

BASES DE DATOS:

PRÁCTICA 1



Rodrigo Díaz-Regañón Ureña
Daniel Martínez Fernández

1. Introducción y objetivos

El objetivo de esta práctica consiste en la toma de contacto con las bases de datos. A lo largo de esta, diseñaremos, desarrollaremos y haremos consultas en una base de datos de vuelos. Las bases de datos se crearán en el gestor de bases de datos PostgreSQL y las consultas se llevarán a cabo usando el lenguaje SQL.

2. Base de datos

Nuestra base de datos ha consistido en la organización de los vuelos de aviones. La base de datos organiza la información de reservas de vuelos de la siguiente manera: cada reserva puede incluir varios pasajeros, y cada pasajero tiene un billete que incluye sus datos, de modo que si la misma persona compra billetes distintos se registra como pasajeros independientes; los billetes pueden cubrir varios vuelos (por conexiones o ida y vuelta) y todos los billetes de una misma reserva comparten los mismos vuelos; cada vuelo tiene un origen y destino definidos por aeropuertos, y vuelos con el mismo número mantienen estos datos aunque cambie la fecha; al realizar el check-in, cada pasajero recibe una tarjeta de embarque con su asiento, sin que se repitan asientos en un mismo vuelo; finalmente, la distribución de asientos depende del modelo de avión que realiza el vuelo, y cada modelo tiene una configuración única de asientos.

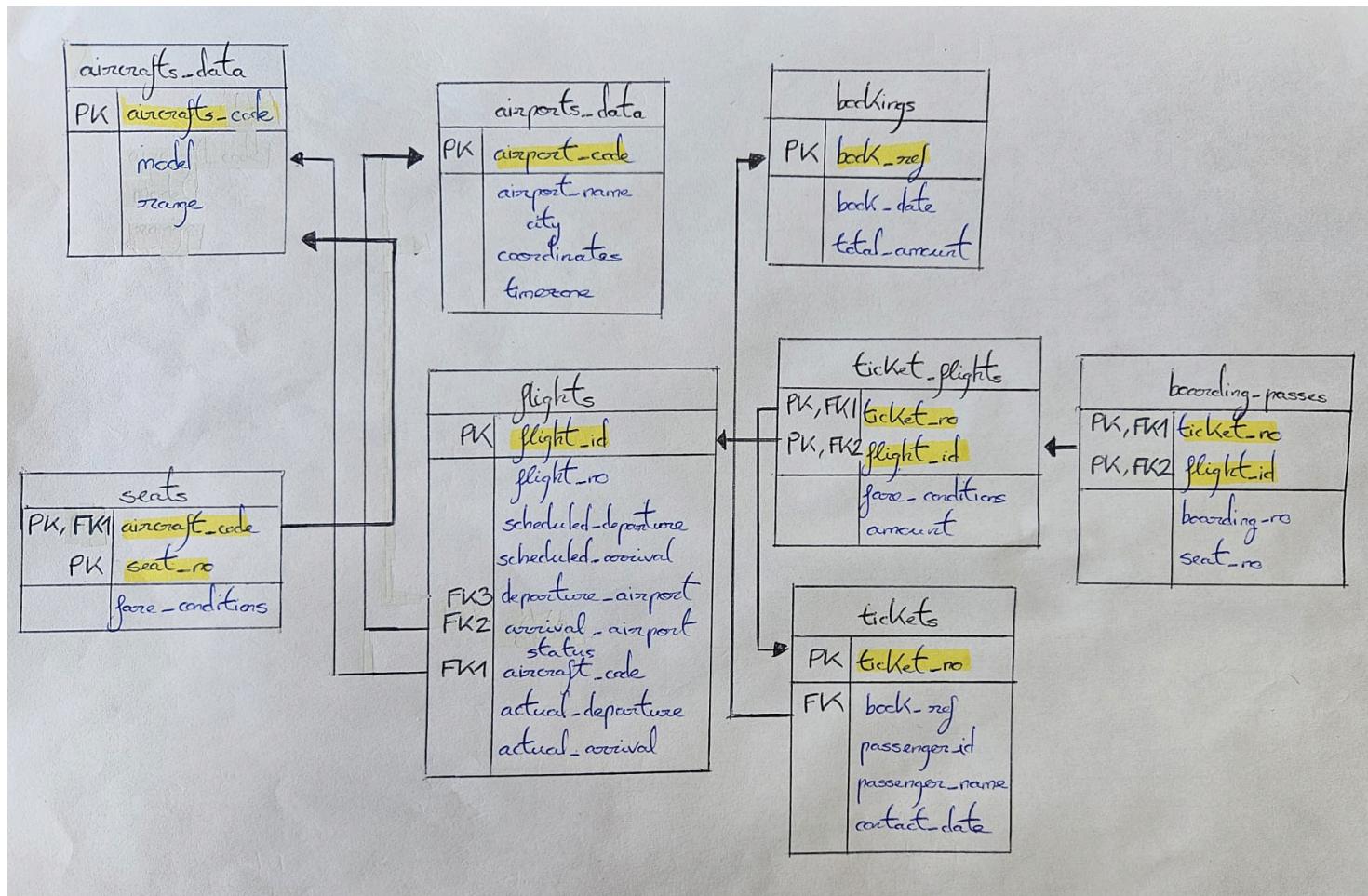
- **El esquema de la base de datos es el siguiente:**

