Universidad Autónoma de Madrid

Departamento de Informática

Fundamentos de Base de Datos
Práctica - 1

Jorge de las Peñas Plana

Registro de Cambios

Versión[1]	**Fecha**	**Autor**	**Descripción**
1.0	01.09.2025	JPP	Primera versión

[1]La asignación de versiones se realizan mediante 2 números $X.Y$. Cambios en Y indican aclaraciones o descripciones más detalladas de algún punto, o traducciones. Cambios en X indican modificaciones más profundas que o bien varían el material suministrado o el contenido de la práctica.

Índice

1. Objetivos

A lo largo de esta práctica 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. Familiarizarse con la Base de Datos

En *Moodle* podéis encontrar la base de datos (`flight.sql`) y el fichero `Makefile` que se usará para crear la base de datos de dominio público "Airlines" (`https://postgrespro.com/docs/postgrespro/10/demodb-bookings`).

El comando `make` además de utilizarse para compilar código puede ser usado para automatizar otras tareas. En nuestro caso lo usaremos para crear y borrar bases de datos así como para ejecutar las consultas a las mismas. En nuestro `Makefile` hemos definido los comandos que se especifican en la Tabla 2.

comando	**función**
createdb	crea una base de datos llamada `flight`
dropdb	borra la base de datos
dump	crea una copia de respaldo de la base de datos
restore	carga la copia de respaldo
shell	invoca el cliente de linea de comandos *psql*
all	ejecuta dropdb, createdb y restore
query?	ejecuta la query ? (donde ? es un número de 1 a 6) y muestra el resultado

Tabla 2: listado de las tareas que se pueden ejecutar usando make.

Cread y poblad la base de datos usando `make all` (la primera vez tendréis que descomprimir el fichero `flight.sql`.zip), examinad la base de datos creada usando un cliente bien sea de línea de comandos (`make shell`) o gráfico (`pgAdmin4`) y proporcionad, en la memoria de la práctica, la información siguiente:

1. Clave primaria de cada tabla.

2. Claves extranjeras que aparezcan en cada tabla.

3. Diagrama del modelo relacional de la base de datos implementada.

Para responder a las dos primeras preguntas escribid el esquema de la base de datos como en el ejemplo siguiente:

nombreTabla(**attrbA**, attrbB, ...)
otraTabla(**attrb1**, attrb2 → nombreTabla.attrA, attr3, ...)
...

donde *nombreTabla* y *otraTabla* son los nombres de las tablas, la clave primaria está en negrita (en este caso es el atributo llamado *attrbA* en la tabla *nombreTabla* y el atributo llamado *attrb1* en la tabla *otraTabla*); las claves extranjeras (en este caso *attrb2*) apuntan al atributo que referencia (en este caso el atributo *attrA* de la tabla *nombreTabla*).

Vuestro diagrama del modelo relacional debe seguir un formato parecido al de la Figura 1 donde tanto las claves primarias como las extranjeras están marcadas explícitamente (PK → clave primaria, FK → clave extranjera). Así mismo, las tablas que contienen atributos relacionados están unidas mediante flechas.

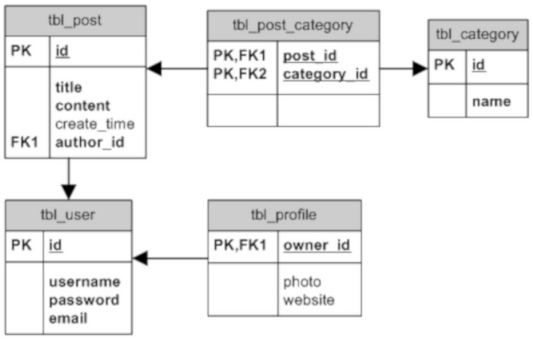

Figura 1: Ejemplo de diagrama relacional

3. Descripción de la base de datos

Cada reserva (booking) puede involucrar a varios pasajeros (passengers), cada uno con su billete (tickets) correspondiente. Los billetes incluyen información sobre los pasajeros de forma que un "pasajero" no es una identidad independiente. Si una misma persona aparece en dos billetes diferentes, en la base de datos se guarda la información como si la compra la hubieran realizado dos personas diferentes.

