BASES DE DATOS

CURSO 2024-2025

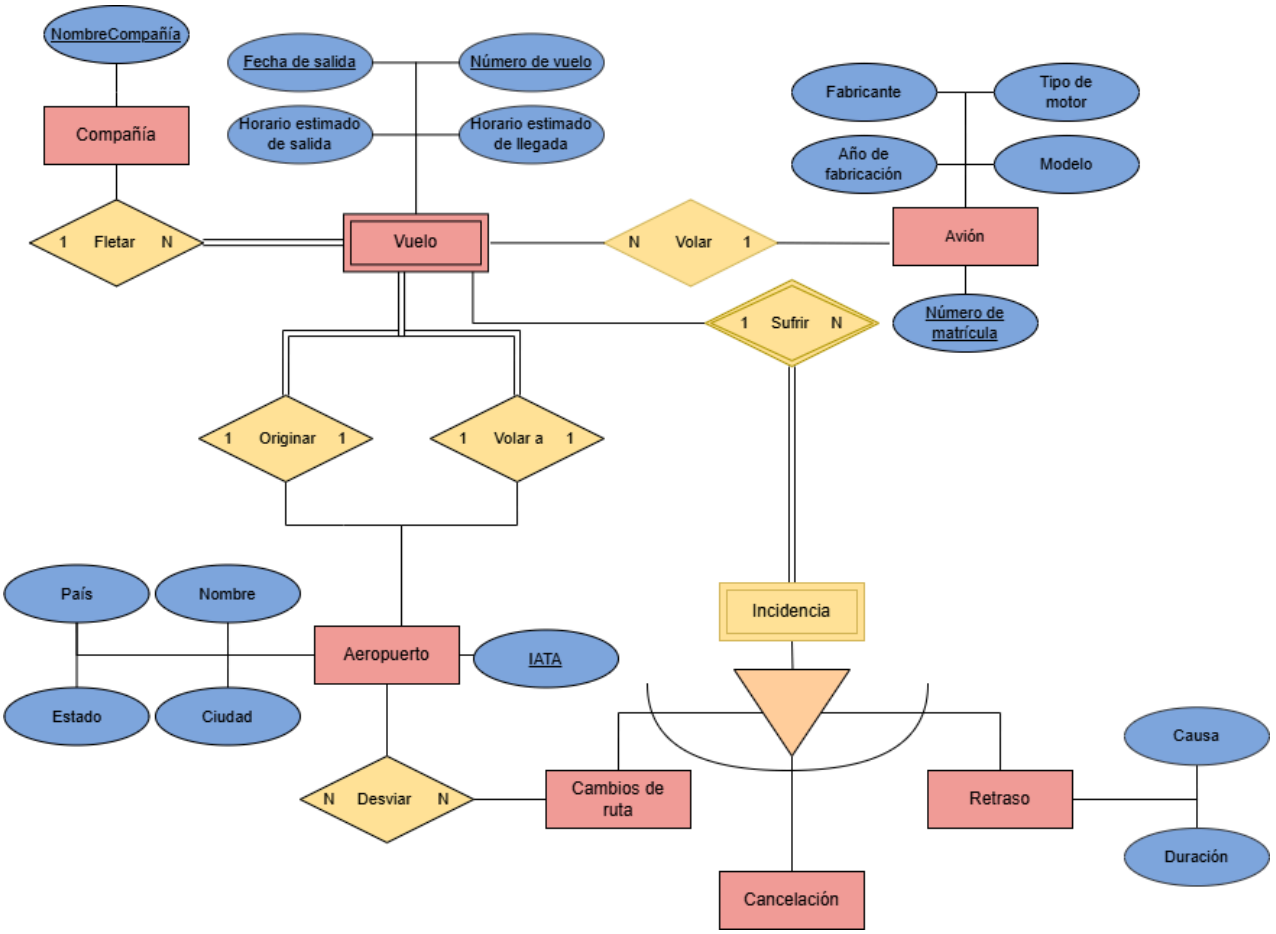
Memoria Segunda Base de Datos: Vuelos de aviones

Grupo: J2.10
Fecha de entrega: 01/05/2025
Autores:
- Fernando Pastor Peralta (897113) .....897113@unizar.es
- Juan José Muñoz Lahoz (902677) .....902677@unizar.es
- Marcos San Julián Fuertes (899849) .....899849@unizar.es

ÍNDICE:

