

Airline Management System

Tema 2 Baze de date:

Dezvoltarea unei baze de date complexe

Sistemul de gestiune a bazelor de date (*Database Management System - DBMS*) dedicat Managementului Companiilor Aeriene este conceput pentru a asigura persistență și interconectarea datelor specifice multiplelor companii aeriene și a zborurilor aferente. Obiectivul principal al sistemului este de a gestiona integral procesele operaționale prin oferirea unui mediu robust care să garanteze integritatea datelor. În cele ce urmează sunt vor fi detaliate: structura bazei de date, soluțiile de implementare tehnice specifice sistemului, precum și cerințele suplimentare implementate.

1. Tables

Această secțiune detaliază structura de reținere a datelor pentru fiecare entitate distinctă din cadrul Sistemului de Gestie a Companiilor Aeriene, incluzând specificațiile atributelor și definirea tipurilor de date asociate. Fiecare tabela conține cheie primara, care va fi folosită pentru integrarea corecta a datelor; acest concept va fi detaliat în secțiunea 2: "Constraints".

1.1 Company

Aceasta tabela are rolul de a retine numele companiei, precum și ID-ul ei unic, astfel putând retine fiecare detaliu al companiei.

```
CREATE TABLE company(
    company_id SERIAL PRIMARY KEY
    company_name VARCHAR(20)
);
```

1.2 Aircraft

Aceasta tabela are rolul de a retine date fundamentale fiecărei aeronave folosite in sistem.

```
CREATE TABLE aircraft(  
    aircraft_id SERIAL PRIMARY KEY,  
    aircraft_model VARCHAR(25),  
    aircraft_mass_tollerance INT,  
    aircraft_passengers_capacity INTEGER,  
    aircraft_cargo_capacity INTEGER,  
    aircraft_crew_members INTEGER,  
    aircraft_year_of_made DATE,  
    aircraft_flights_accomplished INTEGER,  
    aircraft_company_id INT,  
    FOREIGN KEY (aircraft_company_id) REFERENCES company(company_id));
```

- mass_tollerance: masa maxima admisa a aeronavei
- passenger_capacity: numărul maxim de pasageri admisi
- cargo_capacity: masa maxima admisa pentru bagaje
- crew_members: numărul membrilor echipajului solicitat
- year_of_made: anul productiei
- flights_accomplished: numărul total de zboruri realizat
- company_id: id-ul companiei de care apartine aeronava

1.3 Crew

In aceasta tabela nu s-au reținut date specifice, având exclusiv rolul de a retine id-ul specific fiecărui membru din echipaj.

```
CREATE TABLE crew(  
    crew_id INT PRIMARY KEY  
)
```

1.4 Pilot

Tabela creata pentru reținerea informațiilor specifice fiecărui pilot; asemănător acestei structuri au fost create tabelele “copilot” si “steward”.

```
CREATE TABLE pilot(
    pilot_id SERIAL PRIMARY KEY,
    pilot_first_name VARCHAR(12),
    pilot_last_name VARCHAR(12),
    pilot_nationality VARCHAR(12),
    pilot_date_of_birth DATE,
    pilot_experience INTEGER,
    crew_id INTEGER
    foreign key (crew_id_fk) references crew(crew_id));
```

-crew_id: face referire la echipajul căruia-i apartine pilotul.

1.5 Copilot

```
CREATE TABLE copilot (
    copilot_id SERIAL PRIMARY KEY,
    copilot_first_name VARCHAR(12),
    copilot_last_name VARCHAR(12),
    copilot_nationality VARCHAR(12),
    copilot_date_of_birth DATE,
    crew_id INTEGER,
    FOREIGN KEY(copilot_id) REFERENCES crew(crew_id)
);
```

1.6 Steward

```
CREATE TABLE steward(
    steward_id SERIAL PRIMARY KEY,
    steward_first_name VARCHAR(12),
    steward_last_name VARCHAR(12),
    steward_nationality VARCHAR(12),
    steward_date_of_birth DATE,
    crew_id INTEGER,
    FOREIGN KEY(steward_id) REFERENCES crew(crew_id));
```

1.7 Passenger

In tabela passenger au fost reținute date specifice fiecărui pasager, cum ar fi numele, vârsta etc. , cat si date adiționale importante pentru o integrare corecta a acestora.

```
CREATE TABLE passenger(
    passenger_id SERIAL PRIMARY KEY,
    passenger_first_name VARCHAR,
    passenger_last_name VARCHAR,
    passenger_date_of_birth DATE,
    passenger_nationality VARCHAR(12),
    passenger_flight_id INTEGER,
    FOREIGN KEY (passenger_flight_id) REFERENCES flight(flight_id),
    passenger_discount INTEGER,
    passenger_kids INTEGER,
    passenger_crime_record VARCHAR);
```

