Eksamensopgave : Databaser

Faglige Mål

Du skal modellere data, oprette og anvende databaser og analysere egenskaber ved typer af data, samt udvælge og anvende forskellige typer af data

Eksamensopgave:

Ecco har altid haft mange kunder. Det kunne være en god ide at oprette en kundedatabase, for således at firmaet får kendskab til, hvordan de forbedre salg og kundeservice.

Udarbejd en kundedatabase som udgangspunkt består af 3 tabeller.
Databasen bør indeholde følgende:

- Navn
- Køn
- Bopæl By
- Bopæl Postnummer
- Ordre

1. Start med at få opbygget de 3 tabeller der er behov for i databasen (tegn på papir)
2. Angiv hvilke nøglefelter, der indgår i de enkelte tabeller, overvej om der er flere muligheder for valg af datatype på nogle af felterne

3. Normalisér alle tabellerne til 3. normalform.
4. Tegn et E/R-diagram, der viser sammenhængen mellem databasens tabeller.

Bilag 1

Normalisering

https://informatikbeux.systime.dk/?id=p1056

Bilag 2

Datatyper

https://informatikbeux.systime.dk/?id=p1057

Bilag 3

E/R-diagram
https://informatikbeux.systime.dk/?id=p1055

Bilag 1

Normalisering

Normalisering handler om, hvordan databasens tabeller bygges op og sammenkobles, så de relaterede tabeller fungerer mest hensigtsmæssigt og risikoen for fejl minimeres.

NORMALISERING

Normalisering er en metode til at opbygge tabellerne i databasen hensigtsmæssigt, så vi undgår at registrere de samme oplysninger flere gange i databasen. Samtidig reducerer normaliseringen risikoen for, at der er fejl i data i databasen.
I forbindelse med normalisering af en database er vores formål, at få databasens tabeller til at opfylde 3. normalform. Der findes flere normalformer, og vi vil her gennemgå, hvordan vi først opnår 1. normalform, derefter 2. normalform og til sidst 3. normalform. For at en tabel kan komme på 2. normalform, er det et krav, at den opfylder 1. normalform. Ligeledes er det et krav, at 2. normalform er opfyldt, før en tabel kan sættes på 3. normalform.

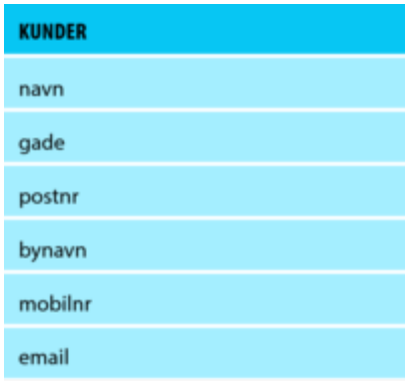
1. normalform

Krav til 1. normalform:

- Tabellen skal have et nøglefelt.
- Alle tabellens felter indeholder netop én værdi.

Lad os se på et eksempel.

Vi starter med de oplysninger, vi typisk ville registrere omkring vores kunder i en kundetabel i databasen:



Figur 7.14
Kontaktoplysninger om kunderne.

I denne tabel har vi alle de basale kontaktoplysninger om vores kunder. Vi mangler dog at få defineret et nøglefelt, så vi entydigt kan identificere vores kunder. Navnet er i dette tilfælde ikke tilstrækkeligt som primært nøglefelt, da der kan være flere kunder, der har det samme navn. Umiddelbart ville vi godt kunne anvende mobilnummer eller email som vores primære nøglefelt, men som tidligere nævnt fravælger vi dette.

For at få en primær nøgle, der entydigt kan identificere de enkelte poster, vælger vi at bruge en sammensat nøgle, der består af felterne *navn*, *gade* og *postnr*. Har vi disse tre oplysninger om en kunde, er der ingen tvivl om, hvilken kunde der præcist er tale om.

At alle felterne indeholder netop én værdi betyder, at vi eksempelvis ikke kan have flere email-adresser registreret på den enkelte kunde.

Figur 7.15
Kundetabellen på 1. normalform med en sammensat nøgle.

2. normalform

Krav til 2. normalform:

- 1. normalform skal være opfyldt.
- Felter der ikke er en del af den primære nøgle, må ikke kun afhænge af en del af den primære nøgle.

Da vores tabel allerede er normaliseret på 1. normalform, ser vi udelukkende på, hvordan de øvrige felter afhænger af den primære nøgle.

Ser vi på feltet *bynavn*, vil dette altid kunne bestemmes ud fra det *postnr*, vi får oplyst. *Postnr* er kun en del af den primære nøgle, og dette strider derfor mod kravene til 2. normalform.