Parte 1: Creación de una base de datos

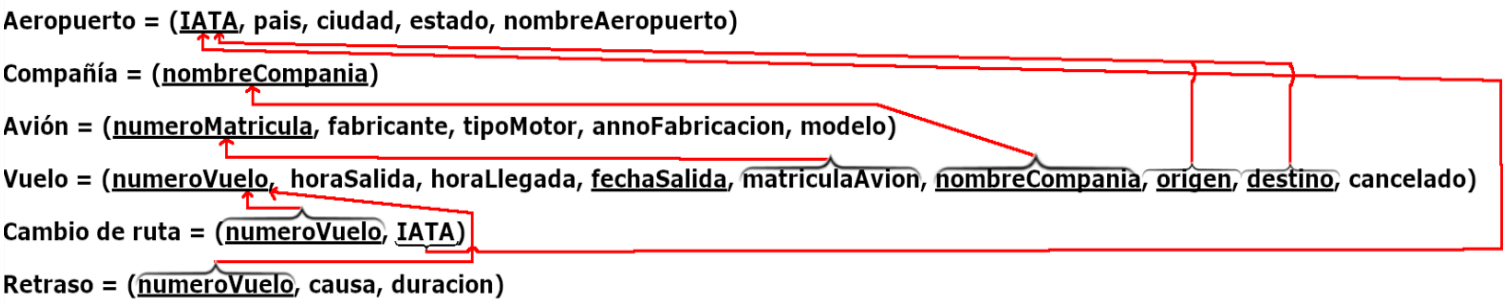
1.1. Esquema E/R global y listado de las restricciones



Restricciones:
- Un vuelo no puede salir del mismo aeropuerto al que debe llegar
- Un vuelo no puede tener un horario de llegada posterior al horario de salida
- El año de fabricación del avión debe ser en una fecha pasada.
- No se puede borrar un aeropuerto si tiene vuelos programados de ida o vuelta en un futuro .
+ restricciones en el apartado 3.2.

Otras opciones alternativas que hemos discutido en la creación del esquema E/R:
- Meter en vuelo directamente el atributo cancelado.

1.2. Esquema relacional correspondiente al E/R y su normalización



El modelo relacional de nuestra base de datos quedaría de la siguiente forma:

Podemos ver que, de primeras, este esquema de modelo relacional parece estar normalizado.
Ahora bien, este ha de estar en la Forma Normal de Boyce-Codd, entonces vamos a ver si hay que modificar algo para que se cumpla:

- Primera Forma Normal: Cada uno de los atributos son valores atómicos, es decir, no hay valores multivaluados. Además, cada entidad es única mediante sus respectivas claves primarias. Ambas premisas se cumplen en nuestra base de datos.
- Segunda Forma Normal: 1FN + No hay dependencias funcionales con parte de la clave. Esto se cumple en nuestra base de datos ya que no existe ninguna entidad cuya clave primaria dependa de tan solo una parte de la clave primaria de otra entidad.
- Tercera Forma Normal: 2FN + No hay dependencias funcionales transitivas. Esto se cumple en nuestra base de datos ya que en ninguna tabla existen atributos que dependen de otro atributo que no es clave primaria. Este sería el caso si, por ejemplo, en vez de tener la entidad estadio, guardamos en la tabla de Equipo, en cada entrada de equipo los datos de su estadio, potencialmente repitiendo la información en la base de datos.
- Forma Normal de Boyce-Codd: Todas las dependencias funcionales dependen de la clave. Esto se cumple en nuestra base de datos, ya que todas las entidades cuya clave depende de la clave de otra entidad, siempre dependen de la primaria entera, ni no primaria ni de forma parcial.

Como hemos visto, nuestro esquema relacional ya está normalizado, por ello no necesitaremos realizar una nueva versión del mismo.

Por comodidad a la hora de manejar la base de datos, hemos creado en esta una clave artificial para la tabla “Vuelo” ya que su clave primaria está compuesta de muchos otros atributos, algunos de ellos claves ajenas.

1.3. Sentencias SQL de creación de tablas

```
/* Creacion de las tablas */
CREATE TABLE Aeropuerto (
        IATA                            VARCHAR(7) PRIMARY KEY,
        pais                            VARCHAR(30),
        estado                          VARCHAR(30),
        ciudad                          VARCHAR(30),
        nombreAeropuerto                VARCHAR(50)
)
PARTITION BY LIST (estado) (
    PARTITION p_alaska_california VALUES ('AK', 'CA'),
    PARTITION p_otros VALUES (DEFAULT)
);

CREATE TABLE Compania (
        nombreCompania                  VARCHAR(100) PRIMARY KEY
);

CREATE TABLE Avion (
        numeroMatricula         VARCHAR(30) PRIMARY KEY,
        fabricante                      VARCHAR(30),
        tipoMotor                       VARCHAR(30),
        annoCompra                      NUMBER,
        modelo                          VARCHAR(30)
);

CREATE SEQUENCE idVuelo START WITH 1 INCREMENT BY 1;

CREATE TABLE Vuelo (
        numeroVuelo                     NUMBER,
        horaSalida                      NUMBER,
        horaLLegada                     NUMBER,
        fechaSalida                     VARCHAR(10),
        nombreCompania                  VARCHAR(100),
        origen                          VARCHAR(7),
        destino                         VARCHAR(7),
        matriculaAvion                  VARCHAR(30),
        cancelado                       NUMBER(1,0), -- Usado como booleano: 0 es false y 1 es true
        FOREIGN KEY (nombreCompania) REFERENCES Compania(nombreCompania),
        FOREIGN KEY (origen) REFERENCES Aeropuerto(IATA),
        FOREIGN KEY (destino) REFERENCES Aeropuerto(IATA),
        FOREIGN KEY (matriculaAvion) REFERENCES Avion(numeroMatricula),
        id                              NUMBER PRIMARY KEY
);

CREATE TABLE CambioDeRuta (
        id                              NUMBER PRIMARY KEY,
        IATAdesvio                      VARCHAR(7),
        FOREIGN KEY (id) REFERENCES Vuelo(id)
);

CREATE TABLE Retraso (
        id                              NUMBER PRIMARY KEY,
        causa                           VARCHAR(50),
        duracion                        NUMBER,
        FOREIGN KEY (id) REFERENCES Vuelo(id)
);
```

