Individuell Uppgift: Kunskapskontroll

Frågor

1, Databasdesign:

A, Förklara skillnaden mellan en relationsdatabas och en dokumentdatabas.
B, Vad är syftet med normalisering, och vilka är de tre första normalformerna?
Ge ett exempel på normalisering från 1NF till 3NF.

Svar

A, En relationsdatabas (RDBMS) och en dokumentdatabas (NoSQL) har olika sätt att lagra och hantera data.
Relationsdatabas:
-Data organiseras i tabeller med rader och kolumner.
-Använder SQL (Structured Query Language) för att manipulera data.
-Strikt schema (datafält måste följa en förutbestämd struktur).
-Lämplig för strukturerad data som följer ett tydligt mönster.
-Exempel PostgreSQL, MySQL, SQL Server.
-Exempel:

ID	Namn	Email
1	Anna	anna@mail.com
2	Johan	johan@mail.com

Dokumentdatabas:
-Data lagras som dokument i JSON- eller BSON-format.
-Schemafritt, vilket innebär att varje dokument kan ha olika struktur.
-Bra för ostrukturerad eller semi-strukturerad data.
-Skalbar och flexibel vid hantering av stora datamängder.
-Exempel MongoDB, CouchDB.
-Exempel:

```
{
 "ID": 1,
 "Namn": "Anna",
 "Email": "anna@mail.com"
}
```

Normalisering är en process för att strukturera en databas för att minimera redundans och förbättra dataintegritet.
-Ej 1NF

OrderID	Kund	Produkter
1	Anna	Dator, Telefon

-1NF (Första normalformen)
Alla kolumner måste innehålla atomära (odelbara) värden.
Exempel = Ingen kolumn ska innehålla flera värden (inga listor i celler).
Exempel:

OrderID	Kund	Produkt
1	Anna	Dator
1	Anna	Telefon

-2NF (Andra normalformen)
Måste vara i 1NF och alla icke-nyckelattribut ska vara fullt funktionellt beroende av primärnyckeln.
Exempel = Separera kunder från ordrar.
Exempel:

Orders

OrderID	KundID	Produkt
1	1	Dator
2	2	Telefon

Kunder

KundID	Kund
1	Anna
2	Johan

-3NF (Tredje normalformen)
Måste vara i 2NF och får inte ha transitiva beroenden.
Exempel = Om priset för produkter ändras måste det ligga i en separat tabell.
Exempel:

OrderID	KundID	ProduktID
1	1	1
2	2	2

ProduktID	Produkt	Pris
1	Dator	1000
2	Telefon	500

2, SQL:

A, Vad är skillnaden mellan PRIMARY KEY och FOREIGN KEY?
B, Beskriv vad en JOIN gör i SQL och ge ett exempel på en enkel JOIN.
C, Du har följande tabeller:**Users(ID, Name, Email)Bookings(ID, UserID, EventID)Events(ID, Title, Date)**
Skriv en SQL-fråga för att lista alla användare och titlar på event de har bokat.

Svar

A, PRIMARY KEY vs FOREIGN KEY:
PRIMARY KEY = Unikt ID för en rad.
FOREIGN KEY = Skapar koppling mellan tabeller.
Exempel:

```
CREATE TABLE Users (
  ID SERIAL PRIMARY KEY,
  Name VARCHAR(100)
);
CREATE TABLE Bookings (
  ID SERIAL PRIMARY KEY,
  UserID INT,
  EventID INT,
  FOREIGN KEY (UserID) REFERENCES Users(ID),
  FOREIGN KEY (EventID) REFERENCES Events(ID)
);
```

B, En JOIN används i SQL för att kombinera data från två eller fler tabeller baserat på en gemensam kolumn.
Detta gör det möjligt att hämta relaterad information från flera tabeller i en enda fråga.
Det finns olika typer av JOIN:
INNER JOIN = Hämtar endast rader där det finns en matchning i båda tabellerna.
LEFT JOIN = Hämtar alla rader från den vänstra tabellen och matchande rader från den högra (eller NULL om ingen match finns).

RIGHT JOIN = Hämtar alla rader från den högra tabellen och matchande rader från den vänstra.
FULL JOIN = Hämtar alla rader från båda tabellerna, även om det inte finns en match.
Exempel på en enkel JOIN med 2 tabeller:

Tabell: `products`

id	product_name	category_id
1	Kexchoklad	2
2	Pasta	1

Tabell: `categories`

id	category_name
1	Torrvaror
2	Godis

Om vi vill hämta produktens namn och dess kategori använder vi en JOIN:

```
SELECT products.product_name, categories.category_name
FROM products
INNER JOIN categories ON products.category_id = categories.id;
```

Resultat:

product_name	category_name
Kexchoklad	Godis
Pasta	Torrvaror

Detta exempel på en INNER JOIN kopplar ihop products.category_id med categories.id och returnerar bara de rader där det finns en matchning.
C, Lista användare och bokade eventtitlar:

```
SELECT Users.Name, Events.Title
FROM Users
JOIN Bookings ON Users.ID = Bookings.UserID
JOIN Events ON Bookings.EventID = Events.ID;
```

3, Constraints:

Vad är en constraint i en databas?
Ge exempel på minst två constraints som kan användas i PostgreSQL.

