

# **Práctica 2**

## **Base de datos de aviones**

Asignatura: Bases de Datos (Grupo L1.01)

Curso 2024/25 – Lunes 8:00 a 10:00

**Entrega: domingo, 4 de mayo de 2025**

**César Tejedo Manovel (818924@unizar.es)**

**Lucía Vázquez Martín (871886@unizar.es)**

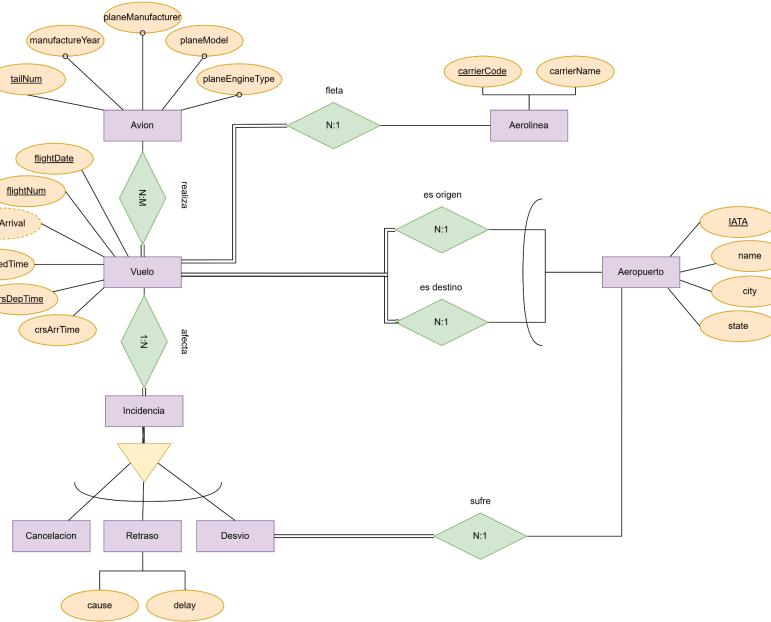
**Luna Zhou Chen (812103@unizar.es)**

## **ÍNDICE**

<b>ÍNDICE.....</b>	<b>1</b>
<b>Parte 1. Creación de una base de datos.....</b>	<b>2</b>
1.1. Esquema E/R.....	2
1.2. Esquema relacional.....	3
1.3. Sentencias SQL.....	4
<b>Parte 2. Introducción de datos y ejecución de consultas.....</b>	<b>6</b>
2.1. Pasos seguidos para poblar la BD.....	6
2.2. Consultas SQL.....	7
<b>Parte 3. Diseño físico.....</b>	<b>10</b>
3.1. Problemas de rendimiento de las consultas.....	10
3.2. Triggers.....	11
Restricción 1: Evitar que se registre un vuelo como “cancelado” y “desviado” o “retrasado” al mismo tiempo.....	11
Restricción 2: Eliminar automáticamente los datos asociados a un vuelo cuando se elimina su asignación desde la tabla Realiza.....	12
Restricción 3: Impide asignar un avión a un vuelo si ya está programado en otro vuelo que aún no ha aterrizado.....	13
<b>Anexo 1. Coordinación del grupo de trabajo.....</b>	<b>14</b>
<b>Anexo 2. Complicaciones durante el trabajo.....</b>	<b>14</b>

## Parte 1. Creación de una base de datos

### 1.1. Esquema E/R



Las restricciones de este modelo E/R son las siguientes:

- El aeropuerto de origen y de destino deben ser distintos.
- La duración de un retraso debe ser mayor a 0 en el caso de que exista, excepto cuando se hayan producido dos desvíos.
- En caso de desvío, el aeropuerto de destino debe ser distinto al de destino original.
- Un vuelo no puede tener una incidencia de tipo cancelación y retraso o desvío al mismo tiempo.
- Un retraso solo puede ser de causa: Carrier, Weather, NAS, Security, LateAircraft.
- Un avión no puede estar asignado a dos o más vuelos realizados en la misma hora.
- El atributo derivado *flightArrival* se calcula con la siguiente fórmula:

$$Fecha_{Llegada} = Fecha_{Salida} + 1 \text{ si } Hora_{Llegada}[min] < Hora_{Salida}[min]$$

Si quisiéramos tener en cuenta los husos horarios, se realizaría el siguiente cálculo:

$$\begin{cases} Fecha_{Salida} - 1 & \text{si } Hora_{Salida}(\text{En minutos}) + Duración + (GMT_{dest} - GMT_{orig})(\text{En minutos}) < 0 : 00 (\text{En minutos}) \\ Fecha_{Salida} & \text{si } Hora_{Salida}(\text{En minutos}) + Duración + (GMT_{dest} - GMT_{orig})(\text{En minutos}) \in [0 : 00, 23 : 59] (\text{En minutos}) \\ Fecha_{Salida} + 1 & \text{si } Hora_{Salida}(\text{En minutos}) + Duración + (GMT_{dest} - GMT_{orig})(\text{En minutos}) \geq 24 : 00 (\text{En minutos}) \end{cases}$$

En cuanto a las soluciones alternativas, se propuso reemplazar la relación *Vuelo–Aerolínea* por *Avión–Aerolínea*. Aunque esto no afectaría al funcionamiento de la BD, se optó por la primera opción, ya que, aunque una compañía podría poseer un avión sin haber realizado ningún vuelo con él, lo que nos interesa son los vuelos realizados por dicha aerolínea.

Por otro lado, aunque un vuelo no puede presentar simultáneamente dos tipos de incidencias, como una cancelación y un desvío, la relación se establece como 1:N. Esto se debe a que un vuelo puede sufrir múltiples retrasos, cada uno con su propia duración y causa, evitando así el uso de un atributo multivaluado para las causas y el cálculo manual de la duración total de los retrasos.

## 1.2. Esquema relacional

El modelo entidad-relación mostrado anteriormente ya se encuentra en *Tercera Forma Normal de Boyce-Codd* (BCNF). A continuación, se presentan las razones por las cuales no es necesario normalizarlo más.

