Jerry Johansson, 2021-10-16

Databaser

Databashantering

Ett databassystems funktioner och uppbyggnad

Läs genom:

http://www.databasteknik.se/webbkursen/databaser/index.html

Centrala Begrepp

Databas (engelska: **database**). En samling data som hör ihop på något sätt. För det mesta brukar man anta att den hanteras av en databashanterare och att den har ett schema. Ibland används ordet "databas" även för att beteckna det som vi här kallar för databashanterare.

Databashanterare, databashanteringssystem, DBHS (engelska: **database management system, DBMS**). Ett program eller ett system av program som kan hantera en eller flera databaser. De flesta databashanterare är generella, och kan hantera olika sorters databaser. Exempel på databashanterare är Oracle och Microsoft Access.

Schema, **databasschema** (engelska: **schema**, **database schema**). En beskrivning av vilka data som kan finnas i en databas, oberoende av vilka data (innehållet) som råkar finnas i databasen just nu. Exempel: I relationsmodellen, där man beskriver världen med hjälp av tabeller, består schemat huvudsakligen av vilka tabeller som finns i databasen, och vilka kolumner de har, men inte vilka värden som råkar finnas i tabellerna just nu. I stället för "schema" säger man ibland intension, åtminstone när man talar om schemat för en relation i relationsmodellen.

Datamodell (engelska: **data model**). Ett sätt att beskriva världen, som man kan använda till exempel när man bygger upp en databas. Exempel: relationsmodellen, där man beskriver världen med hjälp av tabeller. I industrin används ordet "datamodell" ibland för att beteckna det som vi här kallar för schema.

SQL. Av **Structured Query Language**. Ett deklarativt frågespråk som används i de flesta databashanterare. Hette från början **SEQUEL**, och uttalas fortfarande så av en del.

Fråga (engelska: **query**). En sökning i databasen. Den kan vara uttryckt i ett särskilt frågespråk som SQL, men det kan också vara ett program i till exempel C. Ibland kallar man alla operationer, alltså även till exempel att lägga in data i databasen, för "frågor".

Frågespråk (engelska: **query language**). Ett språk som man använder för att göra sökningar i databasen, eller "ställa frågor till databashanteraren", kallas för *frågespråk*. Synonymer: datamanipuleringsspråk, DML.

DDL, **datadefinitionsspråk** (engelska: **data definition language**). Ett språk som används för att tala om för databashanteraren vilka data som ska kunna lagras i databasen.

DML, **datamanipuleringsspråk** (engelska: **data manipulation language**). Ett språk som används för att styra databashanteraren när man vill lagra, ändra eller söka i de data som finns i databasen. Kallas även frågespråk.

Logiskt dataoberoende (engelska: **logical data independence**). Möjligheten att ändra på den logiska strukturen hos en samling data, utan att man också måste ändra på de program som arbetar med dessa data. Exempel på den logiska strukturen i en databas är vilka

kolumner som finns i en tabell.

Fysiskt dataoberoende (engelska: **physical data independence**). Möjligheten att ändra på den fysiska strukturen hos en samling data, utan att man också måste ändra på de program som arbetar med dessa data. Exempel på den fysiska strukturen i en databas är vilka index som finns.

Tre-schema-arkitekturen (engelska: **the three-schema architecture**). Kallas även **tre-nivå-arkitekturen** (engelska: **the three-level architecture**). Samma databas beskrivs på tre olika nivåer, med tre olika scheman: ett externt schema överst (närmast användaren) ett logiskt schema i mitten, och ett fysiskt schema underst (längst in i datorn). Genom att man kan ändra i ett av dessa scheman utan att schemana ovanför påverkas, får man bättre dataoberoende.

Vy (engelska: **view**). En vy är ett sätt att se en databas. Olika användare kan betrakta samma databas genom olika vyer. I SQL är en vy en SQL-fråga som fått ett eget namn. När man tittar på en vy i SQL ser den ut som en tabell, men innehållet beräknas på nytt (genom att SQL-frågan körs) varje gång man tittar på den.

Konceptuell datamodell eller **begreppsmässig datamodell** (engelska: **conceptual data model**). En datamodell som beskriver verkligheten i termer av de saker som finns i den verkligheten. Den säger inget om hur de ska lagras i en databas. Exempel: ER-modellen.

Implementationsmodell eller **implementeringsmodell** (engelska: **implementation data model**). En datamodell som man kan använda i en verklig databashanterare, och alltså skapa eller realisera ("implementera") sin databas med. Den vanligaste implementationsmodellen numera är relationsmodellen.

