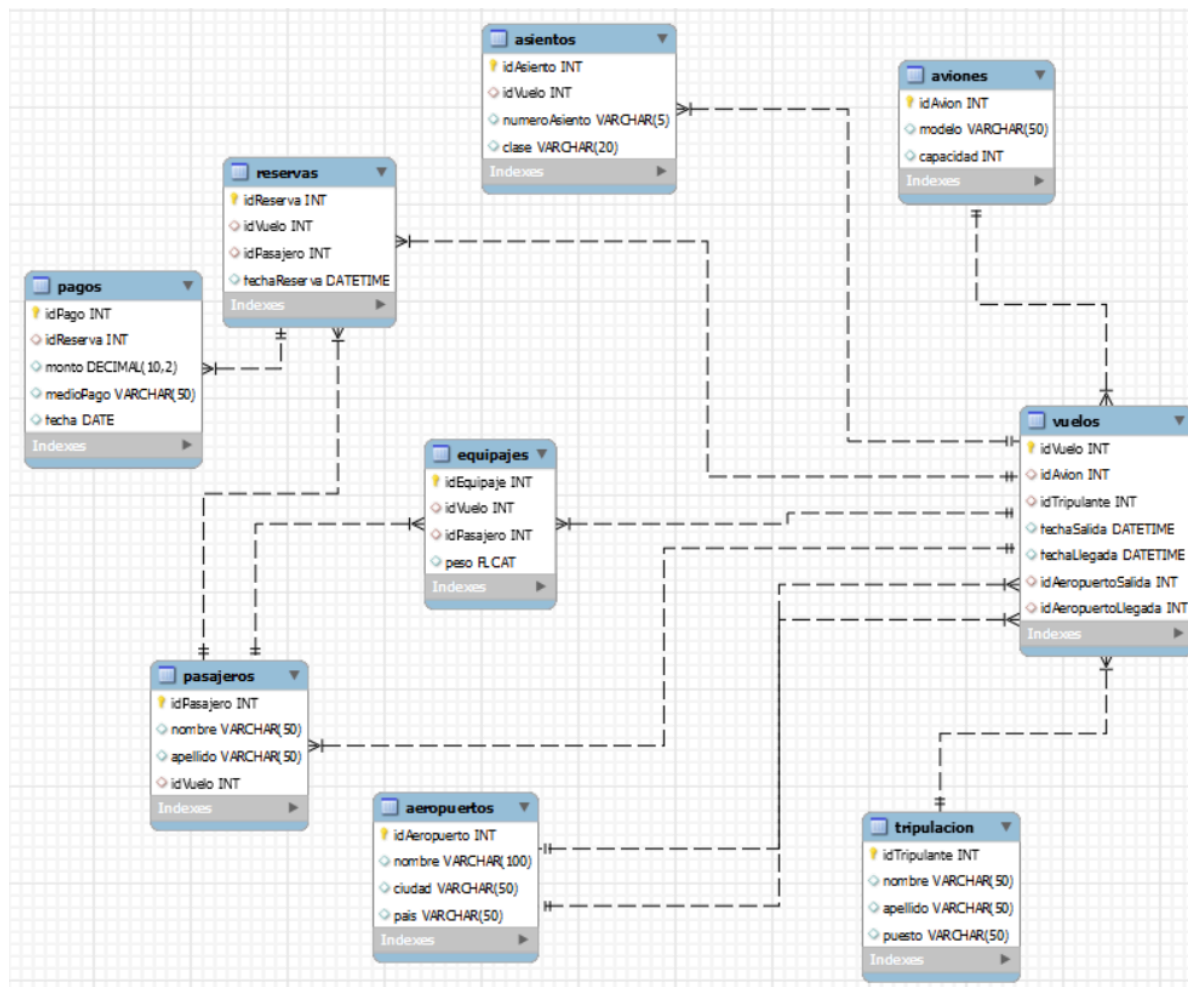
Objetivo: Base de datos para gestionar el servicio de una aerolínea, con el fin de modelar los procesos principales de una aerolínea como administrar vuelos, pasajeros, tripulaciones, aviones, aeropuertos.