- airports_data (**airport_code**, airport_name, city, coordinates, timezone)
- aircrafts_data (**aircraft_code**, model, range)
- flights (**flight_id**, flight_no, scheduled_departure, scheduled_arrival, departure_airport → airports_data.airport_code, arrival_airport → airports_data.airport_code, status, aircraft_code → aircrafts_data.aircraft_code, actual_departure, actual_arrival)
- boarding_passes (**ticket_no** → ticket_flights.ticket_no, **flight_id** → ticket_flights.flight_id, boarding_no, seat_no)
- bookings (**book_ref**, book_date, total_amount)
- seats (**aircraft_code** → aircrafts_data.aircraft_code, **seat_no**, fare_conditions)

-ticket_flights(ticket_no → tickets.ticket_no, flight_id → flights.flight_id, fare_conditions, amount)

-tickets (ticket_no, book_ref → bookings.book_ref, passenger_id, passenger_name, contact_data)

- El diagrama relacional de la base de datos es el siguiente:



3. Consultas

- CONSULTA 1:

```

SELECT f_first.departure_airport,
       Count(DISTINCT b.book_ref) AS num_reservas
  FROM bookings b
 JOIN tickets t

```

```

ON b.book_ref = t.book_ref
JOIN (SELECT tf.ticket_no,
      Min(f.scheduled_departure) AS first_departure_time,
      Max(f.scheduled_departure) AS last_departure_time
FROM ticket_flights tf
JOIN flights f
      ON tf.flight_id = f.flight_id
GROUP BY tf.ticket_no) ftime
ON t.ticket_no = ftime.ticket_no
JOIN flights f_first
ON f_first.scheduled_departure = ftime.first_departure_time
AND f_first.flight_id IN (SELECT flight_id
      FROM ticket_flights
      WHERE ticket_no = t.ticket_no)
JOIN flights f_last
ON f_last.scheduled_departure = ftime.last_departure_time
AND f_last.flight_id IN (SELECT flight_id
      FROM ticket_flights
      WHERE ticket_no = t.ticket_no)
WHERE f_first.departure_airport = f_last.arrival_airport
GROUP BY f_first.departure_airport
ORDER BY f_first.departure_airport;

```

Esta consulta va creando tablas seleccionando lo que nos interesa en cada momento, mediante joins, de forma que al final nos quede una tabla cuyas tuplas cumplen las condiciones necesarias.

