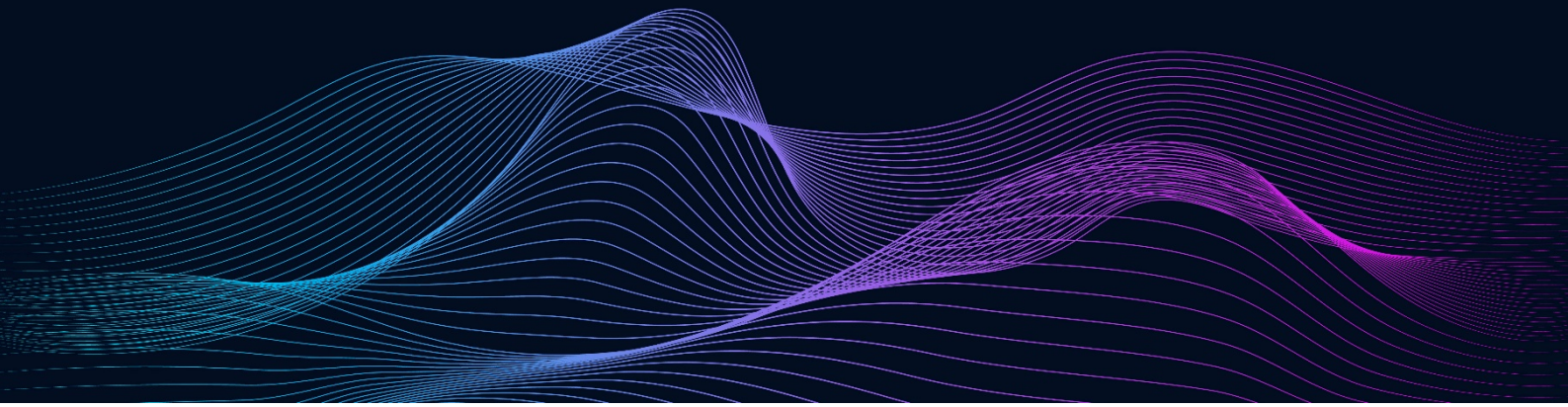

Projekt, uge 8
IT-Arkitektur

Af:

Astrid Gravesen, Daniel Brinkland & Daniel Egelund

Projektopgave – uge 8

Indledning

Projektet tager udgangspunkt i undervisningen. Hertil har projektet til formål at demonstrere viden indenfor HTML, CSS og en videreudbygning af vores viden om MySQL. I rapporten vil hovedpunkter i opgaven demonstreres og ligeledes koden bag. Rapporten er bygget op således, at der først vil være en illustrering af, hvordan undervisningen i webteknologi er en del af vores projekt, dernæst hvordan en database opbygges med udgangspunkt i et given datasæt samt en række opgaver, der definerer dataene.

Problemformulering

I projektbesvarelsen ønskes der en fremstilling af den tekniske kunnen indenfor HTML og CSS, med referencer til undervisningen. Derudover ønskes der en besvarelse af 10 spørgsmål til et opgivet datasæt i databasedesign.

Webteknologi

Websitet i projektbesvarelsen spænder vidt. Der er blevet benyttet layoutteknikker, referencer i form af links, generel syntaks, navigationer og menuer. For at give et fyldestgørende indtryk af besvarelsen rapporten afspejler, vil der nedenstående være et link til GitHub, hvor koden bag websitet er tilgængeligt. Hertil også den pågældende CSS.

Af forsiden fremgår en navigationsbar med henvisninger til resten af websitet. På hvert nyt site er der benyttet grids til at vise vores opgavebesvarelser. CSS'en bidrager til, at vores layout er lige akkurat som vi gerne vil have det, helt ned til den mindste pixel.

https://github.com/DanielEgelund/DataBase-Website

Opgave 1-4. Normalization

Normalization er en teknik brugt i databaser for at reducere dataredundans og eliminere anomalier. Processen består i at udbygge en tabel til flere tabeller ved brug af relationer. Formålet er endvidere at lagre dataene logisk med så få repetitioner som muligt. Projektets besvarelse tager udgangspunkt

i et datasæt, som endnu ikke er blevet *normalized*. Heraf vil besvarelsen rumme hvordan datasættet ser ud i henholdsvis 1NF, 2NF, 3NF og BCNF.