Para que un modelo esté en 1FN, hay que asegurarse de que no tenga atributos multivaluados y que cada atributo se pueda identificar mediante una clave primaria. Esto se observa en los siguientes ejemplos:

- En la entidad *Retraso*, cada incidencia tiene una única causa y un único retraso, evitando así atributos multivaluados.
- En la entidad *Vuelo*, cada atributo está identificado de manera única por la combinación de *flightDate*, *flightNum* y *crsDepTime*.

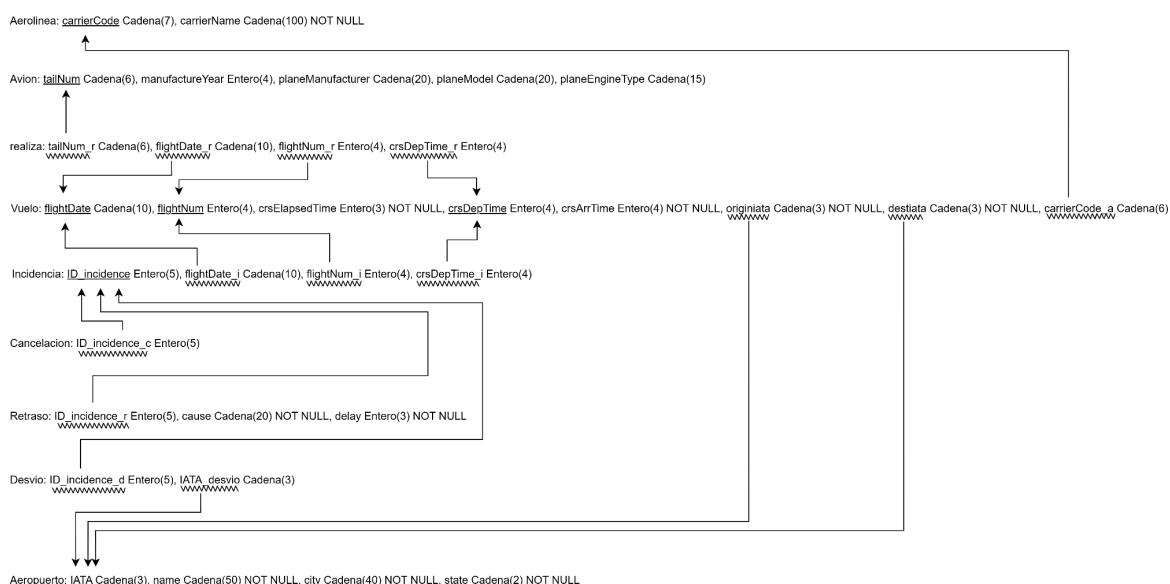
Si, además, se quiere cumplir la 2FN, no debe haber dependencias parciales, es decir, todo atributo no clave debe depender de la clave primaria completa y no solo de una parte de ella (en caso de claves compuestas). Estos son los siguientes ejemplos basados en el modelo:

- En la entidad *Vuelo*, los atributos como *crsElapsedTime*, *crsArrTime*, *originiata*, *destiata* y *carrierCode\_a* dependen de la clave primaria compuesta (*flightDate*, *flightNum* y *crsDepTime*) y no solo de una parte de ella.
- Se han separado las entidades *Retraso*, *Cancelación* y *Desvío*, evitando que haya atributos opcionales dentro de *Incidencia*.

Por último, para cumplir la 3FNBC, no deben existir atributos no clave que dependan de otros atributos no clave. Esto se puede comprobar en este caso hipotético:

- Si se almacenase el atributo *carrierName* en la entidad *Vuelo* en lugar de *Aerolinea*, esto implicaría una dependencia transitiva debido a que *carrierName* dependería de *carrierCode*, y a su vez *carrierCode* dependería de *flightNum* + *flightDate* + *crsDepTime*, violando así la 3FNBC. Esto no ocurre si dejamos el modelo como lo tenemos actualmente.

Por tanto, este es el modelo relacional normalizado.