Svar

En constraint är en regel som säkerställer att data i en tabell är korrekt och konsekvent.
Exempel:

```
-- Skapa en tabell 'categories' med en PRIMARY KEY
CREATE TABLE categories (
```

id SERIAL PRIMARY KEY, -- Unik identifierare, auto-increment
category_name VARCHAR(100) NOT NULL UNIQUE -- Måste vara unikt och ej NULL
);
-- Skapa en tabell 'products' som refererar till 'categories'
CREATE TABLE products (
id SERIAL PRIMARY KEY, -- Unik identifierare, auto-increment
product_name VARCHAR(255) NOT NULL, -- Måste ha ett värde (ej NULL)
price DECIMAL(10,2) NOT NULL CHECK (price >= 0), -- Priset kan ej vara negativt
stock_quantity INT DEFAULT 0 CHECK (stock_quantity >= 0), -- Standardvärde 0, ej negativt
category_id INT NOT NULL, -- Måste vara kopplat till en kategori
CONSTRAINT fk_category FOREIGN KEY (category_id) REFERENCES categories(id) -- Koppling till 'categories'
);
NOT NULL = Förhindrar att en kolumn innehåller NULL-värde.
UNIQUE = Säkerställer att inga duplicerade värden förekommer.
Primary Key = Säkerställer att varje rad är unik och inte NULL.
Foreign Key = Säkerställer att värden i en kolumn matchar värden i en annan tabell.
DEFAULT = Tilldelar ett standardvärde om inget annat anges.
CHECK = Används för att specificera ett villkor som måste vara sant för att lägga till eller uppdatera en rad.

4, Säkerhet:

A, Varför är det viktigt att hantera användarroller i en databas?
B, Beskriv hur du skulle begränsa åtkomst till en specifik tabell i PostgreSQL.

Svar

A, Exempel:
Säkerhet = Begränsar åtkomst till känslig data.
Förebygga fel = Stoppar misstag och sabotage.
Spårbarhet = Ser vem som ändrat vad.
Effektiv hantering = Ger rättigheter baserat på roller istället för per användare.
PostGreSQL Exempel:
-- Skapa en admin-roll med fulla rättigheter
CREATE ROLE admin_role WITH LOGIN PASSWORD 'admin123';
GRANT ALL PRIVILEGES ON DATABASE shop_db TO admin_role;
-- Skapa en kund-roll som bara kan läsa produkter
CREATE ROLE customer_role WITH LOGIN PASSWORD 'customer123';
GRANT CONNECT ON DATABASE shop_db TO customer_role;
GRANT SELECT ON products TO customer_role;

B, Jag skulle använda mig av följande kod:
REVOKE ALL ON TABLE Users FROM public; GRANT SELECT ON Users TO read_only_user;
Koden ger en specifik användare (read_only_user) rätt att endast läsa (SELECT) data från Users, utan att kunna ändra eller radera något.
Vi kan även använda oss av denna kod:
REVOKE ALL ON TABLE Users FROM public;
Koden innebär att vi tar bort alla behörigheter från den generella användargruppen `public`, vilket innebär att ingen användare (utom superanvändare) har åtkomst till tabellen `Users` tills specifika rättigheter ges.

5, Aggregerad Data:

A, Vad är aggregerade funktioner i SQL?
Ge exempel på minst två aggregerade funktioner och hur de används.
B, Skriv en SQL-fråga som räknar det totala antalet deltagare för alla events kombinerat.

Svar

A, Aggregerade funktioner används för att utföra beräkningar på en uppsättning rader och returnera ett enda värde.
De är vanliga vid rapportering och analys av data.
Exempel på aggregerade funktioner:
COUNT() = Räknar antalet rader i en tabell eller en viss kolumn.
SELECT COUNT(*) FROM Bookings;
SUM() = Summerar värden i en kolumn.
SELECT SUM(Price) FROM Orders;
AVG() = Beräknar medelvärdet av en kolumn.
SELECT AVG(Score) FROM Reviews;
MAX() = Returnerar det högsta värdet i en kolumn.
SELECT MAX(Salary) FROM Employees;
MIN() = Returnerar det lägsta värdet i en kolumn.
SELECT MIN(Temperature) FROM WeatherData;
B, SQL-fråga:
SELECT COUNT(*) AS TotalParticipants FROM Bookings;
Förklaring:
SELECT COUNT(*)
COUNT(*) är en aggregerad funktion som räknar det totala antalet rader i en tabell.
***** Betyder att vi räknar alla rader, oavsett kolumninnehåll.
FROM Bookings
Tabellen Bookings innehåller information om bokningar för olika event.
Varje rad representerar en bokning av en användare till ett event.