Se parte desde la tabla bookings (b), pues estamos contabilizando el número de reservas que contienen un viaje de ida y vuelta, unimos esta tabla con la tabla tickets (t) mediante la book_ref en la que se encuentra cada ticket, uniremos ahora esta tabla con una subtabla (ftime), que consta de tres columnas, el número de ticket y el horario de salida del primer vuelo (first_departure_time) y el del

	ticket_no character (13)	first_departure_time timestamp with time zone	last_departure_time timestamp with time zone
1	0005432000987	2017-07-16 11:05:00+02	2017-07-16 11:05:00+02
2	0005432000988	2017-07-16 11:05:00+02	2017-07-16 11:05:00+02
3	0005432000989	2017-07-17 11:05:00+02	2017-07-17 11:05:00+02
4	0005432000990	2017-07-17 11:05:00+02	2017-07-17 11:05:00+02
5	0005432000991	2017-07-18 11:05:00+02	2017-07-18 11:05:00+02
6	0005432000992	2017-07-18 11:05:00+02	2017-07-18 11:05:00+02
7	0005432000993	2017-07-18 11:05:00+02	2017-07-18 11:05:00+02
8	0005432000994	2017-07-19 11:05:00+02	2017-07-19 11:05:00+02
9	0005432000995	2017-07-19 11:05:00+02	2017-07-19 11:05:00+02
10	0005432000996	2017-07-20 11:05:00+02	2017-07-20 11:05:00+02

último vuelo del ticket (last_departure_time), para realizar esta subtabla, se parte de la tabla ticket_flights (tf) y se une con flights (f) mediante los flights_id y finalmente se agrupa por número de tickets, se puede apreciar la subtabla a la derecha.

La unión de lo que llevábamos con ftime se hará mediante los ticket_no.

Ya por último haremos dos uniones con flights, la primera (f_first) en la que se continuarán las tuplas con las columnas de flights, donde first_departure_time sea igual a el horario de salida en f_first, se hará para ticket_no (de ahí la condición AND), hacemos el mismo join para el último vuelo (f_last) y ya solo nos quedará seleccionar las tuplas donde se cumple que el aeropuerto de salida es igual al de llegada. Y por último agrupamos y ordenamos por aeropuertos como nos piden en el enunciado.

Tiempo de ejecución: alrededor de 2.5 segundos. Salida:

	departure_airport character (3)	num_reservas bigint	
1	AAQ	1590	
2	ABA	356	
3	AER	3293	
4	ARH	964	
5	ASF	642	
6	BAX	593	
7	BQS	309	
8	BTK	232	

71	URS	267
72	UUA	60
73	UUD	243
74	UUS	1154
75	VKO	29918
76	VOG	349
77	VOZ	426
78	VVO	1975
79	YKS	929

- CONSULTA 2:

```

SELECT bookings.book_ref,
bookings.total_amount,
Sum(ticket_flights.amount) AS calculated_value
FROM bookings
JOIN tickets
ON bookings.book_ref = tickets.book_ref
JOIN ticket_flights
ON tickets.ticket_no = ticket_flights.ticket_no
GROUP BY bookings.book_ref
ORDER BY bookings.book_ref ASC;

```

La consulta obtiene, para cada reserva, su identificador (book_ref), el importe total registrado (total_amount) y la suma de los importes de todos los vuelos asociados a los billetes de esa reserva (calculated_value). Para ello, une la tabla bookings con tickets mediante book_ref para relacionar cada reserva con sus billetes, y luego une tickets con ticket_flights mediante ticket_no para acceder a los vuelos de cada billete. Despu s agrupa los resultados por book_ref para que la suma de importes se calcule por reserva, y finalmente ordena las reservas de forma ascendente seg n su identificador.

Tiempo de ejecuci n: alrededor de 1.8 segundos. Salida:

	book_ref [PK] character (6)	total_amount numeric (10,2)	calculated_value numeric
1	00000F	265700.00	265700.00
2	000012	37900.00	37900.00
3	000068	18100.00	18100.00
4	000181	131800.00	131800.00
5	0002D8	23600.00	23600.00
6	0002DB	101500.00	101500.00
7	0002E0	89600.00	89600.00
8	0002F3	69600.00	69600.00
9	00034E	73300.00	73300.00
10	000352	109500.00	109500.00
11	000374	136200.00	136200.00
12	00044D	6000.00	6000.00
13	00044E	140100.00	140100.00

	book_ref [PK] character (6)	total_amount numeric (10,2)	calculated_value numeric
262776	FFFC8A	75600.00	75600.00
262777	FFFC8C	24400.00	24400.00
262778	FFFCC2	99200.00	99200.00
262779	FFFCCC	112000.00	112000.00
262780	FFFE39	34000.00	34000.00
262781	FFFE47	212800.00	212800.00
262782	FFFE7B	139200.00	139200.00
262783	FFFEA6	56000.00	56000.00
262784	FFFEF3	56000.00	56000.00
262785	FFFF2C	10800.00	10800.00
262786	FFFF43	78500.00	78500.00
262787	FFFFA8	28800.00	28800.00
262788	FFFFF7	73600.00	73600.00