Parte 2: Introducción de datos y ejecución de consultas

2.1. Población de la base de datos

Creamos el script de poblarT.sql, en el que transportamos los datos necesarios de cada columna de la tabla datosdb.datosvuelos a sus columnas correspondientes en nuestras tablas.

Vamos a desglosar las distintas decisiones tomadas para poblar cada una de nuestras tablas:

- Compania y Avion:
  Tomamos directamente las columnas de la tabla mediante SELECT DISTINCT (evitando así duplicados), asegurándonos de que los PRIMARY KEY no sean NULL (la obtención de los atributos de esta forma se repite en el resto de tablas, dejando de lado las peculiaridades de cada una de las demás).

- CambioDeRuta:
  Seleccionamos los distintos cambios de ruta que encontramos en los mismos vuelos de las tabla de Vuelo y la temporal.

- Vuelo:
  Como vuelo no tiene un atributo identificador primario, hemos tenido que crear un atributo artificial, id, para poder facilitar la población de la tabla. De no hacer esto, la clave primaria de cada vuelo hubiera estado formada por un total de 5 atributos distintos, haciéndola muy ineficiente.

- Aeropuerto:
  Poblamos una tabla con los aeropuertos de origen y otra con los aeropuertos de destino y mediante UNION nos aseguramos de obtener todos los aeropuertos disponibles en nuestra tabla temporal (sin repetidos).

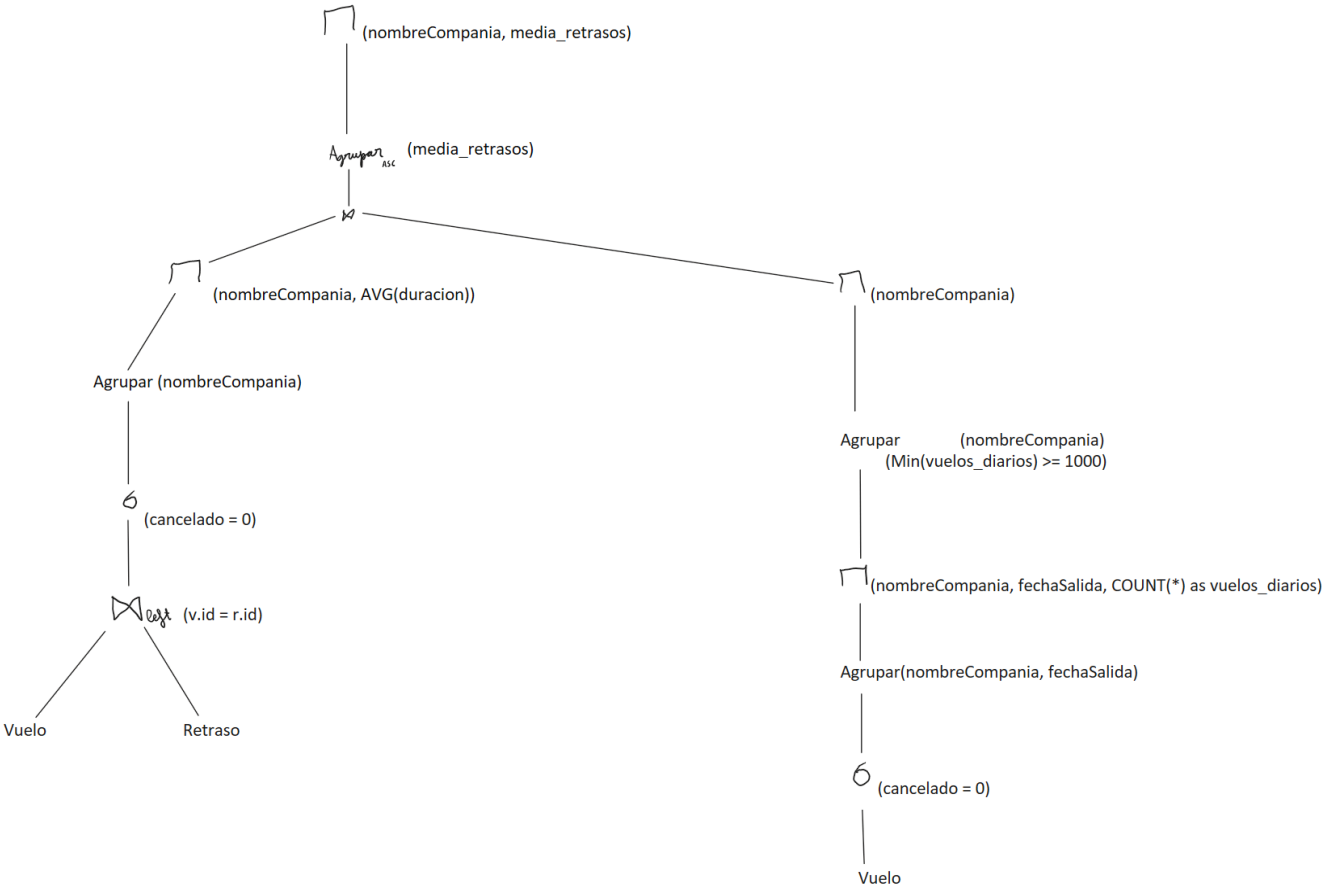
- Retraso:
  Poblamos distintas tablas según la causa de retraso, asegurándonos de que el delay correspondiente sea mayor de 0, y unimos todas las tablas en una sola mediante UNION (además, en todo momento nos aseguramos de tener los mismos vuelos de las tabla de Vuelo y la temporal).