AS TotalDeltagare
Ger det beräknade värdet ett alias (TotalDeltagare), vilket gör det mer läsbart i resultatet.
Den räknar det totala antalet bokningar i Bookings-tabellen.
Varje bokning motsvarar en deltagare på ett event.
Resultatet returnerar en enda siffra som representerar det totala antalet deltagare över alla event.
Exempeldata i Bookings-tabellen:

ID	UserID	EventID
1	101	5
2	102	5
3	103	6
4	104	7

Om Bookings innehåller 4 rader, kommer frågan att returnera:
TotalDeltagare

4
Det betyder att totalt 4 personer har bokat event.

För Väl Godkänt (VG)

6, Avancerad SQL:

Utforma en SQL-fråga som visar alla events där fler än 10 deltagare är registrerade.
Se till att inkludera antalet deltagare och kostnaden per deltagare i resultatet samt beräkna den totala intäkten för eventen.
För att underlätta, använd en aggregerad funktion som SUM och kombinera med GROUP BY och HAVING.
Förklara hur frågan fungerar steg för steg.

Svar

SQL-fråga för events med fler än 10 deltagare:

```
SELECT Events.Title, COUNT(Bookings.ID) AS AntalDeltagare,
    Events.CostPerParticipant,
    SUM(Events.CostPerParticipant * COUNT(Bookings.ID)) AS TotalIntäkt
FROM Events
```

JOIN Bookings ON Events.ID = Bookings.EventID
GROUP BY Events.ID, Events.Title, Events.CostPerParticipant
HAVING COUNT(Bookings.ID) > 10;
Förklaring:
JOIN Events och Bookings = Kombinerar event med deras bokningar genom att matcha Events.ID med Bookings.EventID.
COUNT(Bookings.ID) AS AntalDeltagare = Räknar antalet bokningar (deltagare) per event.
Events.CostPerParticipant = Hämtar kostnaden per deltagare för eventet.
SUM(Events.CostPerParticipant * COUNT(Bookings.ID)) AS TotalIntäkt = Beräknar den totala intäkten genom att multiplicera kostnaden per deltagare med antalet deltagare.
GROUP BY Events.ID, Events.Title, Events.CostPerParticipant = Grupperar resultaten per event så att varje event får sin egen rad i resultatet.
HAVING COUNT(Bookings.ID) > 10 = Filtrerar så att endast event med fler än 10 deltagare visas i resultatet

7, Djuptgående Analys:

Diskutera för- och nackdelar med att använda PostgreSQL jämfört med MongoDB för ett projekt som hanterar stora mängder strukturerad och ostrukturerad data.
Ge konkreta exempel och resonera kring prestanda, skalbarhet och utvecklingsflexibilitet.

Svar

När vi hanterar stora mängder strukturerad och ostrukturerad data finns det flera aspekter att beakta vid valet mellan PostgreSQL och MongoDB.
-Prestanda
PostgreSQL har stark prestanda vid komplexa frågor och relationer tack vare indexering, transaktioner och JOIN-operationer.
MongoDB är snabbare vid hantering av stora mängder ostrukturerad data, särskilt vid snabba insättningar och hämtningar av JSON-baserade dokument.
-Skalbarhet
PostgreSQL skalas vertikalt (starkare servrar), men stöder även partionering och replikering.
MongoDB är byggt för horisontell skalning, där data distribueras över flera servrar med sharding.
-Utvecklingsflexibilitet
PostgreSQL kräver ett fördefinierat schema, vilket ger dataintegritet men minskar flexibilitet.
MongoDB är schemafritt, vilket gör det lättare att snabbt ändra datamodellen.

-Exempel på användningsområden

Användningsområde	PostgreSQL	MongoDB
Finansiella system	ACID-transaktioner, komplexa relationer	Mindre lämplig (ingen stark ACID-stöd)
Realtidsloggning	Begränsad prestanda för skrivintensiv data	Högpresterande för logginsamling
E-handelsplattformar	Strukturerad produkt- och orderhantering	Flexibel hantering av produktdata
IoT och big data	Kräver anpassad hantering av stora datamängder	Naturligt stöd för stora ostrukturerade dataset

-Exempel på funktioner/egenskaper

Funktion/Egenskap	PostgreSQL	MongoDB
Datamodell	Relationsdatabas (tabeller/rader)	Dokumentdatabas (samlingar/dokument)
Schema	Strikt schema	Schemafri
Relationer	Primär- och främmande nycklar	Embedded dokument/referenser
Frågespråk	SQL	MongoDB Query Language
Prestanda	Effektiv för komplexa relationer	Snabb för ostrukturerad data
Användningsområden	Finansiella system, ERP	Webbapplikationer, IoT-data

-Sammanfattning

PostgreSQL är bättre för relationsbaserade system där dataintegritet och komplexa frågor är avgörande.

MongoDB passar bättre för stora ostrukturerade dataset, där flexibilitet och snabb datahantering är viktigare än starka relationer.