BASES DE DATOS: PRACTICA 3

CURSO 2022-2023

17 de Abril de 2023



**Grupo: T9**

Alejandro Benedi Andrés - [843826@unizar.es](mailto:843826@unizar.es)

Javier Julve Yubero - [840710@unizar.es](mailto:840710@unizar.es)

Álvaro de Francisco Nievas - [838819@unizar.es](mailto:838819@unizar.es)

PARTE 1: Creación de una base de datos

# Esquema entidad / relación.

# 

### Entidad *Avión*:

* Un avión solo puede pertenecer a una aerolínea.
* Un avión puede hacer varios vuelos diarios.
* El nombre del modelo del avión es único.

### Entidad *Modelo*:

* El nombre del modelo es único.
* Un modelo corresponde a muchos aviones.

### Entidad *Vuelos*:

* El número de vuelo es un identificador del trayecto del avión, no del vuelo en sí.
* El aeropuerto destino de un vuelo es el aeropuerto donde se prevé que ese vuelo va a aterrizar. Si este vuelo se desvía no se verá reflejado en esta relación.
* La hora de llegada y fecha de llegada de un vuelo es la hora y fecha de cuando se prevé que ese vuelo vaya a aterrizar. Si este vuelo se retrasa no se verá reflejado en esta entidad.
* A un vuelo le pueden corresponder 0 o muchas incidencias.
* Un vuelo será realizado por un único avión.

### Entidad *Aeropuerto*

* El nombre del aeropuerto y el código IATA de este son únicos.
* Se ha considerado la posibilidad de que haya 2 ciudades con el mismo nombre en diferentes estados. Esto impide crear una nueva entidad *Ciudad* con estado como atributo.

### Entidad *Incidencia*

* Un vuelo puede tener más de una incidencia, una incidencia corresponde a un único vuelo.
* La entidad Incidencia tiene como especialización a su vez tres tipos de incidencia.

### Entidad *Retraso*

* Un retraso tendrá una única causa y duración.
* Cada vuelo tiene uno por cada causa de retraso asociada a este. La suma del tiempo retrasado de un vuelo será la suma de la duración de sus retrasos.

### Entidad *Desvío*

* En el caso de desvío, un vuelo podrá ser remitido a uno o más aeropuertos distintos.

### Entidad *Cancelación*

* Esta entidad no almacena atributos.

## **Soluciones Alternativas:**

Valoramos crear una **relación *Aeropuerto-Aerolínea***, sin embargo esta relación sería redundante ya que podríamos este mismo dato mediante la entidad vuelos. Priorizamos la **relación *Aerolínea-Vuelos*** ya que no podemos obtener este dato de otra forma.

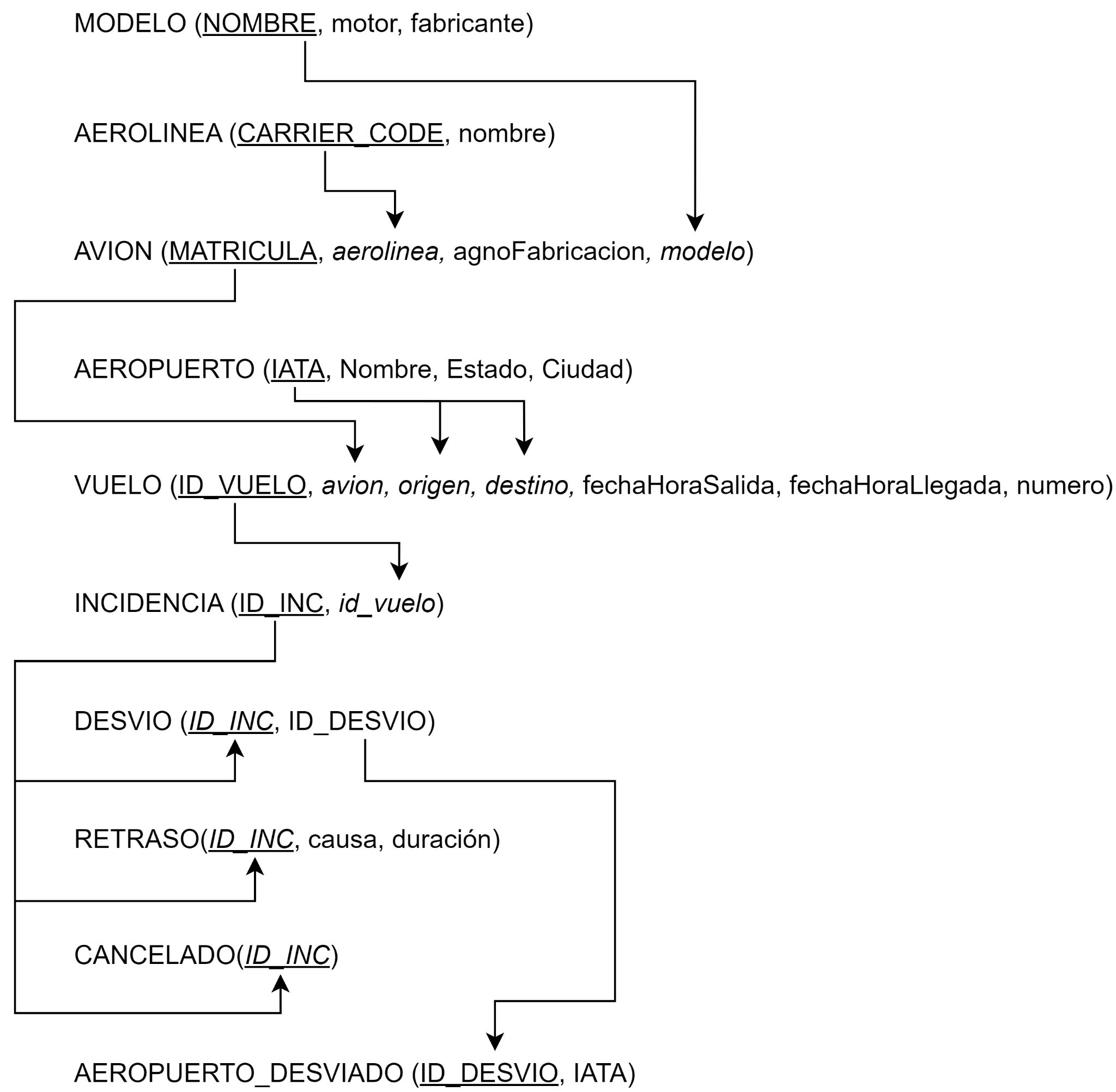
Aunque el nombre de la aerolínea funciona como clave primaria por ser único, no es eficiente como clave extranjera por tener muchos caracteres. Por tanto, optamos por usar *CarrierCode.*

Una alternativa a nuestra relación Cancelación sería un atributo sobre la entidad Vuelo que indique si este ha sido cancelado o no. No obstante nos decidimos por hacerla entidad por diversos motivos:

1. **Uniformidad** con el resto de incidencias posibles sobre un vuelo, que sí necesitan una entidad.
2. **Mantenibilidad** de la BBDD. En los datos proporcionados las cancelaciones tenían motivo. No lo guardamos por no necesitarse ni pedirse explícitamente en el enunciado, pero en un futuro podría necesitar e incluir este cambio sería más fácil de esta manera.

También la entidad Desvío podría ser una relación directa del vuelo con los aeropuertos, pero por los motivos explicados anteriormente optamos por hacerlo una entidad.