Los problemas que hemos encontrado durante la población han sido:
- Creación de una clave artificial que ayude a identificar los distintos vuelos.
- Enfoque de cómo poblar la tabla Vuelo.
- La dificultad para afrontar la unión de datos comunes en tablas como Aeropuerto o Retraso.

2.2. Consultas SQL

2.2.1. Primera consulta

Árbol sintáctico:



Consulta SQL:

```
CREATE MATERIALIZED VIEW MediaRetrasos AS
SELECT v.nombreCompania, AVG(r.duracion) AS media_retrasos
FROM Vuelo v
LEFT JOIN Retraso r ON v.id = r.id
WHERE v.cancelado = 0
GROUP BY v.nombreCompania;
WITH VuelosDiarios AS (
	SELECT nombreCompania, fechaSalida, COUNT(*) AS vuelos_diarios
	FROM Vuelo
	WHERE cancelado = 0
	GROUP BY nombreCompania, fechaSalida
),
CompaniasConMin1000Vuelos AS (
	SELECT nombreCompania
	FROM VuelosDiarios
	GROUP BY nombreCompania
	HAVING MIN(vuelos_diarios) >= 1000
)
```

```
SELECT m.nombreCompania, m.media_retrasos
FROM MediaRetrasos m
JOIN CompaniasConMin1000Vuelos c ON m.nombreCompania = c.nombreCompania
ORDER BY m.media_retrasos ASC;
```

Tanto en esta consulta como en las siguientes hemos utilizado CTEs para simplificar la sintaxis de las consultas. En este caso, necesitamos seleccionar el nombre de la compañía junto con su media de retrasos del JOIN entre el CTE que calcula la media de retrasos por compañía y el que calcula las compañías que tienen al menos 1000 vuelos. Ordenando este resultado en orden ascendente, de menor número de retrasos para arriba, nos da las compañías con al menos 1000 vuelos ordenadas de más a menos puntual. El CTE MediaRetrasos, nos da la media de retrasos de cada compañía, y CompaniasConMin1000Vuelos nos da las compañías con al menos 1000 vuelos cada día. Al mismo tiempo, el segundo de estos CTEs utiliza el CTE VuelosDiarios que devuelve el número de vuelos de cada compañía por día.

Respuesta obtenida:

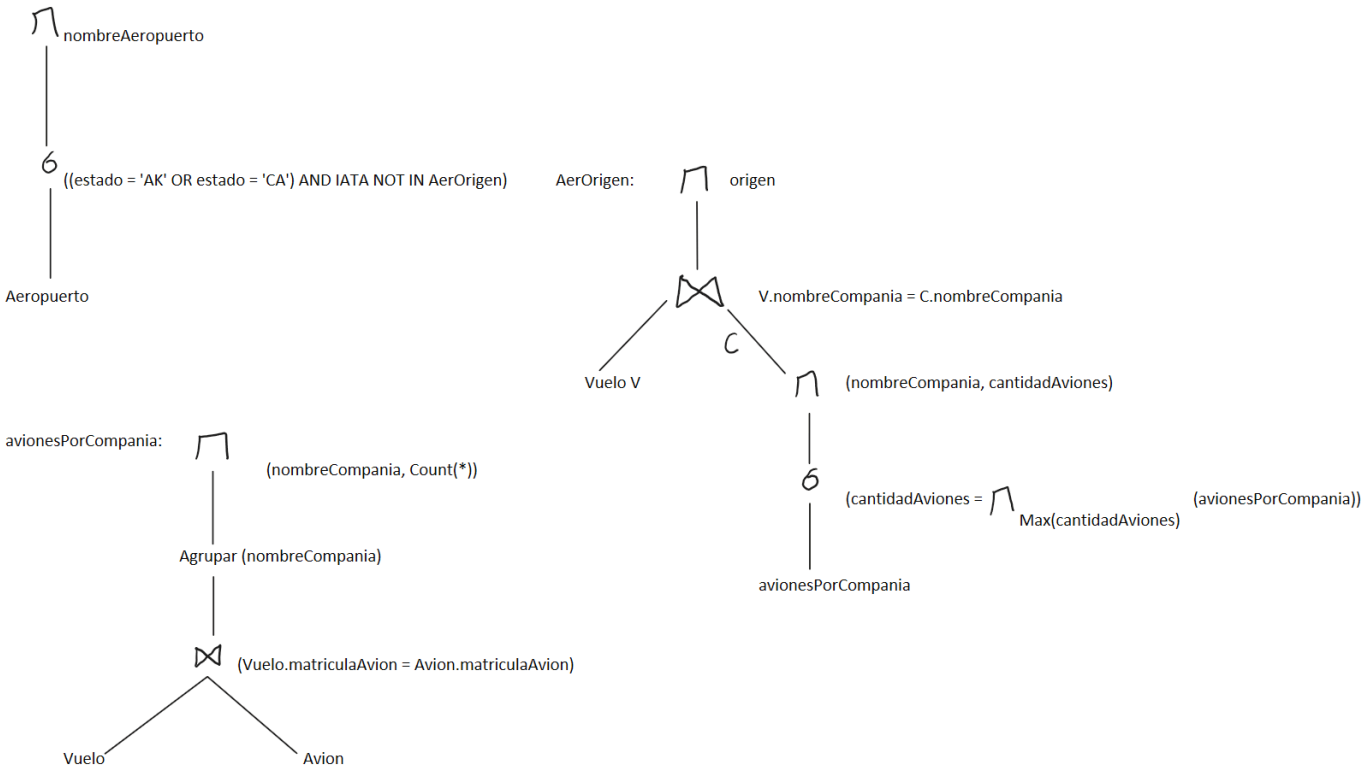
```
NOMBRECOMPANIA
--------------------------------------------------------------------------------
MEDIA_RETRASOS
--------------
Southwest Airlines Co.
    24,1971104

American Airlines Inc.
    33,0696203

Skywest Airlines Inc.
    39,4129301
```

2.2.2. Segunda consulta

Árbol sintáctico:



Consulta SQL:

```
WITH avionesPorCompania AS (
    SELECT nombreCompania, COUNT(*) AS cantidadAviones
    FROM Vuelo
    JOIN Avion ON Vuelo.matriculaAvion = Avion.numeroMatricula
    GROUP BY Vuelo.nombreCompania
),
CompaniaConMasAviones AS (
    SELECT avionesPorCompania.nombreCompania, avionesPorCompania.cantidadAviones
    FROM avionesPorCompania
    WHERE avionesPorCompania.cantidadAviones = (SELECT MAX(cantidadAviones) FROM
avionesPorCompania)
)
SELECT A.nombreAeropuerto
FROM Aeropuerto A
WHERE (A.estado = 'AK' OR A.estado = 'CA')
AND A.IATA NOT IN (
    SELECT V.origen
    FROM Vuelo V
    JOIN CompaniaConMasAviones C ON V.nombreCompania = C.nombreCompania
);
```

En esta consulta utilizamos dos CTEs CompaniaConMasAviones y avionesPorCompania, cuyos nombres son indicativo de que datos devuelven. El segundo CTE nos devuelve el número de aviones que tiene cada compañía y luego CompaniaConMasAviones calcula el mayor valor de este resultado.

Así, solo queda seleccionar los aeropuertos que estén en Alaska o California cuyos códigos IATA no estén como origen o destino de los vuelos en los que opera la compañía con más aviones.