Vores løsning bliver, at vi indfører et nyt felt, vi kalder *kundenr*, som bliver vores nye primære nøgle. Alle kunderne vil herefter kunne identificeres entydigt, hvis vi får oplyst kundenummeret. Vi opnår hermed, at der ikke længere er en sammensat nøgle og vores tabel er hermed på 2. normalform.

Figur 7.16
Kundetabellen på 2. normalform.

3. normalform

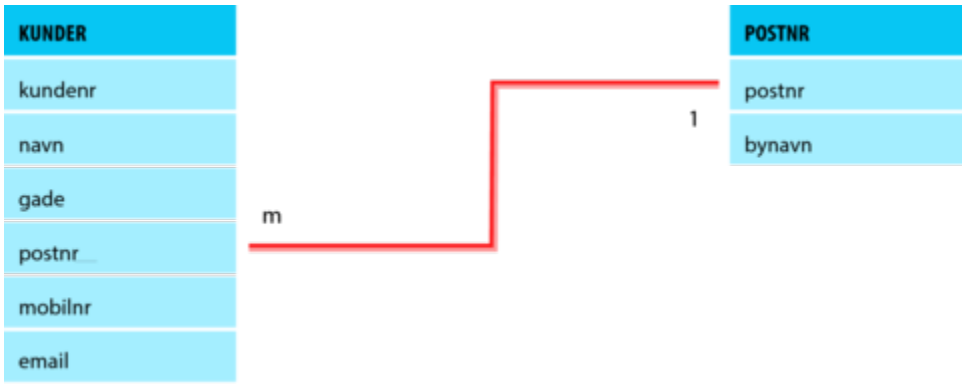
Krav til 3. normalform:

- 2. normalform skal være opfyldt.
- Der må ikke være felter der entydigt kan bestemmes på baggrund af andre felter end det primære nøglefelt.

Igen her er tabellen allerede på 2. normalform, så vi ser på, hvordan felterne afhænger af hinanden.

Som det kan ses i figur 7.16, har vi valgt feltet *kundenr* som vores primære nøglefelt. I tabellen er feltet *bynavn* stadig afhængigt af feltet *postnr*. Dette betyder, at kender vi blot *postnr*, kan vi med sikkerhed sige, hvilket *bynavn* der er tale om. Dette strider mod 3. normalform, da *postnr* ikke er vores primære nøgle.

For at få kundetabellen på 3. normalform er der behov for, at vi splitter op i 2 tabeller i alt: en kundetabel og en postnummertabel. I kundetabellen vil *kundenr* være vores primære nøglefelt og *postnr* være vores fremmednøgle. I postnummertabellen er *postnr* det primære nøglefelt. Der er en 1-m relation mellem tabellerne, fordi hver kunde kun har et postnummer, og der inden for hvert postnummer godt kan være flere kunder.



Figur 7.17
Kundetabel og postnummertabel på 3. normalform og relateret.

Hele normaliseringsprocessen handler om at opbygge databasens tabeller, så hensigtsmæssigt som muligt. Mere præcist ønsker vi at undgå redundans i databasen og dermed undgå at risikere inkonsistente data.

Redundans betyder, at vi har de samme data stående flere steder databasen, hvilket vi gerne vil undgå. Hvis vi eksempelvis, hver gang vi får en ny ordre fra den samme kunde, skal skrive kundens navn og adresse, får vi efterhånden disse oplysninger til at stå i mange poster. Hvis vi så skriver kundens navn og adresse lidt forskelligt i de forskellige ordrer, har vi inkonsistente data. Inkonsistente data betyder, at data, der burde være ens, ikke er det.

I stedet skal vi konstruere databasen således, at det kun er kundens unikke *kundenr*, der registreres om kunden i forbindelse med en ordre. Alle de øvrige kundeoplysninger kan så hentes i kundetabellen, da vores tabeller er relaterede.

REDUNDANS

Redundans betyder, at de samme data er registreret flere steder i databasen og fylder dermed unødig plads.

INKONSISTENTE DATA

Inkonsistente data betyder, at data, der burde være ens, ikke er ens. Vi risikerer at få inkonsistente data, når vi har redundans i vores database.

Bilag 2

Datatyper

Et vigtigt element i opbygningen af en database er valget af datatyper. Datatyper skal vælges for alle vores felter i de forskellige tabeller, der indgår i databasen. Datatypen fortæller noget om, hvilken type indhold der er i feltet. Valget er afgørende for, hvordan vi efterfølgende kan arbejde med feltet.