# **Esquema relacional.**

Tras una traducción del modelo Entidad-Relación obtenemos el siguiente modelo relacional

## Normalización.

Para la elección de claves primarias hemos encontrado en la mayoría de tablas un atributo único y que sabíamos que no se repetiría, como CarrierCode en AEROLINEA, la matrícula en AVION o el IATA de un aeropuerto. En todos estos los códigos son únicos e irrepetibles para cada uno, con lo que eran candidatos idóneos. Otras claves como la de VUELO, INCIDENCIA o DESVIO se han sacado de forma artificial ya que no encontrábamos ningún atributo candidato para ello. Atributos como el número de vuelo se repite al pasar el día, las fechas también o la propia matrícula, pues un avión vuela más de una vez en su vida útil. Lo mismo para INCIDENCIA y DESVIO, ya que en esta última debemos identificar los IATA de desvío con el propio número de la incidencia.

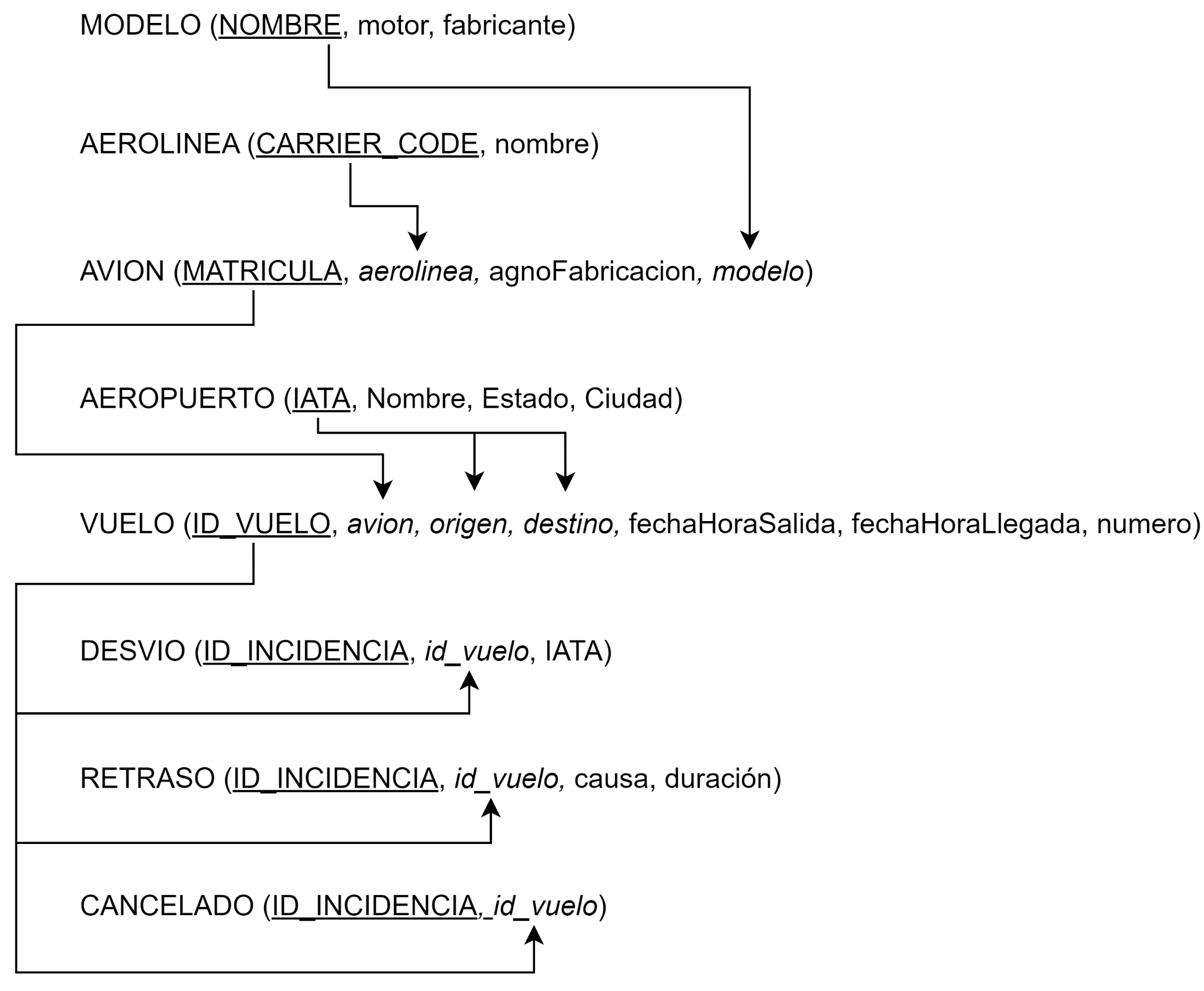
Todas las dependencias se basan en la clave primaria, por lo tanto el siguiente modelo relacional cumple con la forma normal de Boyce-Codd (FNBC).

El esquema cumple también con la primera forma normal (1FN) y segunda forma normal (2FN), por no haber dependencias funcionales que involucren ninguna parte de la clave. Además, el hecho de que no haya dependencias funcionales transitivas implica que cumple con la tercera forma normal (3FN).

## Desnormalización.

Para este esquema hemos tomado la decisión de desnormalizar algunas partes de este. Los motivos de esta decisión han sido:

1. Eliminar JOINs que ralentizan las consultas sobre la BBDD.
2. Evitar tablas con pocos atributos y de carácter transitivo, lo que facilita la comprensión de las consultas, inserciones y de las tablas en sí mismas.



La parte del esquema ER que ha sufrido los cambios está relacionada con la gestión de las INCIDENCIAS. Hemos eliminado la tabla INCIDENCIA y ahora guardamos los identificadores de vuelo en las tablas correspondientes a especializaciones de la entidad. Para restar una tabla en la relación desvío, ya que esta entidad no tiene atributos extra como la causa o momento del desvío; cada incidencia corresponderá a un desvío de los muchos que puede sufrir un vuelo. Aunque conceptualmente será una única incidencia, almacenaremos una entrada por aeropuerto al que sea desviado.

# Creación de tablas.

Creamos las siguientes 8 tablas: aerolínea, modelo, avión, aeropuerto, vuelo, cancelado y retrasado. Añadimos cláusulas ON DELETE CASCADE en las tablas de incidencia para que al borrarse un vuelo, se borren sus incidencias (cancelación, desvíos o retrasos). Además, introducimos comprobaciones para asegurar que el año de fabricación es mayor que 1903 (construcción del primer avión de la historia), la fecha y hora de salida es menor que la fecha y hora de llegada, que el aeropuerto origen y destino de un vuelo son distintos y que el tiempo de retraso es mayor que 0.