Los billetes (tickets) pueden incluir varios vuelos (ticket_flights). Bien sea porque son vuelos de conexión o porque es un viaje de ida y vuelta. Podéis asumir que todos los billetes en una reserva tienen los mismos vuelos.

Cada vuelo (flights) sale de un aeropuerto y llega a otro aeropuerto. Los vuelos con el mismo número de vuelo (flight_no) tienen el mismo origen y destino pero pueden tener diferente fecha de partida.

Al realizar el check-in, cada pasajero recibe una tarjeta de embarque (boarding_passes), donde se especifica su número de asiento. No pueden existir dos tarjetas de embarque para el mismo asiento en un vuelo dado.

El número de asientos (seats) en una aeronave y su distribución depende del modelo de avión (aircrafts) que realiza el vuelo. Se asume que cada modelo de avión tiene una única configuración posible.

4. Consultas

A continuación, se enumeran un conjunto de consultas a la base de datos que deberéis implementar usando SQL (leed las notas relativas a la implementación de las consultas antes de implementar esta parte). Por favor, NO añadáis el fichero `flight`.sql (o `flight`.sql.zip) al zip, el fichero es muy grande. Además hará imposible que entreguéis una copia de la práctica en *Moodle*.

1. Lista el número de reservas (bookings) que contengan un billete de ida y vuelta agrupadas por aeropuerto. Un billete es de ida y vuelta si el aeropuerto de origen del primer vuelo es idéntico al aeropuerto de destino del último vuelo. El resultado mostrará dos columnas con el código del aeropuerto de salida así

como el número de reservas que contienen billetes de ida y vuelta para ese aeropuerto. Igualmente, el resultado se ordenará de forma ascendente usando el atributo (departure_airport).

2. El precio de una reserva (booking.total_amount) puede ser calculado a partir del precio de cada vuelo (ticket_flights.amount). Crea una consulta que para cada reserva muestre el precio de la misma calculado a partir del valor almacenado en ticket_flights.amount. La salida debe mostrar tres columnas conteniendo los atributos booking.book_ref, booking.total_amount y el valor calculado por vuestra consulta. Ordena la consulta de forma ascendiente usando el atributo bookigs.book_ref

3. Crea una consulta que muestre por pantalla el código de aeropuerto y el número de pasajeros recibidos, y ordena la misma de forma ascendiente por el número de pasajeros recibidos. Se considera que un pasajero ha volado al aeropuerto si se ha emitido una tarjeta de embarque. Una tarjeta de embarque para un vuelo del aeropuerto X al Y incrementa en 1 el número de pasajeros recibidos en el aeropuerto Y.

4. Vuelo con más asientos vacíos. Muestra en la salida el atributo flight_id y el número de asientos vacíos. En caso de empate muestra todos los vuelos que hayan empatado.

5. Muestra reservas para los cuales no se haya emitido alguna de las tarjetas de embarque. La salida debe mostrar los atributos book_ref y flight_id. Ordenar el resultado de forma ascendiente por la pareja (book_ref, flight_id).

6. Por definición dos vuelos (flights) realizan el mismo trayecto si su número de vuelo (flight_no) es el mismo. Se solicita el trayecto donde la media de los retrasos acumulados sea mayor. Se define retraso como la diferencia entre los valores actual_arrival - scheduled_arrival. Mostrar en la línea de salida flight_no y el retraso medio. En caso de empate deben aparecer todos los trayectos con mayor retraso.

4.1. Notas relativas a la implementación de las consultas

Makefile

Se desea que las consultas se puedan ejecutar usando `make`. En particular `make query1` debería ejecutar la primera consulta, `make query2` debería ejecutar la segunda consulta y así sucesivamente.

Para conseguirlo, deberéis crear un fichero por cada consulta llamado `queryX.sql` (donde X es el número de la consulta) que contenga el código SQL y modificar en el fichero `makefile` las líneas que corresponda (por cada consulta):

```
query1:
  @echo consulta-1: "query␣description"
  @cat consulta1.sql | $(PSQL) | tee consulta1.log
```

Recordad que los espacios a la izquierda de cada orden en un fichero `makefile` son tabuladores.