Der findes mange forskellige datatyper, også flere end vi nævner her. Mange af datatyperne har mest relevans for meget store databaser, fordi det korrekte valg af datatype kan spare plads i databasen og derved få den til arbejde hurtigere ved forespørgsler.

Overordnet kan vi dele datatyperne ind i følgende kategorier:

- **Tekst:** Datatypen tekst skal vi anvende, hvis der forekommer bogstaver i feltet, men kan også anvendes når der udelukkende er tal.
- **Heltal:** Fordelen ved at anvende datatypen tal er, at man kan foretage beregninger på disse felter. Hvis vi eksempelvis i en varetabel ønsker at optælle en varebeholdning og lægge nogle antal sammen, kræver det, at felterne har en datatype, der kan håndtere heltal.
- **Decimaltal:** Har samme egenskaber som datatypen Heltal, men kan også håndtere tal med decimaler. Hvis vi eksempelvis i en ordretabel ønsker at lægge nogle beløb sammen, kræver det, at felterne har en datatype, der kan håndtere decimaltal.
- **Boolean:** En datatype, der kun kan antage to værdier. Denne datatype kan anvendes til felter, hvor vi kun har to svarmuligheder. Eksempelvis kan vi ønske at registrere, om vores kunder ønsker at modtage vores nyhedsbrev pr. mail. Svaret her er enten ja eller nej.
- **Dato og tid:** Disse datatyper kan anvendes, når vi ønsker at registrere en tidsangivelse. Fordelen er, at disse datatyper bliver fremvist på et let genkendeligt format for datoer. Samtidig vil det være muligt, ved hjælp af nogle indbyggede funktioner, at beregne hvor lang tid der eksempelvis er mellem to tidsangivelser.

I vores eksempler hidtil vil vi skulle anvende datatyperne tekst og heltal. I MySQL kaldes disse datatyper for VARCHAR og INT.

Når vi vælger datatyper for felterne i en relationsdatabase, er én af begrundelserne for at vælge den rigtige, at den rigtige datatype og feltlængde vil spare plads i vores database. Når databasen anvender mindre dataplads, vil den også kunne arbejde hurtigere. Der er samtidig et krav om, at når vi har relationer mellem tabeller, skal datatyperne for den sammenkoblede primære nøgle og tilkoblede fremmednøgle være ens.

Ved valg af de rigtige datatyper får vi også mulighed for såkaldt validering på felter og for at have skabeloner for indtastning. Ved validering forstås, at vi sætter begrænsninger på, hvad der kan indtastes i feltet. Hvis vi eksempelvis har valgt datatypen INT(4), kan der kun indtastes tal i feltet og maksimalt fire cifre. Vi kan så yderligere tilføje begrænsninger i form af en valideringsregel, der siger, at tallet måske skal ligge mellem 3000 og 8000.

VALIDERING

Ved validering af et felt i en tabel forstås, at det kontrolleres om indtastningen i feltet er gyldig. Ved konstruktion af tabeller kan vi selv bestemme, hvad det skal være gyldigt at indtaste i de enkelte felter.
Ved brug af skabeloner kan man vise, hvordan man ønsker, at data skal indtastes i feltet. I et datofelt kan der eksempelvis stå dd-mm-yyyy, hvilket betyder, at man gerne vil have indtastningen på en bestemt form. En skabelon kaldes også for en *inputmaske*.

Ofte vil vi også lade vores database selv angive et nyt nummer for primærnøglen, der automatisk tæller én op, når der kommer en ny post i en tabel. Dette kaldes autonummerering. En autonummerering fortolkes af databaseprogrammet som værende af samme datatype som heltal.

Bilag 3

E/R-diagram

Et E/R-diagram er en god måde at skitsere sammenhængen mellem tabeller i en database.

E/R-diagrammet skal give os et overblik over, hvilke tabeller vi har behov for i vores database og hvilken sammenhæng, der er mellem disse tabeller.

E/R-DIAGRAM

E/R-diagram står for *Entitets/Relations-diagram*. Diagrammet viser, hvilke sammenhænge der er mellem de enkelte tabeller i en database.
Entitet betyder i database-sammenhæng at være en selvstændig enhed eller på anden vis at have specielle egenskaber, der giver data sin egen eksistens. Hver tabel, vi opretter i vores database, vil derfor repræsentere en entitet.

I en virksomhed vil der være en masse forskellige entiteter, vi har informationer om. Eksempelvis vil kunder, varer, ordrer og medarbejdere mv. være forskellige entiteter.