CREATE TABLE AEROLINEA (

CARRIER\_CODE VARCHAR(6) PRIMARY KEY,

NOMBRE VARCHAR(150));

CREATE TABLE MODELO (

NOM\_MODELO VARCHAR(50) PRIMARY KEY,

FABRICANTE VARCHAR(50),

TIPO\_MOTOR VARCHAR(50));

CREATE TABLE AVION (

MATRICULA VARCHAR(10) PRIMARY KEY,

AEROLINEA VARCHAR(6) REFERENCES AEROLINEA(CARRIER\_CODE),

MODELO VARCHAR(50) REFERENCES MODELO(NOM\_MODELO),

AGNO\_FABRICACION NUMBER(4),

CONSTRAINT CHK\_AGNO\_FABRICACION CHECK (AGNO\_FABRICACION > 1903));

CREATE TABLE AEROPUERTO (

IATA VARCHAR(3) PRIMARY KEY,

NOMBRE VARCHAR(50),

CIUDAD VARCHAR(50),

ESTADO VARCHAR(50));

CREATE TABLE VUELO (

ID\_VUELO VARCHAR(15) PRIMARY KEY,

NUMERO VARCHAR(10),

AVION VARCHAR(10) REFERENCES AVION(MATRICULA),

ORIGEN VARCHAR(3) REFERENCES AEROPUERTO(IATA),

DESTINO VARCHAR(3) REFERENCES AEROPUERTO(IATA),

SALIDA DATE,

LLEGADA DATE,

CONSTRAINT CHK\_FECHA CHECK (SALIDA <= LLEGADA),

CONSTRAINT CHK\_AEROPUERTOS\_DISTINTOS CHECK (ORIGEN <> DESTINO));

CREATE TABLE CANCELADO (

ID\_INCIDENCIA VARCHAR(15) PRIMARY KEY,

ID\_VUELO VARCHAR(15) REFERENCES VUELO(ID\_VUELO) ON DELETE CASCADE);

CREATE TABLE RETRASO (

ID\_INCIDENCIA VARCHAR(15) PRIMARY KEY,

ID\_VUELO VARCHAR(15) REFERENCES VUELO(ID\_VUELO) ON DELETE CASCADE,

CAUSA VARCHAR(50),

TIEMPO\_RETRASO NUMBER(6),

CONSTRAINT CHK\_TIEMPO\_RETRASO CHECK (TIEMPO\_RETRASO > 0));

CREATE TABLE DESVIO (

ID\_INCIDENCIA VARCHAR(15) PRIMARY KEY,

ID\_VUELO VARCHAR(15) REFERENCES VUELO(ID\_VUELO) ON DELETE CASCADE,

IATA VARCHAR(3) REFERENCES AEROPUERTO(IATA));

PARTE 2: Introducción de datos y ejecución de consultas

# Población de la base de datos.

En el .csv de la práctica se encuentra toda la información necesaria para poblar, sin embargo toda ella está desordenada. Por tanto fue necesario hacer 8 ficheros .csv diferentes (uno por cada tabla) en los cuales cada columna .csv será un tipo de datos de la tabla SQL.

Una vez realizados estos ficheros .csv se ha utilizado la aplicación SQL \*Loader de Oracle (sqlldr2) para poblar toda la base de datos a partir de estos 8 csv. Es necesario implementar un fichero de control .ctl para cada fichero .csv, en los cuales se especifican el nombre del fichero .csv de donde se va a extraer los datos, el nombre de la tabla destino, el carácter que se separa los datos en el fichero csv (en este caso separador es ‘;’ ) y los datos que se van a introducir (columnas de la tabla ).

Aunque este proceso no ha costado demasiado, no hemos tenido que enfrentar a los siguientes problemas:

* La dificultad principal de este proceso fue **entender el fichero** csv de la práctica. Este fichero tenía un gran número de columnas “innecesarias” que no aportan ningún tipo de información relevante para nuestra base de datos. Por ello fue necesario eliminar todas estas columnas y centrarnos solo en las importantes.
* Para poblar la tabla VUELO es necesario saber la **fecha de llegada prevista** de cada vuelo. Sin embargo esta información no estaba en el fichero .csv entregado con el guión de la práctica, por tanto fue necesario calcularla. No es un cálculo muy complejo, pero para facilitar la población decidimos incorporar la fecha en el fichero .csv que usamos para poblar esta tabla.
* La realización de los ficheros csv que usamos para poblar las tablas RETRASOS, DESVÍOS Y CANCELACIÓN fue la tabla que más tiempo nos llevó. Esto fue debido al gran número de columnas muy parecidas “innecesarias” que tenía el csv.
* En el csv de la práctica había aviones con **modelo desconocido**. Para estos casos, en los cuales no se conoce el modelo, se ha insertado *“no model”* en vez de un modelo. Además se ha añadido una nueva tupla en la tabla MODELO que indica que no se tiene datos, ni del modelo, ni del motor, ni del fabricante.
* En el .csv de la práctica también había **vuelos sin avión**. Estos vuelos también han sido considerados como válidos y se han insertado en la tabla VUELOS, sin embargo, en estos casos, en vez de tener la matrícula de un avión el valor del atributo será NULL.

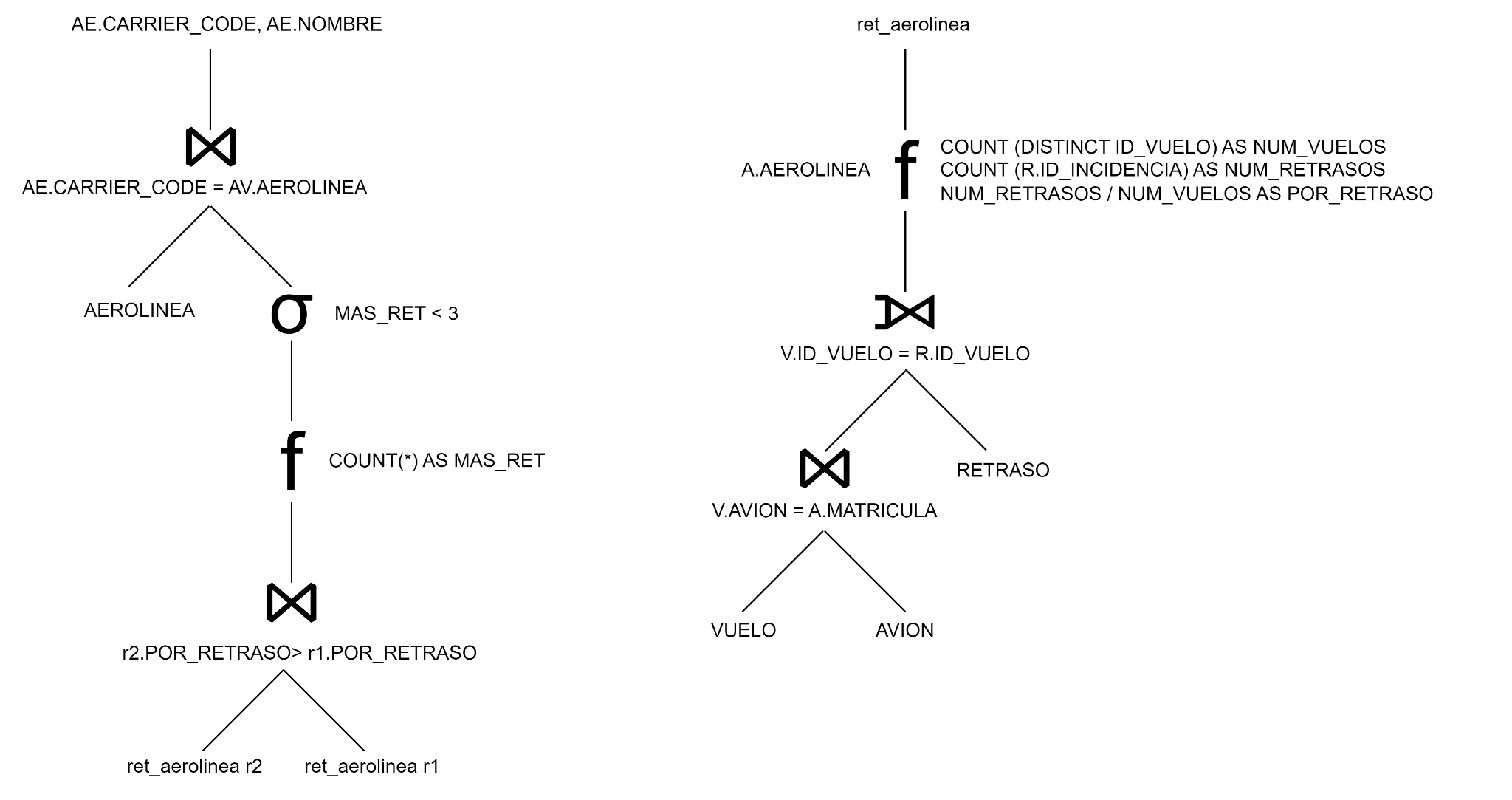
# 

# Consultas SQL.

## **1ª consulta.** Tres compañías aéreas menos puntuales (% vuelos retrasados).

Para esta consulta, primero obtenemos el número de vuelos realizados y cuantos de estos han sido retrasados y calculamos el % de vuelos retrasados por compañía. Una vez tengamos los resultados, seleccionamos las compañías que cumplen que el número de compañías con un porcentaje de retrasos mayor que ellas es menor que tres.