Ovenstående normal forms har hver især specifikke krav til, hvordan datasættet skal være i en tabel. First normal form (1NF) kræver, at der i hver celle er unikke data, og at hver celle højst må indeholde en værdi. I det udleverede datasæt er 1NF hurtigt opfyldt, eftersom der i det pågældende datasæt ikke er flere værdier i en celle, og at der samtidig fremgår unikke data, som opfylder kravene.

orderNr	productNr	orderDate	customerType	Quantity	price
20210226001	10	2021-02-26	Wholesaler	150	22
	14			100	45
20210226002	2	2021-02-26	Retailer	5	560
	5			8	40
	10			1	25

Dernæst skal datasættet opfylde second normal form (2NF). Helt grundlæggende er kravet her, at datasættet er i 1NF. Herudover skal tabellens *primary key*, være funktionelt uafhængig af andet data i datasættet. Det betyder at tabellen skal deles op, da flere af dataene i ovenstående sæt relaterer til den samme data og skaber unødvendigt redundans i tabellen. Så for at datasættet opfylder kravene til second normal form, skal OrderNr og ProductNr have deres egen tabel.

Det ser ud som følgende:

Order table

Order_ID	Order_Date	Customer_Type
20210226001	26-02-2021	Wholesaler

Products table

Product_Nr	Product_Name	Product_Price
10	Laks	22

Af second normal form fremgår her ”transitive dependency”, altså at non-key værdier kan ændres, såfremt der opstår nye data i vores ordresystem.

I third normal form (3NF) er kravet at datasættet optræder i 2NF, og at der ikke forekommer noget transitive dependency. Det gøres ved, at vi udfolder vores tabeller yderligere:

Order table

Order_ID	Order_Date	Customer_ID
20210226001	26-02-2021	1

Product table

Product_Nr	Product_Name	Product_Price
10	Laks	22

Order_line table

Order_Line_ID	Order_ID	Quantity	Line_Price	Product_Nr
12	20210226001	12	22	10

Customer table

Customer_ID	Type_ID
1	2

Type table

Type_ID	Type
2	Wholesaler

I ovenstående tabeller er alt transitive dependency fjernet fra datasættet. Dette ses ved at non-key kolonner ikke længere har påvirkning på hinanden. Derudover er datasættet pakket ud til et punkt, hvor det ikke længere giver mening at opbryde tabellerne, da relationerne tabellerne imellem, refererer til den korrekte data og al redundans er fjernet.
Heldigvis er ovenstående derfor også grundlag for at datasættet nu opfylder Boyce-Codd normal form, eftersom der ikke forekommer anomalier, da vi har reduceret til højst en candidate key pr tabel.

Opgave 5. Brugeroprettelse og uddelegering af privilegier

Formålet med at uddele privilegier i SQL er at fortælle databasen, hvem der kan se den samt lave ændringer. Databaseobjekterne kan altså kun ændres, slettes eller opdateres af brugere med de rette rettigheder. Først og fremmest skal der i databasen oprettes brugere, der kan tildeles disse privilegier. I opgaven fremgår det, at tre brugere skal oprettes i databasen; to med mange privilegier og en restriktiv bruger. De tre brugere hedder henholdsvis Harald, Kurt og Abelone. To af dem kan manipulere data i form af insert, update og delete, hvorimod den sidste kun har select-rettigheder og altså kun kan læse indholdet af databasen. Scriptet bag viser syntaksen til at uddele privilegierne samt hvordan brugerne oprettes i databasen.

```
CREATE USER Kurt identified by 'Password';
CREATE USER Harald identified by 'Password';
CREATE USER Abelone identified by 'Password';

Grant select, insert, delete on OrdreSystem.* to Harald, Abelone;

Grant select on OrdreSystem.* to Kurt;
```

