Università
Ca'Foscari
Venezia

Progetto Basi di Dati 19-20

Compagnia aerea - PlatypOS

MEMBRI GRUPPO

- Leonardo Mazzon 868445
- Giulio Nicola 875297
- Alessandro Piazza 869386

FUNZIONALITÀ

L'applicazione web fornisce funzionalità relative alla prenotazione di voli aerei. Le due categorie principali di utenti sono "cliente" e "operatore" con le seguenti funzioni:

Cliente:

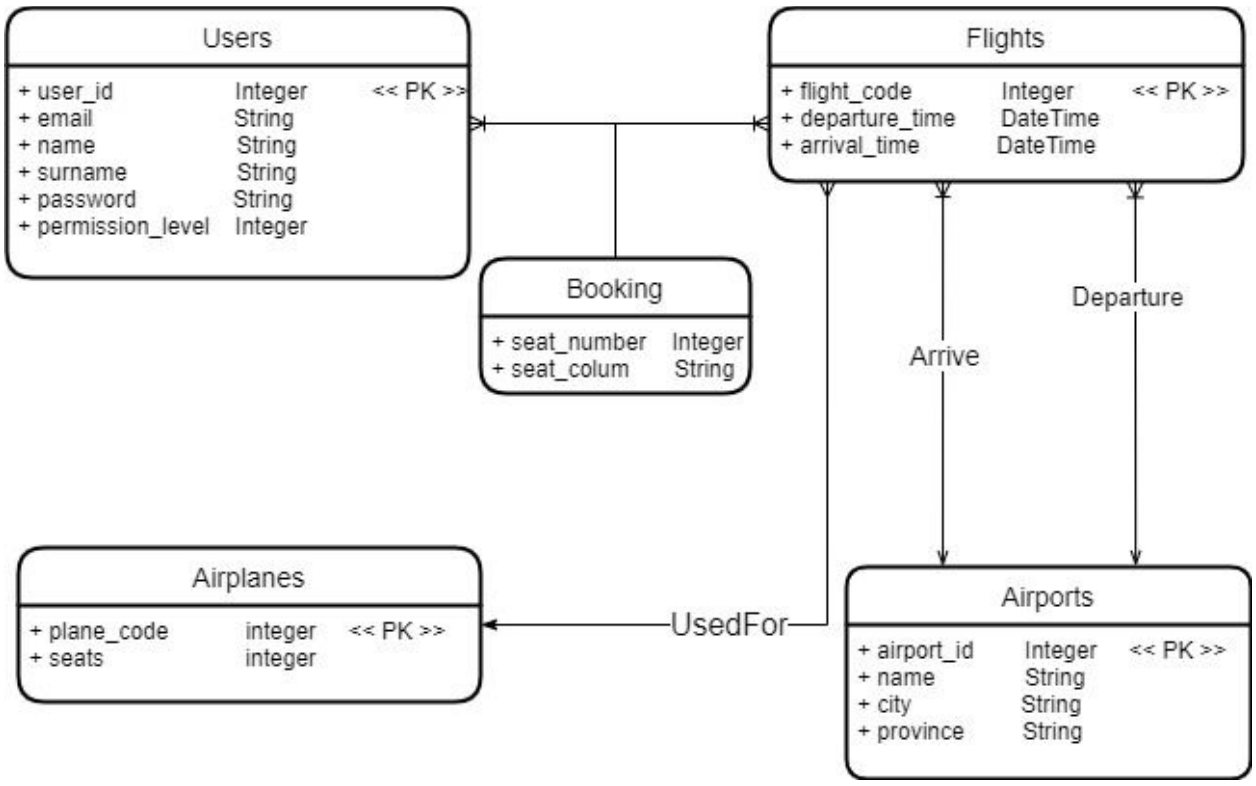
- Ricerca e prenotazione voli disponibili per andata e ritorno
- Visualizzazione prenotazioni
- Cambio password

Operatore:

- Inserzione di nuovi voli
- Inserzione aeroporti
- Inserzione aerei
- Ricerca voli disponibili
- Visualizzazione di statistiche

Il sito utilizza il framework Bootstrap per gestire le componenti grafiche, permettendo un interfaccia responsive che si adatta al dispositivo in uso.