### **Árbol sintáctico.**



### **Consulta SQL.**

WITH ret\_aerolinea AS (

SELECT

AVION.AEROLINEA,

COUNT(DISTINCT VUELO.ID\_VUELO) AS NUM\_VUELOS,

COUNT(RETRASO.ID\_INCIDENCIA) AS NUM\_RETRASOS,

COUNT(RETRASO.ID\_INCIDENCIA)/COUNT(DISTINCT VUELO.ID\_VUELO) AS POR\_RETRASO

FROM VUELO

INNER JOIN AVION ON AVION.MATRICULA = VUELO.AVION

LEFT JOIN RETRASO ON RETRASO.ID\_VUELO = VUELO.ID\_VUELO

GROUP BY AVION.AEROLINEA

)

SELECT AEROLINEA, NOMBRE

FROM retrasos\_aerolineas r1

INNER JOIN AEROLINEA ON AEROLINEA.CARRIER\_CODE = r1.AEROLINEA

WHERE (

SELECT COUNT(\*)

FROM retrasos\_aerolineas r2

WHERE r2.POR\_RETRASO > r1.POR\_RETRASO

) < 3;

### **Resultado.**

| **AEROLÍNEA** | **NOMBRE** |
| --- | --- |
| AA | American Airlines Inc. |
| MQ | American Eagle Airlines Inc. |
| EV | Atlantic Southeast Airlines |

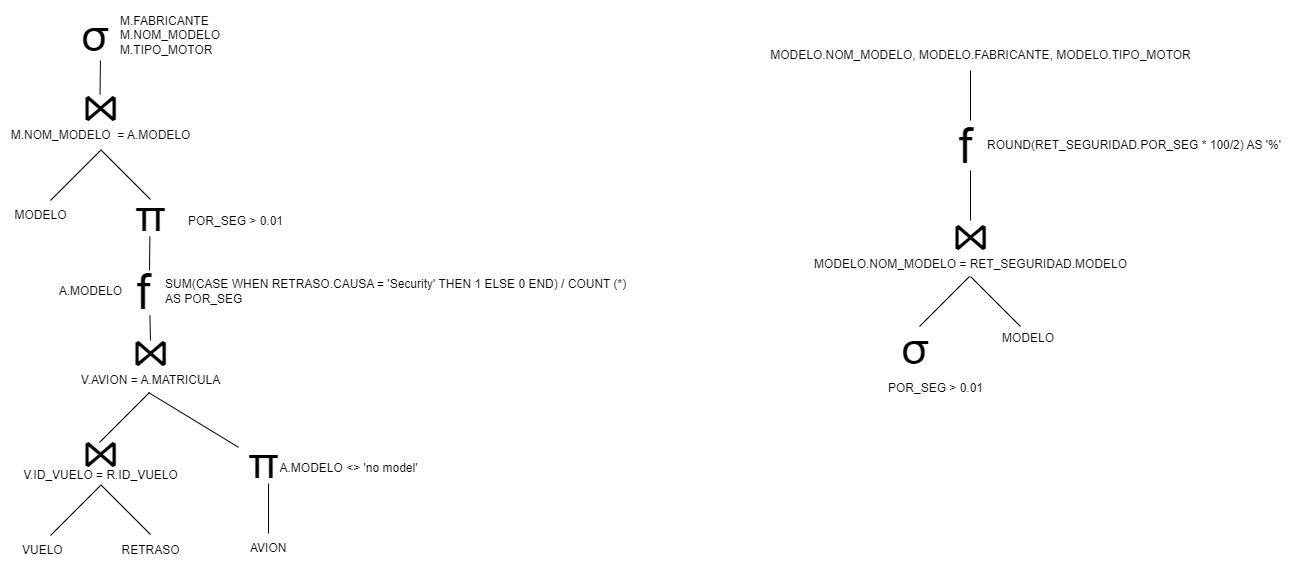
## 

## 

## **2ª consulta.** Lista los fabricantes, modelos y motores en los que el número de vuelos retrasados por motivos de seguridad sea mayor del 1% de su total de retrasos.

Para esta consulta primero comprobamos cuántos casos de retrasos son debido a seguridad y los dividimos entre los totales para obtener un porcentaje, descartando en el camino aquellos vuelos donde no se especifique el modelo del avión. Una vez con estos datos, averiguamos las tres cosas que se nos piden al ya saber el modelo y tan solo tener que buscar estos datos en la tabla MODELO y por último solo se muestran aquellos con un porcentaje mayor a 1 y en orden descendiente para que se pueda ver con mayor claridad.

## **Árbol sintáctico.**



### **Consulta SQL.**

WITH RET\_SEGURIDAD AS (

SELECT AVION.MODELO,

SUM(CASE WHEN RETRASO.CAUSA = 'Security' THEN 1 ELSE 0 END) / COUNT(\*) AS POR\_SEG

FROM RETRASO

INNER JOIN VUELO ON RETRASO.ID\_VUELO = VUELO.ID\_VUELO

INNER JOIN AVION ON AVION.MATRICULA = VUELO.AVION

WHERE AVION.MODELO <> 'no model'

GROUP BY AVION.MODELO

)

SELECT

ROUND(RET\_SEGURIDAD.POR\_SEG \* 100, 2) AS "%",

MODELO.NOM\_MODELO,MODELO.FABRICANTE,MODELO.TIPO\_MOTOR

FROM RET\_SEGURIDAD

INNER JOIN MODELO ON MODELO.NOM\_MODELO = RET\_SEGURIDAD.MODELO

WHERE POR\_SEG > 0.01

ORDER BY "%" DESC;