- flight_id: zborul pasagerului
- discount: reducere aferenta pasagerului așa prețul zborului (daca exista)
- kids: numărul de copii (daca exista)
- criminal record: cazier (daca exista)

1.8 Flight

Aici s-au reținut doar datele relevante fiecare zbor in parte, utile in mod special pentru pasageri; detalii ulterioare au fost implementate in tabela “Flight id”.

```
CREATE TABLE flight(
    flight_id SERIAL PRIMARY KEY,
    flight_duration VARCHAR (10),
    departure_airport_id INTEGER,
    arrival_airport_id INTEGER,
    FOREIGN KEY (departure_airport_id) REFERENCES airport(airport_id),
    FOREIGN KEY (arrival_airport_id) REFERENCES airport(airport_id),
    flight_price DECIMAL (4, 2));
```

- airport_id: acest parametru a fost folosit de doua ori, pentru stocarea aeroporturilor dictate zborului
- flight_price: se retine prețul biletului pentru calcularea costului total al fiecărui pasager

1.9 Flight_schedule

Aici sunt reținute date specifice zborului dedicăte companiilor, pentru un “brief” asupra modului în care acesta se va desfășura.

```
CREATE TABLE flight_schedule(
    flight_schedule_id INTEGER,
    flight_aircraft_id INTEGER,
    FOREIGN KEY (flight_schedule_id) REFERENCES flight(flight_id),
    FOREIGN KEY (flight_aircraft_id) REFERENCES aircraft(aircraft_id),
    flight_passengers INTEGER);
```

- aircraft_id: reține id-ul aeronavei folosit pentru zbor
- flight_passengers: numărul total de pasageri participanți (a nu se confunda cu numărul maxim de pasageri al aeronavei)

1.10 Airport

În tabela următoare au fost reținute datele specifice fiecărui aeroport.

```
CREATE TABLE airport(
    airport_id SERIAL PRIMARY KEY,
    airport_name VARCHAR(20),
    airport_country VARCHAR(20),
    airport_city VARCHAR(20),
    airport_runways INT,
    airport_time_zone VARCHAR);
```

- runways: numărul de piste disponibile aeroportului
- time_zone: fusul orar al tării adiacente acestuia

2. Constraints

Fiecare entitate din modelul de date i s-a alocat un identificator unic prin definirea constrangerii de cheie primara (PRIMARY KEY). Acest mecanism fundamental a permis:

- garantarea integritatii entitatii si a unicitatii inregistrarilor
- stabilirea eficienta a relatiilor prin utilizarea cheilor externe (FOREIGN KEY)
- definirea unei logici operationale interne a sistemului.

Pornind de la tabela “company”, acesteia i-au fost atribuite doar doua variabile:

- nume, de tip VARCHAR
- ID unic, de tip SERIAL PRIMARY KEY.

De aceasta tabela se leaga ID-ul aeronavelor din tabela “Aircraft”, crearea aeronavei fiind posibila doar prin atribuirea unei companii existente, in caz contrar apărând eroare in sistem. In acest mod, unei companii pot fi atribuite mai multe aeronave, insa unei aeronave doar o singura companie (legare 1 la N).

Codul folosit pentru referinta:

FOREIGN KEY (aircraft_company_id) REFERENCES company (company_id)

Ca o ultima observatie, prin legarea aeronavelor de companii, este subinteleasa apartenenta zborurilor pentru o anumita companie, economisind memorie si sporind astfel integritatea datelor.

Tabela “Flight” are doua referinte, reprezentand ambele aeroporturi aferente zborului;

Acstea s-au realizat cu doua FOREIGN KEY-uri, si in mod logic un zbor nu se poate crea daca nu face referire la doua aeroporturi valide din baza de date.

Codul folosit pentru legarea zborului este:

FOREIGN KEY (departure_airport_id) REFERENCES airport(airport_id),

FOREIGN KEY (arrival_airport_id) REFERENCES airport(airport_id).

Prin urmare, tabela “Flight_schedule” preia informatii importante asupra zborului precum si aeronava folosita (legat de observatia precedenta), folosind in același mod doua referiri cheilor primare din fiecare obiect:

FOREIGN KEY (flight_schedule_id) REFERENCES flight(flight_id),

FOREIGN KEY (flight_aircraft_id) REFERENCES aircraft(aircraft_id).

In tabela “Passenger” avem o referinta catre tabela “Flight”, acest lucru fiind vital pentru stabilirea zborului aferent pasagerului, cat si pentru controlul si integritatea pasagerilor participant la un anumit zbor, in acest sens putand tine numaratoarea.

Cod: FOREIGN KEY (passenger_flight_id) REFERENCES flight(flight_id).