Situación problemática: La falta de una base de datos estructurada puede llevar a errores en la administración de vuelos, como sobrecupos de pasajes en vuelos debido a sobreventa. Esto causaría que pasajeros con pasajes confirmados no puedan abordar. Cancelaciones de vuelos por problemas climáticos u operativos. Esto genera inconvenientes para reubicar pasajeros y tripulaciones. Retrasos en las salidas o llegadas de vuelos, ocasionando pérdida de conexiones para pasajeros con viajes multidestino. Pérdida o demora del equipaje de pasajeros durante el transporte a destino. Falta de registro e información de horarios y disponibilidad de tripulaciones. Esto dificulta la asignación eficiente de recursos humanos. Escasez o sobreasignación de aviones en rutas muy demandadas, por falta de optimización de flota. Errores o fraudes durante el check-in y embarque de pasajeros en aeropuertos. Por estas situaciones problemáticas, entre otras, es necesario e indispensable contar con un modelo de entidad relación como paso inicial para desarrollar bases de datos aeronáuticas completas y eficientes.

En resumen, la base de datos que surge del modelo sirve como columna vertebral para todo sistema de información de pasajeros, equipaje, estadísticas de vuelo. Además optimiza el rendimiento de consultas y análisis de información relevante para la toma de decisiones gerenciales.






Modelo de negocio: Nuestra empresa ***FastFlight*** cuenta con un modelo híbrido que tiene en cuenta combinar clases con algunos servicios incluidos y otras tarifas básicas, establecer alianzas estratégicas con otros transportadores, y ofrecer programa de viajero frecuente atractivo.



Diagrama Entidad - Relación de MySQL Workbench




Tablas y atributos:

Tablas	Atributos
<ul style="list-style-type: none"> Aviones: contiene información sobre cualquier avión que es utilizado para algún vuelo 	<ul style="list-style-type: none"> idAvion 🔑: integer (entero) modelo: varchar(50) capacidad: integer (entero)
<ul style="list-style-type: none"> Tripulación: contiene información de la tripulación de la tripulación 	<ul style="list-style-type: none"> idTripulante 🔑: integer (entero) nombre: varchar(50) apellido: varchar(50) puesto: varchar(50)

<ul style="list-style-type: none"> • Aeropuertos: contiene información sobre los aeropuertos 	<ul style="list-style-type: none"> • idAeropuerto :integer (entero) • nombre: varchar(100) • ciudad INDEX: varchar(50) • país INDEX: varchar(50)
<ul style="list-style-type: none"> • Vuelos: contiene información sobre los vuelos 	<ul style="list-style-type: none"> • idVuelo : integer (entero) • idAvion FK : integer (entero) • idTripulante FK: integer (entero) • idEquipaje FK: integer (entero) • idAsiento FK: integer (entero) • idReserva FK: integer (entero) • fechaSalida INDEX: datetime (fecha y hora) • fechaLlegada INDEX: datetime (fecha y hora) • idAeropuertoSalida FK:integer (entero) • idAeropuertoLlegada FK: integer (entero)
<ul style="list-style-type: none"> • Pasajeros: contiene información sobre los pasajeros 	<ul style="list-style-type: none"> • idPasajero :integer (entero) • nombre: varchar(50) • apellido: varchar(50) • idVuelo FK: integer (entero)
<ul style="list-style-type: none"> • Equipajes: contiene información del equipaje 	<ul style="list-style-type: none"> • idEquipaje : integer (entero) • idVuelo FK: integer (entero) • idPasajero FK:integer (entero) • peso: float
<ul style="list-style-type: none"> • Asientos: contiene información de los asientos 	<ul style="list-style-type: none"> • idAsiento :integer (entero) • idVuelo FK: integer (entero) • númeroAsiento: varchar(5) • clase: varchar(20)

<ul style="list-style-type: none"> Reservas: contiene información de las reservas 	<ul style="list-style-type: none"> idReserva : integer (entero) idVuelo FK: integer (entero) idPasajero FK: integer (entero) fechaReserva: datetime (fecha y hora)
<ul style="list-style-type: none"> Pagos: contiene información de los pagos 	<ul style="list-style-type: none"> idPago : integer (entero) idReserva FK: integer (entero) monto: decimal(10,2) medioPago: varchar(50) fecha: date (fecha)

Cada tabla tiene su clave primaria () que identifica de manera única a cada tupla.

Las tablas Vuelos y Pasajeros tienen foreign keys (**FK**) que referencian las tablas principales a las que están relacionadas.