### 1.3. Sentencias SQL

Para representar el modelo relacional descrito previamente, hay que crear las tablas de la siguiente manera:

```
-- Tabla que almacena información sobre las aerolíneas.  
CREATE TABLE Aerolinea (  
    carrierCode      VARCHAR(7) PRIMARY KEY, -- Código único de la aerolínea  
    carrierName     VARCHAR(100) NOT NULL -- Nombre de la aerolínea  
);  
  
-- Tabla que almacena información sobre los aeropuertos  
CREATE TABLE Aeropuerto (  
    IATA      VARCHAR(3) PRIMARY KEY, -- Código único del aeropuerto (IATA)  
    name      VARCHAR(50) NOT NULL, -- Nombre del aeropuerto  
    city      VARCHAR(40) NOT NULL, -- Ciudad donde se ubica el aeropuerto  
    state     VARCHAR(2) NOT NULL -- Estado donde se ubica el aeropuerto  
);  
  
-- Tabla que almacena información sobre los vuelos  
CREATE TABLE Vuelo (  
    flightDate      VARCHAR(10), -- Fecha del vuelo  
    flightNum       NUMBER(4), -- Número de vuelo  
    crsDepTime     NUMBER(4) CHECK(crsDepTime > 0), -- Hora estimada de salida (24h), debe ser positiva  
    crsElapsedTime NUMBER(3) NOT NULL CHECK(crsElapsedTime > 0),  
    crsArrTime     NUMBER(4) NOT NULL CHECK(crsArrTime > 0),  
    originiata     VARCHAR(3) NOT NULL, -- Aeropuerto de origen (código IATA)  
    destiata       VARCHAR(3) NOT NULL, -- Aeropuerto de destino (código IATA)  
    carrierCode_a  VARCHAR(7), -- Código de la aerolínea que realiza el vuelo.  
    FOREIGN KEY(originiata) REFERENCES Aeropuerto(IATA), -- Relación con el aeropuerto de origen  
    FOREIGN KEY(destiata) REFERENCES Aeropuerto(IATA), -- Relación con el aeropuerto de destino  
    FOREIGN KEY(carrierCode_a) REFERENCES Aerolinea(carrierCode),  
    PRIMARY KEY (flightDate, flightNum, crsDepTime),  
    CHECK (originiata <> destiata) -- Un vuelo no puede tener el mismo origen y destino  
);  
  
-- Tabla que almacena incidencias en los vuelos  
CREATE TABLE Incidencia (  
    ID_incidence   NUMBER(5) PRIMARY KEY, -- Identificador único de la incidencia  
    flightDate_i    VARCHAR(10), -- Fecha del vuelo asociado a la incidencia  
    flightNum_i     NUMBER(4), -- Número de vuelo asociado  
    crsDepTime_i   NUMBER(4), -- Hora programada de salida del vuelo asociado  
    FOREIGN KEY(flightDate_i, flightNum_i, crsDepTime_i) -- Relación con la tabla de vuelos  
        REFERENCES Vuelo(flightDate, flightNum, crsDepTime)  
);  
  
-- Tabla que almacena información sobre cancelaciones de vuelos  
CREATE TABLE Cancelacion (  
    ID_incidence_c NUMBER(5), -- Identificador de la incidencia asociada a la cancelación  
    FOREIGN KEY(ID_incidence_c) -- Relación con la tabla de incidencias  
        REFERENCES Incidencia(ID_incidence)  
);
```

```
-- Tabla que almacena información sobre retrasos en los vuelos
CREATE TABLE Retraso (
    ID_incidence_r NUMBER(5), -- Identificador de la incidencia asociada al retraso
    cause          VARCHAR(20) NOT NULL, -- Causa del retraso
    delay         NUMBER(3) NOT NULL CHECK(delay > 0), -- Duración retraso (min), debe ser positiva
    FOREIGN KEY(ID_incidence_r) -- Relación con la tabla de incidencias
        REFERENCES Incidencia(ID_incidence)
);

-- Tabla que almacena información sobre desvíos de vuelos
CREATE TABLE Desvio (
    ID_incidence_d NUMBER(5), -- Identificador de la incidencia asociada al desvío
    IATA_desvio    VARCHAR(3), -- Aeropuerto alternativo al que se desvió el vuelo
    FOREIGN KEY(ID_incidence_d) -- Relación con la tabla de incidencias
        REFERENCES Incidencia(ID_incidence),
    FOREIGN KEY(IATA_desvio) -- Relación con la tabla de aeropuertos
        REFERENCES Aeropuerto(IATA)
);

-- Tabla que almacena información sobre los aviones
CREATE TABLE Avion (
    tailNum           VARCHAR(6) PRIMARY KEY, -- Id del avión que realizó el vuelo
    manufactureYear  NUMBER(4), -- Año de fabricación del avión
    planeManufacturer VARCHAR(20), -- Fabricante del avión
    planeModel        VARCHAR(20), -- Modelo del avión
    planeEngineType   VARCHAR(15) -- Tipo de motor del avión
);

-- Tabla que almacena qué avión realizó qué vuelo
CREATE TABLE Realiza (
    tailNum_r      VARCHAR(6), -- Identificador del avión que realizó el vuelo
    flightDate_r   VARCHAR(10), -- Fecha del vuelo realizado
    flightNum_r    NUMBER(4), -- Número de vuelo realizado
    crsDepTime_r   NUMBER(4), -- Hora programada de salida del vuelo
    FOREIGN KEY(tailNum_r) -- Relación con la tabla de aviones
        REFERENCES Avion(tailNum),
    FOREIGN KEY(flightDate_r, flightNum_r, crsDepTime_r) -- Relación con la tabla de vuelos
        REFERENCES Vuelo(flightDate, flightNum, crsDepTime)
);
```

## **Parte 2. Introducción de datos y ejecución de consultas**

### **2.1. Pasos seguidos para poblar la BD**

Antes de poblar la base de datos, fue necesario adaptar el archivo *DatosVuelo.csv*, ya que todas las incidencias de un vuelo aparecían agrupadas en una sola fila, a pesar de ser de tipos distintos. Dado que nuestro modelo E/R establece una relación 1:N entre *Vuelo* e *Incidencia*, fue imprescindible separar cada incidencia en filas independientes para generar correctamente el archivo *Incidencias.csv*. De este modo, cada fila representa una única incidencia asociada a un vuelo.

Para semi automatizar la tarea, y evitar modificar manualmente más de 40.000 registros, se desarrolló el script *genIncidencias.py*, incluido en el archivo .zip, el cuál genera un nuevo archivo csv donde cada fila es una nueva incidencia. Si en un vuelo suceden múltiples incidencias (Por ejemplo, distintos tipos de retraso), el script genera una incidencia por cada incidente sufrido en el vuelo. El archivo .csv resultado, es uno que contiene todos los datos de las incidencias, y del cuál es mucho más fácil separar y obtener los respectivos archivos de población para las tablas *Incidencia*, *Cancelacion*, *Retraso* y *Desvio*.

Durante este proceso, surgieron algunas incidencias técnicas. Por un lado, detectamos que varios archivos .csv contenían el indicador BOM, lo que provocaba errores de carga. Esto se solucionó ajustando la codificación a *UTF-8 sin BOM* usando el siguiente comando: `sed '1s/^xEFxBB\xBF/' < [input].csv > [output].csv`. Por otro lado, ciertos campos como *carrierCode* y *carrierName* excedían la longitud definida inicialmente, por lo que se ampliaron sus tamaños. También fue necesario corregir el formato de fecha generado por el script, que producía fechas en formato *AAAA-DD-MM*, cuando en la tabla *Avion* el formato definido era *DD/MM/AAAA*, lo cual pudo ser solucionado fácilmente mediante herramientas como Excel o LibreOffice Calc.

Respecto a la carga de datos, primero se definieron las sentencias SQL para crear las tablas, y después se prepararon los archivos .ctl y .csv seleccionando las columnas necesarias de *DatosVuelo.csv*. Las tablas se poblaron en este orden: *Aerolinea*, *Aeropuerto*, *Vuelo*, *Incidencia*, *Cancelacion*, *Retraso*, *Desvio*, *Avion* y *realiza*.

El script de creación contenía un error de sintaxis, un CHECK que impedía la población de todas las filas de la tabla *Vuelo*, debido a que, como no tenemos en cuenta los husos horarios, establecimos que los vuelos no podían tener la misma hora de salida y llegada. Sin embargo, los datos sí que tienen en cuenta los husos horarios, por lo que existen casos de vuelos que van a ciudades con un huso una hora menos, y el viaje una duración estimada de una hora, dando como resultado un vuelo con misma hora de salida y llegada.