Respuesta obtenida:

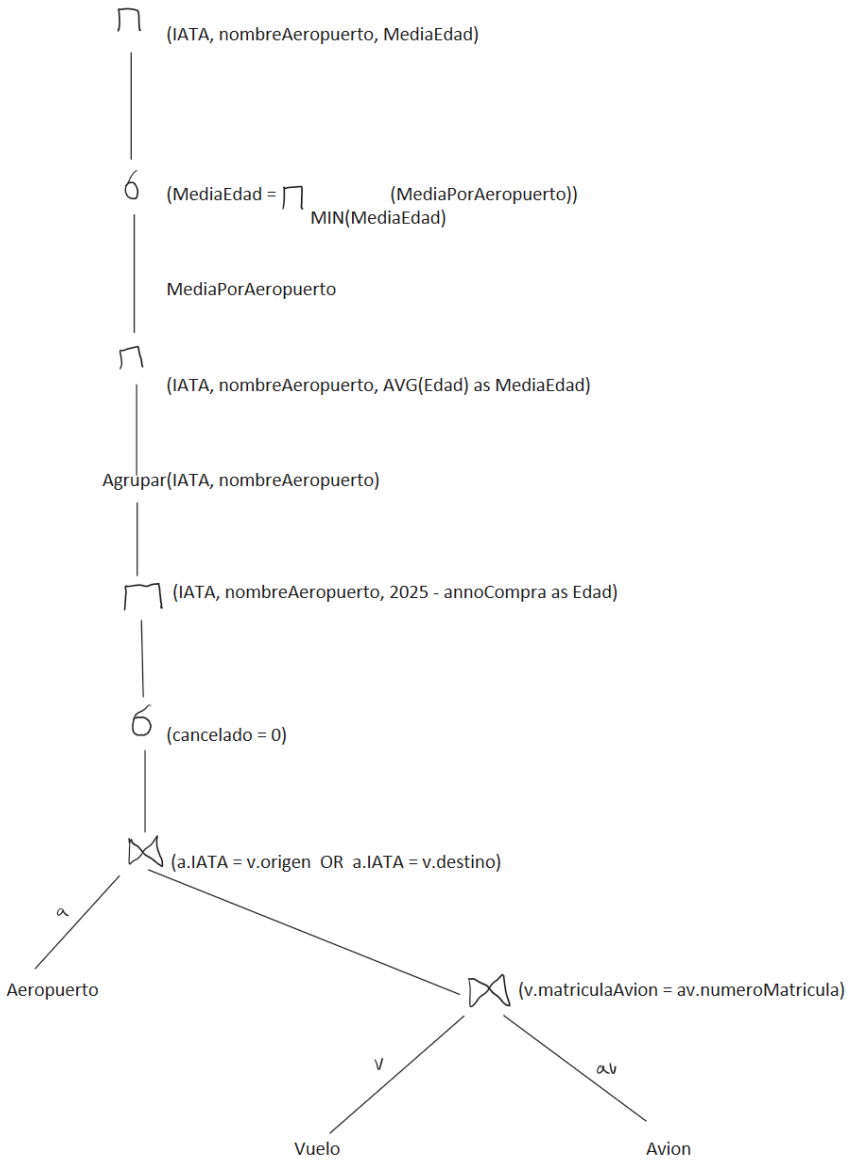
```
NOMBREAEROPUERTO
--------------------------------------------------
Arcata
Adak
Kodiak
Ted Stevens Anchorage International
Bethel
Meadows
Wiley Post Will Rogers Memorial
Merle K (Mudhole) Smith
Jack McNamara
Chico Municipal
Fairbanks International

NOMBREAEROPUERTO
--------------------------------------------------
Fresno Yosemite International
Imperial County
Inyokern
Juneau International
Ketchikan International
Long Beach (Daugherty )
Modesto City-County-Harry Sham
Monterey Peninsula
Nome
Ralph Wien Memorial
Oxnard

NOMBREAEROPUERTO
--------------------------------------------------
Palmdale Production Flight
James C. Johnson Petersburg
Palm Springs International
Redding Municipal
Santa Barbara Municipal
San Luis Obispo Co-McChesney
Deadhorse
Sitka
Santa Maria Pub/Capt G Allan Hancock
Wrangell
Yakutat
```

2.2.3. Tercera consulta

Árbol sintáctico:



Consulta SQL:

```
WITH EdadesPorVuelo AS (
    SELECT a.IATA, a.nombreAeropuerto, 2025 - av.annoCompra AS Edad
    FROM Aeropuerto a
    JOIN Vuelo v ON (a.IATA = v.origen OR a.IATA = v.destino)
    JOIN Avion av ON v.matriculaAvion = av.numeroMatricula
    WHERE v.cancelado = 0
),
MediaPorAeropuerto AS (
    SELECT IATA, nombreAeropuerto, AVG(Edad) AS MediaEdad
    FROM EdadesPorVuelo
    GROUP BY IATA, nombreAeropuerto
)
```

```
SELECT IATA AS CodigoAeropuerto, nombreAeropuerto AS NombreAeropuerto, MediaEdad
FROM MediaPorAeropuerto
WHERE MediaEdad = (SELECT MIN(MediaEdad) FROM MediaPorAeropuerto);
```

En esta consulta utilizamos los CTEs EdadesPorVuelo y MediaPorAeropuerto, el primero nos devuelve las distintas edades de los aviones que operan en un aeropuerto, estos datos los utiliza el segundo CTE para agruparlos por IATA y nombre de aeropuerto, sacando la media de edad de los aviones que operan en cada aeropuerto. Por último, solo queda seleccionar la fila cuyo valor de la media de edad sea menor.