Y se definieron índices (**INDEX**) en los campos de fechas de Vuelos para agilizar búsquedas y en los campos de ubicación de Aeropuertos para optimizar consultas espaciales.

Relaciones:

- Un avión puede ser asignado a varios vuelos.
- Un vuelo utiliza un solo avión.
- Un tripulante puede estar asignado a varios vuelos.
- Un vuelo tiene asignada una tripulación.
- Un vuelo despegue de un aeropuerto y aterriza en otro.
- Un aeropuerto puede ser origen o destino de varios vuelos.
- Un pasajero toma un vuelo.
- Un vuelo lleva varios pasajeros.
- Un vuelo lleva varios equipajes.
- Un pasajero puede tener varios equipajes.
- A un vuelo se le asignan varios asientos.
- Un vuelo tiene varias reservas.
- Un pasajero tiene una reserva.
- Una reserva puede tener un pago.

Creación de la Base de datos e inserción

```
-- -----  
-- Creación de la BASE.  
-- -----
```

```
CREATE DATABASE Flights;  
USE Flights;
```

```
-- -----  
-- Creación de las TABLAS.  
-- -----
```

```
-- Tabla Aviones-----  
CREATE TABLE Aviones (  
    idAvion INT PRIMARY KEY,  
    modelo VARCHAR(50),  
    capacidad INT  
);
```

```
-- Tabla Tripulacion-----  
CREATE TABLE Tripulacion (  
    idTripulante INT PRIMARY KEY,  
    nombre VARCHAR(50),  
    apellido VARCHAR(50),  
    puesto VARCHAR(50)  
);
```

```
-- Tabla Aeropuertos-----  
CREATE TABLE Aeropuertos (  
    idAeropuerto INT PRIMARY KEY,  
    nombre VARCHAR(100),  
    ciudad VARCHAR(50),  
    pais VARCHAR(50)  
);
```

```
-- Tabla Vuelos-----  
CREATE TABLE Vuelos (  
    idVuelo INT PRIMARY KEY,  
    idAvion INT,  
    idTripulante INT,  
    fechaSalida DATETIME,  
    fechaLlegada DATETIME,  
    idAeropuertoSalida INT,  
    idAeropuertoLlegada INT,  
    FOREIGN KEY (idAvion) REFERENCES Aviones(idAvion),  
    FOREIGN KEY (idTripulante) REFERENCES Tripulacion(idTripulante),  
    FOREIGN KEY (idAeropuertoSalida) REFERENCES  
Aeropuertos(idAeropuerto),  
    FOREIGN KEY (idAeropuertoLlegada) REFERENCES  
Aeropuertos(idAeropuerto)  
);
```

```
CREATE INDEX idx_fecha ON Vuelos(fechaSalida, fechaLlegada);
```

```
-- Tabla Pasajeros-----
```

```
CREATE TABLE Pasajeros (  
    idPasajero INT PRIMARY KEY,  
    nombre VARCHAR(50),  
    apellido VARCHAR(50),  
    idVuelo INT,  
    FOREIGN KEY (idVuelo) REFERENCES Vuelos(idVuelo)  
);
```

```
-- Tabla Equipajes-----
```

```
CREATE TABLE Equipajes (  
    idEquipaje INT PRIMARY KEY,  
    idVuelo INT,  
    idPasajero INT,  
    peso FLOAT,  
    FOREIGN KEY (idVuelo) REFERENCES Vuelos(idVuelo),  
    FOREIGN KEY (idPasajero) REFERENCES Pasajeros(idPasajero)  
);
```

```
-- Tabla Asientos-----
```

```
CREATE TABLE Asientos(  
    idAsiento INT PRIMARY KEY,  
    idVuelo INT,  
    numeroAsiento VARCHAR(5),  
    clase VARCHAR(20),  
    FOREIGN KEY (idVuelo) REFERENCES Vuelos(idVuelo)  
);
```

```
-- Tabla Reservas-----
```

```
CREATE TABLE Reservas(  
    idReserva INT PRIMARY KEY,  
    idVuelo INT,  
    idPasajero INT,  
    fechaReserva DATETIME,  
    FOREIGN KEY (idVuelo) REFERENCES Vuelos(idVuelo),  
    FOREIGN KEY (idPasajero) REFERENCES Pasajeros(idPasajero)  
);
```

```
-- Tabla Pagos-----
```

```
CREATE TABLE Pagos(  
    idPago INT PRIMARY KEY,  
    idReserva INT,  
    monto DECIMAL(10,2),  
    medioPago VARCHAR(50),  
    fecha DATE,  
    FOREIGN KEY (idReserva) REFERENCES Reservas(idReserva)  
);
```