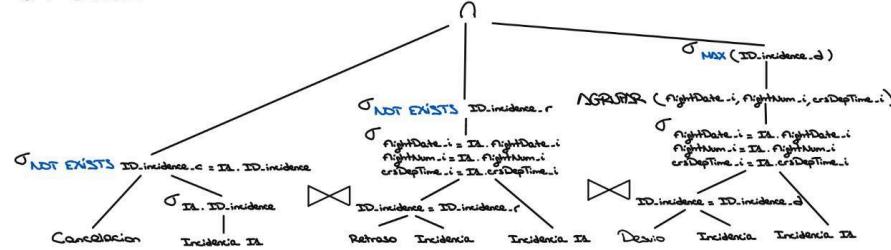
Para agilizar el proceso de eliminar tablas, en caso de que fuese necesario, se creó el archivo *borrar.sql*, el cual borra las tablas en el siguiente orden: *Avion*, *Cancelacion*, *Retraso*, *Desvio*, *Incidencia*, *Vuelo*, *Aerolinea* y *Aeropuerto*.

## 2.2. Consultas SQL

1) Lista todas las compañías aéreas, de más a menos puntual (según su media de retrasos, que también debe listarse), siempre que hayan operado al menos 1000 vuelos cada día.

Algebra Relacional:

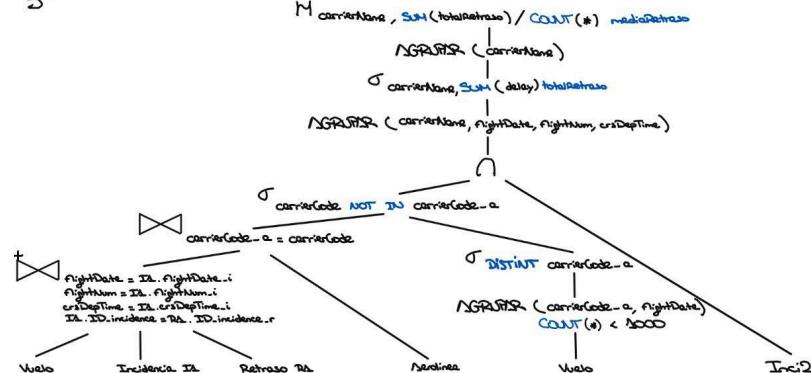
1º. Inc1Δ



2º. Inc2



3º



Sentencia SQL:

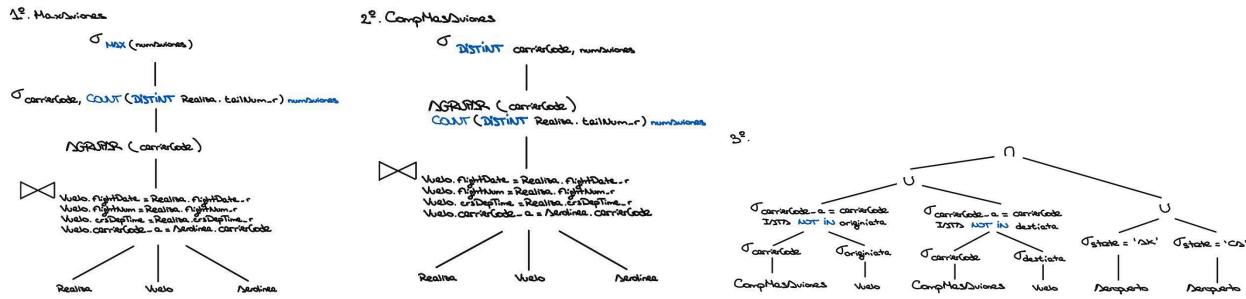
```
-- Calculamos la media de retraso por aerolinea
SELECT CARRIERNAME, SUM(TOTALRETRASO) / COUNT(*) AS MEDIARETRASO FROM (
    SELECT CARRIERNAME, SUM(DELAY) AS TOTALRETRASO FROM VUELO
    -- Creamos tabla con todos los vuelos y aquellos que tienen incidencias
    LEFT JOIN INCIDENCIA I1 ON I1.FLIGHTDATE_I=FLIGHTDATE AND I1.FLIGHNUM_I=FLIGHNUM AND I1.CRSDEPTIME_I=CRSDEPTIME
    -- Si las incidencias son retrasos añadimos tambien los "delays"
    LEFT JOIN RETRASO R1 ON I1.ID_INCIDENCE = R1.ID_INCIDENCE_R
    -- Añadimos el nombre de la aerolinea
    JOIN AEROLINEA ON CARRIERCODE = CARRIERCODE_A
    -- Comprobamos que el vuelo pertenece a una aerolinea que hace 1000 o mas vuelos diarios
    WHERE CARRIERCODE NOT IN (
        SELECT DISTINCT CARRIERCODE_A FROM VUELO
        GROUP BY CARRIERCODE_A, FLIGHTDATE
        HAVING COUNT(*) < 1000
    )
    AND (
        I1.ID_INCIDENCE IS NULL -- Es nulo -> el vuelo no tiene incidencias -> lo incluimos (Sólo saldrá una vez el vuelo)
        OR R1.ID_INCIDENCE IS NOT NULL -- No es nulo -> Esta fila asocia un vuelo con uno de sus retrasos -> lo incluimos
        -- En este punto, esta fila asocia un vuelo con un desvío/cancelación -> si existe, es una cancelación -> No la incluimos
        OR (NOT EXISTS (SELECT ID_INCIDENCE_C FROM CANCELACION WHERE ID_INCIDENCE_C = I1.ID_INCIDENCE))
        -- Esta fila asocia un vuelo con un desvío -> Si existe, este vuelo ya tiene retrasos asociados -> No la incluimos
        AND NOT EXISTS (SELECT ID_INCIDENCE_R FROM RETRASO JOIN INCIDENCIA ON ID_INCIDENCE_R = ID_INCIDENCE
                        WHERE I1.FLIGHTDATE_I=FLIGHTDATE_I AND I1.FLIGHNUM_I=FLIGHNUM_I AND I1.CRSDEPTIME_I=CRSDEPTIME_I)
        -- Este vuelo solo tiene desvíos, solo incluimos 1 (El de mayor ID por ejemplo) para que cuente como retraso = 0
        AND I1.ID_INCIDENCE = (SELECT MAX(ID_INCIDENCE_D) FROM DESVIO JOIN INCIDENCIA ON ID_INCIDENCE_D = ID_INCIDENCE
                               WHERE I1.FLIGHTDATE_I=FLIGHTDATE_I AND I1.FLIGHNUM_I=FLIGHNUM_I AND I1.CRSDEPTIME_I=CRSDEPTIME_I
                               GROUP BY FLIGHTDATE_I, FLIGHNUM_I, CRSDEPTIME_I)
    )
    GROUP BY CARRIERNAME, FLIGHTDATE, FLIGHNUM, CRSDEPTIME
)
GROUP BY CARRIERNAME
ORDER BY MEDIARETRASO;
```