- CONSULTA 3:

```

SELECT f.arrival_airport,
       Count(*) AS num_pasajeros_recibidos
  FROM  boarding_passes bp
        JOIN ticket_flights tf
          ON ( bp.flight_id = tf.flight_id
                AND bp.ticket_no = tf.ticket_no )
        JOIN flights f
          ON tf.flight_id = f.flight_id
 GROUP BY f.arrival_airport
 ORDER BY num_pasajeros_recibidos ASC
    
```

Para realizar esta consulta partimos de la tabla `boarding_passes`(bp), ya que la tarjeta de embarque será lo que entendamos como un pasajero. Uniremos esta tabla con `ticket_flights`, por medio de las dos columnas `flight_id` y `ticket_no`, pues hacerlo únicamente por medio de una añadiría pasajeros de más al estar haciendo producto cartesiano. Proseguimos haciendo la unión con `flights` (f), por medio del `flight_id` y agrupamos por aeropuertos de llegada para hacer el count de cada aeropuerto.

Tiempo de ejecución: alrededor de 0.2 segundos. Salida:

	arrival_airport character (3)	num_pasajeros_recibidos bigint
1	SWT	8
2	RGK	51
3	CEE	80
4	NYA	82
5	USK	131
6	KXX	158
7	KLF	168
8	UCT	169

87	PEE	18122
88	AER	19182
89	SVX	20628
90	OVB	27477
91	VKO	29701
92	LED	35867
93	DME	76137
94	SVO	77909

- CONSULTA 4:

WITH tabla1

```

AS (SELECT flights.flight_id,
          Count(*) AS asientos_vacios
       FROM flights
     JOIN seats
      ON flights.aircraft_code = seats.aircraft_code
    LEFT JOIN boarding_passes
       ON seats.seat_no = boarding_passes.seat_no
          AND boarding_passes.flight_id = flights.flight_id
 WHERE boarding_passes.seat_no IS NULL
 GROUP BY flights.flight_id)

SELECT *
FROM tabla1
WHERE asientos_vacios = (SELECT Max(asientos_vacios)
                           FROM tabla1)

```

Esta consulta primero crea una tabla temporal (tabla1) que contiene, para cada vuelo (`flight_id`), el número de asientos vacíos (`asientos_vacios`). Para calcularlo, se unen las

tablas flights y seats según el avión (aircraft_code) y luego se hace un left join con boarding_passes para relacionar cada asiento con los pasajeros que han hecho check-in; los asientos sin tarjeta de embarque (boarding_passes.seat_no IS NULL) se cuentan como vacíos. Luego, en la consulta principal, se seleccionan todos los registros de tabla1 cuyo número de asientos vacíos sea igual al máximo de asientos vacíos de toda la tabla, por lo que se devuelve el vuelo con más asientos disponibles.

Tiempo de ejecución: alrededor de 2.25 segundos. Salida:

	flight_id [PK] integer	asientos_vacios bigint
1	9924	402
2	7722	402
3	5255	402
4	10921	402
5	257	402
6	14545	402
7	14536	402
8	9843	402
9	5349	402
10	270	402
11	5335	402
12	10923	402
13	5338	402

	flight_id [PK] integer	asientos_vacios bigint
288	9855	402
289	290	402
290	9884	402
291	9895	402
292	9871	402
293	10880	402
294	9923	402
295	9881	402
296	9894	402
297	5313	402
298	9887	402
299	30588	402
300	9898	402

- CONSULTA 5:

```

SELECT DISTINCT t.book_ref,
    tf.flight_id
FROM tickets t
JOIN ticket_flights tf
    ON t.ticket_no = tf.ticket_no
WHERE NOT EXISTS (SELECT 1
    FROM boarding_passes bp
    WHERE bp.flight_id = tf.flight_id
        AND bp.ticket_no = t.ticket_no)
ORDER BY t.book_ref,
    tf.flight_id;