Inserción de datos

```
-- Insertar datos en tabla Aviones
INSERT INTO Aviones VALUES
(1, 'Airbus A320', 150),
(2, 'Boeing 737', 180),
(3, 'Boeing 747', 400),
(4, 'Boeing 787', 300),
(5, 'Airbus A380', 550);

-- Insertar datos en tabla Tripulacion
INSERT INTO Tripulacion VALUES
(1, 'Juan', 'Perez', 'Piloto'),
(2, 'Maria', 'Garcia', 'Copiloto'),
(3, 'Pedro', 'Suarez', 'Asistente'),
(4, 'Karla', 'Mendez', 'Piloto'),
(5, 'Esteban', 'Lopez', 'Copiloto');

-- Insertar datos en tabla Aeropuertos
INSERT INTO Aeropuertos VALUES
(1, 'Aeropuerto Intl JFK', 'Nueva York', 'Estados Unidos'),
(2, 'Aeropuerto Intl LHR', 'Londres', 'Reino Unido'),
(3, 'Aeropuerto Intl CDG', 'Paris', 'Francia'),
(4, 'Aeropuerto Intl PEK', 'Beijing', 'China'),
(5, 'Aeropuerto Intl GRU', 'San Pablo', 'Brasil');

-- Insertar datos en tabla Vuelos
INSERT INTO Vuelos VALUES
(101, 1, 1, '2023-03-01 10:00', '2023-03-01 14:00', 1, 2),
(102, 2, 2, '2023-03-15 12:00', '2023-03-15 17:30', 2, 3),
(103, 3, 3, '2023-04-01 09:00', '2023-04-01 11:30', 3, 1),
(104, 4, 4, '2023-05-12 07:00', '2023-05-12 22:00', 4, 5),
(105, 5, 5, '2023-06-20 15:00', '2023-06-21 17:00', 5, 2);

-- Insertar datos en tabla Pasajeros
INSERT INTO Pasajeros VALUES
(1, 'Alice', 'Martinez', 101),
(2, 'Bob', 'Johnson', 101),
(3, 'Charlie', 'Garcia', 102),
(4, 'Diana', 'Flores', 103),
(5, 'Eric', 'Diaz', 103),
(6, 'Frank', 'Perez', 104),
(7, 'Gloria', 'Rodriguez', 104),
(8, 'Hugo', 'Gutierrez', 105),
(9, 'Isabel', 'Campos', 105);

-- Insertar datos en tabla Equipajes
INSERT INTO Equipajes (idEquipaje, idVuelo, idPasajero, peso)
VALUES
(1, 101, 1, 23.5),
(2, 102, 3, 17.2),
```



```

(3, 103, 4, 55.8),
(4, 101, 2, 22.0),
(5, 103, 5, 24.1),
(6, 104, 6, 29.0),
(7, 104, 7, 22.5),
(8, 105, 8, 20.3),
(9, 105, 9, 21.0);

-- Insertar datos en tabla Asientos
INSERT INTO Asientos VALUES
(1, 101, '1A', 'Primera'),
(2, 101, '1B', 'Primera'),
(3, 102, '14C', 'Turista'),
(4, 103, '18J', 'Ejecutiva'),
(5, 103, '22A', 'Turista'),
(6, 104, '38B', 'Primera'),
(7, 105, '5C', 'Turista');

-- Insertar datos en tabla Reservas
INSERT INTO Reservas VALUES
(1, 101, 1, '2023-01-29'),
(2, 102, 3, '2023-01-30'),
(3, 103, 4, '2023-02-01'),
(4, 103, 5, '2023-01-28'),
(5, 104, 6, '2023-02-02'),
(6, 105, 8, '2023-02-05'),
(11, 101, 1, '2023-01-29'),
(22, 102, 5, '2023-02-14'),
(33, 105, 9, '2023-02-20'),
(44, 104, 6, '2023-03-01'),
(55, 103, 8, '2023-03-05'),
(66, 105, 7, '2023-03-15');

-- Insertar datos en tabla Pagos
INSERT INTO Pagos VALUES
(1, 1, 125.50, 'Tarjeta de credito', '2023-02-10'),
(2, 2, 155.80, 'PayPal', '2023-02-11'),
(3, 3, 83.90, 'Transferencia bancaria', '2023-02-15'),
(101, 11, 223.5, 'Efectivo', '2023-01-29'),
(202, 22, 127.5, 'Transferencia', '2023-02-18'),
(303, 33, 95.6, 'Tarjeta', '2023-02-25'),
(404, 44, 354.1, 'PayPal', '2023-03-01'),
(505, 55, 194.8, 'Cripto', '2023-03-07'),
(606, null, 453.2, 'Tarjeta', '2023-03-16');