Relationsmodellen (engelska: **the relational model**). En datamodell där man beskriver verkligheten genom att lagra data i tabeller. Relationsmodellen är den överlägset vanligaste datamodellen i databaser idag.

DBA, **databasadministratör** (engelska: **database administrator**). En person eller en grupp av personer som är ansvariga för driften av ett databassystem.

Applikationsprogram (engelska: **application program**) eller **tillämpningsprogram**. Ett program som är avsett för ett specifikt ändamål, till exempel för att låta biljettförsäljarna på SJ boka tågresor. I databassammanhang tänker vi oss att alla data lagras i en databas som hanteras av databashanteraren, och applikationsprogrammet kommunicerar med databashanteraren, men applikationsprogrammet är enklare att använda för biljettförsäljaren. I det programmet kan man kanske klicka på knappar för att välja resmål. Det är lättare än att kommunicera med databashanteraren direkt, i värsta fall genom att skriva SQL-kommandon.

Förklara kortfattat följande begrepp:

1. Databashanteringssystem, DBHS

2. Konceptuell datamodell

3. Schema

4. Vy

5. Fysiskt dataoberoende

6. Logiskt dataoberoende

7. Tre-schema arkitekturen

8. DDL

9. DML

10. SQL

Relationsmodellen

Läs genom:

http://www.databasteknik.se/webbkursen/relationer/index.html

Centrala begrepp

Relationsmodellen (engelska: **the relational model**). En datamodell där man beskriver verkligheten genom att lagra data i tabeller. Relationsmodellen är den överlägset vanligaste datamodellen i databaser idag.

Relationsdatabas (engelska: **relational database**). En databas organiserad enligt relationsmodellen, dvs med alla data lagrade i tabeller. Det är en vanlig missuppfattning att det är kopplingarna mellan tabellerna med hjälp av referensattribut som kallas relationer.

Relation. En tabell av den typ som används i relationsmodellen. Den här sortens relationer har inget att göra med ER-modellens relationships. Det är också en vanlig missuppfattning att det är kopplingarna mellan tabellerna med hjälp av referensattribut som är relationer. Bland annat i den svenska versionen av Microsoft Access används ordet (felaktigt) på det viset.

Tabell (engelska: **table**). I relationsmodellen lagras alla data i tabeller, som också kallas relationer. En sån här tabell skiljer sig från andra tabeller, som man till exempel ritar upp på papper, bland annat genom att raderna inte har någon särskild ordning.

Tupel (engelska: **tuple**). En rad i en tabell i relationsmodellen.

Rad (engelska: **row**). I databassammanhang menar man för det mesta en rad i en tabell i relationsmodellen, även kallad tupel.

Attribut (engelska: **attribute**). Betyder "egenskap", och används både om kolumnerna i tabellerna i relationsmodellen, och om egenskaperna hos entiteterna i ER-modellen.

Kolumn (engelska: **column**). I databassammanhang menar man för det mesta en kolumn i en tabell i relationsmodellen, även kallad attribut.

Primärnyckel (engelska: **primary key**).

1. I en relationsdatabas: En kolumn, eller en kombination av kolumner, som alltid har ett unikt värde för varje rad i tabellen. (Man får dock inte ta med några onödiga kolumner.) Om det finns flera möjliga primärnycklar säger man att man har flera kandidatnycklar, och man väljer en av dem som primärnyckel.
2. När man talar om fysiska lagringsstrukturer: Ett fält, eller en kombination av fält, som alltid har ett unikt värde för varje post i filen, och som filen dessutom är sorterad efter.
3. I verkligheten, eller i en konceptuell datamodell: Något som unikt identifierar en viss sak, till exempel personnumret på en person.

Referensattribut (engelska: **reference attribute**). Ett attribut (dvs en kolumn) i en tabell som refererar till (dvs "pekar ut rader i") en annan (eller ibland samma) tabell. Det är inga "pekare" av samma typ som man har i många programmeringsspråk, utan referensen består i att det står ett värde, och sen ska det stå samma värde på en rad i den refererade tabellen. Kallas även **främmande nyckel** (engelska: **foreign key**).

Främmande nyckel (engelska: **foreign key**). Samma sak som referensattribut.

Referensintegritet (engelska: **referential integrity**). Om två tabeller är hopkopplade med

referensattribut, så ska det värde som refereras till alltid existera: Om det står i tabellen **Anställd** att en viss anställd jobbar på avdelning nummer **17**, så ska det också finnas en avdelning med nummer **17**, i tabellen **Avdelning**.

