Gruppuppgift

Uppgift

Scenario:

Ni arbetar som utvecklare på ett företag som behöver en enkel bokningslösning för att hantera aktiviteter och användare.
Målet är att designa och implementera en fungerande relationsdatabas i PostgreSQL som uppfyller grundläggande krav på funktionalitet och struktur.

Kravspecifikation:

Databasdesign:

Skapa en ER-modell som inkluderar följande tabeller:
Users = ID, namn, e-post (ska vara unikt).
Events = ID, titel, datum, kostnad per deltagare, max antal deltagare.
Bookings = Kopplingstabell som visar vilken användare som bokat vilket event.

Implementering i PostgreSQL:

Skapa databasen och tabellerna baserat på ER-modellen.
Lägg till PRIMARY KEY och FOREIGN KEY för att definiera relationer mellan tabellerna.
Implementera constraints för att säkerställa att e-postadresser är unika och att antalet bokningar per event inte överskrider maxantalet.

SQL-frågor:

Skriv SQL-frågor för att:
Hämta alla bokningar (användare och event).
Visa alla events med deras deltagare.
Lägg till en ny bokning.
Ta bort en bokning.
Beräkna den totala intäkten för ett specifikt event baserat på antalet deltagare och kostnaden per deltagare.

Grundläggande Säkerhet:

Begränsa antalet deltagare per event till det angivna maxantalet.

Examinationskriterier för Godkänt (G):

Databasdesign: En ER-modell som visar korrekt struktur och relationer mellan tabeller.
Implementering: Tabeller är korrekt skapade med PRIMARY KEY och FOREIGN KEY.
SQL-frågor: Frågorna fungerar som förväntat och returnerar korrekta resultat.

Säkerhet: Begränsningen på max antal deltagare per event är korrekt implementerad.
Unika e-postadresser: Implementeringen säkerställer att e-postadresser är unika.
Aggregerad data: Beräkningen av total intäkt per event fungerar korrekt.

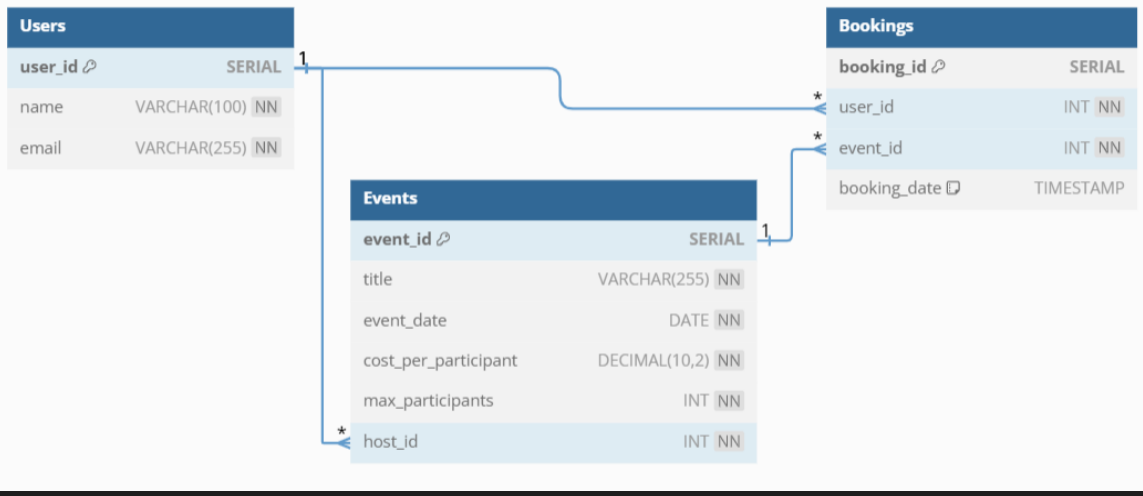
Leverabler:

En ER-modell (diagram och kort beskrivning).
SQL-kod för att skapa och hantera databasen.
SQL-frågor som uppfyller uppgiftskraven.
Presenation av Grupparbetet [MER INFO KOMMER]

Lösning

Databasdesign och Implemetering i PostGreSQL

ER-Diagram




```
Table Users {
  user_id SERIAL [primary key]
  name VARCHAR(100) [not null]
  email VARCHAR(255) [unique, not null]
}
Table Events {
  event_id SERIAL [primary key]
  title VARCHAR(255) [not null]
  event_date DATE [not null]
  cost_per_participant DECIMAL(10,2) [not null]
  max_participants INT [not null]
  host_id INT [not null, ref: > Users.user_id]  // Foreign Key till Users
}
Table Bookings {
  booking_id SERIAL [primary key]
  user_id INT [not null, ref: > Users.user_id]   // Foreign Key till Users
  event_id INT [not null, ref: > Events.event_id] // Foreign Key till Events
  booking_date TIMESTAMP [default: `CURRENT_TIMESTAMP`]
}
```

Beskrivning

Users:

Lagrar personer som kan boka eller skapa event.