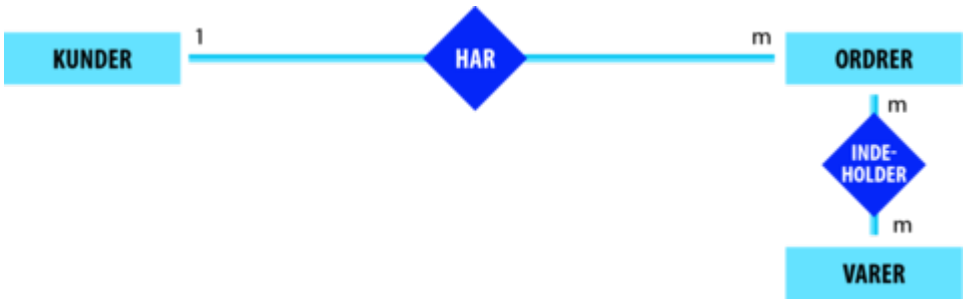
Relationerne viser så de sammenhænge, der er mellem entiteterne (tabellerne).

Vi kan skitsere E/R-diagrammet for vores kundetabel og ordretabel på følgende måde:

Figur 7.11
E/R-diagram mellem kundetabel og ordretabel.
Dette diagram angiver, at der er en relation mellem tabellen *Kunder* og tabellen *Ordrer*. Vi læser dette diagram sådan, at en kunde kan have mange ordrer, mens en ordre kun kan have en enkelt kunde tilknyttet. Derfor står der 1 og m i diagrammet. Ordet har i midten af diagrammet udtrykker blot, på hvilken måde tabellerne er relateret.

I ovenstående tilfælde er der tale om det, der kaldes en 1-m relation mellem de to tabeller. Når du selv skal opbygge en database, er det denne type relation, du i langt de fleste tilfælde skal forsøge at få opbygget mellem tabellerne.

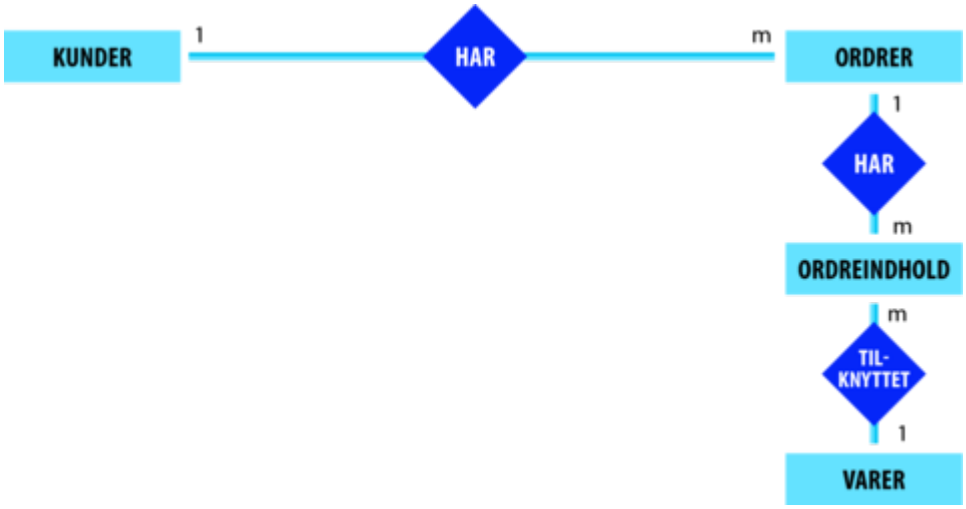
Vi kan nu udvide ovenstående eksempel til også at omfatte tabellen med varer. Vi får derefter følgende E/R-diagram:



Figur 7.12
E/R-diagram mellem kundetabel, ordretabel og varetabel.

Dette diagram viser ligesom det forrige diagram, at en kunde kan placere flere ordrer, mens en ordre kun kan have én kunde tilknyttet. Den nye relation, der er tilføjet, viser, at en ordre kan indeholde mange forskellige grupper af varer. Omvendt kan en varegruppe være indeholdt i mange forskellige ordrer. Derfor er der opstået en m-m relation mellem ordretabellen og varetabellen. Denne relationstype kan vi ikke arbejde med, når vi konstruerer vores database elektronisk, fordi vores valgte databasesystem ikke kan håndtere m-m relationer.

Vi løser dette ved at opbygge en ny tabel med relevante data. Denne tabel skal oprettes mellem de to tabeller, der har m-m relationen, hvorefter vi i stedet får to nye 1-m relationer.



Figur 7.13
E/R-diagram med udelukkende 1-m relationer.

Hvordan vi præcist opbygger denne nye tabel i databasen kan ses i afsnit 7.5 Konstruktion af database, hvor der gennemgås et eksempel på opbygning af en database fra start til slut.