```

Vistas

La vista (view) es una consulta almacenada en la base de datos que se guarda con

un nombre y puede ser referenciada y utilizada como una tabla normal. Las vistas no almacenan datos por sí mismas; en cambio, son consultas predefinidas que se pueden utilizar para acceder y presentar datos de una o más tablas en una forma específica.

Las vistas en SQL son herramientas poderosas que ofrecen simplificación, seguridad y reutilización en el acceso a datos almacenados en una base de datos relacional.

Para este caso, se presentaron 5 vistas:

- Vista de vuelos con origen Estados Unidos = Vuelos con origen a EE.UU. Tablas que necesito son Vuelos y Aeropuertos.
- Vista de pasajeros por vuelo = Número de pasajeros por vuelo. Tablas que necesito son Vuelos y Pasajeros.
- Vista de tripulación con puesto de piloto = Pilotos . Tabla que necesito es Tripulación
- Vista de vuelos por modelo de avión = Cantidad de vuelos por modelo de avión. . Tablas que necesito son Vuelos y Aviones.
- Vista de aeropuertos de Latinoamérica = Aeropuertos de Latinoamérica. Necesito la tabla de Aeropuertos.

Scripts:

```
-- Vista de vuelos con origen Estados Unidos
CREATE VIEW vw_vuelos_usa AS
SELECT v.idVuelo, v.fechaSalida, v.fechaLlegada, a.nombre AS
aeropuertoSalida
FROM Vuelos v
INNER JOIN Aeropuertos a ON a.idAeropuerto = v.idAeropuertoSalida
WHERE a.pais = 'Estados Unidos';
```

```
-- Vista vuelos por pasajeros
CREATE VIEW vw_pasajeros_por_vuelo AS
SELECT v.idVuelo, v.fechaSalida, COUNT(p.idPasajero) AS pasajeros
FROM Vuelos v
LEFT JOIN Pasajeros p ON p.idVuelo = v.idVuelo
GROUP BY v.idVuelo;
```

```
-- Vista de tripulación con puesto de piloto
CREATE VIEW vw_pilotos AS
SELECT *
FROM Tripulacion
WHERE puesto = 'Piloto';
```

```
-- Vista de vuelos por modelo de avión
CREATE VIEW vw_vuelos_por_avion AS
SELECT a.modelo, COUNT(v.idVuelo) AS cantidad
FROM Vuelos v
INNER JOIN Aviones a ON a.idAvion = v.idAvion
```

```

GROUP BY a.modelo;

-- Vista de aeropuertos de Latinoamerica
CREATE VIEW vw_aeropuertos_latam AS
SELECT *
FROM Aeropuertos
WHERE pais IN ('Argentina', 'Brasil', 'Chile', 'Colombia', 'México',
'Perú');

```

Funciones

Las funciones son rutinas predefinidas que realizan una tarea específica y devuelven un valor único. Estas funciones pueden utilizarse en diversas partes de una consulta SQL para realizar operaciones en los datos, manipular valores y realizar cálculos.