Förklara kortfattat följande begrepp:

1. Relationsmodellen

2. Tupel

3. Referensattribut

4. Primärnyckel

5. Främmande nyckel

6. Referensintegritet

7. Kolumn

MySQL

Ladda ner MySQL, välj MySQL Community Server:

http://dev.mysql.com/downloads/mysql/

Vi använder MySQL Workbench för att bl a skriva SQL-frågor mot databasen. Man kan också designa och modellera databaser i Workbench.

http://dev.mysql.com/downloads/workbench/

https://www.mysql.com/products/workbench/

Datatyper:

http://www.w3schools.com/sql/sql_datatypes.asp

Skapa databasen bokhandel med SQL

```
CREATE DATABASE bokhandel;
```

Man kan också göra det i designläget i MySQL Workbench.

Skapa tabellen bok med SQL

```
CREATE TABLE bok(
        bokId INT NOT NULL AUTO_INCREMENT,
        bokTitel VARCHAR(50),
        bokForfattare VARCHAR(50),
        bokBeskrivn VARCHAR(50),
        bokIsbn VARCHAR(50),
        bokPris INT,
        PRIMARY KEY (bokId)
);
```

AUTO_INCREMENT fungerar som en räknare.

SQLite

https://www.sqlite.org/index.html

För Windows, ladda ner (Precompiled Binaries for Windows):

https://www.sqlite.org/download.html

För Mac OS X är SQLite redan installerat.

Tutorial SQLite:

https://www.tutorialspoint.com/sqlite/index.htm

SQL-syntax för SQLite:

https://www.sqlite.org/lang.html

Datatyper:

https://www.tutorialspoint.com/sqlite/sqlite_data_types.htm

Skapa en ny databas i SQLite som heter bokhandel.

Skapa tabellen bok med SQL

```
CREATE TABLE bok(
        bokId INTEGER PRIMARY KEY,
        bokTitel TEXT,
        bokForfattare TEXT,
        bokBeskrivn TEXT,
        bokIsbn TEXT,
        bokPris REAL
);
```

Skriver man ”INTEGER PRIMARY KEY” får man automatiskt en räknare.

SQL

Läs genom och för referens i SQL:

http://www.databasteknik.se/webbkursen/sql/index.html

http://www.w3schools.com/sql/default.asp

Codecademy har en kurs i SQL (Learn SQL).
https://www.codecademy.com

SQL – DDL och DML

DDL Data Defintion Language. SQL är ett datadefinitionsspråk som används för att tala om för databashanteraren vilken data som ska kunna lagras i databasen.

DML Data Manipulation Language. SQL är samtidigt ett datamanipuleringsspråk som används för att styra databashanteraren när man vill lagra, ändra eller söka i de data som finns i databasen.

Vad kan man göra med SQL?

SQL kan köra frågor mot en databas.

SQL hämta data från en databas.

SQL kan lägga till data i en databas.

SQL kan uppdatera data i en databas.

SQL kan radera data i en databas.

SQL kan skapa en ny databas.

SQL kan skapa en ny tabell i en databas.

SQL kan skapa lagrade procedurer i en databas.

SQL kan skapa vyer i en databas.

SQL kan bestämma rättigheter för tabeller procedurer, vyer och tabeller.

Kommandon i SQL:

SELECT - hämtar data från en databas.

UPDATE - uppdaterar data från en databas.

DELETE - raderar data från en databas.

INSERT INTO – lägger till data i en databas.

CREATE DATABASE – skapar en databas.

ALTER DATABASE – modifierar en databas.

CREATE TABLE – skapar en tabell.

ALTER TABLE – modifierar en tabell.

DROP TABLE – raderar en tabell.

CREATE INDEX – skapar ett index.

DROP INDEX – tar bor ett index.

Här går vi genom vanliga frågor i SQL utifrån tabellen för ”bok”

(Lite exempeldata för tabellen bok)

bok			
bokId	**bokForfattare**	**bokTitel**	**bokPris**
2	Strindberg, August	Röda rummet	100
5	Carroll, Lewis	Alice i underlandet	150
9	Lindgren, Astrid	Vi på saltkråkan	80
10	Tolkien, J.R.R	Sagan om ringen	70

SELECT

SELECT * FROM bok;
Hämtar alla kolumner i tabellen bok.

SELECT bokTitel, bokForfattare FROM bok;
Hämtar kolumnerna titel och författare från tabellen bok.

SELECT DISTINCT bokTitel FROM bok;
Hämtar bara unika värden i kolumnen titel.