DATABASE



Users(<u>user_id</u>, email, name, surname, password, permission_level)

Airports(<u>airport_id,</u> name, city, province)

Airplanes(<u>plane_code</u>, seats)

Flights(<u>fligh_code</u>, departure_time, arrival_time, *departure_airport**, *arrival_airport**, *plane_code**)

Bookings(<u>booking_id</u>, seat_column, seat_number, *user_id**, *flight_code**)

departure_airport,*arrival_airport* FK(Airports),*plane_code* FK(Airplanes)

user_id FK(Users),*flight_code* FK(Airplanes)

Airplanes → Flights 1:N, un aereo può essere usato per fare più voli, un volo ha un solo aereo.

User → Flights N:N, un utente può prenotare più voli e un volo ha più passeggeri.

Flights → Airports 1:N, un volo ha un solo aeroporto di partenza o di arrivo e un aeroporto ha più voli.

DESCRIZIONE

Abbiamo optato per l'utilizzo del DBMS **PostgreSQL**.

Il database rappresenta la gestione dei dati di una compagnia aerea che organizza voli. Ogni volo è identificato da un codice univoco, un aeroporto di partenza, uno di arrivo con le relative date. Di un aeroporto si conosce il nome, la città e la provincia. Ogni volo ha un solo aereo identificato da un codice e dai numero di posti totali disponibili per i passeggeri. Alla prenotazione di un volo da parte di un utente viene salvato il posto, identificato da una riga e una colonna. Di un utente si conosce l'email,il nome, il cognome, la password di accesso.

Gli utenti vengono distinti da un livello di permessi, i clienti hanno un livello 0 mentre gli operatori possono assumere valori maggiori di 0. Per questo progetto ci si è limitati ad assegnare il valore 1 per i permessi massimi. Per poter assegnare ad un utente il ruolo di operatore è necessario impostare i suoi permessi al valore 1 con una query o dall'interfaccia web del DBMS, all'inizializzazione del database non viene creato nessun utente e dal sito è possibile creare solo clienti.

Ogni posto è identificato da una colonna, tramite una carattere, e una riga, tramite un intero. Ogni aereo della compagnia ha sempre 5 colonne, quindi i posti totali di un aereo dovranno essere un multiplo di 5. Dovrà essere un multiplo di 5 per rendere standard la disposizione dei posti.

Il database di prova caricato con il progetto contiene tre utenti: admin@admin, mariorossi@gmail.com e billgates@outlook.com. Tutti e tre con password "password" per motivi di test. Il database ha inoltre dati di aeroporti, voli e prenotazioni già inseriti per comodità.

QUERY

Tramite SQLAlchemy Core le query vengono scritte sia in modalità Expression Language che Textual SQL. Si è preferito Textual SQL per query più particolari mentre Expression Language per query banali.

Ricerca dei voli

```
SELECT f.flight_code as flight_code ,a1.name as departure_airport ,
a2.name as arrival_airport , f.departure_time as departure_time,
f.arrival_time as arrival_time, a1.province as province_from,
a2.province as province_to, f.plane_code as plane_code, p.seats as
seats

FROM airports a1 JOIN flights f ON a1.airport_id = f.departure_airport
JOIN airports a2 ON a2.airport_id = f.arrival_airport JOIN airplanes p
ON f.plane_code=p.plane_code

WHERE a1.province='Treviso' AND a2.province='Venezia' AND
date(f.departure_time)='2020/08/31'AND f.flight_code

NOT IN (

      SELECT f1.flight_code

      FROM bookings b RIGHT JOIN flights f1 ON b.flight_code =
      f1.flight_code JOIN airplanes a1 ON f1.plane_code = a1.plane_code

      GROUP BY f1.flight_code, a1.seats

      HAVING COUNT(b.booking_id) >= a1.seats)
```

Questa query viene eseguita quando un utente ricerca un volo. Nella query sovrastante vengono visualizzati i voli che partono dalla provincia di Treviso e arrivano nella provincia di Venezia il 31 agosto 2020. Vengono fatti due join, il primo join per unire la tabella volo con la tabella aeroporto di partenza e il secondo join per unire la tabella con la tabella aeroporti di arrivo. Facendo così viene fatta una copia delle due tabelle aeroporti, quindi è possibile selezionare aeroporti diversi tramite il where, ritornando i campi necessari per l'identificazione. Il NOT IN serve per verificare che il codice del volo non sia presente tra i risultati della sottoquery che ritorna i voli con tutti i posti occupati.