### **Resultado:**

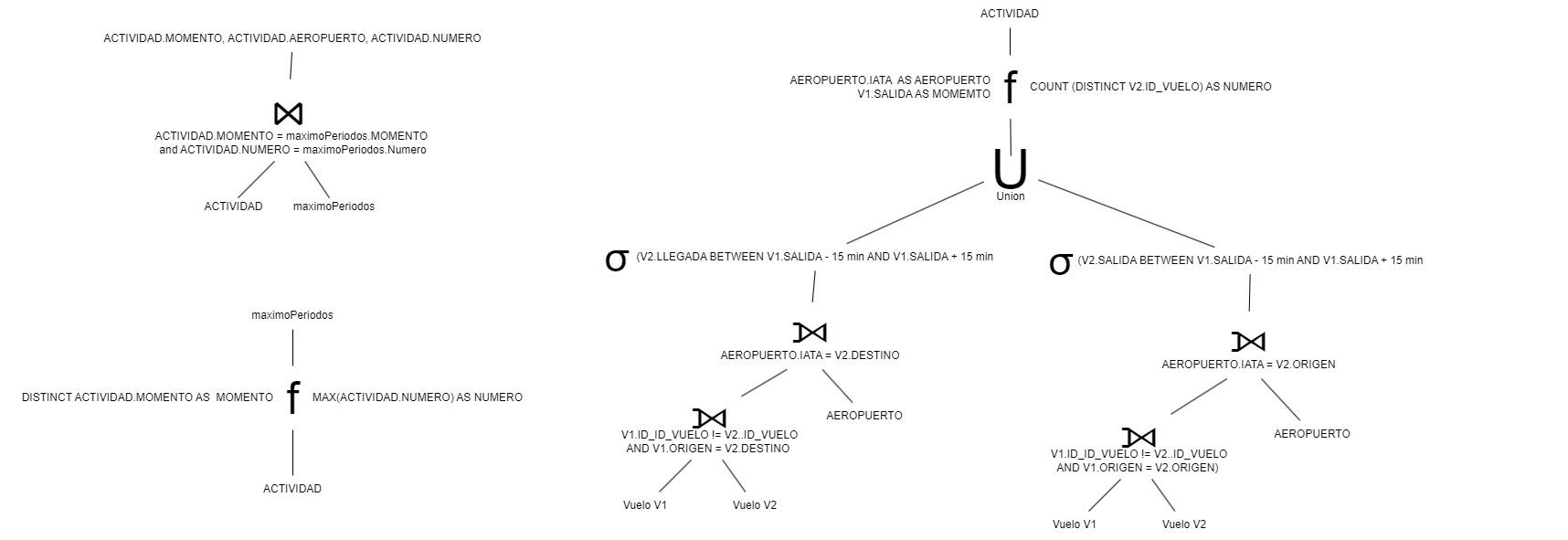
| **%** | **MODELO** | **FABRICANTE** | **MOTOR** |
| --- | --- | --- | --- |
| 7.69 | 757-33N | BOEING | Turbo-Jet |
| 7.14 | 767-33A | BOEING | Turbo-Fan |
| 5 | 737-924ER | BOEING | Turbo-Fan |
| 3.57 | 737-3G7 | BOEING | Turbo-Jet |
| 2.7 | 737-3TO | BOEING | Turbo-Jet |
| 1.85 | A319-132 | AIRBUS | Turbo-Jet |
| 1.75 | 737-524 | BOEING | Turbo-Fan |
| 1.35 | CL600-2D24 | BOMBARDIER INC | Turbo-Fan |
| 1.11 | A321-211 | AIRBUS | Turbo-Jet |
| 1.08 | 737-3H4 | BOEING | Turbo-Fan |
| 1.04 | 737-5H4 | BOEING | Turbo-Jet |

## 

## **3ª consulta.** Aeropuertos que han tenido el momento con mayor actividad, y momento en el que ha ocurrido.

Para esta tercera consulta, primero obtenemos el número de vuelos totales que despegan y aterrizan que tiene cada aeropuerto en intervalos de 15 minutos. Tras ello calculamos el máximo de todos los intervalos. Finalmente juntamos el máximo de los intervalos con el aeropuerto donde se ha producido.

**Árbol sintáctico.**



### Consulta SQL.

– Calculamos la actividad de cada aeropuerto en intervalos de 15 minutos

WITH ACTIVIDAD AS (

SELECT AEROPUERTO.IATA AS AEROPUERTO, V1.SALIDA AS MOMENTO, COUNT(DISTINCT V2.ID\_VUELO) AS NUMERO

FROM VUELO V1, VUELO V2, AEROPUERTO

WHERE V1.ID\_VUELO <> V2.ID\_VUELO

AND (V2.LLEGADA BETWEEN V1.SALIDA - 15/1440 AND V1.SALIDA + 15/1440

AND V1.ORIGEN = 2.DESTINO AND AEROPUERTO.IATA = V2.DESTINO)

OR (V2.SALIDA BETWEEN V1.SALIDA - 15/1440 AND V1.SALIDA + 15/1440

AND AEROPUERTO.IATA = V2.ORIGEN AND V1.ORIGEN = V2.ORIGEN)

GROUP BY AEROPUERTO.IATA, V1.SALIDA

ORDER BY NUMERO DESC

),

– Calculamos el máximo de cada intervalo

maximoPeriodos AS(

SELECT DISTINCT ACTIVIDAD.MOMENTO AS MOMENTO, MAX(ACTIVIDAD.NUMERO) AS NUMERO

FROM ACTIVIDAD

GROUP BY ACTIVIDAD.MOMENTO

)

– Juntamos el máximo de cada intervalo junto con el aeropuerto donde se ha producido

SELECT DISTINCT TO\_CHAR(ACTIVIDAD.MOMENTO, 'DD/MM/YYYY HH24:MI:SS') AS MOMENTO,

ACTIVIDAD.AEROPUERTO AS AEROPUERTO, ACTIVIDAD.NUMERO AS NUMERO

FROM ACTIVIDAD, maximoPeriodos

WHERE ACTIVIDAD.MOMENTO = maximoPeriodos.MOMENTO

and ACTIVIDAD.NUMERO = maximoPeriodos.NUMERO

ORDER BY NUMERO DESC;

Cabe destacar que los atributos LLEGADA y SALIDA son de tipo DATE y por tanto guardan la fecha y la hora. Así pues, si quieres hacer intervalos de 15 minutos basta con restarle o sumarle 15. Para indicar a Oracle que queremos restar 15 minutos lo hacemos de la siguiente forma: +-15/1440. Si le sumamos o le restamos solo 15 el ORACLE interpreta que quieres sumar 15 días por lo tanto para sumar minutos hay que dividir esos 15 días por el número de minutos que tiene un día. Para mostrar la hora es necesario hacer uso de la función “TO\_CHAR”.

### Resultado.

Obtenemos la actividad máxima de cada uno de los intervalos con sus correspondientes aeropuertos. Al ser un gran número de intervalos tan solo vamos a mostrar las 10 primeras filas.

| **ACTIVIDAD** | **MOMENTO** | **AEROPUERTO** |
| --- | --- | --- |
| 164 | 27/10/2008 8:45 | ATL |
| 154 | 26/10/2008 8:45 | ATL |
| 151 | 25/10/2008 8:45 | ATL |
| 147 | 27/10/2008 8:40 | ATL |
| 145 | 27/10/2008 8:46 | ATL |
| 142 | 27/10/2008 8:42 | ATL |
| 142 | 27/10/2008 8:49 | ATL |
| 141 | 27/10/2008 8:50 | ATL |
| 140 | 27/10/2008 8:41 | ATL |
| 138 | 25/10/2008 10:00 | ATL |

## 

PARTE 3: Diseño Físico

**Optimizaciones**

## Consulta 1. Tres compañías aéreas menos puntuales (% vuelos retrasados).

### **Índices.**

CREATE INDEX idx\_vuelo\_avion ON VUELO (AVION); -- join VUELO-AVION

CREATE INDEX idx\_retraso\_id\_vuelo ON RETRASO (ID\_VUELO); --join VUELO-RETRASO

Mediante la creación de un índice en la columna AVION de la tabla VUELO, el SGBD puede localizar eficientemente las filas coincidentes en función de los valores de AVION. Dado que la consulta une estas dos tablas utilizando la columna AVION, el índice permitiría una recuperación más rápida de las filas relevantes, lo que reduciría el tiempo de ejecución general.