Para este caso, se presentaron 2 funciones:

- Función para calcular la duración de un vuelo en horas. Esta función devuelve un entero, que es la cantidad de horas que dura un vuelo. Los parámetros que se necesitan, son las fechas de salidas y las fechas de llegada de los vuelos.
- Función para verificar si un tripulante está en un vuelo. Esta función devuelve Si o No, si la persona está o no en un vuelo, respectivamente. Los parámetros que se necesitan son el idTripulante, y el idVuelo de la tabla Vuelos

Scripts:

```

-- Creando Funciones -----
-- Función para calcular la duración de un vuelo en horas:

DELIMITER //

CREATE FUNCTION fn_duracion_vuelo(fechaHoraSalida DATETIME,
fechaHoraLlegada DATETIME)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE duracion INT;

    SET duracion =
        TIMESTAMPDIFF(HOUR, fechaHoraSalida, fechaHoraLlegada);

    RETURN duracion;
END //

DELIMITER ;
SELECT idVuelo, idAvion, fn_duracion_vuelo(fechaSalida, fechaLlegada)
AS duracion
FROM Vuelos;

-- Función para verificar si un tripulante está en un vuelo:

```

```

DELIMITER //

CREATE FUNCTION fn_tripulante_en_vuelo(pidTripulante INT, pidVuelo INT)
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
    DECLARE esta BOOLEAN;

    SELECT COUNT(*) INTO esta
    FROM Vuelos
    WHERE idVuelo = pidVuelo
    AND idTripulante = pidTripulante;

    IF esta > 0 THEN
        RETURN 'SI';
    ELSE
        RETURN 'NO';
    END IF;
END //

DELIMITER ;

SELECT nombre, apellido, fn_tripulante_en_vuelo(1, 102) AS Vuela
from tripulacion;

```

Store procedure

Un stored procedure (procedimiento almacenado) es un conjunto de instrucciones SQL que se almacenan en una base de datos para su reutilización. Un stored procedure puede aceptar parámetros, realizar operaciones en la base de datos y devolver resultados. Los stored procedures son utilizados para encapsular lógica de negocios o tareas específicas en el nivel de base de datos.

Para este caso, se crearon dos SP:

- Stored procedure que permite ordenar una tabla por diferentes campos en orden ascendente o descendente según parámetros, que son Tabla, Campo de la tabla, true si se quiere que esté ordenado de manera descendente, o false si se quiere de manera ascendente.
- Stored procedure que permite eliminar un registro de una tabla. En este caso los parámetros son Tabla, campo de la tabla, y el valor que se quiere eliminar.

Scripts:

```

-- SP1-----
DELIMITER //

CREATE PROCEDURE sp_ordenar_tabla(
    IN nombreTabla VARCHAR(50),
    IN nombreCampoOrden VARCHAR(50),
    IN esDescendente BOOLEAN

```

```

)
BEGIN

    DECLARE consulta VARCHAR(200);

    SET consulta = CONCAT('SELECT * FROM ', nombreTabla, ' ORDER BY ',
nombreCampoOrden);

    IF esDescendente THEN
        SET consulta = CONCAT(consulta, ' DESC');
    ELSE
        SET consulta = CONCAT(consulta, ' ASC');
    END IF;

    SET @sql = consulta;
    PREPARE stmt FROM @sql;
    EXECUTE stmt;

END //

DELIMITER ;

CALL sp_ordenar_tabla('Vuelos','fechaSalida', false); -- Ascendente
CALL sp_ordenar_tabla('Pasajeros','idPasajero', true); -- Descentente

-- SP2
-----
----

DELIMITER //

CREATE PROCEDURE sp_eliminar_registro(
    IN tabla VARCHAR(50),
    IN nombre_columna_primaria VARCHAR(50),
    IN valor_columna_primaria INT
)
BEGIN

    SET @sql = CONCAT('DELETE FROM ', tabla, ' WHERE ',
nombre_columna_primaria, ' = ', valor_columna_primaria);

    PREPARE stmt FROM @sql;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;

END //

DELIMITER ;

CALL sp_eliminar_registro('pasajeros', 'idPasajero', 5);

```

Triggers

Los triggers son un tipo de objeto en bases de datos relacionales que responden automáticamente a ciertos eventos, como la inserción, actualización o eliminación de registros en una tabla. Los triggers contienen instrucciones o procedimientos almacenados que se ejecutan de forma automática en respuesta a un evento específico en la base de datos. Los triggers se utilizan para automatizar acciones y mantener la integridad de los datos en la base de datos.

En este caso se utilizan dos triggers para las tablas Vuelos y Pasajeros, permitiendo la actualización de las mismas, sabiendo quien realiza los cambios y a la fecha en que se realiza.