Din fiecare dintre tabelele “Pilot”, “Copilot” si “Steward” a fost direcționată o referință către tabela “Crew”, astfel fiecare dintre instanțele fiecărui obiect putând forma un echipaj, căruia la rândul lui i se va atribui un ID specific.

Acestea sunt cheile străine folosite pentru integrarea datelor, asigurând unicitatea acestora în baza de date aferentă, și o utilizare corespunzătoare a acestora.

3.Functions

Pentru administrarea și manipularea seturilor de date aferente, sistemul de gestiune a bazei de date utilizează o serie de funcționalități integrate, precum și funcții definite de utilizator, esențiale pentru optimizarea proceselor de prelucrare a informațiilor.

3.1 Count

Aceasta este o funcție prestabilită de SQL, care are rolul de a număra elemente (SELECT COUNT), pe baza unei condiții explicate (WHERE), pe coloana dorită (FROM). În acest caz am folosit funcția aferentă pentru a număra zborurile programate pentru o anumita data din an.

3.2 Ceck In

Aceasta funcție are scopul de a returna un set de date aferente unei tabele interesante, sub formă de tabel. În acest caz se dorește afișarea numelui și prenumelui pasagerului, precum și zborului asociat, asigurând pasagerului ca acesta a fost introdus în baza de date. Linia de cod “LANGUAGE SQL” indică faptul că nu au fost folosite funcții prestabilite de un alt limbaj (cum ar fi plpgsql), ci doar cod aferent SQL.

O observație importantă este că în funcții și în procedurile care urmează să fie prezentate în acest document a fost folosită linia de cod “AS \$\$” pentru schimbarea delimitatorului. În acest caz a fost modificat și un parametru de securitate, care va fi discutat anterior, căruia îi conferă pasagerului control limitat, în acest sens fiind creată și o politică.

3.3 Update Timestamp

Aceasta funcție a fost realizata pe urma introducerii unei coloane in tabela “passenger”, numita “last_modified”, cu scopul de a ilustra data ultimei modificări asupra unui anumit pasager; aceasta este folosita pe urma declanșării unui trigger, asigurând ca înainte de introducerea datelor într-o linie a tablei “passenger”, sa fie modificata valoarea “last_modified” in data si ora curenta.

3.4 Update Flight Occupancy

Aceasta funcție are un rol asemănător cu funcția “Update Timestamp”, asigurând ca numărul de pasageri aferent unui zbor este modificat pe urma inserării sau ștergerii unui pasager din tabela acestora; acest lucru asigura o precizie constantă asupra numărului efectiv de pasageri, nefiind necesara modificarea manuală in tabela “Flight Schedule”. Funcționarea acesteia este pusa in practica pe urma declanșării unui trigger, renumărarea pasagerilor fiind realizata pe urma ștergerii sau adăugării unui pasager.

4. Procedures

În cuprinsul acestei secțiuni vor fi prezentate în detaliu procedurile stocate și funcțiile definite de utilizator integrate în baza de date supusă analizei. Se va acorda o atenție deosebită specificațiilor operaționale și rolului acestora în automatizarea proceselor și în optimizarea performanței generale a sistemului.

4.1 Insert Passenger

Aceasta procedura a fost implementata cu scopul adăugării de noi pasageri unui anumit zbor, precum și datele personale ale acestuia; primește ca parametrii valorile dorite pentru adăugarea

acestuia, fiind obligatorie inserarea ID-ului pentru zborul dorit. In acest sens au fost luate masuri de securitate.

4.2 Add Flight

Aceasta procedura este mai complexa din punct de vedere al codului folosit, fiind folosit limbajul plpgsql. Se începe prin a declara o noua variabila in interiorul procedurii, care va primi ca valoare ID-ul noului zbor, in cazul in care condiția de egalitate a datelor nu este îndeplinită, asigurând ca nu sunt doua zboruri cu aceeași data si ora exacta (condiție verificata cu ajutorul funcției PERFORM). Daca data si ora este egala cu un zbor deja existent, va fi “aruncata” o eroare si procedura se va opri. Continuând, sunt introduse valorile in noul zbor, fiind returnata valoarea cheii primare in variabila declarata precedent (cheia primara este generata automat). Se procedează apoi prin crearea unui nou obiect in tabela Flight Schedule care va retine cheia zborului cat si cea a aeronavei, precum si anumite date prestabile.

5. Joins and Views

Pentru a facilita extragerea de informații complexe si a reconstituи seturi de date coerente din multiple tabele normalize, au fost utilizate operațiuni de îmbinare (JOIN), precum si proceduri de afișare (VIEW).