La columna ID\_VUELO se utiliza como clave externa en la tabla RETRASO para hacer referencia a la clave primaria en la tabla VUELO. Al crear un índice en la columna ID\_VUELO de la tabla RETRASO, el SGBD puede hacer coincidir eficientemente los valores de ID\_VUELO y recuperar las filas correspondientes. Este índice mejoraría el rendimiento de la unión, especialmente al unir conjuntos de datos grandes, lo que resultaría en una ejecución más rápida de la consulta.

| **OPERACIÓN** | **SIN ÍNDICES** | | **CON ÍNDICES** | |
| --- | --- | --- | --- | --- |
| **BYTES** | **COST (%CPU)** | **BYTES** | **COST (%CPU)** |
| SELECT STATEMENT | 4970K | 236 (2) | 4970K | 236 (2) |
| HASH GROUP BY | 217K | 127 (3) | 217K | 127 (3) |
| VIEW | 217K | 127 (3) | 217K | 127 (3) |
| HASH GROUP BY | 369K | 127 (3) | 369K | 127 (3) |
| HASH JOIN RIGHT OUTER | 2286K | 125 (1) | 2286K | 125 (1) |
| TABLE ACCESS FULL | RETRASO | 219K | 15 (0) | 219K | 15 (0) |
| HASH JOIN | 1276K | 109 (0) | 1276K | 109 (0) |
| TABLE ACCESS FULL | AVION | 49488 | 7 (0) | 49488 | 7 (0) |
| TABLE ACCESS FULL | VUELO | 730K | 102 (0) | 730K | 102 (0) |
| HASH JOIN | 4970K | 57 (0) | 4970K | 57 (0) |
| TABLE ACCESS FULL | AEROLINEA | 114K | 5 (0) | 114K | 5 (0) |
| VIEW | 894K | 52 (0) | 894K | 52 (0) |
| TABLE ACCESS FULL | 1342K | 52 (0) | 1342K | 52 (0) |
| VIEW | 646K | 52 (0) | 646K | 52 (0) |
| TABLE ACCESS FULL | 1342K | 52 (0) | 1342K | 52 (0) |
| **TOTAL** | 19,864K | 1,220 | 19,864K | 1,220 |

Los índices implementados no mejoran el rendimiento de la consulta. La causa puede ser la alta selectividad de las operaciones, por lo que el SGBD puede decidir utilizar el mismo plan en ambos casos.

### 

### **Vista materializada.**

CREATE MATERIALIZED VIEW retrasos\_aerolineas

REFRESH FAST ON COMMIT

AS

SELECT

AVION.AEROLINEA,

COUNT(DISTINCT VUELO.ID\_VUELO) AS NUM\_VUELOS,

COUNT(RETRASO.ID\_INCIDENCIA) AS NUM\_RETRASOS,

COUNT(RETRASO.ID\_INCIDENCIA) / COUNT(DISTINCT VUELO.ID\_VUELO) \* 100 AS PORCENTAJE\_RETRASOS

FROM VUELO

INNER JOIN AVION ON AVION.MATRICULA = VUELO.AVION

LEFT JOIN RETRASO ON RETRASO.ID\_VUELO = VUELO.ID\_VUELO

GROUP BY AVION.AEROLINEA;

La vista materializada mejora el rendimiento de la consulta al almacenar los resultados de la consulta original en forma de una tabla materializada. En lugar de calcular los datos en el momento, la vista materializada calcula previamente los resultados. Además, probamos a implementar el siguiente índice sobre la vista materializada para mejorar la unión con la tabla AEROLINEA.

CREATE INDEX idx\_retrasos\_aerolineas\_aerolinea ON retrasos\_aerolineas(AEROLINEA);

| **OPERACIÓN** | **BYTES** | **COST (%CPU)** |
| --- | --- | --- |
| **SIN ÍNDICE** | | |
| SELECT STATEMENT | 1800 | 11 (0) |
| HASH JOIN | 1800 | 8 (0) |
| MAT\_VIEW ACCESS FULL | RET\_AEROLINEAS | 324 | 3 (0) |
| TABLE ACCESS FULL | AEROLINEA | 114K | 5 (0) |
| MAT\_VIEW ACCESS FULL | RET\_AES | 13 | 3 (0) |
| **TOTAL** | 118K | 30 |
| **CON ÍNDICE** | | |
| SELECT STATEMENT | 1800 | 11 (10) |
| MERGE JOIN | 1800 | 8 (13) |
| MAT\_VIEW ACCESS BY INDEX ROW | RET\_AE | 324 | 2 (0) |
| INDEX FULL SCAN | IDX\_RET\_AE\_AEROLA | - | 1 (0) |
| SORT JOIN | 114K | 6 (17) |
| TABLE ACCESS FULL | AEROLINEA | 114K | 5 (0) |
| MAT\_VIEW ACCESS FULL | RET\_AE | 13 | 3 (0) |
| **TOTAL** | 231K | 36 |

Observamos que el rendimiento de la consulta sin el uso del índice es muy superior en uso de bytes (x1.95) y significativamente en uso de CPU (x1.2).

Concluimos que el coste de realizar la consulta utilizando la vista materializada es mucho menor, sin esta la consulta es x168 más intensiva en uso de bytes y x40 en CPU. No obstante, la decisión sobre si implementarla o no dependerá del número de veces que se haga la consulta y del número de inserciones en las tablas implicadas.

# Consulta 2: Fabricantes, modelos y motores en los que el número de vuelos retrasados por motivos de seguridad sea mayor del 1% de su total de retrasos.

### **Índices.**

CREATE INDEX idx\_retraso\_causa ON RETRASO (CAUSA);

CREATE INDEX idx\_vuelo\_id\_vuelo ON VUELO (ID\_VUELO);

CREATE INDEX idx\_avion\_matricula ON AVION (MATRICULA);

El índice **idx\_retraso\_causa mejora la búsqueda uniendo las columnas CAUSA y RETRASO.**

Al crear un índice en la columna CAUSA de la tabla RETRASO, el SGBD puede hacer coincidir eficientemente los valores de CAUSA y recuperar las filas correspondientes. Este índice mejoraría el rendimiento de la búsqueda por su alta cardinalidad, especialmente al unir conjuntos de datos grandes, lo que resultaría en una ejecución más rápida de la consulta.

El índice **idx\_vuelo\_id\_vuelo mejora la búsqueda uniendo las columnas VUELO y ID\_VUELO**.

Mediante la creación de un índice en la columna ID\_VUELO de la tabla VUELO, el SGBD puede localizar eficientemente las filas coincidentes en función de los valores de VUELO. Dado que la consulta hace una gran búsqueda para relacionar vuelo con otros datos, sería conveniente tener este índice para así tener una mayor velocidad a la hora de relacionar datos, lo que reduciría el tiempo de ejecución general.

El índice **idx\_avion\_matricula mejora la búsqueda uniendo las columnas AVION y MATRICULA**.

Mediante la creación de un índice en la columna MATRICULA de la tabla AVION, el SGBD puede localizar eficientemente las filas coincidentes en función de los valores de AVION. Dado que MATRICULA es clave primaria de avión esto podría reducir los costes de joins futuros entre otras tablas, el índice permitiría una recuperación más rápida de las filas relevantes, lo que reduciría el tiempo de ejecución general.