6. Stored procedure - Totalpris for en ordre

Først og fremmest bruges in ”Stored procedure” for hurtigere at kunne eksekvere statements, da de bliver precompiled på serveren. Derudover bidrager stored procedures til datasikkerhed, da der, lige som ved privilegierne, kan gives rettigheder til den enkelte bruger.
I opgaven skal en stored procedure afregne summen af en ordre. Dette gøres ved at benytte sig af den korrekt syntaks, som indledningsvist starter med ”Create procedure”, hertil det pågældende tabelnavn, og dernæst de værdier man ønsker at få den samlede sum af.

```
DELIMITER $$
CREATE PROCEDURE GetOrderAmount(
        IN ID varchar(255)
)
BEGIN
SELECT
SUM(Line_Price)
FROM Order_Line
WHERE Order_ID = ID;
END$$
DELIMITER ;
```

7. Stored procedure – Meddelelse ved lavt lagerantal

Denne stored procedure bidrager til, at vi i det givne datasæt får en meddelelse hver gang en af vores produkter understiger 5 i den samlede lagerbeholdning. Proceduren oprettes ved at tage udgangspunkt i vores ”Products”- tabel og sørger for at brugerne og databasedesigneren, bliver gjort opmærksomme på at det er på tide at bestille produktet hjem igen. Proceduren viser her, hvordan vi har refererer til de korrekte værdier, samt hvilken syntaks der gør det muligt at sende meddelelsen.

```
DELIMITER $$
CREATE PROCEDURE LowStockError(
        IN Product varchar(255)
)
BEGIN
        DECLARE In_Stock int default 0;
SELECT Stock
INTO In_Stock
FROM Products
WHERE Product_Nr = Product;
                IF In_Stock < 5
                THEN Select 'Der er mindre end 5 stk på lager af denne vare';
                END IF;
        END $$
DELIMITER ;
```

8. Opret en trigger der gemmer alle actions på log_table

SQL-triggers giver serveren klar besked om, hvilke specifikke funktioner der skal udføres ved brug af forskellige statements i databasen. Hver gang der forekommer et DML-statement (Insert, update, delete) på et specifikt table, aktiveres triggeren og udfører den pågældende opgave. Herudover

bidrager triggers til at sikre integritet af data, samt at automatisere processor, som gør det muligt at logge de ændringer der må forekomme.

I opgave 8 bliver triggeren implementeret, så den er i stand til at dokumentere hvilke handlinger, der bliver foretaget hvornår, samt hvilken bruger der gør det. Herudover er den også i stand til at tilføje det pågældende ordrer nummer, så der ikke kan herske tvivl om, hvor der bliver manipuleret data. For at dataene bliver lagret korrekt, opstilles der en log-tabel, hvori dataene tilføjes.

```
Create trigger order_delete
        Before delete on Orders
    For each row
insert into log_table
set action = 'delete',
User_Name = USER(),
action_time = NOW(),
Order_Nr = OLD.Order_ID;
```

9. Transaction og tilføjelse af nye data

En transaction er en entity, som enten skal eller ikke skal eksekveres. Det kan være:

- En gruppe af SQL-statements
- En kombination af statements og ”stored-procedure call”

Med transaction kan man tilbagerulle evt. fejl, så en database kan opretholde sin korrekthed.
Således bliver dele af et eksekveret statement ikke gemt i databasen.
Formålet med transaction i vores opgave, er ganske enkelt at sørge for, at nye ordrer ikke tilføjes til serveren, hvis transaktionen ikke gennemføres. Såfremt det derimod virker, tilføjes eller ændres den pågældende data permanent i databasen.

10. Tilsvarende XML og XML-schema

XML står for Extensible Markup Launguage og er designet til at indeholde den mængde data, der ikke vises som HTML. XML-tags er ikke prædefineret som HTML-tags. Det betyder ganske enkelt, at XML-tags bliver oprettet af designeren selv. XML gemmer data som en ren tekst, hvilket gør den uafhængig fra software og hardware. Det betyder altså, at alle er i stand til at læse XML - computere såvel som mennesker. Opgave 10 bliver derfor relevant, da alle skal være i stand til at læse ordretabellen.