Introduktion till databaser

I första lektionen kommer vi bekanta oss med några grundläggande databasbegrepp som databasmodell, databasarkitektur, databas och databashanterare. Vi kommer även kort att ta en titt på den idag mest använda databasmodellen, nämligen relationsmodellen.

Var används databaser?

Databaser har blivit en väldigt viktigt del för att klara av vardagslivet i ett modernt samhälle. Under loppet av en dag kommer många av oss att utföra eller vara involverad i aktiviteter som på något sätt kan sammankopplas med en databas. Det sker till exempel när vi använder vårt bank- eller kreditkort för att betala varor i en butik eller ta ut pengar från en bankomat, bokar resor via en resebyrå, lånar böcker i ett bibliotek, tecknar en bilförsäkring eller registrerar oss på en kurs vid Mittuniversitetet. Vid alla dessa aktiviteter, och många fler, finns det ett behov av att lagra relevant data för aktiviteten. En bank behöver till exempel lagra hur mycket pengar som ska dras från ditt bankkonto när du använt ditt bankkort.

Databaser används överallt där det finns ett behov av att lagra data och där lagrad data behöver bearbetas, användas eller presenteras på olika sätt. Tänk bara på alla register du själv kan finnas registrerade i. Folkbokföringsregistret, körkortsregistret, bokklubbar, prenumerationer på olika tidskrifter, försäkringsföretag med flera. Alla dessa register lagras i någon databas. De data som finns av dig i dessa register kan till exempel presenteras för dig på olika sätt. Data om dina personuppgifter kan sammanställas och skickas hem till dig i pappersformat eller göras tillgänglig för dig via en hemsida på internet.

Det är inte bara stora företag eller organisationer som har behov av att lagra data i databaser. Även mindre företag, föreningar eller privatperson kan ha nytta av en databas. Ett företag kan lagra data om sina produkter, en förening kan lagra data om sina medlemmar och som privatperson kanske vill lagra data om en skiv- eller frimärkssamling.

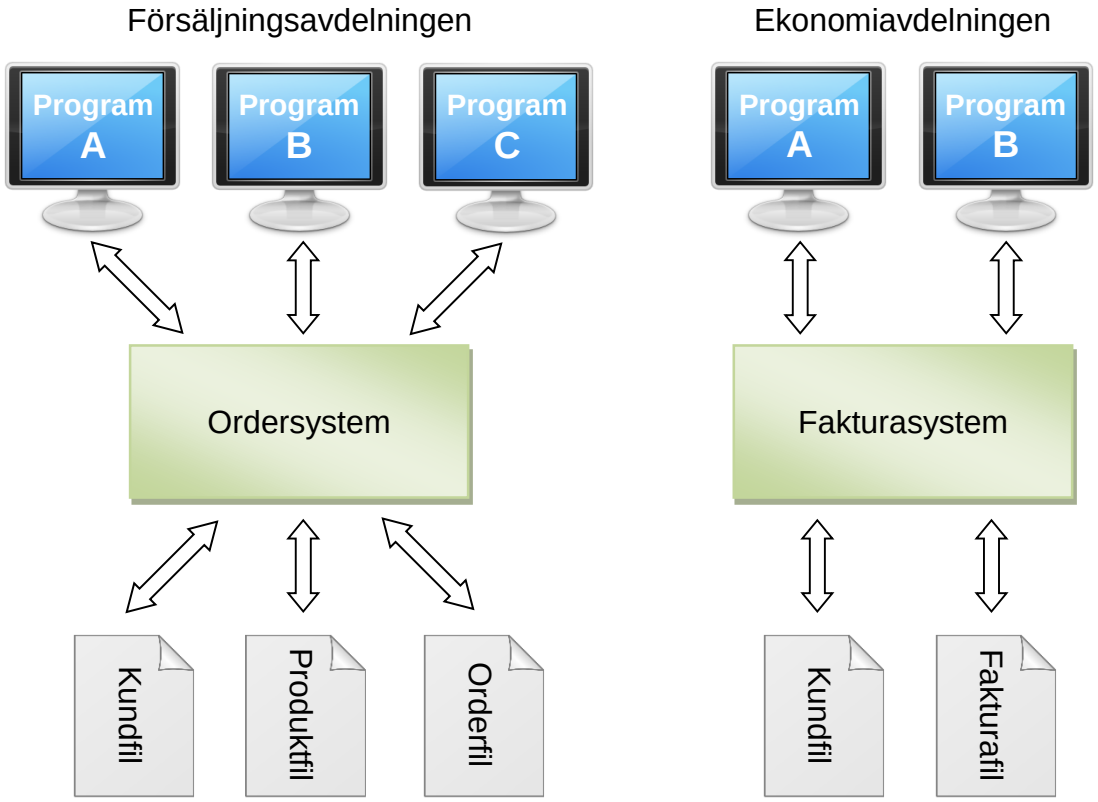
Att skapa och lagra data i en databas är inte någon ny företeelse som enbart kan kopplas till den elektroniska tidsåldern. Redan långt innan datorer fanns var ett vanligt sätt att lagra data, på ett systematiskt sätt, att använda ett så kallat kartotek. I ett kartotek lagras data oftast på papperskort som sen sorterades i alfabetisk eller numerisk ordning. På biblioteken användes kartotek för att föra register över bibliotekets alla böcker. För att till exempel finna i vilken hylla en viss bok var placerad fick vi manuellt bläddra igenom kartoteket för att finna det kort som tillhörde den bok vi letade efter. På detta kort kunde vi sen finna information om bokens placering. Figur 1 nedan visar ett exempel på hur ett kartotek kan se ut.