Si bien la mejora es mínima, afectando al HASH JOIN y su número de bytes reduciendo drásticamente de 1045K a 12444 bytes, un número más elevado de datos podría hacer que esta mejora fuera más significativa.

| **OPERACIÓN** | **CON ÍNDICE** | | **SIN ÍNDICE** | |
| --- | --- | --- | --- | --- |
| **BYTES** | **COST (%CPU)** | **BYTES** | **COST (%CPU)** |
|
| SELECT STATEMENT | 2159K | 130 (3) | 2159K | 130 (3) |
| SORT ORDER BY | 2159K | 130 (3) | 2159K | 130 (3) |
| HASH GROUP BY | 2159K | 130 (3) | 2159K | 130 (3) |
| HASH JOIN | 2159K | 128 (1) | 2159K | 128 (1) |
| TABLE ACCESS FULL | MODELO | 12927 | 3 (0) | 12927 | 3 (0) |
| HASH JOIN | 12444 | 125 (1) | 1045K | 125 (1) |
| TABLE ACCESS FULL | AVION | 117K | 7 (0) | 117K | 7 (0) |
| HASH JOIN | 633K | 118 (1) | 633K | 118 (1) |
| TABLE ACCESS FULL | RETRASO | 438K | 15 (0) | 438K | 15 (0) |
| TABLE ACCESS FULL | VUELO | 715K | 102 (0) | 715K | 102 (0) |

### **Vista materializada.**

CREATE MATERIALIZED VIEW ret\_seguridad\_mv AS

SELECT

AVION.MODELO,

SUM(CASE WHEN RETRASO.CAUSA = 'Security' THEN 1 ELSE 0 END) / COUNT (\*) AS POR\_SEG

FROM RETRASO

INNER JOIN VUELO ON RETRASO.ID\_VUELO = VUELO.ID\_VUELO

INNER JOIN AVION ON AVION.MATRICULA = VUELO.AVION

WHERE AVION.MODELO <> 'no model'

GROUP BY AVION.MODELO;

La vista materializada mejora el rendimiento de la consulta al almacenar los resultados de la consulta original en forma de una tabla materializada. En lugar de calcular los datos en el momento, la vista materializada calcula previamente los resultados.

| **OPERACIÓN** | **SIN ÍNDICES** | | **CON ÍNDICES** | |
| --- | --- | --- | --- | --- |
| **BYTES** | **COST (%CPU)** | **BYTES** | **COST (%CPU)** |
|
| SELECT STATEMENT | 1331 | 7(15) | 1331 | 7(15) |
| SORT ORDER BY | 1331 | 7(15) | 1331 | 7(15) |
| HASH JOIN | 1331 | 6(0) | 1331 | 6(0) |
| MAT\_VIEW ACCESS FULL| RET\_SEGURIDAD\_MV | 440 | 3(0) | 440 | 3(0) |
| TABLE ACCESS FULL | MODELO | 11340 | 3 (0) | 11340 | 3 (0) |

Podemos apreciar una gran mejoría con las vistas materializadas que minimiza a más de la mitad del coste de la consulta. Pasamos de 11 operaciones a tan solo 5 y además al consultar los datos de la vista materializada apenas gastarán recursos en bytes o tiempo. Con la vista materializada la ejecución pasa de unos 11 segundos a tan solo 1 segundo. Si bien los índices en este caso no afectan en nada, consideramos que no importa puesto que la optimización que se obtiene es muy grande.

# 

# Consulta 3: Aeropuertos que han tenido el momento con mayor actividad, y momento en el que ha ocurrido.

### Índices.

CREATE INDEX idx\_vuelo\_origen\_destino ON VUELO (ORIGEN, DESTINO); -- Join ORIGEN- DESTINO

CREATE INDEX idx\_vuelo\_salida\_llegada ON VUELO (SALIDA, LLEGADA); -- Precalculo SALIDA-LLEGADA

Mediante la creación de un índice en las columnas ORIGEN, DESTINO, SALIDAS, LLEGADA de la tabla VUELO, el SGBD puede localizar eficientemente las filas coincidentes en función de los valores de AVION. Dado que la consulta une estas dos tablas utilizando la columna AVION el índice permitiría un cálculo más rápido, lo que reduciría el tiempo de ejecución general.

Si nos fijamos en el tiempo de ejecución ( set timing on ) podemos ver los siguientes resultados:

* Sin Índices: 2 minutos y 8 segundos
* Con índices: 1 minuto y 38 segundos

Se puede ver que el tiempo de ejecución con índices es 30 segundos más rápido. Sin embargo, esos datos no solo dependen del programa, sino que también puede llegar a depender de otros factores.

Debido a la inexactitud del tiempo de ejecución, hacemos uso de la función “Explain Plan For” para saber el número de operaciones y el porcentaje de uso de la CPU:

| **OPERACIÓN** | **SIN ÍNDICES** | | **CON ÍNDICES** | |
| --- | --- | --- | --- | --- |
| **BYTES** | **COST (%CPU)** | **BYTES** | **COST (%CPU)** |
| SELECT STATEMENT | 47 | 553 (12) | 47 | 553 (12) |
| SORT ORDER BY | 1001K | 469 (14) | 1001K | 469 (14) |
| HASH GROUP BY | 1001K | 469 (14) | 1001K | 469 (14) |
| VIEW | 1001K | 465 (13) | 1001K | 465 (13) |
| HASH GROUP BY | 1001K | 465 (13) | 1001K | 465 (13) |
| VIEW | 1001K | 463 (12) | 1001K | 463 (12) |
| NESTED LOOPS | 858K | 231 (12) | 858K | 231 (12) |
| HASH JOIN | 786K | 230 (12) | 786K | 230 (12) |
| TABLE ACCESS FULL | 536K | 102 (0) | 536K | 102 (0) |
| TABLE ACCESS FULL | 939K | 102 (0) | 939K | 102 (0) |
| INDEX UNIQUE SCAN | 3 | 0 (0) | 3 | 0 (0) |
| NESTED LOOPS | 1358K | 232 (13) | 1358K | 232 (13) |
| HASH JOIN | 1287K | 230 (12) | 1287K | 230 (12) |
| TABLE ACCESS FULL | 939K | 102 (0) | 939K | 102 (0) |
| TABLE ACCESS FULL | 1476K | 102 (0) | 1476K | 102 (0) |
| INDEX UNIQUE SCAN | 3 | 0 (0) | 3 | 0 (0) |
| SORT UNIQUE | 47 | 84 (5) | 47 | 84 (5) |
| HASH JOIN | 47 | 83 (4) | 47 | 83 (4) |
| VIEW | 22 | 42 (16) | 22 | 42 (16) |
| HASH GROUP BY | 22 | 42 (5) | 22 | 42 (5) |
| VIEW | 1049K | 40 (0) | 1049K | 40 (0) |
| TABLE ACCESS FULL | 1001K | 40 (0) | 1001K | 40 (0) |
| VIEW | 1192K | 40 (0) | 1192K | 40 (0) |
| TABLE ACCESS FULL | 1001K | 40 (0) | 1001K | 40 (0) |

Como se puede ver en la tabla, el usos de índices no mejora casi nada el rendimiento de la consulta por lo que se va a hacer uso de otras técnicas de diseño físico

**Partición horizontal.**

Los índices no eran suficientes para optimizar la consulta. Por lo tanto se ha hecho particiones horizontales sobre la tabla VUELO de tipo “hash”:

CREATE TABLE VUELO (

ID\_VUELO VARCHAR(15) PRIMARY KEY,

NUMERO VARCHAR(10),

AVION VARCHAR(10) REFERENCES AVIÓN(MATRICULA),

ORIGEN VARCHAR(3) REFERENCES AEROPUERTO(IATA),

DESTINO VARCHAR(3) REFERENCES AEROPUERTO(IATA),

SALIDA DATE,

LLEGADA DATE,

CONSTRAINT CHK\_FECHA CHECK (SALIDA <= LLEGADA),

CONSTRAINT CHK\_AEROPUERTOS\_DISTINTOS CHECK (ORIGEN <> DESTINO)

)

STORAGE (INITIAL 50K)

PARTITION BY HASH(SALIDA)