Resultado Consulta:

CARRIERNAME	MEDIARETRASO
Southwest Airlines Co.	5,27076543
Skywest Airlines Inc.	9,37890819
American Airlines Inc.	11,428259
American Eagle Airlines Inc.	12,2482497

## 2) Listado de aeropuertos en Alaska o California en los que no opera la compañía con más aviones.

Algebra Relacional:



Sentencia SQL:

```
-- Vista que calcula la compañía con mas aviones
WITH COMPMASAVIONES AS ( -- Contamos el nº de tailnums distintos en los vuelos realizados por cada compañía
    SELECT CARRIERCODE, COUNT(DISTINCT REALIZA.TAILNUM_R) AS NUMAVIONES
    FROM REALIZA JOIN VUELO ON (REALIZA.FLIGHTDATE_R=VUELO.FLIGHTDATE AND REALIZA.FLIGHNUM_R=VUELO.FLIGHNUM
        AND REALIZA.CRSDEPTIME_R=VUELO.CRSDEPTIME)
    JOIN AEROLINEA ON (VUELO.CARRIERCODE_A=AEROLINEA.CARRIERCODE)
    GROUP BY CARRIERCODE
    -- Seleccionamos la aerolinea cuyo nº de aviones sea igual al máximo.
    HAVING COUNT(DISTINCT REALIZA.TAILNUM_R) =
    -- Nos quedamos con el nº mayor
    SELECT MAX(NUMAVIONES) FROM ( -- Contamos el nº de tailnums distintos en los vuelos realizados por cada compañía
        SELECT DISTINCT CARRIERCODE, COUNT(DISTINCT REALIZA.TAILNUM_R) AS NUMAVIONES
        FROM REALIZA
        JOIN VUELO ON (REALIZA.FLIGHTDATE_R=VUELO.FLIGHTDATE AND REALIZA.FLIGHNUM_R=VUELO.FLIGHNUM
            AND REALIZA.CRSDEPTIME_R=VUELO.CRSDEPTIME)
        JOIN AEROLINEA ON (VUELO.CARRIERCODE_A=AEROLINEA.CARRIERCODE)
        GROUP BY CARRIERCODE)
    )
    SELECT DISTINCT * FROM AEROPUERTO
    -- Seleccionamos los aeropuertos que están en alaska ó california y cuyo iata no sea el origen ó destino de un vuelo de la
    -- compañía con más aviones
    WHERE (STATE='AK' OR STATE='CA') AND (
        IATA NOT IN (SELECT ORIGINIATA FROM VUELO WHERE CARRIERCODE_A=(SELECT CARRIERCODE FROM COMPMASAVIONES))
        OR IATA NOT IN (SELECT DESTIATA FROM VUELO WHERE CARRIERCODE_A=(SELECT CARRIERCODE FROM COMPMASAVIONES)));

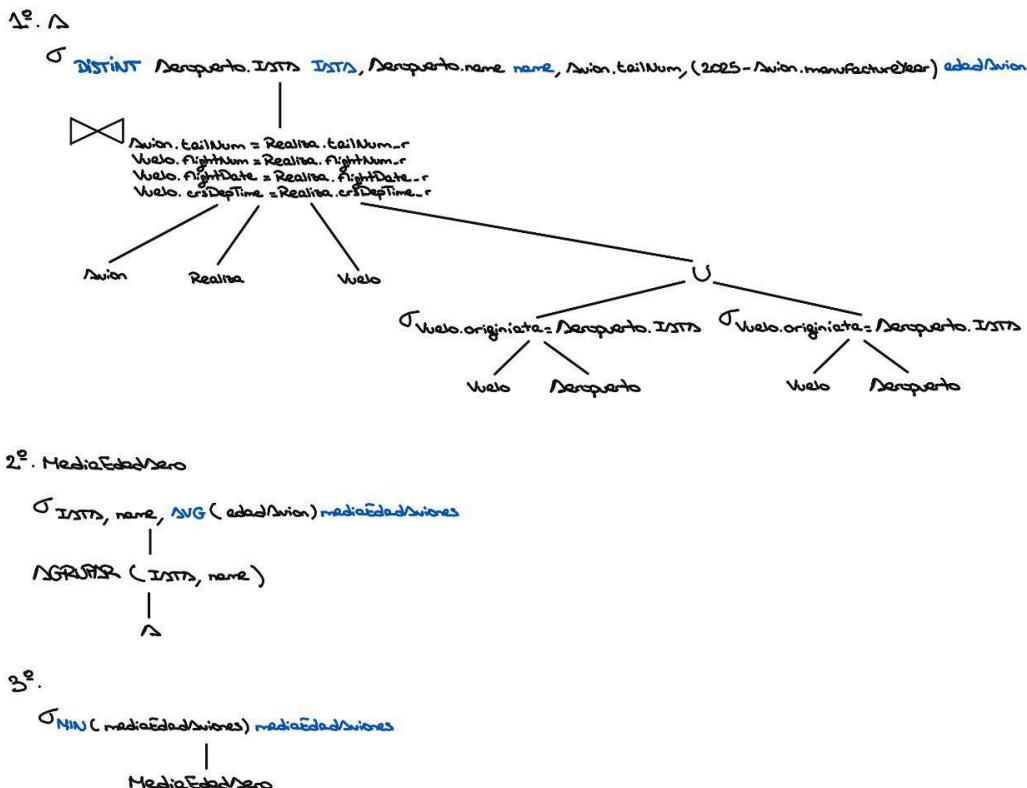
```