Figur 1. Exempel på kartotek.

Senare, när den tekniska utvecklingen gjorde det möjligt för större företag och organisationer att lagra och behandla stora mängder data med en dator, introducerades de så kallade filbaserade systemen. Dessa system kom helt att ersätta de manuella kartoteken.

Filbaserade system

Det som kännetecknar ett filbaserat system är att data lagras i många olika filer och att ett antal olika klientprogram i sin tur använder dessa filer för att utföra diverse tjänster, som att producera rapporter, åt användarna av programmen. Varje program definierar och styr sina egna data. Detta är vad vi kan kalla ett decentraliserat system där till exempel varje avdelning i ett företag har sin egen uppsättning av data och program (figur 2).



Figur 2. En schematisk beskrivning av ett filbaserat system.

Från de filbaserade systemen introducerades begrepp som data, fält (field), post (record) och fil (file), vilka beskrivs i tabell 1.

Tabell 1

Data	Ett tecken, det vill säga en bokstav, en siffra 0 till 9 eller en symbol (?,+,-,#,*,& med flera).
Fält	Innehåller ett antal tecken som har en viss innebörd. Till exempel ett namn, ett tal, en tidpunkt, en adress.
Post	En uppsättning fält som hör ihop med varandra. Till exempel namn och adress för en viss person.
Fil	Ett antal poster som relaterar till varandra på ett naturligt sätt. Till exempel alla medlemmar i en idrottsförening, alla anställda i ett företag, alla produkter i ett lager.

En fil är en samling poster som innehåller logiska data. Varje post består av en logisk uppsättning fält. Varje fält representerar egenskaper som tillhör objekt från den verkliga världen.

Exempel på innehåll i en fil som lagrar data om medlemmar i en idrottsförening kan se ut så här:

```
Namn;Adress;Telefon;Födelseår;Sektion
Kalle Svensson;Storgatan 1;123456;1977;Innebandy
Maria Bergfors;Trossgränd 11;070123456;1994;Curling
Peter Pettersson;Gågatan 32;5551234;1990;Innebandy
Magnus Svensson;Akademigatan 1;457765;1979;Fotboll
Anna Andersson;Vattuvägen 2;0908776;2001;Fotboll
```

Begränsningar

Filbaserade system har ett antal begränsningar. Data ligger ofta isolerade i separata filer och det kan vara svårt att komma åt och kombinera data från olika filer. Det är även lätt att data dubbellagras, vilket leder till redundans. Det vill säga att samma data lagras i många olika filer. Här finns det då stor risk att data blir inkonsistent, den innehåller motsägelser. Vi uppdaterar data i en fil, men glömmer att uppdatera samma data som finns lagrat i en annan fil.

En annan begränsning är att definitionen av data, det vill säga innebörden av data (att första fältet i en post är medlemmens namn) definieras i applikationsprogrammet och inte definieras oberoende och separat. Därmed blir data beroende av programmet för att kunna användas. Programmet i sin tur är även beroende på hur data i filen lagras. Vi kan till exempel inte ändra på ordningen av fälten i en post utan att uppdatera programmet samtidigt så att det kan hantera ändringen.

I applikationsprogrammet är möjligheten att bearbeta data på ny olika sätt ofta begränsat. Vilka olika typer av rapporter som till exempel kan sammanställas av data är fast och behöver slutanvändaren en ny typ av rapport måste ett nytt program utvecklas eller det befintliga uppdateras.