```

Para realizar esta consulta es fundamental tener claro el concepto de las tarjetas de embarque. Al hacer una reserva, se nos guarda un ticket_no con sus flights_ids asociados (esto se encuentra dentro de ticket_flights), pero para poder viajar es necesario que se emita la tarjeta de embarque, que nos la suelen dar al hacer el check-in, por tanto en boarding_passes se encuentran los ticket_no con sus flight_ids, cuya tarjeta de embarque ha sido ya emitida. Por lo que aquellos billetes que no cuentan con tarjeta de embarque serán esos (ticket_no, flight_id) que se encuentren en ticket_flights pero no en boarding_passes.

En nuestra consulta, haremos una tabla que une tickets (t), con ticket_flights (tf) por ticket_no y se cogerán sólo las tuplas que no existan en la tabla que parte desde boarding_passes(bp) y se une con ticket_flights de forma correlacionada por (ticket_no y flight_id), cuando esta tabla esté vacía para unos (ticket_no, flight_id) dados, estos serán elegidos. Finalmente ordenamos por book_refs como nos piden.

Tiempo de ejecución: alrededor de 0.8 segundos. Salida:

	book_ref character (6) 	flight_id integer 			
1	000068	1039	333332	FFFEA6	5276
2	000068	17082	333333	FFFEA6	9905
3	000181	1039	333334	FFFF2C	3092
4	000181	11238	333335	FFFF2C	31134
5	000181	17082	333336	FFFFA8	5343
6	000181	20881	333337	FFFFA8	10888

- CONSULTA 6:

```

WITH tabla_def
AS (WITH tabla
AS (SELECT *2
actual_arrival - scheduled_arrival AS retraso
FROM flights)
SELECT flight_no,
Avg(retraso) AS retraso_medio
FROM tabla
GROUP BY flight_no)
SELECT flight_no,

```

```

retraso_medio
FROM tabla_def
WHERE retraso_medio = (SELECT Max(retraso_medio)
FROM tabla_def)

```

Esta consulta calcula, para cada número de vuelo (flight_no), el retraso medio y luego devuelve el vuelo con el mayor retraso promedio. Primero, en la subconsulta interna ‘tabla’, se añade una columna llamada ‘retraso’ que calcula la diferencia entre la llegada real y la programada (actual_arrival - scheduled_arrival) para cada vuelo. Luego, en la siguiente subconsulta ‘tabla_def’, se agrupan los datos por flight_no y se calcula el promedio de retraso (Avg(retraso)) como retraso_medio. Finalmente, la consulta principal selecciona los vuelos cuyo retraso_medio es el máximo de todos los retrasos medios de tabla_def.

Tiempo de ejecución: alrededor de 0.128 segundos. Salida:

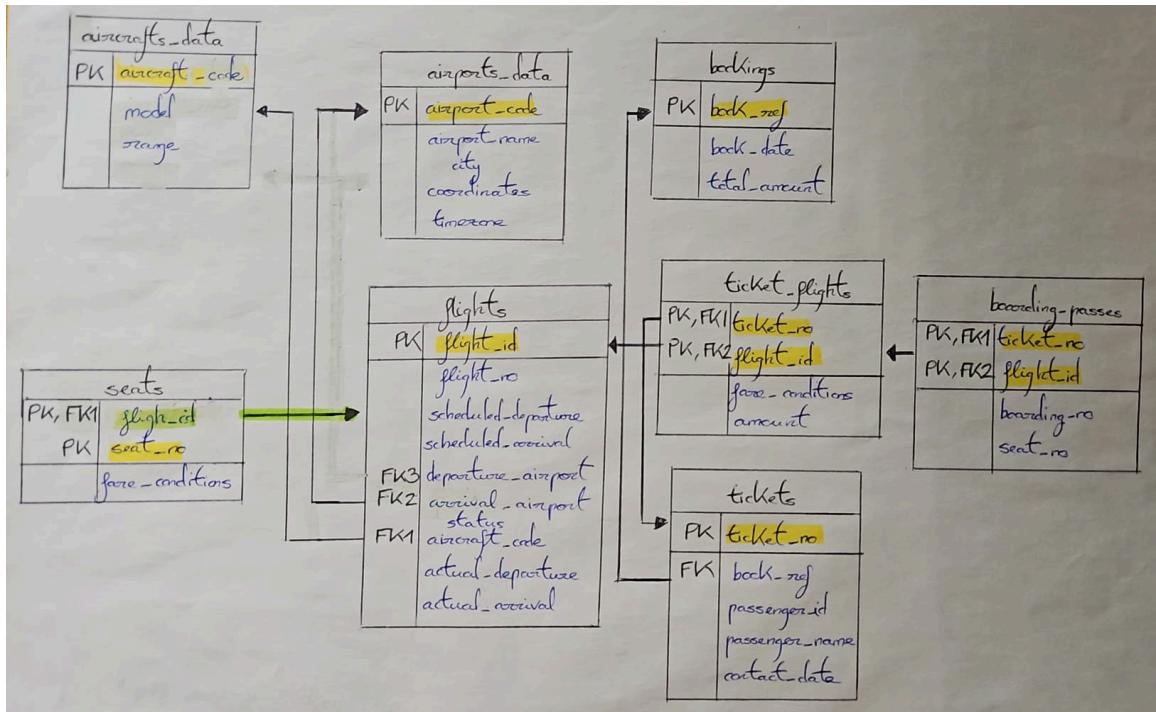
	flight_no character (6) 	retraso_medio interval 
1	PG0450	01:47:15

4. Rediseño

Pensando en cómo funciona la asignación de asientos para cada vuelo según la base de datos dada se llega rápidamente a la conclusión de que hay 0 libertad a la hora de configurar los asientos para un vuelo, pues el tipo de asientos y los asientos ocupados dependen del modelo del avión. Esto lleva a varios problemas por ejemplo, imaginemos que se va a llevar a cabo un vuelo de Madrid a Japón, este vuelo es de larga duración y dejar ocupados todos los asientos puede resultar en incomodidades para los tripulantes, por lo que se prefiere dejar un asiento vacío entre dos asientos, pero con la actual base de datos eso no se puede hacer.

Nuestra propuesta ha sido modificar la tabla seats, de forma que se cambie la columna aircraft_code, ya que de esta forma existen sólo 9 posibles configuraciones de los asientos (1 por cada aircraft_code), por flight_id y por lo tanto que existan tantas configuraciones de asientos como vuelos (flight_id) haya. Al eliminar la columna aircrafts_code no se estará perdiendo nada de información, pues hacemos flight_id una clave fórranea que hace referencia a flights y en allí

encontramos una columna que hace referencia al aircraft_code de cada vuelo. El diagrama relacional quedaría de la siguiente manera, resaltando en verde los cambios introducidos:



La implementación de lo comentado sería la siguiente:

```

nuevabase.sql
1  CREATE TABLE seats_new AS
2  SELECT f.flight_id,
3         s.seat_no,
4         s.fare_conditions
5     FROM flights f
6     JOIN seats s
7       ON f.aircraft_code = s.aircraft_code;
8
9  ALTER TABLE seats_new
10 ADD CONSTRAINT seats_new_pkey PRIMARY KEY (flight_id, seat_no);
11
12 ALTER TABLE seats_new
13 ADD CONSTRAINT seats_new_flight_id_fkey FOREIGN KEY (flight_id) REFERENCES
14 flights(flight_id);
15
16 DROP TABLE seats CASCADE;
17
18 ALTER TABLE seats_new
19   RENAME TO seats;

```

Lo que hacemos es crearnos una tabla auxiliar donde seleccionaremos las dos columnas antiguas de seats y flight_id relacionando seats con flights, por aircraft_code, que la tabla pondrá a cada

flight_id los seat_no y fare_conditions del aircraft_code que tenga ese avión (esto lo hacemos para llenar la tabla, cada flight_id se puede modificar con la configuración que se desee dentro de seats).

Finalmente, añadimos las constraints, eliminamos seats y renombramos la tabla auxiliar a seats.