Events:

Lagrar event med titel, datum, pris och max antal deltagare.

Har en host_id som visar vem som skapat eventet.

Bookings:

Kopplar användare och event, visar vilka som bokat vilka event.

Relationer

Users ↔ Events (1:N) = En användare kan skapa flera event, men varje event har bara en värd.
Users ↔ Bookings (1:N) = En användare kan boka flera event, men varje bokning hör till en användare.
Events ↔ Bookings (1:N) = Ett event kan ha flera bokningar, men varje bokning gäller bara ett event.

Begrepp

Not Null = Förhindrar att en kolumn innehåller NULL-värden, det vill säga att den måste ha ett värde (Används för att säkerställa att obligatoriska fält fylls i).
Primery Key = Identifierar unikt varje rad i en tabell (måste vara unikt och får aldrig vara NULL).
Foreign Key = Skapar en relation mellan två tabeller (ser till att en kolumn bara innehåller värden som finns i en annan tabell).
SERIAL = Skapar en automatiskt ökande unik ID-kolumn.
TIMESTAMP = Lagrar datum och tid (exakt ner till millisekunder).
VARCHAR(n) = Lagrar textsträngar upp till en viss längd, n = max antal tecken (tar bara upp så mycket plats som behövs, till skillnad från CHAR).
INT = Lagrar heltal (-2,147,483,648 till 2,147,483,647).
$$ LANGUAGE plpgsql = Den här syntaxen används för att definiera en lagrad funktion (stored function) i PostgreSQL med PL/pgSQL, som är ett programmeringsspråk inbyggt i PostgreSQL.

Skapning av databasen

Vi skapar en relationsdatabas i PostgreSQL med tre tabeller:
Users (för användare).
Events (för evenemang).
Bookings (för bokningar mellan användare och event).

Kod och förklaring

Users:

```
CREATE TABLE Users (
  user_id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL
);
```

user_id SERIAL PRIMARY KEY = Skapar en unik ID-kolumn (user_id) som ökar automatiskt för varje ny användare.
name VARCHAR(100) NOT NULL = Skapar en kolumn för användarnamn (max 100 tecken), som måste fyllas i (NOT NULL).

email VARCHAR(255) UNIQUE NOT NULL = Skapar en kolumn för e-post, som måste vara unik och måste fyllas i (NOT NULL).
Events:

```
CREATE TABLE Events (
event_id SERIAL PRIMARY KEY,
title VARCHAR(255) NOT NULL,
event_date DATE NOT NULL,
cost_per_participant DECIMAL(10,2) NOT NULL,
max_participants INT NOT NULL CHECK (max_participants > 0),
host_id INT NOT NULL, FOREIGN KEY (host_id) REFERENCES Users(user_id) ON
DELETE CASCADE );
```

event_id SERIAL PRIMARY KEY = Skapar en unik ID-kolumn (event_id) som ökar automatiskt för varje nytt event.
title VARCHAR(255) NOT NULL = Skapar en kolumn för eventets titel (max 255 tecken), som måste fyllas i (NOT NULL).
event_date DATE NOT NULL = Lagrar datumet för eventet och måste fyllas i (NOT NULL).
cost_per_participant DECIMAL(10,2) NOT NULL = Lagrar priset per deltagare i eventet (ex. 150.00 SEK).
max_participants INT NOT NULL CHECK (max_participants > 0) = Begränsar antalet deltagare per event och ser till att värdet måste vara positivt.
host_id INT NOT NULL = Lagrar ID:t för eventets värd (host), som måste vara en användare i Users.
FOREIGN KEY (host_id) REFERENCES Users(user_id) ON DELETE CASCADE = Kopplar eventet till en användare (Users.user_id).
Om användaren tas bort, tas även eventen bort (ON DELETE CASCADE).
Bookings:

```
CREATE TABLE Bookings (
booking_id SERIAL PRIMARY KEY,
user_id INT NOT NULL,
event_id INT NOT NULL,
booking_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
FOREIGN KEY (user_id) REFERENCES Users(user_id) ON DELETE CASCADE,
FOREIGN KEY (event_id) REFERENCES Events(event_id) ON DELETE CASCADE,
CONSTRAINT unique_booking UNIQUE (user_id, event_id)
```

booking_id SERIAL PRIMARY KEY = Skapar en unik ID-kolumn (booking_id) för varje bokning.
user_id INT NOT NULL = Lagrar ID:t för användaren som bokar eventet (måste finnas i Users).

event_id INT NOT NULL = Lagrar ID:t för eventet som bokas (måste finnas i Events).
booking_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP = Sparar datum och tid för bokningen automatiskt när den görs.
FOREIGN KEY (user_id) REFERENCES Users(user_id) ON DELETE CASCADE = Kopplar bokningen till en användare.
Om användaren tas bort, tas också bokningarna bort (CASCADE).
FOREIGN KEY (event_id) REFERENCES Events(event_id) ON DELETE CASCADE = Kopplar bokningen till ett event.
Om eventet tas bort, tas också bokningarna bort.
CONSTRAINT unique_booking UNIQUE (user_id, event_id) = Förhindrar att samma användare bokar samma event fler än en gång.