Samma data som lagras i olika filer kan vi inte garantera är kompatibla med varandra. Data kan vara lagrad i olika filformat och data kanske inte är uppbyggd på exakt samma sätt. Applikationsprogrammen kunde även vara skrivna i olika programspråk som tolkar och hanterar datatyper (vad är en teckensträng, ett heltal, ett tecken, ett decimaltal) på annorlunda sätt. I ett programspråk kan det största heltalet som kan hanteras skiljas från ett annat programspråk.

Databasbaserade system

För att komma till rätta med de problemen filbaserade system led av utvecklades så småningom databasbaserade system. 1970 skrev E.F. Codd ett dokument i vilket han med matematik beskrev ett sätt att lagra stora mängder data på smartare sätt. Utifrån detta dokument definierades relationsmodellen, vilket i sin tur innebar utvecklingen av relationsdatabaserna.

Databaserna ersatte de filbaserade systemen och tekniskt sett är en databas en betydligt mer avancerad lösning. En databas (DB) kan definieras som en samling av logiska data som hör ihop och en beskrivning av samma data, som avser att uppfylla de informationskrav som finns i en organisation. En databas är en typ av en enhetlig lagring av data, definierade en gång och som kan används samtidigt av många olika slutanvändare.

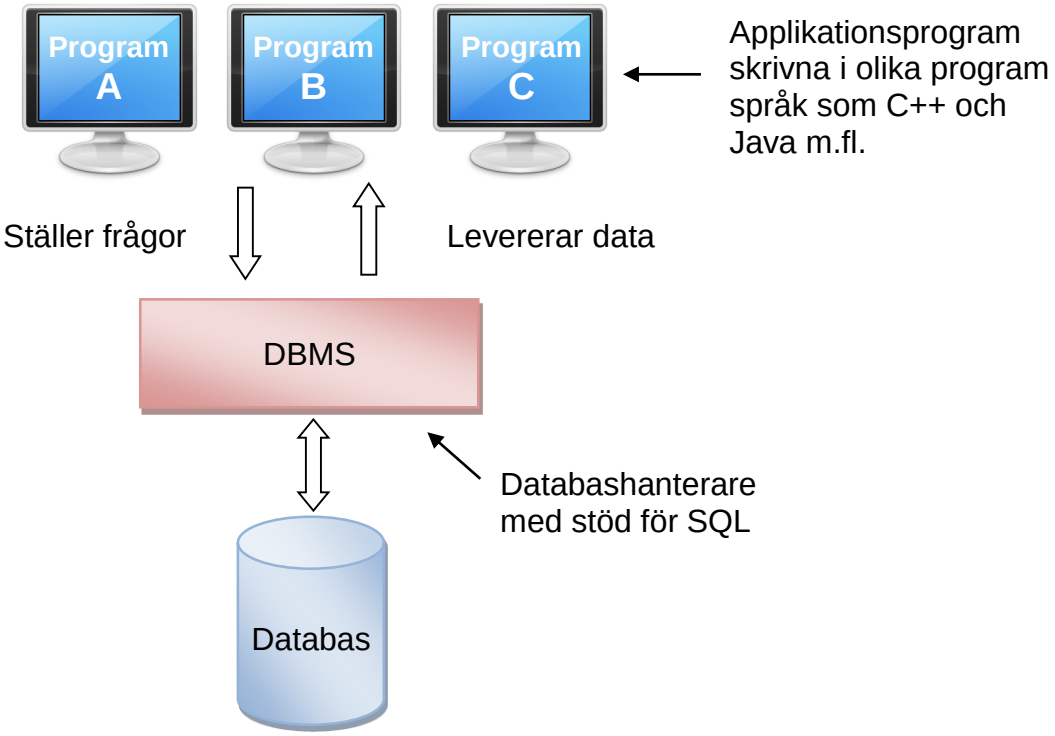
Databashanterare

En databashanterare (DBMS, database management system) är en samling program som gör det möjligt att skapa och använda databaser samt kontrollera åtkomsten till data på olika sätt. DBMS håller reda på databasens struktur, vilka tabeller den innehåller (data lagras i tabeller, mer om det senare), vilka kolumner tabellerna innehåller och vilka typ av data vi kan lagra i dessa. Detta kallas för databasens schema och bestämmer alltså vilka data som kan lagras i databasen.

DBMS är användarens gränssnitt mot databasen och genom DBMS kan användare till exempel ställa olika frågor mot databasen för att söka bland de data som finns lagrat. Förenklat sett kan en sådan förfrågan gå till så här:

1. En användare efterfrågar data från databasen och ställer en fråga, en sökning i databasen, med hjälp av ett frågespråk (till exempel SQL, Structured Query Language).
2. DBMS tar emot frågan och analyserar den (vad ska göras?).
3. DBMS undersöker schemat och gör en mappning från detta till hur data faktiskt är lagrat rent fysiskt (en mappning mellan det externa schemat ner till det interna schemat, mer om detta senare).
4. DBMS utför nu de nödvändiga operationerna för att få fram de data som efterfrågades och leverera dessa till användaren.