SELECT * FROM bok WHERE bokTitel = ’Röda rummet’;
Hämtar alla kolumner från tabellen bok där titeln är ”Röda rummet”.

SELECT * FROM bok WHERE bokId = 1;
Hämtar alla kolumner från tabellen bok där bokid är 1.

SELECT * FROM bok WHERE bokTitel LIKE ’R%’;
Hämtar alla kolumner från tabellen bok där titeln börjar på R. LIKE ’%t’ hämtar istället titel som slutar på t.

SELECT * FROM bok WHERE bokTitel = ’Röda rummet’ AND bokForfattare = ’ Strindberg, August’;
Hämtar alla kolumner från tabellen bok där titeln är ”Röda rummet” och författare är ”Strindberg, August’”.

SELECT * FROM bok WHERE bokTitel = ’Röda rummet’ OR bokTitel = ’ Sagan om ringen’;
Hämtar alla kolumner från tabellen bok där titeln är ”Röda rummet” eller ”Sagan om ringen”.

SELECT * FROM bok WHERE bokPris > 100;
Hämtar alla böcker som kostar mer än 100 kr.

SELECT * FROM bok ORDER BY bokTitel ASC;
Hämtar alla kolumner från tabellen bok och sorterar stigande. Man kan också utesluta ASC för att sortera stigande.

SELECT * FROM bok ORDER BY bokTitel DESC;
Hämtar alla kolumner från tabellen bok och sorterar fallande.

SELECT * FROM bok ORDER BY bokTitel, bokForfattare;
Hämtar alla kolumner från tabellen bok och sorterar först på titel och sedan på författare.

UPDATE

UPDATE bok
SET bokTitel='Röda rummet, bokForfattare='August Strindberg'
WHERE bokId=2;
Uppdaterar raden där bokid är 2.

INSERT INTO

INSERT INTO bok (bokTitel, bokForfattare)
VALUES ('Jazz i Göteborg',Rune Johansson');
Lägger till en ny rad i tabellen bok.

DELETE

DELETE FROM bok
WHERE bokId=2;
Raderar raden där id är 2 i tabellen bok.

BETWEEN

SELECT * FROM bok
WHERE bokPris BETWEEN 50 AND 100;
Hämtar böcker där bokens pris är mellan 50 och 100.

SELECT * FROM bok
WHERE bokPris NOT BETWEEN 10 AND 40;
Hämtar böcker där bokens pris inte är mellan 10 och 40.

NULL

SELECT * FROM bok
WHERE bokTitel IS NULL;
Hämtar bara böcker som har NULL-värden för titel.

SELECT * FROM bok
WHERE bokTitel IS NOT NULL ;
Hämtar bara böcker som inte har NULL-värden för titel.

NULL fungerar inte tillsammans med operatorer som t ex =, <, >, <>

CREATE TABLE

CREATE TABLE bok
(
bokId INT NOT NULL,
bokTitel VARCHAR(255),
bokForfattare VARCHARr(255),
PRIMARY KEY (bokId)
);
Skapar tabellen bok.

ALTER

ALTER TABLE bok
ADD bokIsbn varchar(50);
Lägger till kolumnen bokIsbn i tabellen bok.

DROP

DROP TABLE bok;
Raderar tabellen bok.

Övningsuppgifter:

Du kan göra uppgifterna i både MySQL och SQLite.
Skapa en ny databas och spara den med namnet ”uppgifter”.

1. Skapa en tabell som du kallar för ”dator”.

Följande kolumner skall finnas med, välj passande namn. Tänk ut vilka datatyper som passar bäst, varchar, int etc.

Artikelnummer

Fabrikat

Processortyp

HastighetGhz

RAM

Hårddiskstorlek

Skriv SQL för att skapa tabellen. Mata in minst 3 poster i din tabell.

2. Skriv en SQL-fråga som hämtar alla datorer från tabellen ”datorer” som är av processortypen ”Intel”.

3. Skriv en SQL-fråga som visar datorer som har större RAM än 4 (GB).

4. Skapa en tabell för ”skivor”.

Tabellen skall kunna användas av t.ex. en butik för CD, LP-skivor, så att all nödvändig information om varje skiva finns med.

Skapa själv kolumner för tabellen, och välj rätt datatyp.

Man skall kunna hitta information om vad skivan heter, vad den kostar, när den är utgiven etc.

Minst 5 kolumner ska du skapa.

Skriv SQL för att skapa tabellen. Mata in minst 3 poster i din tabell.