Resultado Consulta:

IAT NAME	CITY	ST
ADK Adak	Adak	AK
ACV Arcata	Arcata/Eureka	CA
BET Bethel	Bethel	AK
CIC Chico Municipal	Chico	CA
SCC Deadhorse	Deadhorse	AK
FAI Fairbanks International	Fairbanks	AK
IPL Imperial County	Imperial	CA
IYK Inyokern	Inyokern	CA
CEC Jack McNamara	Crescent City	CA
PSG James C. Johnson Petersburg	Petersburg	AK
JNU Juneau International	Juneau	AK
KTN Ketchikan International	Ketchikan	AK
ADQ Kodiak	Kodiak	AK
LGB Long Beach (Daugherty )	Long Beach	CA
BFL Meadows	Bakersfield	CA
CDV Merle K (Mudhole) Smith	Cordova	AK
OAK Metropolitan Oakland International	Oakland	CA
MOD Modesto City-County-Harry Sham	Modesto	CA
MRY Monterey Peninsula	Monterey	CA
OME Nome	Nome	AK
OXR Oxnard	Oxnard	CA
PMD Palmdale Production Flight	Palmdale	CA
OTZ Ralph Wien Memorial	Kotzebue	AK
RDD Redding Municipal	Redding	CA
SBP San Luis Obispo Co-McChesney	San Luis Obispo	CA
SBA Santa Barbara Municipal	Santa Barbara	CA
SMX Santa Maria Pub/Capt G Allan Hancock	Santa Maria	CA
SIT Sitka	Sitka	AK
ANC Ted Stevens Anchorage International	Anchorage	AK
BRW Wiley Post Will Rogers Memorial	Barrow	AK
WRG Wrangell	Wrangell	AK
YAK Yakutat	Yakutat	AK

**3) Obtener el aeropuerto en el que operan los aviones más modernos (es decir, con menor media de edad). Obtener tanto el nombre y el código del aeropuerto como la media de edad de los aviones que operan en él.**

Algebra Relacional:



Sentencia SQL:

```
-- Vista con la edad media de los aviones que operan en cada aeropuerto
WITH MEDIAEDADAERO AS (
    -- Usamos AVG al calcular la media para omitir los valores nulos
    SELECT IATA, NAME, AVG(EDADAVION) AS MEDIAEDADAVIONES
    FROM (
        -- Seleccionamos de cada aeropuerto, cada avión distinto que haya
        -- realizado un vuelo en dicho aeropuerto y su edad.
        SELECT DISTINCT AEROPUERTO.IATA AS IATA,
        AEROPUERTO.NAME AS NAME, AVION.TAILNUM,
        (2025 - AVION.MANUFACTUREYEAR) AS EDADAVION
        FROM AVION JOIN REALIZA ON (AVION.TAILNUM = REALIZA.TAILNUM_R)
        JOIN VUELO ON (REALIZA.FLIGHTNUM_R = VUELO.FLIGHTNUM
                    AND REALIZA.FLIGHDATE_R = VUELO.FLIGHDATE
                    AND REALIZA.CRSDEPTIME_R = VUELO.CRSDEPTIME)
        JOIN AEROPUERTO ON (VUELO.ORIGINIATA = AEROPUERTO.IATA
                            OR VUELO.DESTIATA = AEROPUERTO.IATA)
    )
    GROUP BY IATA, NAME
) -- Seleccionamos el aeropuerto con menor edad media de los aviones que operan en él
SELECT * FROM MEDIAEDADAERO
-- Seleccionamos la menor edad media de la vista
WHERE MEDIAEDADAVIONES = (SELECT MIN(MEDIAEDADAVIONES) AS MEDIAEDADAVIONES FROM MEDIAEDADAERO);
```

Resultado:

IAT	NAME	MEDIAEDADAVIONES
HDN	Yampa Valley	20,8888889

## Parte 3. Diseño físico

### 3.1. Problemas de rendimiento de las consultas

Para evaluar el rendimiento, usamos *EXPLAIN PLAN* y *DBMS\_XPLAN.DISPLAY()* de Oracle y la instrucción *SET TIMING ON* para medir el tiempo de ejecución.

#### Consulta 1 (Coste original: 244, 0,87 segundos)

El principal problema eran los JOIN entre *Vuelo*, *Incidencia* y *Retraso/Desvio*, además de los GROUP BY y ORDER BY. Como el problema no era el acceso, sino la agrupación y el ordenamiento, se crearon dos vistas materializadas independientes en la consulta original:

- *c1\_mv\_COMMILVUELOS.sql* guarda las aerolíneas con  $\geq 1000$  vuelos diarios (Coste: 2488, 0.87 segundos).
- *c1\_mv\_TOTALRETRASOVUELO.sql* extrae el retraso de cada vuelo (Coste: 30, 0.02 segundos).

Dado que el coste de **COMMILVUELOS** es mucho mayor que el original, y el coste temporal no mejora prácticamente nada, nos quedamos con la vista materializada **TOTALRETRASOVUELO** por dar un mejor resultado. Aún así, habría que realizar más pruebas, debido a que este elevado coste puede ser un sobrecoste del optimizador de Oracle.

Por último aplicamos un índice a la consulta que optimiza el acceso a la tabla *Vuelo*.

```
CREATE INDEX idx_vuelo ON VUELO (CARRIERCODE_A, FLIGHTDATE, FLIGHTNUM, CRSDEPTIME);
```

Sin embargo, este no mejoraba el coste de **TOTALRETRASOVUELO** (No usa el índice), produce una mejora poco apreciable en **COMMILVUELOS** (Coste: 2450), y mejora notablemente la consulta base (Coste: 120).

#### Consulta 2 (Coste original: 416, 0.14 segundos)

Gran parte del coste se debía a la subconsulta *CompMasAviones*, evaluada repetidamente con agregaciones anidadas, y a los accesos completos a las tablas *realiza* y *Vuelo*.