En schematisk bild över detta tillvägagångssätt kan du se i figur 3.



Figur 3. En schematisk bild över förfrågningar mot en databas.

Vad innehåller ett DBMS?

En databashanterare innehåller många olika delar och erbjuder många olika tjänster år slutanvändare. En DBMS innehåller bland annat ett datadefinitionsspråk (DDL, Data Definition Language) och ett datamanipuleringsspråk (DML, Data Manipulation Language). SQL är ett exempel på båda dessa typer av språk. Det vill säga SQL innehåller både DDL och DML.

DDL använder vi för att definiera databasens struktur, vilka tabeller, kolumner och datatyper med mera som ska användas. DML använder vi för att komma åt och ändra på data i databasen. Vi ställer frågor som kan lägga till, uppdatera, ta bort och hämta data från databasen.

En DBMS måste också innehålla en systemkatalog i vilken metadata om databasen lagras. Metadata är data om data, det vill säga data som beskriver annan data. I databas-sammanhang innehåller metadata beskrivningar av data som lagras. Det kan vara namn på själva databaserna, namn på alla tabeller i databaserna, namn på alla attribut/kolumner i tabellerna, vilken datatyp som används (ska det lagras tal, text etc), vilka eventuella begränsningar som gäller för data (till exempel enbart tillåtet med tal mellan 1 och 10, eller max tre bokstäver). Information vilka användare som har tillgång till databasen samt vilka data de kommer åt och vad användaren kan göra med detta (läsa, skriva, uppdatera). Systemkatalogen lagras också som tabeller vilket innebär att det går att ställa frågor till den på samma sätt som till databasens övriga tabeller.

Dessutom måste en DBMS bestå av mekanismer som stöder dataintegritet, säkerhet, säkerhetskopiering, optimering och mycket annat. Eftersom fokus i denna kurs inte ligger i hur en databashanterare fungerar internt är detta inget vi tittar på i detalj.

Applikationsprogram

DBMS är som sagt användarens gränssnitt mot databasen. Ofta används en specifikt framtaget applikation, med ett grafiskt användargränssnitt (GUI), för att använda de funktioner DBMS tillhandahåller. GUI anpassas till de krav som en specifik grupp av slutanvändare har. En grupp av slutanvändarna kan till exempel få information om anställda i företaget medan en annan grupp får information om produkter företaget producerar.

En applikation brukar traditionellt sett inte vara en del av själva DBMS utan är separata applikationer skrivna i ett programmeringsspråk som C++, Java och C#. Nu för tiden erbjuder faktiskt många DBMS möjligheten att skapa applikationer direkt i DBMS. En användare kan till exempel skapa applikationer direkt i DBMS som innehåller formulär för att presentera och lägga till data.

Ett annat vanligt sätt att komma åt en databas är via applikationer som körs på en webbserver. Med hjälp av tekniker som HTML, PHP och ASP med mera kan det grafiska användargränssnittet nås via hemsidor på internet.

Även om applikationer kan användas för att komma åt data i en databas erbjuder också DBMS denna möjlighet. På olika sätt kan vi skapa nya databaser och tabeller samt lägga till och ta bort data med mera direkt från DMBS.

Roller i ett databassystem

I ett databassystem finns ett antal olika grupper av användare. En databasadministratör (DBA) och dataadministratör (DA) har det fulla tekniska och operativa ansvaret för DBMS. I hans/hennes arbete kommer följande att vara viktigt:

- Att definiera det konceptuella schemat, att avgöra vilken information som ska lagras i databasen. Detta kallas logiska eller konceptuell databasdesign.
- Att definiera det interna schemat, att besluta hur data ska representeras i databasen, så kallade fysisk databasdesign.
- Kommunicera med och utbilda slutanvändare. För att få information om vilken typ av data slutanvändarna har behov av att lagra samt lära dem hur applikationerna ska användas.
- Att definiera och utföra säkerhets- och integritetskontroller.
- Att definiera och utföra rutiner för säkerhetskopiering och återställning av data.
- Att övervaka databasens prestanda och säkerställa driften av DBMS.

Andra grupper av användare är programmerare av de olika applikationer och givetvis slutanvändarna som ska använda databasen.

Kontroll av redundant data

Väldigt ofta använder olika applikationer samma data i en databas. Ett företag kan ha en applikation som hanterar lönerna för de anställda, en annan applikation för att hantera kunder och ännu en för att hantera varje säljares vinst. Alla dessa applikationer använder data om anställda och kunder. En databas kan med andra ord användas i ett fleranvändarsystem. Detta måste hanteras på ett säkert sätt och är en uppgift för DBMS.