Clienti fedeli

```
SELECT u.name, u.surname, b.user_id , COUNT(*) as flights_number

FROM bookings b JOIN users u ON b.user_id = u.user_id

GROUP BY b.user_id, u.name, u.surname

ORDER BY flights_number DESC

LIMIT 10
```

Questa è una query necessaria agli amministratori per individuare i 10 clienti che hanno effettuato più prenotazioni. Viene eseguito un join tra la tabella prenotazioni e la tabella utenti, per permettere di conoscere il nome e il cognome dei clienti. In seguito viene fatto un group by, raggruppando per id, nome e cognome dell'utente, che ci permettere di usare il count in modo tale da sapere quante prenotazioni sono state effettuate da un singolo utente. Tramite l'order by è possibile visualizzare in ordine decrescente il numero delle prenotazioni. Infine tramite il limit vengono visualizzati solo i primi 10 risultati.

Voli per anno

```
SELECT  CAST(date_part('year', departure_time) as int) as
years,COUNT(*) as flights_number

FROM flights

GROUP BY years

ORDER BY years desc
```

Questa query è necessaria agli amministratori per individuare quanti voli sono stati effettuati per ogni anno. Viene raggruppato per anno tramite il group by per contare tramite il count quanti voli sono stati effettuati. Il risultato viene ordinato in base all'anno in ordine decrescente mediante l'order by.Infine per visualizzare l'anno viene prima applicata la funzione date_part con il parametro year e successivamente applicata la funzione cast per convertirlo in intero dato che la funzione date_part ritorna un valore in double precision.

Città più visitate

```
SELECT a.province, COUNT(f.flight_code) as numero_voli

FROM airports a JOIN flights f ON a.airport_id = f.arrival_airport
JOIN bookings b ON b.flight_code = f.flight_code

GROUP BY a.province

ORDER BY numero_voli DESC

LIMIT 10
```

La query permette di visualizzare le città(o provincie) più popolari come destinazione. Vengono coinvolte la tabella degli aeroporti, per ottenere il nome della città, e la tabella dei voli per contare i voli e bookings per avere solo i voli con passeggeri. Siccome i voli hanno sia la FK per la partenza che per la destinazione dobbiamo effettuare il join solo su quest'ultima. Le città vengono raggruppate e viene conteggiato il numero di voli associati come destinazione ordinati in modo decrescente. Viene applicato un limite ai risultati.

Media prenotazioni per cliente per anno

```
SELECT CAST(date_part('year', f.departure_time) as int) as years ,

CASE WHEN COUNT(DISTINCT b.user_id) > 0

THEN CAST(COUNT(b.booking_id) as DOUBLE PRECISION)/CAST(COUNT(DISTINCT
b.user_id) as DOUBLE PRECISION)

ELSE 0 END as average

FROM flights f LEFT JOIN bookings b ON f.flight_code = b.flight_code

GROUP BY years
```

Questa query è necessaria agli amministratori per conoscere la media delle prenotazioni effettuate dai clienti per ogni anno.Viene fatto un left join tra voli e prenotazioni per visualizzare anche i valori NULL della tabella prenotazioni. Viene raggruppato per anno tramite il group by, per contare il numero di prenotazioni e il numero di utenti tramite il count. Prima di effettuare il calcolo della media si controlla tramite case when che il numero di persone sia maggiore di 0, perché non è possibile effettuare una divisione per 0, in caso il controllo risultasse falso viene ritornato 0. Funzione di cast per convertirlo in intero dato che la funzione date_part ritorna un valore in double precision.