Vad är syftet med dessa tabeller?

Users lagrar användare med en unik e-postadress.
Events lagrar evenemang och är kopplade till en användare som värd.
Bookings hanterar bokningar och skapar en många-till-många-relation mellan Users och Events.
Om en användare tas bort, tas även bokningar och event där användaren är värd bort (ON DELETE CASCADE).

Triggers

Skapar en funktion check_max_participants() som:
Räknar antalet befintliga bokningar för ett event.
Jämför antalet bokningar med eventets maxantal (max_participants).
Om maxgränsen är nådd, kastar den ett felmeddelande och bokningen stoppas.
Skapar en trigger enforce_max_participants som:
Anropar funktionen varje gång en ny bokning görs.
Stoppar bokningen om maxgränsen är uppnådd.

```
CREATE OR REPLACE FUNCTION check_max_participants()
RETURNS TRIGGER AS $$
BEGIN
```

Skapar en funktion i PostgreSQL.
RETURNS TRIGGER betyder att funktionen används med en trigger.

```
IF (SELECT COUNT(*) FROM Bookings WHERE event_id = NEW.event_id) >=
  (SELECT max_participants FROM Events WHERE event_id = NEW.event_id) THEN
   RAISE EXCEPTION 'Max antal deltagare för detta event har uppnåtts.';
```

Funktionen räknar hur många bokningar som finns för eventet (COUNT(*) från Bookings).
Jämför med eventets maxantal (max_participants från Events).
Om bokningsantalet är lika med eller större än maxantalet → kastas ett felmeddelande.

```
  END IF;
  RETURN NEW;
```

END;
$$ LANGUAGE plpgsql;
Om bokningen är tillåten, returnerar den den nya raden (NEW), vilket betyder att bokningen sparas.
CREATE TRIGGER enforce_max_participants BEFORE INSERT ON Bookings FOR EACH ROW EXECUTE FUNCTION check_max_participants();
BEFORE INSERT → Triggern körs innan en ny bokning infogas i Bookings.
FOR EACH ROW → Den körs för varje ny rad som försöker läggas in.
EXECUTE FUNCTION check_max_participants(); → Kör funktionen check_max_participants().

Slutsats

Users är kopplad till både Events och Bookings.
Events och Users hänger ihop via host_id (eventets skapare).
Bookings fungerar som en kopplingstabell mellan Users och Events, eftersom en användare kan delta i flera event.

SQL Frågor

Förklaring

Fråga1:
SELECT
b.booking_id,
u.name AS user_name,
e.title AS event_title
FROM bookings b
INNER JOIN users u
ON u.user_id = b.user_id
INNER JOIN
events e ON e.event_id = b.event_id;
Hämtar alla bokningar tillsammans med användarnamn och eventtitel.
INNER JOIN används för att koppla bookings till users och events.
Fråga2:
SELECT
e.event_id,
e.title AS event_title,
u.name AS user_name
FROM events e
INNER JOIN bookings b
ON b.event_id = e.event_id
INNER JOIN users u

ON u.user_id = b.user_id;
Hämtar alla event tillsammans med namnen på deltagarna.
INNER JOIN används för att koppla events, bookings och users.
Fråga3:
INSERT INTO bookings (user_id, event_id, booking_date) VALUES (
9,
3,
'2025-03-15'
);
Lägger till en ny bokning där användaren (user_id = 9) bokar eventet (event_id = 3).
booking_date sätts till 2025-03-15.
Möjliga fel = Om UserID 9 eller EventsID 3 inte finns (kontrollera att både användare och event finns).
SELECT * FROM users WHERE user_id = 9;
SELECT * FROM events WHERE event_id = 3;
Fråga4:
DELETE FROM bookings WHERE booking_id = 1;
Tar bort en bokning med booking_id = 1 från bookings.
Möjliga fel = Om BookingID 1 inte finns (kontrollera om bookningen finns).
SELECT * FROM bookings WHERE booking_id = 1;
Fråga5:
SELECT
e.title AS event_title,
e.cost_per_participant AS ticket_price,
COUNT(b.booking_id) AS tickets_sold,
SUM(e.cost_per_participant) AS revenue
FROM
events e
LEFT JOIN
bookings b ON e.event_id = b.event_id
GROUP BY
e.title, e.cost_per_participant;
Räknar antal sålda biljetter (COUNT(b.booking_id)) för varje event.
Beräknar total intäkt (SUM(e.cost_per_participant)).
LEFT JOIN används så att även event utan bokningar visas (med tickets_sold = 0).