Fördelen med att använda en DBMS är att data om anställda och kunder lagras i bara en databas på ett ställe. På detta sätt kan vi undvika att data dubbellagras (redundans) och eventuella förändringar som görs i data kommer alla användare av databasen att se direkt.

Ett problem med de filbaserade systemen var att samma data kunde lagras i olika filer, vilket ledde till dataredundans. Samma data kan finnas i flera olika filer som i sin tur kan innebära att dataintegriteten förloras: samma person har fått två olika telefonnummer i två olika filer. Hur ska vi säkerställa att samma data uppdateras överallt?

Vad ska hända när vi ta bort data? Om en säljare avslutar sin anställning på ett företag, vad ska hända med alla kunder säljaren hade?

Frågeställningarna ovan kan vara svåra att hantera så här i början. Vi återkommer till detta senare i kursen. Poängen är att när vi konstruera en databas måste dessa typer av frågeställningar beaktas. Genom att använda en DBMS kan vi lättare undvika en del av problemen som nämns ovan.

Effektiv hantering av data

En mycket viktig egenskap hos en DBMS är att de har optimerade algoritmer för att lagra och inte minst söka efter data på det fysiska lagringsmediet (ofta en hårddisk). En DBMS kommer alltid kommer att försöka använda det mest snabba och effektiva sättet att hantera data i databasen. Om du till exempel skapar ett eget filbaserat system, måste du själv skriva dessa algoritmer för att hämta data. Är du en professionell C++-programmerare kommer du säkert kunna skriva mycket effektiva sökalgoritmer som kan mäta sig mot de en DMBS använder. Att i stället lagra data i en databas är i de flesta fallen en mycket enklare lösning.

SQL

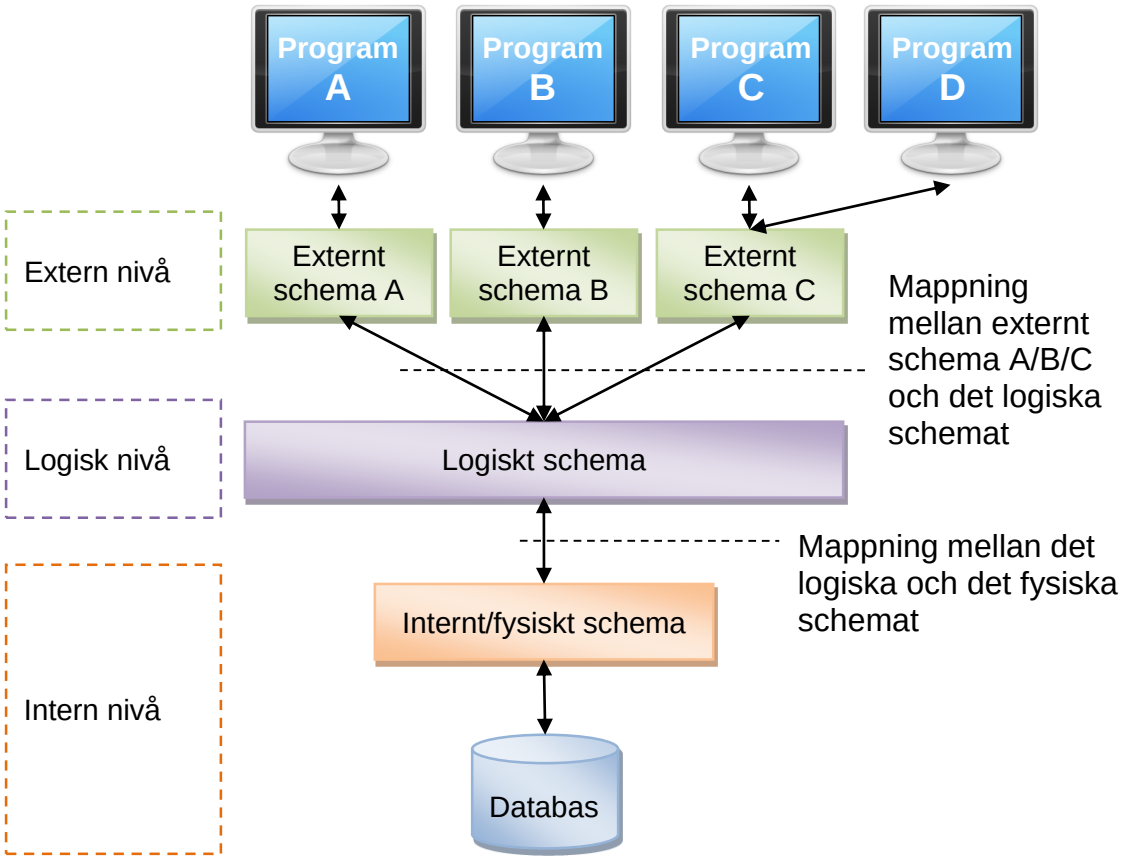
En annan viktig egenskap hos ett DBMS är stödet för ett frågespråk, vanligtvis SQL. Detta gör det möjligt att komma åt och uppdatera data i databasen på ett relativt enkelt och effektivt sätt. Om du en gång har lärt sig att använda SQL kan du använda den på olika plattformar så länge DBMS stöder det. Denna typ av standardiserat frågespråk finns inte bland de traditionella programmeringsspråken.