Para optimizarla, se probaron tres alternativas:

- Dos índices, que redujeron el coste a 180.
  - Índice para búsquedas por aerolínea y aeropuerto de origen  
`CREATE INDEX idx_carrier_orig ON VUELO (carrierCode_a, originiata);`
  - Índice para búsquedas por aerolínea y aeropuerto de destino  
`CREATE INDEX idx_carrier_dest ON VUELO (carrierCode_a, destiata);`
- Solo la vista materializada (*c2\_mv\_COMPMASAVIONES.sql*), con un coste de 214 y tiempo de 0.1 segundos.
- Combinación de ambos métodos, con un coste de 7 y tiempo de 0.05 segundos, siendo esta la opción elegida.

#### Consulta 3 (Coste original: 78, 29.2 segundos)

El elevado coste temporal se debía a accesos completos a las tablas *Vuelo*, *realiza*, *Avion* y *Aeropuerto*, junto con duplicación de JOIN para origen y destino, y uso de DISTINCT, que incrementaban el consumo de CPU. Para optimizarla, diseñamos una vista materializada, *c3\_mv\_AEROMEDIAEDADAVION.sql*, que redujo el coste a 6 y un tiempo aproximadamente nulo. Dado este buen resultado, se consideró innecesario aplicar índices, ya que el beneficio adicional sería insignificante frente al aumento en el uso de memoria.

### 3.2. Triggers

Hay restricciones de integridad que no pueden definirse en Oracle únicamente con claves primarias, foráneas o cláusulas CHECK. Para validar condiciones complejas o comparaciones entre tablas, es necesario utilizar triggers.

A continuación, se enumeran algunas restricciones que requieren este tipo de solución:

#	Restricción	Explicación
1	Validar que la causa de retraso esté entre las opciones predefinidas.	CHECK permite definir listas fijas, pero no es flexible ante cambios. Si se modifica el conjunto de valores permitidos, habría que alterar la estructura de la tabla. Un trigger permite controlarlo con mayor flexibilidad.
2	Evitar que se registre un vuelo como “cancelado” y “desviado” o “retrasado” al mismo tiempo.	Se necesita validar exclusividad entre registros de diferentes tablas ( <i>Cancelacion</i> , <i>Desvio</i> , <i>Retraso</i> ), lo cual no puede expresarse con CHECK, ya que este no permite acceder a otras filas ni otras tablas.
3	Asegurar que si un vuelo está desviado, debe tener al menos un aeropuerto alternativo registrado.	Se necesita validar una dependencia condicional entre campos de distintas tablas ( <i>Vuelo</i> y <i>Desvio</i> ), y CHECK no permite hacer referencias cruzadas entre relaciones.
4	El año de fabricación de un avión no puede ser posterior a la fecha de un vuelo realizado con ese avión.	Se necesita comparar atributos de tablas diferentes ( <i>Avion.manufactureYear</i> y <i>Vuelo.flightDate</i> ), y CHECK solo permite condiciones dentro de la misma fila.
5	Eliminar automáticamente los datos asociados a un vuelo cuando se elimina su asignación desde la tabla <i>Realiza</i> .	Se necesita realizar eliminaciones en tablas relacionadas ( <i>Incidencia</i> , <i>Cancelacion</i> , <i>Desvio</i> , <i>Retraso</i> y <i>Vuelo</i> ), CHECK no puede implementarlo ya que solo permite condiciones dentro de una misma fila y tabla, y no acciones de borrado en cascada entre tablas diferentes.
6	Impedir que un avión esté asignado a dos o más vuelos programados para la misma hora.	Se necesita comparar múltiples registros de la tabla <i>Realiza</i> para el mismo avión en una misma fecha y hora, lo cual no puede ser validado mediante CHECK, ya que este solo evalúa una fila a la vez y no permite restricciones entre tuplas.

#### Restricción 1: Evitar que se registre un vuelo como “cancelado” y “desviado” o “retrasado” al mismo tiempo

El trigger se ejecuta antes de insertar o actualizar una fila en *Cancelacion*. Busca en las tablas *Retraso* y *Desvio* si el identificador de incidencia coincide con el asociado a la cancelación. En ese caso, lanza un error.

```
CREATE OR REPLACE TRIGGER trg_BI_cancelacion_exclusiva
BEFORE INSERT ON Cancelacion
FOR EACH ROW
DECLARE
    existe_retraso NUMBER;
    existe_desvio NUMBER;
```

```
BEGIN
    -- Comprobamos si la incidencia se encuentra ya en Retraso
    SELECT COUNT(*) INTO existe_retraso FROM Retraso
    WHERE ID_incidence_r = :NEW.ID_incidence_c;
    -- Comprobamos si la incidencia se encuentra ya en Desvio
    SELECT COUNT(*) INTO existe_desvio FROM Desvio
    WHERE ID_incidence_d = :NEW.ID_incidence_c;
    -- Si está presente en cualquiera, no se puede insertar en Cancelacion
    IF existe_retraso > 0 OR existe_desvio > 0 THEN
        raise_application_error(-20000, 'La incidencia ya ha sido registrada
            como retraso o desvio. No puede ser tambien una cancelacion.');
    END IF;
END;
/
```

## Restricción 2: Eliminar automáticamente los datos asociados a un vuelo cuando se elimina su asignación desde la tabla Realiza