5. Skriv en SQL-fråga som visar alla skivor för en viss artist från tabellen ”skivor”.

6. Skriv en SQL-fråga som visar alla skivor för en viss artist som har givits ut senare än 2008 från tabellen ”skivor”.

7. Skriv en SQL-fråga som lägger till en ny skiva i tabellen ”skivor”.

8. Skapa en tabell för ”anställda”.

Tabellen skall kunna användas av t.ex. Ett företag med många anställda, så att all nödvändig information om varje person finns med.

Skapa själv kolumner för tabellen, och välj rätt datatyp.

Man skall kunna hitta information om vad den anställde heter, vad han har för lön, vad han har för befattning etc.

Minst 5 kolumner ska du skapa.

Skriv SQL för att skapa tabellen. Mata in minst 2 poster i din tabell.

9. Skriv en SQL-fråga som hämtar alla anställda från tabellen ”anställda”, sorterat på senast inlagd.

10. Skriv en SQL-fråga som lägger till en ny anställd i tabellen ”anställda”.

11. Skriv en SQL-fråga som tar bort alla anställda från tabellen ”anställda” som bor i Göteborg.

12. Skriv en SQL-fråga som uppdaterar någon av kolumnerna för alla anställda som är ”chefer”.

13. Lägg till en ny kolumn med SQL i tabellen ”anställda”.

Fördjupning:
Skriv egna SQL-frågor kopplat till tabellerna vi har skapat. Försök att använda samtliga delar vi har gått genom: SELECT, UPDATE, INSERT INTO, DELETE, ALTER.

Databasdesign och normaliserade databaser

Metoder för att rita konceptuella datamodeller

Läs genom:

http://www.databasteknik.se/webbkursen/er/index.html

http://www.databasteknik.se/webbkursen/er2relationer/index.html

Kopplingar och relationer:

- *Ett-till-ett-samband* eller *1:1-samband*. En sambandstyp där en sak av något slag kan höra ihop med en sak av ett annat slag, och varje sak av det andra slaget kan höra ihop med en sak av det första slaget. Exempel: En person kan vid ett och samma tillfälle bara köra en bil, och varje bil kan bara köras av en person.

- *Ett-till-många-samband* eller *1:N-samband*. En sambandstyp där en sak av något slag kan höra ihop med flera saker av ett annat slag, men varje sak av det andra slaget kan bara höra ihop med en sak av det första slaget. Exempel: En person kan äga flera bilar, men varje bil kan bara ägas av en person. Om man vänder på det och börjar med bilarna blir det i stället ett *många-till-ett-samband* (*N:1-samband*).

- *Många-till-många-samband* eller *N:M-samband*. En sambandstyp där en sak av något slag kan höra ihop med flera saker av ett annat slag, och varje sak av det andra slaget kan höra ihop med flera saker av det första slaget. Exempel: En person kan äga flera hus, och varje hus kan ägas gemensamt av flera personer.

Centrala begrepp

ER-modellen, **Entity-Relationship-modellen**, **entitets-sambands-modellen** (engelska: **the Entity-Relationship model**). En konceptuell datamodell där man beskriver verkligheten genom att ange de typer av entiteter (engelska: **entities**) (ungefär "saker") som finns i den verkligheten, och vilka typer av samband (engelska: **relationships**) som finns mellan dessa. Beskrivningen ritas oftast upp som ett ER-diagram.

Entitetstyp (engelska: **entity type**). I ER-modellen beskriver man världen med hjälp av två grundläggande byggblock: typer av saker, eller entiteter, och typer av samband mellan dessa saker.

Sambandstyp (engelska: **relationship type**). I ER-modellen beskriver man världen med hjälp av två grundläggande byggblock: typer av saker, eller entiteter, och typer av samband mellan dessa saker. (Undvik att säga "relation" eller "relationstyp", för en relation är en tabell i relationsmodellen.)

Kardinalitetsförhållande (engelska: **cardinality ratio**). Hur många saker som kan delta i en och samma instans av en sambandstyp. Används främst i ER-modellen. Se ett-till-ett-samband, ett-till-många-samband och många-till-många-samband.

Fullständigt deltagande (engelska: **total participation**). Kravet att varje förekomst av en entitetstyp måste delta i en viss sambandstyp. Används i ER-modellen.

Attribut (engelska: **attribute**). Betyder "egenskap", och används både om kolumnerna i tabellerna i relationsmodellen, och om egenskaperna hos entiteterna i ER-modellen.