Respuesta obtenida:

```
CODIGOA NOMBREAEROPUERTO                                      MEDIAEDAD
------- ---------------------------------------------------- ----------
HDN     Yampa Valley                                            20,9375
```

Parte 3: Diseño Físico

3.1. Rendimiento de las consultas SQL

Al aplicar los comandos

EXPLAIN PLAN FOR <consulta SQL>

SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY())

sobre las 3 diferentes consultas...

Veamos los resultados obtenidos en cuanto a tiempo y porcentaje de uso de CPU:

	Primera consulta	Segunda consulta	Tercera consulta
Porcentaje de CPU*	3%	1%	1%
Coste	178	608	947
Tiempo de ejecución*	00:00:01	00:00:01	00:00:01

* *Los valores son aproximados.*

Se nos ocurren las siguientes modificaciones:

1. La creación de una vista materializada.
2. La creación de particiones en las tablas.
3. La utilización de índices.
4. Quitar la unión en la segunda consulta.

De estas 3 opciones hemos empleado la primera en la consulta 1, reduciendo el coste a diez veces menos, y la segunda en la consulta 2 reduciendo el coste en 40. La 4ª modificación ha sido llevada a cabo porque observamos un mismo resultado, sin importar que esta unión esté o no, por lo que ha sido eliminada para mejorar el coste de la consulta.

Vista materializada de la consulta 1:

```
CREATE MATERIALIZED VIEW MediaRetrasos AS
SELECT v.nombreCompania, AVG(r.duracion) AS media_retrasos
FROM Vuelo v
LEFT JOIN Retraso r ON v.id = r.id
WHERE v.cancelado = 0
GROUP BY v.nombreCompania;
```

Partición de la consulta 2 en la tabla Aeropuerto:

```
CREATE TABLE Aeropuerto (
        IATA                            VARCHAR(7) PRIMARY KEY,
        pais                            VARCHAR(30),
        estado                          VARCHAR(30),
        ciudad                          VARCHAR(30),
        nombreAeropuerto                VARCHAR(50)
)
PARTITION BY LIST (estado) (
    PARTITION p_alaska_california VALUES ('AK', 'CA'),
    PARTITION p_otros VALUES (DEFAULT)
);
```

3.2. Restricciones solucionadas mediantes triggers

Algunas de las restricciones que podemos encontrar son:

- ➢ Un vuelo no puede salir del mismo aeropuerto al que debe llegar.
- ➢ Un vuelo no puede tener un horario de llegada posterior al horario de salida.
- ➢ La duración de un retraso tiene que ser mayor que 0.
- ➢ Un vuelo no puede tener un horario de llegada posterior al horario de salida.
- ➢ El año de fabricación del avión debe ser en una fecha pasada.
- ➢ No se puede borrar un aeropuerto si tiene vuelos programados de ida o vuelta en un futuro.
- ➢ No se puede hacer un cambio de ruta hacia el mismo destino que en un principio.
- ➢ Si se borra un vuelo hay que borrar: cambios de ruta, cancelaciones y retrasos de dicho vuelo.
- ➢ No se pueden añadir cambios de ruta ni desvíos si cancelamos un vuelo.
- ➢ Cuando se añade un vuelo con un avión y/o compañía que no existen, se añaden.

3.2.1. Primer trigger

Inserción de un vuelo en el que el origen y el destino es el mismo:

```
CREATE OR REPLACE TRIGGER mismoOrigenYDestino
BEFORE INSERT OR UPDATE ON Vuelo
FOR EACH ROW
WHEN (NEW.origen = NEW.destino)
BEGIN
    RAISE_APPLICATION_ERROR(-20001, 'Un vuelo no puede tener origen y destino
iguales.');
END;
/
```

Nuestro primer trigger se encarga de uno de los errores más obvios que puede provocar un usuario, y que por ello es bueno cubrirlo. El error ocurre cuando se intenta introducir un vuelo en el que el origen y el destino del vuelo es el mismo, algo que no debe ocurrir en una agencia de viajes.

Acerca del funcionamiento del trigger, este es muy básico: el trigger se activa antes de una inserción en la tabla Vuelo (línea 2), y se lleva a cabo el código que tiene cuando el nombre del origen del vuelo que intentas introducir coincide con el nombre del destino de ese mismo vuelo. El código del trigger simplemente imprime el error “Un vuelo no puede tener origen y destino iguales.” por pantalla, además de impedir la inserción de dicho vuelo.

3.2.2. Segundo trigger

Eliminar todos los retrasos y cambios de ruta asociados a un vuelo al borrar dicho vuelo:

```
CREATE OR REPLACE TRIGGER limpiarIncidenciasVuelo
BEFORE DELETE ON Vuelo
FOR EACH ROW
BEGIN
    DELETE FROM Retraso r WHERE r.id = :NEW.id;
    DELETE FROM CambioDeRuta cr WHERE cr.id = :NEW.id;
END;
/
```

Si borramos un vuelo directamente, no se borrarán los cambios de ruta y retrasos asociados a ese mismo vuelo en otras tablas. Por ello, para que no haya datos sin conexión, si borramos un vuelo, borramos todos los cambios de ruta y retrasos asociados al mismo.