El trigger se activa tras la eliminación de una fila en *Realiza*. Utiliza los valores de *:OLD* para identificar el vuelo correspondiente y eliminar manualmente los registros relacionados en *Incidencia*, *Cancelacion*, *Desvio*, *Retraso* y *Vuelo*, evitando los datos huérfanos.

```
CREATE OR REPLACE TRIGGER trg_AD_avion_descatalogado
AFTER DELETE ON Realiza
FOR EACH ROW
BEGIN
    -- Eliminar cancelaciones del vuelo eliminado
    DELETE FROM Cancelacion WHERE ID_incidence_c IN (
        SELECT ID_incidence FROM Incidencia
        WHERE flightDate_i = :OLD.flightDate_r
        AND flightNum_i = :OLD.flightNum_r
        AND crsDepTime_i = :OLD.crsDepTime_r
    );
    -- Eliminar desvios del vuelo eliminado
    DELETE FROM Desvio WHERE ID_incidence_d IN (
        SELECT ID_incidence FROM Incidencia
        WHERE flightDate_i = :OLD.flightDate_r
        AND flightNum_i = :OLD.flightNum_r
        AND crsDepTime_i = :OLD.crsDepTime_r
    );
    -- Eliminar retrasos del vuelo eliminado
    DELETE FROM Retraso WHERE ID_incidence_r IN (
        SELECT ID_incidence FROM Incidencia
        WHERE flightDate_i = :OLD.flightDate_r
        AND flightNum_i = :OLD.flightNum_r
        AND crsDepTime_i = :OLD.crsDepTime_r
    );
    -- Eliminar incidencia del vuelo eliminado
    DELETE FROM Incidencia
    WHERE flightDate_i = :OLD.flightDate_r
    AND flightNum_i = :OLD.flightNum_r
    AND crsDepTime_i = :OLD.crsDepTime_r;
```

```
-- Eliminar el vuelo (si no está ya eliminado)
DELETE FROM Vuelo
WHERE flightDate = :OLD.flightDate_r
    AND flightNum = :OLD.flightNum_r
    AND crsDepTime = :OLD.crsDepTime_r;
END;
/
```

### Restricción 3: Impide asignar un avión a un vuelo si ya está programado en otro vuelo que aún no ha aterrizado

El trigger se ejecuta antes de insertar o actualizar una fila en *realiza*. Calcula la hora estimada de llegada del nuevo vuelo y comprueba si el avión (*:NEW.tailNum\_r*) ya está asignado, en la misma fecha, a otro vuelo cuyo intervalo horario se solape. Si detecta un conflicto, lanza un error para evitar asignaciones incompatibles.

```
CREATE OR REPLACE TRIGGER trg_BI_realiza_sin_conflictos
BEFORE INSERT OR UPDATE ON realiza
FOR EACH ROW
DECLARE
conflictos NUMBER; -- Variable para contar vuelos en conflicto
hora_salida_nueva NUMBER; -- Hora de salida del nuevo vuelo
hora_llegada_nueva NUMBER; -- Hora estimada de llegada del nuevo vuelo
BEGIN
-- Obtenemos la hora de salida y llegada del vuelo que se quiere asignar
SELECT TRUNC(crsDepTime / 100) * 60 + MOD(crsDepTime, 100),
       TRUNC(crsDepTime / 100) * 60 + MOD(crsDepTime, 100) + crsElapsedTime
  INTO hora_salida_nueva, hora_llegada_nueva
 FROM Vuelo
 WHERE flightDate = :NEW.flightDate_r
       AND flightNum = :NEW.flightNum_r
       AND crsDepTime = :NEW.crsDepTime_r;
-- Revisamos si el avión ya tiene otro vuelo solapado ese día
SELECT COUNT(*) INTO conflictos
FROM realiza
JOIN Vuelo ON realiza.flightDate_r = Vuelo.flightDate
           AND realiza.flightNum_r = Vuelo.flightNum
           AND realiza.crsDepTime_r = Vuelo.crsDepTime
WHERE realiza.tailNum_r = :NEW.tailNum_r
      -- Mismo avión
      AND realiza.flightDate_r = :NEW.flightDate_r
      -- Mismo día
      AND NOT (-- Verificamos que los vuelos se solapan
                -- Vuelo anterior termina antes de que empiece el nuevo
                TRUNC(crsDepTime / 100) * 60 + MOD(crsDepTime, 100) + crsElapsedTime < hora_salida_nueva
                OR
                -- Vuelo anterior empieza después de que termine el nuevo
                TRUNC(crsDepTime / 100) * 60 + MOD(crsDepTime, 100) > hora_llegada_nueva
);
IF conflictos > 0 THEN
raise_application_error(-20000,
'El avión ya está asignado a otro vuelo que aún no ha aterrizado.');
END IF;
END;
/
```

## Anexo 1. Coordinación del grupo de trabajo

La división del trabajo y las horas dedicadas en cada parte han sido muy similares entre los tres integrantes. Desde el inicio, acordamos que el objetivo principal era que todos comprendiésemos en profundidad los conceptos involucrados en el diseño y desarrollo de la base de datos. Por ello, adoptamos una metodología colaborativa.

Para facilitar el aprendizaje equitativo y la organización, establecimos fechas clave para:

- Acordar el enfoque conceptual de cada parte.
- Revisar y validar el trabajo individual de cada miembro.
- Distribuir tareas que luego eran rotadas o revisadas entre todos.

Fecha	Apartados del trabajo realizados	César	Lucía	Luna
31/03	Esquema E/R, esquema relacional, normalización	5	6	6
31/03	Sentencias SQL para crear las tablas	1	2	2
31/03, 07/04	Población de las tablas	5	5	3
07/04, 08/04, 09/04, 10/04, 11/04, 30/04, 02/05, 03/05	Consultas SQL y análisis de rendimiento	21	17	17
08/04, 09/04, 10/04, 11/04, 28/04, 30/04	Implementación de triggers	6	7	7
31/03, 07/04, 08/04, 09/04, 10/04, 11/04, 28/04, 30/04, 02/05, 03/05	Documentación	6	6	6
		44	43	41

## Anexo 2. Complicaciones durante el trabajo

Durante el desarrollo de la base de datos, hemos tenido que enfrentarnos a algunas complicaciones que han influido en la distribución del trabajo a lo largo de los días. La principal dificultad ha sido la falta de disponibilidad, debido a la coincidencia con entregas de otras asignaturas. A pesar de ello, logramos coordinarnos eficazmente a través de *WhatsApp*, ya que no siempre fue posible coincidir para trabajar juntos presencialmente.

Por otra parte, al partir de una única tabla (*DatosVuelo.csv*) y tener que convertirlos en varias entidades relacionadas, hemos tenido que hacer muchos ajustes y transformaciones antes de poblar las tablas. Esto ha consumido bastante tiempo de prueba y error.

Además, para entender y aplicar correctamente las técnicas de optimización en Oracle, dedicamos un esfuerzo adicional a analizar el rendimiento de nuestras consultas SQL, así como el uso de índices y vistas materializadas para mejorar su eficiencia.