5.1 Avioanele Companiilor

Acet view alcătuiește informațiile cuprinse in tabela “Aircraft” asupra aeronavei, pe baza unui join cu tabela company. Folosind ca relație de adevar cheia aeronavei care face referință la cheia companiei, fiecare aeronava este etichetata pe baza numele companiei aferente, sub forma unui tabel.

5.2 Detalii zbor

In acest view au fost selectate detaliiile zborurilor cat si numele aeroporturilor, pe baza mai multor join-uri; in acest caz au fost necesare doua join-uri pe aceeași tabela “Airport”, deoarece accesul numelor a fost realizat cu ajutorul ID-ului fiecărui dintre acestea, iar tabela este unica. In acest sens au fost redenumite tabele temporar cu funcția AS, putând fi realizabila distingerea de către sgbd. Al treilea join a fost realizat cu scopul de a culege date aferente zborului din tabela “Flight Schedule” prin intermediul condiției de egalitate a cheilor, asemănător cu cea a aeroporturilor. In plus, a fost folosit un ORDER BY, care a realizat așezarea liniilor pe baza datei zborurilor, in mod crescător.

5.3 Membrii echipaj

Acest “view” selectează și afișează datele aferente fiecărui membru al fiecărui echipaj; pentru o mai bună vizuire a datelor a fost folosita funcția STRING AGG care unește numele și prenumele fiecărui “steward” într-o singura coloana. In acest sens au fost folosite trei join-uri pentru fiecare membru in parte, grupându-i după datele piloților și copiloților.

5.4 Detalii pasageri

Asemănător cu “Membrii echipaj”, view-ul “Detalii pasageri” a fost ideat pentru afișarea datelor fiecărui pasager, precum și pentru anumite calcule necesare pentru o logica îndeplinită a bazei de date. Variabila “total_cost” retine costul total pentru fiecare pasager in parte, calculul fiind următorul:

pretul biletului + (prețul biletului * număr copii) -30% -discount

In acest sens a fost aplicata o reducere standard fiecărui copil (reducere ce se aplica doar in cazul in care copii au fost declarati la cumpărarea biletului, ci nu ulterior); in plus se aplica reducerea costului pasagerului in cazul in care acesta exista.

A fost folosita o îmbinare asupra tablei “Flight”, pentru a vizualiza date specifice zborului selectat de către pasager.

6. Security

Pentru protejarea datelor au fost implementate masuri de securitate, in mod specific functiilor si procedurilor. Acest lucru a permis ca datele sa nu fie accesate de persoane neautorizate, pentru a fi distruse sau folosite in mod nefiresc.

Fiecarui functie cat si proceduri i-au fost implementate definiții “SECURITY SQL” aferente, pentru a specifica contextul de manipulare. De asemenea a fost creat si un role “l” pentru testare.

A fost folosita functia de “GRANT” pentru a permite accesul unui utilizator specific, cum ar fi admin (postgres), si “REVOKE” pentru a anula anumite permisiuni, acest lucru garantand accesul exclusiv unor utilizatori specifici.

Pentru gruparea unor anumite entitati a fost creata grupa “passenger_role”, care a fost folosita in definitia politicii “only_own_ticket”.

6. Cerințe suplimentare

6.1 Forma nominală a treia (3NF)

A fost efectuat procesul de normalizare a schemei bazei de date, asigurând ca toate tabelele respectă forma normală a treia (3NF) prin eliminarea completă a dependentelor funcționale tranzitive. Acest lucru a fost realizat prin descompunerea tabelelor, garantând ca fiecare atribut non-cheie depinde direct și integral doar de cheia primara.

6.2 Crearea de vederi (views)

Au fost implementate multiple vederi (views) care au contribuit la simplicitatea accesului datelor complexe. În acest sens a fost îmbunătățită performanța interogărilor frecvente, menținând astfel integritatea și confidențialitatea datelor implicate.

6.3 Utilizarea de proceduri stocate (procedures)

Implementarea multipla de proceduri stocate în baza de date subiacentă a permis execuția operațiunilor critice repetitive, centralizând logica de Airline Management. De asemenea au fost luate măsuri de securitate asupra modului de accesare și datele implicate în interiorul procedurilor.

6.4 Declanșatori (triggers)

Au fost adăugați declanșatori pentru integritatea datelor și manipularea eficientă a acestora, asigurând o bună gestionare a întregului proces de ștergere/adăugare de pasageri.

6.5 Securitate

S-a acordat atenție sporita securității datelor, asigurând o protecție solida asupra potențialelor manipulări în interiorul bazei de date. Acest lucru a contribuit la o mai bună înțelegere a întregului proces de management asupra bazei de date supuse analizei.

6.6 Documentație

În final s-a realizat o documentație asupra bazei de date complexe “Airline Management system”, cu rolul de a crea un “grand view” asupra tuturor tabelelor și implementărilor respective, direcționat eventualilor utilizatori.