Grundläggande Säkerhet

Kod

CREATE OR REPLACE FUNCTION kontrollera_bokningskapacitet()
RETURNS TRIGGER AS $$

```
DECLARE bokningar INT;
max_deltagare INT;
BEGIN
SELECT (SELECT COUNT(*) FROM bookings WHERE event_id = NEW.event_id),
max_participants
INTO bokningar, max_deltagare
FROM events
WHERE event_id = NEW.event_id;
IF bokningar >= max_deltagare THEN RAISE EXCEPTION 'Event % har nått max antal
deltagare.',
NEW.event_id;
END IF;
RETURN NEW; END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trigger_bokningskapacitet
BEFORE INSERT ON bookings
FOR EACH ROW
EXECUTE FUNCTION kontrollera_bokningskapacitet();
```

Förklaring

Den här koden är en PL/pgSQL-funktion och en trigger i PostgreSQL som används för att kontrollera bokningskapaciteten för ett event innan en ny bokning läggs till.

kontrollera_bokningskapacitet()

Denna funktion används som en triggerfunktion som körs innan en ny bokning läggs till i tabellen bookings.
Funktionen ser till att antalet bokningar för ett specifikt event inte överskrider det maximala deltagarantalet (max_participants) för eventet.

```
DECLARE
  bokningar INT;
  max_deltagare INT;
```

bokningar håller antalet nuvarande bokningar för ett event.
max_deltagare lagrar maxgränsen för deltagare i eventet.
Hämtar antalet bokningar och maxdeltagare för det aktuella eventet:

```
SELECT (SELECT COUNT(*) FROM bookings WHERE event_id = NEW.event_id),
max_participants
INTO bokningar, max_deltagare
FROM events
WHERE event_id = NEW.event_id;
```

Räknar antalet rader i bookings där event_id matchar det event som den nya bokningen (NEW.event_id) gäller.
Hämtar max_participants från events-tabellen för att veta maxgränsen för eventet.

Dessa två värden lagras i variablerna bokningar och max_deltagare.
Kontrollerar om maxgränsen har uppnåtts:

```
IF bokningar >= max_deltagare THEN
   RAISE EXCEPTION 'Event % har nått max antal deltagare.', NEW.event_id;
END IF;
```

Om antalet nuvarande bokningar (bokningar) är lika med eller större än maxgränsen (max_deltagare), kastas ett felmeddelande.
Detta stoppar bokningen från att genomföras.
Returnerar det nya värdet (NEW) om allt är okej:

```
RETURN NEW;
```

Om maxgränsen inte är nådd, tillåts bokningen att genomföras.

trigger_bokningskapacitet

```
CREATE TRIGGER trigger_bokningskapacitet
BEFORE INSERT ON bookings
FOR EACH ROW
EXECUTE FUNCTION kontrollera_bokningskapacitet();
```

Skapas på bookings-tabellen.
Körs innan (BEFORE) en ny rad (INSERT) läggs till.
Gäller för varje enskild ny rad (FOR EACH ROW).
Anropar funktionen kontrollera_bokningskapacitet(), som gör kontrollen och avgör om bokningen ska accepteras eller avvisas.

Hur fungerar detta i praktiken?

En användare försöker boka en plats på ett event genom att lägga till en rad i bookings-tabellen.
Triggersystemet i PostgreSQL aktiverar kontrollera_bokningskapacitet() innan bokningen sparas.
Funktionen räknar antalet bokningar för eventet och jämför det med maxantalet.
Om eventet är fullt, avbryts insättningen och ett felmeddelande returneras.
Om det finns lediga platser, tillåts bokningen att genomföras.

Exempel

Tabelldata innan en ny bokning görs

events:

event_id	name	max_participants
1	Konsert A	2

bookings:

booking_id	user_id	event_id
1	101	1
2	102	1

Scenario:

En tredje användare försöker boka en plats till event_id = 1.
Funktionen hämtar bokningar = 2 (antal befintliga bokningar för eventet).
max_deltagare = 2 (maxgränsen för eventet).
Eftersom bokningar >= max_deltagare, kastas ett fel:
ERROR: Event 1 har nått max antal deltagare.
Bokningen stoppas, och eventet överfylls inte.

Sammanfattning

Funktionen kontrollerar antalet bokningar innan en ny läggs till.
Om eventet är fullbokat, stoppas bokningen med ett felmeddelande.
Triggersystemet automatiserar denna kontroll innan en ny bokning görs.