Formateo de Consultas

Un buen código debe ser legible tanto para el programador que lo crea como para cualquier otra persona que revise el código. Por favor escribid las consultas de forma consistente usando sangrías y saltos de línea para mejorar su legibilidad. Antes de entregar el código pasarlo por un "formateador". Se sugiere usar `http://www.dpriver.com/pp/sqlformat.htm`. Si no os gusta buscad otro que os satisfaga y tenga una funcionalidad similar, e incluid el URL en la primera línea del `makefile` como un comentario.

4.1.1. ¿Cuándo es una consulta correcta?

Una consulta es correcta si satisface los siguientes requerimientos:

1. Se puede ejecutar en los ordenadores de los laboratorios usando los comandos definidos en el fichero `makefile` y descritos en esta memoria.

2. Proporciona el resultado correcto para **cualquier** instancia de la base de datos.

3. El código es eficiente y legible. Se acepta como código legible el obtenido usando la utilidad accesible en `http://www.dpriver.com/pp/sqlformat.htm` u otra utilidad semejante y accesible on-line en la dirección dada en la primera línea del fichero `makefile`. La consulta será considerada eficiente si el tiempo necesario para ejecutarla no es superior a 20 veces el tiempo requerido por la implementación de la consulta desarrollada por los profesores de esta asignatura.

Si cualquiera de estos puntos no se satisface, la consulta se calificará con cero puntos.

5. Rediseño de la Base de Datos

El diseño de la base de datos muestra múltiples deficiencias. Por ejemplo, no es posible configurar los asientos de forma diferente para cada vuelo.

Diseña una base de datos que evite el inconveniente citado. Incluye en la memoria: (1) el nuevo diagrama relacional y comenta cómo tus cambios solucionan el problema planteado y (2) añade un nuevo comando en el fichero `makefile` que se ejecute con `makefile nuevabase` y que borre todas las tablas y datos de la base de datos y cree las tablas de tu nuevo diseño (los comandos SQL necesarios para crear las tablas se almacenarán en un fichero llamado nuevabase.sql).

6. Resumen del material a entregar

Subir a *Moodle* el fichero obtenido al ejecutar el comando `practica1.zip` que debería contener:

1. Fichero `makefile` y todos los ficheros necesarios para crear la base de datos, borrarla, poblarla y ejecutar cada consulta. IMPORTANTE: **NO incluyáis el fichero** `flight.sql`.
2. Memoria en formato pdf que incluya: el esquema de la base de datos, el diagrama relacional de la base de datos y responda a las preguntas realizadas en esta práctica. Adicionalmente, para cada consulta en SQL incluid un listado con el código y una breve descripción de la implementación. No os olvidéis de incluir el material solicitado en el apartado 5

7. Criterios de corrección

Para aprobar es necesario:

- Que tanto el esquema como el diagrama relacional de la base de datos sean correctos. (Se admitirán un máximo de tres errores).
- Tener 3 consultas totalmente correctas.

Recordad que los criterios para decidir si una consulta es correcta están definidos en la subsección 4.1.1

Para obtener una nota en el rango 5-6.9 es necesario:

- Satisfacer todos los requerimientos del apartado anterior.
- Tener 5 consultas correctas

Para obtener una nota en el rango 7-7.9 es necesario:

- Satisfacer todos los requerimientos del apartado anterior.
- Que tanto el esquema como el diagrama relacional de la base de datos sean totalmente correctos.
- Igualmente, se espera que la memoria esté correctamente redactada y demuestre vuestra comprensión de la práctica.

Para obtener una nota en el rango 8-8.9 es necesario:

- Satisfacer todos los requerimientos del apartado anterior.
- Que todas las consultas sean correctas.

Para obtener una nota en el rango 9-10 es necesario:

- Satisfacer todos los requerimientos del apartado anterior.
- Haber rediseñado la base datos tal y como se solicita en la práctica.

NOTA: Cada día (o fracción) de retraso en la entrega de la práctica se penalizará con un punto.
NOTA: El código usado en la corrección de la práctica será el entregado en *Moodle*.