Schema, **databasschema** (engelska: **schema**, **database schema**). En beskrivning av vilka data som kan finnas i en databas, oberoende av vilka data (innehållet) som råkar finnas i databasen just nu. Exempel: I relationsmodellen, där man beskriver världen med hjälp av tabeller, består schemat huvudsakligen av vilka tabeller som finns i databasen, och vilka kolumner de har, men inte vilka värden som råkar finnas i tabellerna just nu. I stället för "schema" säger man ibland intension, åtminstone när man talar om schemat för en relation i relationsmodellen.

Normalformer

Läs genom:

http://www.databasteknik.se/webbkursen/normalisering/index.html

Centrala begrepp

Normalform (engelska: **normal form**).
En regel som förbjuder vissa typer av dum design i en databas. Det finns flera olika normalformer, till exempel BCNF.

Normalisering (engelska: **normalisation**).
En teori för (främst) relationsdatabaser som kan användas för att undvika vissa typer av dum design i en databas, bland annat när det gäller onödig dubbellagring av information.

Redundans (engelska: "redundancy"). Upprepning av data eller funktionalitet. Kan vara nyttig eller skadlig, avsiktlig eller oavsiktlig.

Scheman och nycklar

När man kommer till steget då man skapar tabeller utifrån schemat så behöver man olika nycklar för att kunna skapa önskade relationer.

Det har vi tidigare gått genom i ”Relationsmodellen”.

Repetition av centrala begrepp:

Primärnyckel

Främmandenyckel, referensattribut

Referensintegritet

Förklara kortfattat följande begrepp:

1. Entitetstyp

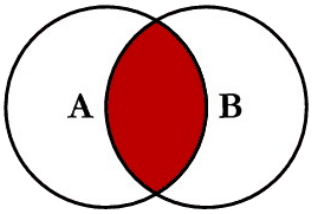
2. Sambandstyp

3. Attribut

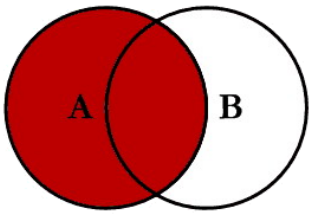
4. Normalisering

5. Redundans

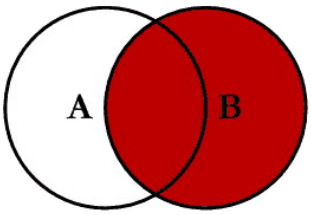
SQL JOINS



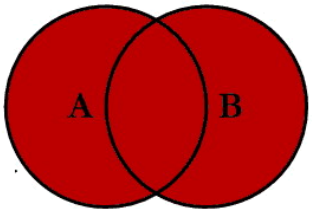
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key



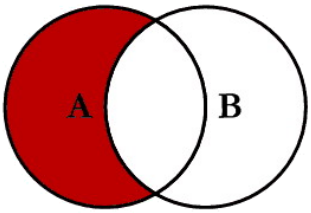
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key



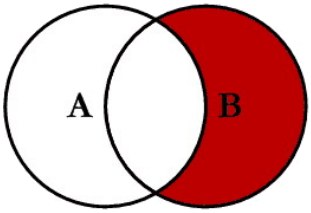
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key



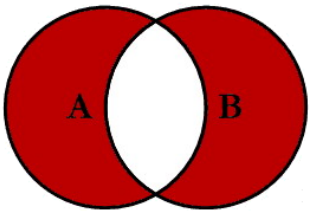
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key



SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL



SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL



SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL

SQL, fördjupning (kopplingar och relationer)

JOIN

Den vanligaste typen av JOIN i SQL är INNER JOIN.

INNER JOIN returnerar de rader som har träffar på båda sidorna i relationen.

LEFT JOIN returnerar alla rader i den vänstra tabellen och de rader i den högra tabellen som matchar.

RIGHT JOIN returnerar alla rader i den högra tabellen och de rader i den vänstra tabellen som matchar.

FULL JOIN returnerar alla rader när det är en träff i någon av tabellerna.

SQL JOIN - koppling mellan två tabeller (ett-till-många)

MySQL: SQL för att skapa tabellerna bok och kategori

BokId är primärnyckel i boktabellen och kategoriId är primärnyckel i kategoritabellen. bokKategoriId är främmandenyckel i kategoritabellen som skapar relationen mellan bok och kategori.

```
CREATE TABLE kategori(
        kategoriId INT NOT NULL AUTO_INCREMENT,
        kategoriNamn VARCHAR(50),
        PRIMARY KEY (kategoriId)
);
CREATE TABLE bok(
        bokId INT NOT NULL AUTO_INCREMENT,
        bokTitel VARCHAR(50),
        bokForfattare VARCHAR(50),
        bokBeskrivn VARCHAR(50),
        bokIsbn VARCHAR(50),
        bokPris INT,
        bokKategoriId INT,
        PRIMARY KEY (bokId),
        FOREIGN KEY (bokKategoriId) REFERENCES kategori(kategoriId)
);
```

När man skapar två tabeller som här med bok och kategori behöver man tänka på *Referensintegritet*. Vilket innebär att det alltid måste finnas ett kategoriId i kategoritabellen

om man ska kunna lägga till en bok och koppla det till den kolumnen. Detta skapar vi i SQL genom REFERENCES.