ANSI/SPARC-arkitekturen

ANSI/SPARC är en förkortning för American National Standard Institute/Standard Planning and Requirements Committee och är en arkitektur som användas av många DBMS. Målet är

att separera varje användares vy av databasen från hur databasen fysiskt är implementerad (hur data lagras). En användare behöver inte känna till hur data är organiserat eller lagras för att använda den. Om tabellerna i databasen växer i antal kolumner eller rader, ska inte det påverka användarens befintliga vyer av data i databasen. För att uppnå detta skiljer ANSI/SPARC mellan fysisk och logisk representation av data: fysisk och logisk dataoberoende.

ANSI/SPARC-arkitekturen är indelad i tre nivåer som beskrivs av var sitt schema. I varje nivå är det samma data som används, men som representeras på olika sätt. De tre nivåerna är: extern, logisk och intern. Mellan varje nivå sker det en mappning mellan nivåernas olika scheman. Se exempel i figur 4 nedan.



Figur 4. De tre nivåerna i ANSI/SPARC och mappningen mellan de olika schemana.

Alla tre nivåer och hur mappningen mellan nivåerna hanteras av DBMS.

Extern nivå

Detta är den nivå som finns närmast användaren. Det är användarens syn på databasen. Beskriver endast den del av databasen som är relevant för varje användare. En användare har kanske endast åtkomst till del av en tabell, medan en annan användare kommer åt hela tabellen. En slutanvändare använder något frågespråk för att kommunicera med databasen, vanligtvis SQL. SQL innehåller två underspråk (data sublanguage, DSL), data definition language (DDL) och data manipulation language (DML). Det kan finnas många olika externa vyer av samma data. Den externa nivån är oberoende av den logiska nivån. DBA kan göra

ändringar i den logiska nivån som kommer att återspeglas i den externa nivån utan att göra några ändringar i den.

Logisk nivå

Den logiska (konceptuella) nivån är en koppling mellan den externa nivån och den interna nivån. Denna nivå representerar alla relationer, attribut, samband, användare, säkerhets- och integritetsregler med mera i databasen. Det är denna nivå en DBA arbetar med.

Intern nivå

Denna nivå representerar det närmaste vi kommer den fysiska lagringen av data. Säger något om hur data lagras i databasen. Som användare märker man inte så mycket av denna nivå. Här kan man till exempel ändra och optimera sätt att lagra data på för att uppnå bättre prestanda. Den interna nivån vet dock inget om hur data i databasen organiseras på till exempel hårddisken i form av diskblock och disksidor.

Datamodeller och konceptuell modellering

En datamodell är en integrerad samling av begrepp som beskriver data, förhållandet mellan data och regler som är förknippade med data i en organisation. Den försöker ge en abstraktion av den verklighet vi vill lagra data om. Verkligheten är så komplex att den information vi kan lagra i en databas bara kommer att vara en förenkling av den. Ju bättre informationen är strukturerad, desto lättare är det dock att tillgodogöra sig de möjligheter en databas erbjuder, som sökning och analys av data. Därför är det bra att ha en klar och tydlig bild över vilken information vi vill ha i databasen. En datamodell är tänkt att ge en sådan översikt. I denna kurs är det två datamodeller vi tittar närmare på, nämligen relationsmodellen och Entity-Relationship-modellen (ER-modellen). Båda dessa modeller har vissa gemensamma drag.

Relationsmodellen är ett till viss del baserad på matematisk mängdlära (relation är den matematiska termen för tabell). Relationsmodellen är den modell många stora DBMS baseras på och kallas då för relationsdatabashanterare (RDBMS). Denna modell är en så kallad implementationsmodell, en modell som implementeras (används) av en DBMS. Relationsdatamodellen beskriver ett sätt att se på data från en logisk synvinkel. Tre aspekter av data tas upp i modellen: datastrukturer, datamanipulation och dataintegritet.