Aerei con numero posti prenotati dal 2000 all'anno corrente

```
SELECT f1.plane_code, COUNT(f1.plane_code) as numero_posti
FROM airplanes a1 JOIN flights f1 ON a1.plane_code=f1.plane_code
JOIN bookings b1 ON f1.flight_code=b1.flight_code
WHERE CAST(date_part('year', f1.departure_time) as int) BETWEEN 2000
AND date_part('year', CURRENT_DATE)
GROUP BY f1.plane_code
ORDER BY numero_posti DESC
LIMIT 5
```

Questa query è necessaria agli amministratori per conoscere i 5 aerei che hanno trasportato più passeggeri.Viene fatto un join tra aerei , voli, prenotazioni. Tramite il where vengono filtrati i voli effettuati nell'arco temporale dal 2000 fino all'anno corrente compreso, raggruppando per aereo, il risultato sarà ordinato in modo decrescente e tramite il limit visualizzati solo i primi 5 risultati.

VINCOLI

Come vincoli si è scelto di impostare tutti gli attributi delle tabelle come *NOT NULL* poiché gli attributi inseriti sono fondamentali per la gestione della compagnia. Vincolo *UNIQUE* per il campo email dell'utente in modo tale da permettere di essere identificato dal sito nel log-in. Il vincolo *UNIQUE* è stato inoltre utilizzato nella tabella delle prenotazioni nei campi (`seat_column, seat_number, user_id`) per avere la certezza che il posto sia stato prenotato una sola volta, inoltre si è ritenuto opportuno utilizzare dei *CHECK* nella tabella flights per far gestire al database operazioni del tipo: data di arrivo di un volo siano maggiori della data di partenza, e , che l'aeroporto di arrivo sia diverso da quello di partenza.

TRANSAZIONI

Per le transazioni abbiamo utilizzato il metodo di SQLAlchemy Core *`engine.connect()`* insieme alle opzioni per specificare il livello di isolamento. Questo permette una gestione automatica in caso di conflitto. Si è cercato di mantenere il database con livelli di isolamento non troppo stretti.

FLASK

STRUTTURA

Il progetto è stato strutturato cercando di seguire le linee guida di Flask, adattandole alle dimensioni del progetto. Sono state utilizzate le blueprint per suddividere le parti principali del sito: homepage con prenotazione(*main/*) e gestione degli utenti(*users/*). La cartella *templates/* contiene i file html che rappresentano le varie pagine, mentre la cartella *static/* contiene css e javascript personalizzati.

Il file target per l'esecuzione è *run.py* mentre *__init__.py* contiene l'inizzializzazione di vari elementi dell'app. Sono presenti altri file *__init__.py* dentro le cartelle *main/* e *users/* che servono a visualizzare queste ultime come packages per le blueprint.

Il file *models.py* contiene la definizione della classe Users per flask-login, la connessione con il database e la definizione delle tabelle. Il modulo di SQLAlchemy si occuperà di creare tali tabelle a patto che esista già un database+password come specificati nel *uri*.

ACCESSO E SICUREZZA

L'accesso agli account è gestito da **flask-login** tramite le sessioni. Quando l'utente è autenticato avrà accesso solo alle pagine consentite al suo ruolo, nel caso cerchi di accedere a pagine riservate ad altri utenti verrà reindirizzato. È possibile ottenere le informazioni dell'utente autenticato tramite dei metodi come `get_name()` o `get_email()` forniti dalla classe Users.

L'accesso alle pagine è stato regolato tramite il metodo `is_authenticated` e `get_permission()` che permettono di effettuare redirect efficaci in base alle richieste.

L'autenticazione viene affiancata da **flask-bcrypt** per creare un hash delle password in fase di accesso e registrazione tramite la funzione di hashing bcrypt, cifratura basata su Blowfish.

Quando un utente crea un account viene direttamente creato l'hash della password e durante l'accesso viene comparato l'hash della password inserita e quello nel database. Bcrypt utilizza il "salt" per salvare le password quindi ogni hash sarà diverso.

La password dell'utente viene obbligata ad essere lunga almeno 8 caratteri, non è un vincolo presente nel database e può essere disabilitato. È utile per indirizzare l'utente ad un corretto uso delle password. La lunghezza finale nel database è sempre di 60 caratteri a causa della funzione di hashing.