SQLite: SQL för att skapa tabellerna bok och kategori.

```
CREATE TABLE kategori(
        kategoriId INTEGER PRIMARY KEY,
        kategoriNamn TEXT
);
CREATE TABLE bok(
        bokId INTEGER PRIMARY KEY,
        bokTitel TEXT,
        bokForfattare TEXT,
        bokBeskrivn TEXT,
        bokIsbn TEXT,
        bokPris REAL,
        bokKategoriId INT
);
```

(Lite exempeldata för bok och kategori, till SQL-exemplen)

bok				
bokId	**bokForfattare**	**bokTitel**	**bokPris**	**bokKategoriId**
2	Strindberg, August	Röda rummet	100	1
5	Carroll, Lewis	Alice i underlandet	150	1
9	Lindgren, Astrid	Vi på saltkråkan	80	2
10	Tolkien, J.R.R	Sagan om ringen	30	1

kategori	
kategoriId	**kategoriNamn**
1	Skönlitteratur
2	Barnböcker

```
SELECT bok.bokTitel, kategori.kategoriNamn
FROM kategori
INNER JOIN bok
ON kategori.kategoriId = bok.bokKategoriId;
```
Returnerar böcker och kategorier där det matchar.

Övning:

Skapa en ny databas med tabellerna bok och kategori. Fyll tabellerna med lite relevant data. Skriv SQL-frågor som visar t ex:
- Böcker tillsammans med kategori
- Böcker för en viss kategori
- Samtliga kategorier (även dem som inte har en koppling till någon bok) tillsammans med de böcker som har koppling till kategori
- Samtliga böcker (även dem som inte har en koppling till någon kategori) tillsammans med de kategorier som har koppling till bok
- Radera någon rad i bok-tabellen
- Radera någon rad i kategori-tabellen

SQL JOIN - koppling mellan tre tabeller (många-till-många)

MySQL: SQL för att skapa tabellerna bok, kategori och bokKategori

BokId är primärnyckel i boktabellen och kategoriId är primärnyckel i kategoritabellen. Tabellerna bok och kategori har främmandenycklar i ”mellantabellen” bokKategori som gör att man kan ha många-till-många förhållande mellan böcker och kategorier.

```
CREATE TABLE kategori(
        kategoriId INT NOT NULL AUTO_INCREMENT,
        kategoriNamn VARCHAR(50),
        PRIMARY KEY (kategoriId)
);
CREATE TABLE bok(
        bokId INT NOT NULL AUTO_INCREMENT,
        bokTitel VARCHAR(50),
        bokForfattare VARCHAR(50),
        bokBeskrivn VARCHAR(50),
        bokIsbn VARCHAR(50),
        bokPris INT,
        PRIMARY KEY (bokId)
);
CREATE TABLE bokKategori(
        bokKategoriId INT NOT NULL AUTO_INCREMENT,
        bokKategoriBid INT,
        bokKategoriKid INT,
```

```
        PRIMARY KEY (bokKategoriId),
        FOREIGN KEY (bokKategoriBid) REFERENCES bok(bokId),
        FOREIGN KEY (bokKategoriKid) REFERENCES kategori(kategoriId)
);
```

Tänk på att skapa tabellerna bok och kategori innan mellantabellen bokKategori då primärnycklarna måste skapas innan främmandenycklarna.