ER-modellen å andra sidan är en så kallad konceptuell datamodell, och används för datamodellering oavsett vilken DBMS vi använder för att förverkliga databasen i. Vi använder en konceptuell modell för att snabbt och tydligt visa den verklighet vi försöker modellera. En konceptuell modell måste översättas till den implementationsmodell DBMS använder.

Implementationsmodeller

Det finns ett antal olika implementationsmodeller och tidigare var den hierarkiska datamodellen och nätverksmodellen vanliga. Nu förtiden är dock relationsmodellen som är den mest förekommande implementationsmodellen i en DBMS.

Relationsmodellen

Relationsmodellen bygger på begreppet från matematiska relationer. Data och samband representeras som tabeller och bara tabeller, som alla har ett antal kolumner och rader (se tabell 2).

Tabell 2

Person

personnr	förnamn	efternamn	adress	postnr
730527	Eva	Andersson	Gatan 1	21775
860630	Tord	Svensson	Vägen 13	83432
791123	Sandra	Nilsson	Bye 3	79026

Postort

postnr	postort
79026	Enviken
83432	Brunflo
21775	Malmö

Primärnyckel — Främmandenyckel — Primärnyckel

Tabellerna kopplas samman med hjälp av primärnycklar (primary key (pk)) och främmande nycklar (foreign key (fk)).

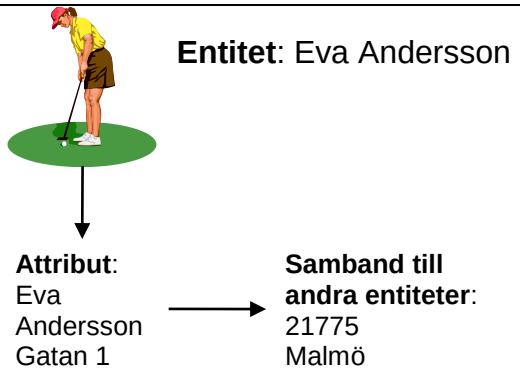
Utöver relationsmodellen finns även databaser som använder en implementationsmodell baserad på objektorienterade koncept. Två exempel är objektorienterade databaser och objekt-relationella databaser.

Konceptuella datamodeller

En konceptuell datamodell kan vi använda för att skapa en beskrivning av hur någonting fungerar. Det finns olika typer av konceptuella modeller, men när vi pratar om databaser är det oftast ER-modellen som används.

ER-modellen

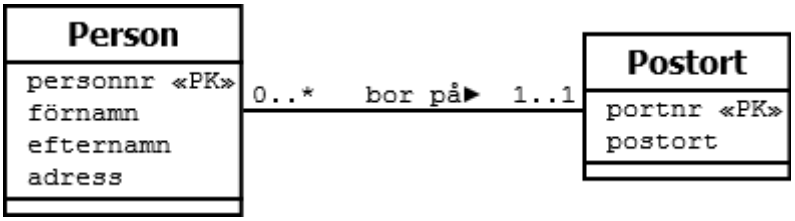
ER-modellen innehåller entiteter och samband mellan entiteter. En entitet är föremål eller objekt från den verkliga världen som vi önskar lagra information om i databasen. I figur 5 ser du ett exempel på en entitet.



Figur 5. Exempel på entitet.

En entitet har vissa egenskaper som representeras genom ett antal attribut. Traditionellt är dessa verkliga fakta om objektet i verkligheten som efternamn, förnamn, adress och telefonnummer. Utöver detta kan en entitet delta i en eller flera samband med andra entiteter. I figur 4 finns ett samband mellan entiteten ”Eva Andersson” och entiteten ”21755”.

Det är viktigt att skilja mellan entitetstyper och förekomster av entiteter. En entitet är av en viss entitetstyp. Entitetstypen säger något om vilka data vi kan lagra, vilka typer av samband som tillåts och hur förekomsterna ska identifieras i databasen. Förekomsten ”Eva Andersson” är av en viss entitetstyp som vi kan kalla Person. Förekomsten ”21775” är av en annan entitetstyp som vi kan kalla Postort. Mellan dessa entitetstyper tillåts ett samband av typen ett-till-många. En förekomst av entitetstypen Person bor på en viss postort, och en förekomst av entitetstypen Postort kan vara hem för många personer. Figur 6 illustrerar detta.