(PARTITION par1, PARTITION par2,

PARTITION par3, PARTITION par4, PARTITION par5);

| **OPERACIÓN** | **Particion Horizontal** | |
| --- | --- | --- |
| **BYTES** | **COST (%CPU)** |
| SELECT STATEMENT | 47 | 239 (3) |
| SORT ORDER BY | 21 | 232 (2) |
| HASH GROUP BY | 21 | 232 (2) |
| VIEW | 21 | 230 (1) |
| HASH GROUP BY | 57 | 230 (1) |
| NESTED LOOPS | 57 | 229 (1) |
| MERGE JOIN CARTESIAN | 36 | 115 (0) |
| INDEX FAST FULL SCAN | 3 | 2 (0) |
| BUFFER SORT | 1592K | 113 (0) |
| PARTITION HASH ALL | 1592K | 113 (0) |
| TABLE ACCESS FULL | 1592K | 113 (0) |
| PARTITION HASH ALL | 21 | 114 (1) |
| TABLE ACCESS FULL | 21 | 114 (1) |
| SORT UNIQUE | 47 | 6 (20) |
| HASH JOIN SEMI | 47 | 5 (12) |
| VIEW | 25 | 2 (0) |
| TABLE ACCESS FULL | 21 | 2 (0) |
| VIEW | 22 | 3 (16) |
| HASH GROUP BY | 22 | 3 (20) |
| VIEW | 22 | 2 (0) |
| TABLE ACCESS FULL | 21 | 2 (0) |
| SELECT STATEMENT | 47 | 239 (3) |

Se puede ver en la tabla como el número de operaciones ha disminuido con respecto a la optimización con únicamente índices. Además la carga del procesador se mantiene muy parecida por lo tanto podemos decir que esta consulta es más óptima con la partición horizontal.

# Triggers.

En nuestra base de datos encontramos restricciones que Oracle no puede solucionar con “**CONSTRAINTS”** en la sentencia de creación de tablas. Algunas de estas son:

1. Si los desvíos de un vuelo no son al mismo aeropuerto.
2. Comprobar que la fecha de fabricación del avión es anterior a la fecha de llegada y de salida.
3. Comprobar que la fecha de salida de un vuelo es anterior a la fecha de llegada.

## 1. Comprueba si los desvíos que tiene un avión no son al mismo aeropuerto

CREATE OR REPLACE TRIGGER ControlDesvio

BEFORE INSERT ON DESVIO

FOR EACH ROW

DECLARE

count\_exist NUMBER;

BEGIN

-- Verificar si existe una combinación de ID\_VUELO y IATA en la tabla DESVIO

SELECT COUNT(\*) INTO count\_exist

FROM DESVIO

WHERE ID\_VUELO = :NEW.ID\_VUELO AND IATA = :NEW.IATA;

-- Si existe, generar un error y cancelar la inserción

IF count\_exist > 0 THEN

raise\_application\_error(-20000, 'Ya existe una desviacion para este vuelo y aeropuerto de destino');

END IF;

END;

/

La información de cada aeropuerto de destino de desvío se encuentra en la tabla VUELO, en la columna IATA. En ella se encuentra el ID\_VUELO, un identificador de cada vuelo cuya información es relevante para este trigger. Por ello basta con hacer una consulta extrayendo los aeropuertos al que se dirija ese vuelo y compararlo con los ya existentes.

Para la comprobación miramos si en la tabla DESVIO existe algun vuelo con su mismo id y que comparta mismo IATA, en caso afirmativo nuestro contador de coincidencias aumentará y una vez recorrida la tabla si el contador ha llegado a aumentar es que ha habido alguna coincidencia, con lo cual mandaremos un error para evitar la inserción.

## 

## 

## 2. Comprueba que la fecha fecha de fabricación del avión es anterior a la fecha de salida y de llegada de un vuelo

CREATE OR REPLACE TRIGGER controlFabricación

BEFORE INSERT ON VUELO

FOR EACH ROW

DECLARE

AGNO NUMBER(4);

BEGIN

SELECT AGNO\_FABRICACION INTO AGNO

FROM AVION

WHERE MATRICULA = :NEW.AVION;

IF (AGNO > EXTRACT(year FROM :NEW.SALIDA) OR AGNO > EXTRACT(year FROM :NEW.LLEGADA)) THEN

RAISE\_APPLICATION\_ERROR(-20000, 'El avión no puede volar antes de su fabricación');

END IF;

END;

/

La información de cada avión se encuentra en la tabla AVIÓN. En ella se encuentra la fecha de fabricación del avión que es la información relevante para este trigger. Por ello basta con hacer una consulta filtrando con la matrícula del nuevo vuelo y guardando el año en la variable AGNO.

El atributo AGNO\_FABRICIÓN, al igual que la variable AGNO, es de tipo NUMBER(4). Por lo tanto hará falta extraer únicamente el año de los atributos SALIDA y LLEGADA (ambas son de tipo DATE). Para hacer esto se utiliza la función EXTRACT que en este caso extrae el año y lo convierte en tipo NUMBER para poder hacer la comparación.

Si el año de fabricación es mayor que el año de salida o del año de llegada, ese vuelo no se puede llevar a cabo ya que el avión aún no habría sido fabricado.

## 

## 3. Comprueba que la fecha de salida de un vuelo es anterior de la fecha de llegada

CREATE OR REPLACE TRIGGER ControlFechas

BEFORE INSERT ON VUELO

FOR EACH ROW

BEGIN

IF :NEW.SALIDA > :NEW.LLEGADA THEN

RAISE\_APPLICATION\_ERROR(-20000, 'La fecha de salida tiene que ser anterior a la de llegada');

END IF;

END;

/

La información relevante para poder hacer la comprobación se encuentra únicamente en el propio INSERT que se realice. Nos tenemos que fijar únicamente en los atributos SALIDA y LLEGADA.

En este trigger si que se puede comparar sin extraer el año ya ambos atributos (SALIDA Y LLEGADA) son de tipo de DATE. Cabe destacar que estos atributos también guardan la hora en cual se despega o aterriza, así pues las horas también las tendrá en cuenta a la hora de hacer la comparación.

Si la fecha de salida es mayor que la fecha de llegada significa que el avión ha despegado más tarde que cuando ha aterrizado. Esto, como es lógico, no puede pasar por lo tanto este vuelo no se insertaría

Conclusiones

# Horas dedicadas:

|  | **ALEJANDRO BENEDI** | **ALVARO DE FRANCISCO** | **JAVIER JULVE** | **TOTAL** |
| --- | --- | --- | --- | --- |
| **PARTE I** | 2 horas | 2 horas | 2 horas | 6 horas |
| **PARTE II** | 3 horas | 5 horas | 2 horas y 30 minutos | 10 horas y media |
| **PARTE III** | 3 horas y media | 2 horas y media | 3 horas y media | 9 horas y media |
| **TOTAL** | 8 horas y media | 9 horas | 8 horas |  |

# División del trabajo:

Al dividirnos el trabajo, acordamos dividir de forma equitativa las tareas para poder hacer el mayor número de tareas de forma paralela. Por ejemplo, cada miembro del grupo ha hecho una de las consultas con su respectiva optimización (diseño físico). Esto se ha podido hacer debido a la experiencia y a la soltura que hemos obtenido tras 2 bases. Aunque las tareas han estado repartidas, todas han sido supervisadas por todos los miembros del equipo y hemos estado en constante comunicación para optimizar el tiempo lo máximo posible.

### **Dificultades:**

Como tal en esta práctica no hemos tenido grandes complicaciones. Esto es debido principalmente a la experiencia que hemos obtenido tras 2 bases de datos. Lo que más nos ha costado han sido las consultas ya que consideramos que han sido más complicadas que las de la práctica anterior. En las demás parte de la práctica no hemos tenido ningún tipo de problema.