Scripts:

```
CREATE TABLE nuevos_vuelos (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    fecha DATE,  
    hora TIME,  
    usuario VARCHAR(50),  
    accion VARCHAR(20),  
    tabla VARCHAR(50),  
    registro_afectado INT  
);  
  
delimiter //  
CREATE TRIGGER trg_vuelos_audit  
before INSERT ON Vuelos  
FOR EACH ROW  
BEGIN  
    INSERT INTO nuevos_vuelos VALUES(NULL, CURRENT_DATE(),  
CURRENT_TIME(), USER(), 'INSERT', 'Vuelos', NEW.idVuelo);  
END//  
  
delimiter //  
CREATE TRIGGER trg_vuelos_delete  
after delete ON Vuelos  
FOR EACH ROW  
BEGIN  
    INSERT INTO nuevos_vuelos VALUES(NULL, CURRENT_DATE(),  
CURRENT_TIME(), USER(), 'DELETE', 'Vuelos', OLD.idVuelo);  
END//  
  
delimiter ;  
  
CREATE TABLE nuevos_pasajeros (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    fecha DATE,
```

```

    hora TIME,
    usuario VARCHAR(50),
    mensaje VARCHAR(100),
    nombre VARCHAR(50),
    apellido VARCHAR(50)
);

delimiter //
CREATE TRIGGER trg_before_insert_pasajeros
BEFORE INSERT ON Pasajeros
FOR EACH ROW
BEGIN
    INSERT INTO nuevos_pasajeros VALUES(NULL, CURRENT_DATE(),
CURRENT_TIME(), USER(),'Insertando pasajero:',NEW.nombre,
NEW.apellido);
END//

delimiter //
CREATE TRIGGER trg_after_delete_pasajeros
AFTER DELETE ON Pasajeros
FOR EACH ROW
BEGIN
    INSERT INTO nuevos_pasajeros VALUES(NULL, CURRENT_DATE(),
CURRENT_TIME(), USER(),'Eliminando pasajero:',OLD.nombre,
OLD.apellido);
END//

delimiter ;
INSERT INTO Vuelos (idVuelo, fechaSalida, fechaLlegada, idAvion)
VALUES (106, '2023-07-01 10:00', '2023-07-01 12:00', 1);

select * from nuevos_vuelos;

INSERT INTO Pasajeros(idPasajero, nombre, apellido, idVuelo)
VALUES (301, 'Juan', 'Ditz', 105);

select * from nuevos_pasajeros;

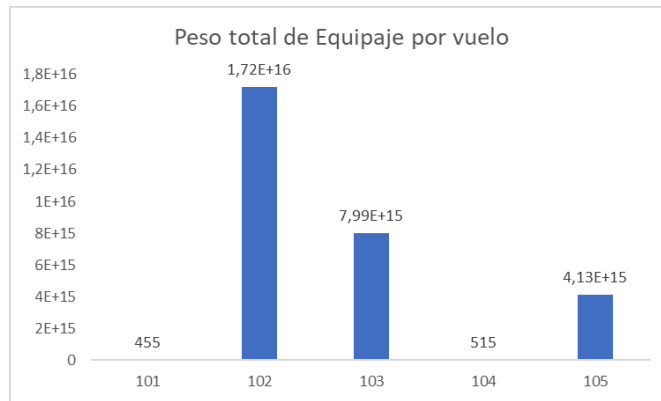
DELETE FROM Pasajeros WHERE idPasajero = 301;
select * from nuevos_pasajeros;

DELETE FROM Vuelos WHERE idVuelo = 106;
select * from nuevos_vuelos;

```

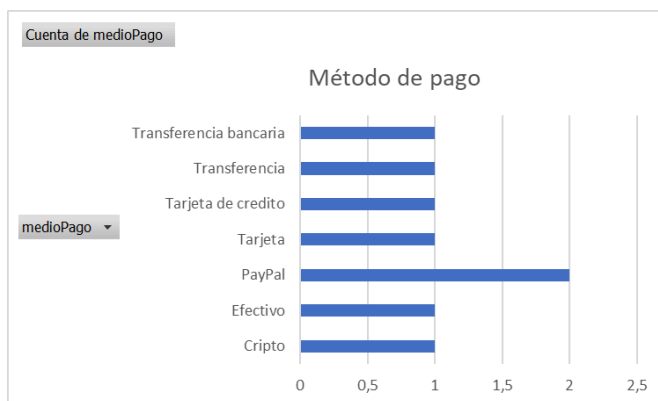
Informe: A partir de los datos insertados se generan los siguientes reportes. Es importante destacar que los datos son pocos y, además, ficticios.