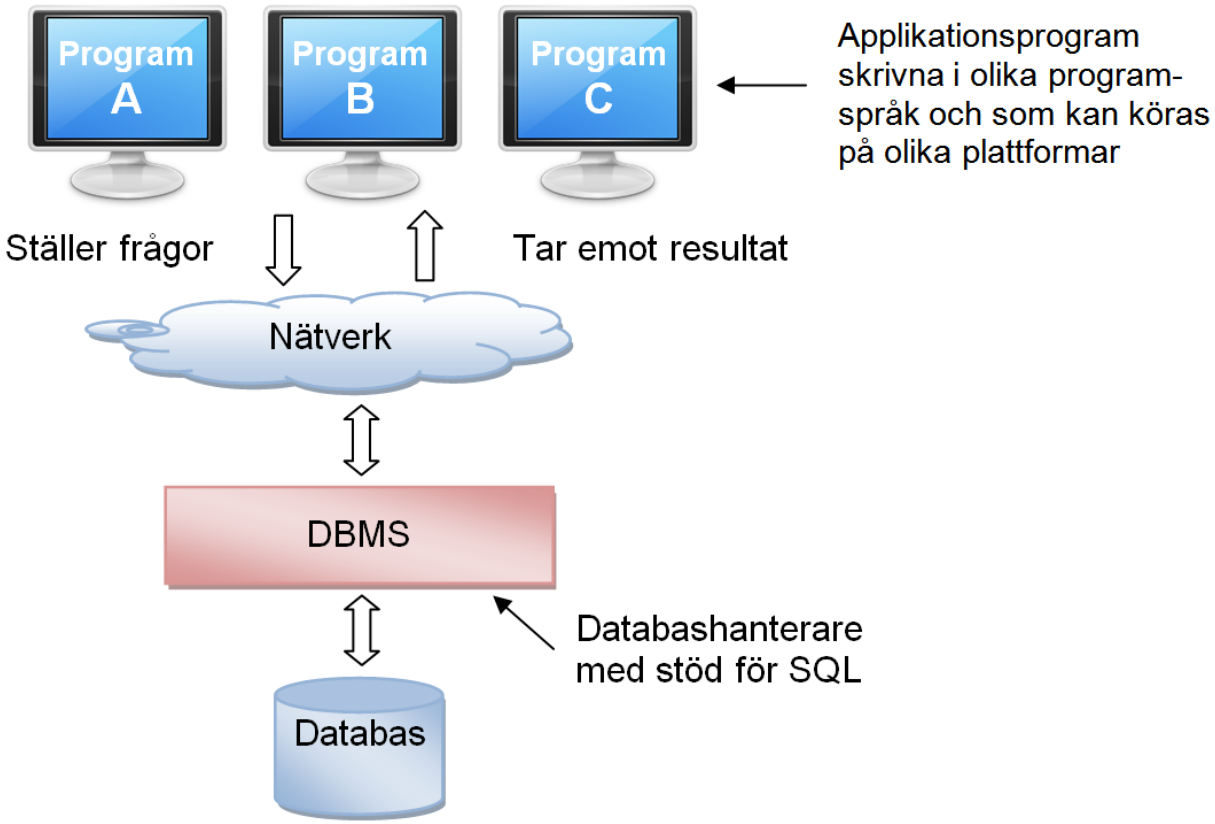
Figur 6. Exempel på ER-modell.

Du behöver inte förstå figur 6 i detta skede utan vi återkommer till detta ämne senare i kursen där vi i detalj tittar på hur detta representeras i både relationsmodellen och ER-modellen.

Klient/Server-modellen

Ett vanligt sätt att sätta upp ett databassystem är att dela upp systemet i en klientdel och en serverdel. Data lagras i en databas på serverdelen och en eller flera klienter ansluter till servern och hämtar data från denna. På klienten körs ett program som samlar in data och frågor från användaren, förbereder dem för servern och gör sen en förfrågan (med SQL) till servern. Servern väntar på förfrågningar från klienter och när en sådan kommer analyserar

och bearbetar servern förfrågan och skickar tillbaka ett resultat till servern. Klienten tar emot resultatet och visar detta för användaren i programmet (figur 7).



Figur 7. Klient/Server-modellen.

En sådan modell har flera fördelar:

- Klient- och serverdelen kan köras på olika plattformar.
- Nätverkstrafiken minskar (jämför med andra modeller). Endast förfrågan och resultat skickas över nätverket.
- Klientapplikationer kan skrivas i olika språk och anpassas för olika plattformar.
- Databasen kan göras åtkomlig för alla och överallt (via t.ex. webbläsare och mobiltelefoner).
- Servern kan vara dedikerad till att enbart hantera databasen vilket leder till ökad effektivitet. Det är även enkelt att i efterhand bygga ut servern med nya och bättre hårdvara allteftersom behov uppstå.