3.2.3. Tercer trigger

No permitir actualizar un vuelo cancelado:

```
CREATE OR REPLACE TRIGGER noActualizarVueloCancelado
BEFORE UPDATE ON Vuelo
FOR EACH ROW
WHEN (OLD.cancelado = 1)
BEGIN
    RAISE_APPLICATION_ERROR(-20002, 'No puedes actualizar un vuelo cancelado');
END;
/
```

Este trigger impide actualizar los valores de un vuelo ya cancelado. Para ello, antes de una actualización de un vuelo mira si el atributo ‘cancelado’ de dicho vuelo es igual a 1. Si es así (el vuelo está cancelado) salta un mensaje de error y no realiza la actualización.

Impedir insertar un vuelo en Cambios de ruta o Retrasos:

```
CREATE OR REPLACE TRIGGER noInsertarRetrasoEnCancelado
BEFORE INSERT ON Retraso
FOR EACH ROW
DECLARE
    vueloExiste NUMBER;
    vueloCancelado NUMBER;
BEGIN
    SELECT COUNT(*) INTO vueloExiste FROM Vuelo WHERE id = :NEW.id;
    IF vueloExiste = 0 THEN
```

```
        RAISE_APPLICATION_ERROR(-20003, 'No puedes insertar un retraso a un vuelo
que no existe');
    ELSE
        SELECT cancelado INTO vueloCancelado FROM Vuelo WHERE id = :NEW.id;
        IF vueloCancelado = 1 THEN
            RAISE_APPLICATION_ERROR(-20004, 'No puedes insertar un retraso a un
vuelo cancelado');
        END IF;
    END IF;
END;
/

CREATE OR REPLACE TRIGGER noInsertarCambioDeRutaEnCancelado
BEFORE INSERT ON CambioDeRuta
FOR EACH ROW
DECLARE
    vueloExiste NUMBER;
    vueloCancelado NUMBER;
BEGIN
    SELECT COUNT(*) INTO vueloExiste FROM Vuelo WHERE id = :NEW.id;
    IF vueloExiste = 0 THEN
        RAISE_APPLICATION_ERROR(-20003, 'No puedes insertar un cambio de ruta a un
vuelo que no existe');
    ELSE
        SELECT cancelado INTO vueloCancelado FROM Vuelo WHERE id = :NEW.id;
        IF vueloCancelado = 1 THEN
            RAISE_APPLICATION_ERROR(-20004, 'No puedes insertar un cambio de ruta a
un vuelo cancelado');
        END IF;
    END IF;
END;
/
```

Estos dos triggers se encargan de impedir las inserciones de retrasos o cambios de ruta a un vuelo ya cancelado. Los trigger se diferencian por la tabla en la que afectan y el nombre que tienen.

En cuanto a su funcionamiento, antes de una inserción de un retraso o un cambio de ruta, comprueba si el vuelo objetivo existe. En caso de que no exista, salta un mensaje de error; si este sí existe, comprueba si está cancelado o no. De nuevo, si este está cancelado salta un mensaje de error y en caso contrario no hace nada.

Organización del grupo

Nos organizamos de manera que todos los integrantes del grupo nos dedicamos a un mismo punto, cada uno creando una parte del apartado. En caso de que uno tuviese problemas, nos reunimos con el objetivo de solucionarlo en conjunto.

El trabajo en grupo ha sido bastante efectivo y ecuánime.

Tareas realizadas	Miembros del grupo implicados	Horas dedicadas	Problemas observados
Esquema E/R global, normalización y listado de las restricciones.	Todos los miembros del grupo.	3 horas	Problemas relacionados con qué atributos usar en nuestra base.
Esquema relacional.	Todos los miembros del grupo.	1 hora	Sin problemas.
Creación de la base de datos.	Todos los miembros del grupo.	2 horas	Creación de una clave artificial idVuelo.
Población de la base de datos.	Todos los miembros del grupo.	4,5 horas	Dificultades para poblar Vuelo.
Creación y pruebas de las consultas SQL.	Todos los miembros del grupo.	8 horas	Sin problemas.
Árboles relacionales de las consultas anteriores.	Todos los miembros del grupo.	2 horas	Sin problemas.
Rendimiento de las consultas.	Todos los miembros del grupo.	1,5 horas	Sin problemas.
Triggers.	Todos los miembros del grupo.	4 horas	En un principio, se nos hizo difícil encontrar triggers diferentes y con cierta dificultad.
Creación de la memoria.	Todos los miembros del grupo.	4,5 horas	Sin problemas.