- Peso total de equipaje por vuelo:



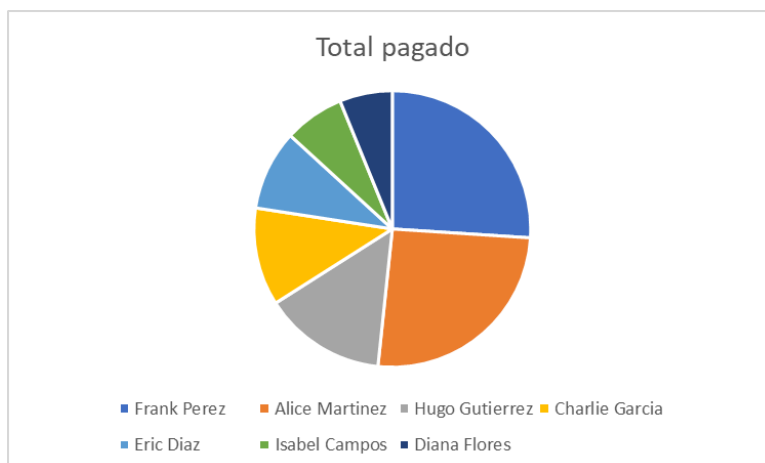
El vuelo que más peso lleva es el 102, mientras que el vuelo que menos peso carga es el 101, con 455.

- Métodos de pago:



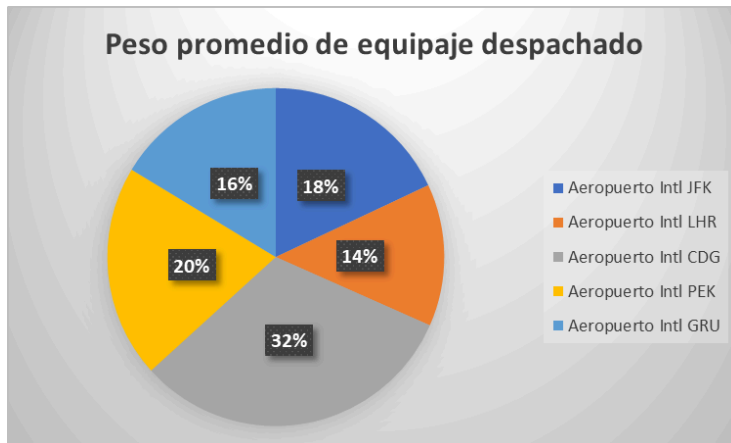
El método de pago más utilizado es PayPal.

- Pago realizado por pasajeros:



El pasajero que más gastó fue Frank Perez, mientras que Diana Flores gastó menos.

- Peso promedio de equipaje despachado por aeropuerto:



El aeropuerto que, en promedio, despacha, mayor peso de equipaje es el Aeropuerto Intl CDG, de París. Y el que menos peso medio de equipaje despacha es el de Londres.

Herramientas y tecnologías usadas:

- **Google Docs:** utilizado para crear el documento donde se describe la base de datos.
- **Herramienta Recortes:** utilizado para recortar partes del código.
- **MySQL Workbench:** utilizado para diseñar, gestionar y hacer mantenimiento de la base de datos.
- **Excel:** utilizado para realizar los gráficos del reporte.
- **Github:** Repositorio de donde encontrar los scripts para crear la BBDD.

Líneas futuras: Posibles líneas de trabajo futuras para mejorar y ampliar el modelo entidad-relación desarrollado para el sistema de la aerolínea:

1. Incorporar un módulo de inteligencia de negocios con un data warehouse y OLAP para análisis multidimensional y minería de datos que apoye la toma de decisiones.
2. Integrar el modelo con sistemas de empresas asociadas como hotelería, alquiler de vehículos, actividades turísticas para ofrecer planes complejos.
3. Extender el modelo para soportar operaciones de mantenimiento de flota vinculadas al historial de cada avión.
4. Incluir funcionalidades de movilidad como aplicaciones móviles para check-in, pase abordar, seguimiento de vuelos y manejo de equipaje.
5. Crear una capa de inteligencia artificial y machine learning para recomendación de rutas, predicción de demoras, atención automatizada.

Click aquí para descargar el script SQL completo