SQLite: SQL för att skapa tabellerna bok, kategori och bokKategori.

```
CREATE TABLE kategori(
        kategoriId INTEGER PRIMARY KEY,
        kategoriNamn TEXT
);
CREATE TABLE bok(
        bokId INTEGER PRIMARY KEY,
        bokTitel TEXT,
        bokForfattare TEXT,
        bokBeskrivn TEXT,
        bokIsbn TEXT,
        bokPris REAL,
        bokKategoriId INT
);
CREATE TABLE bokKategori(
        bokKategoriId INTEGER PRIMARY KEY,
        bokKategoriBid INT,
        bokKategoriKid INT
);
```

(Lite exempeldata för bok, kategori och bokKategori)

bok			
bokId	**bokForfattare**	**bokTitel**	**bokPris**
2	Strindberg, August	Röda rummet	100
5	Carroll, Lewis	Alice i underlandet	150
9	Lindgren, Astrid	Vi på saltkråkan	80

kategori	
kategoriId	**kategoriNamn**
1	Skönlitteratur
2	Barnböcker

bokKategori		
bokKategoriId	**bokKategoriBid**	**bokKategoriKid**
1	5	1
2	5	2
3	2	1

```
SELECT bok.bokTitel, kategori.kategoriNamn
FROM kategori INNER JOIN bokKategori
ON kategori.kategoriId = bokKategori.bokKategoriKid
INNER JOIN bok
ON bokKategori.bokKategoriBid = bok.bokId;
```

Returnerar böcker och kategorier där det matchar.

Övning:

Skapa en ny databas med tabellerna bok, kategori och bokKategori. Fyll tabellerna med lite relevant data. Skriv SQL-frågor som visar t ex:

- Böcker tillsammans med kategori
- Böcker för en viss kategori
- Samtliga kategorier (även dem som inte har en koppling till någon bok) tillsammans med de böcker som har koppling till kategori
- Samtliga böcker (även dem som inte har en koppling till någon kategori) tillsammans med de kategorier som har koppling till bok
- Radera någon rad i bok-tabellen
- Radera någon rad i kategori-tabellen

(Håll koll på hur kopplingarna är i mellantabellen och i vilken ordning man behöver radera i tabellerna för referensintegritet)

SQL, fördjupning (funktioner och vyer)

SQL Functions (AVG, COUNT, MAX etc):

http://www.w3schools.com/sql/sql_functions.asp

Vyer

Vyer är ofta för att förenkla krångliga frågor. Med vyer kan man dela upp frågan i flera steg. En vy innehåller rader och kolumner som en vanlig tabell. Man kan använda WHERE och JOINS för att presentera data i en vy så att det ser ut som det kommer från en tabell.

http://www.w3schools.com/sql/sql_view.asp

AVG()

SELECT AVG(bokPris) AS bokMedelPris FROM bok;
Hämtar medelvärdet för böckernas pris.

SELECT bokTitel, bokPris FROM bok
WHERE bokPris>(SELECT AVG(bokPris) FROM bok);
Hämtar de böcker som kostar mer än medelvärdet för böckernas pris.

COUNT()

SELECT COUNT(bokId) AS bokAntal FROM bok;
Räknar antal böcker i tabellen.

MAX()

SELECT MAX(bokPris) AS hogstaPris FROM bok;
Visar priset för den bok som kostar mest.

MIN()

SELECT MIN(bokPris) AS minstaPris FROM bok;
Visar priset för den bok som kostar minst.

SUM()

SELECT SUM(bokPris) AS totalPris FROM bok;
Visar den totala summan för alla böcker.

GROUP BY och HAVING

När man vill använda aggreagatfunktioner som t ex AVG() finns det tillfällen då man vill gruppera svaret. Som när man t ex vill visa anställdas genomsnittslön kopplat till avdelningar där de arbetar.

```
SELECT anstalldaAvdelning, AVG(anstalldaLon)
FROM anstallda
GROUP BY anstalldaAvdelning;
```

Anstallda	
AnstalldaAvdelning	**AVG(anstalldaLon)**
1	18000
3	19000

Man kan också använda WHERE-villkor. Det ska isåfall före GROUP BY.

```
SELECT anstalldaAvdelning, AVG(anstalldaLon)
FROM anstallda
WHERE anstalldaNamn = 'Kalle' OR anstalldaNamn = 'Lisa'
GROUP BY anstalldaAvdelning;
```

Genom att använda HAVING kan man bara ta med vissa av grupperna i svaret.

```
SELECT anstalldaAvdelning, AVG(anstalldaLon)
FROM anstallda
GROUP BY anstalldaAvdelning
HAVING avg(anstalldaLon) > 18000;
```

Man kan också kombinera WHERE, GROUP BY och HAVING i samma fråga.

```
SELECT anstalldaAvdelning, AVG(anstalldaLon)
FROM anstallda
WHERE anstalldaNamn = 'Kalle' OR anstalldaNamn = 'Lisa'
GROUP BY anstalldaAvdelning
HAVING avg(anstalldaLon) > 18000;
```

Övningar (Funktioner och vyer):

Skriv SQL-frågorna utifrån tabellerna nedan. Skapa också vyer för några av SQL-frågorna t ex i samband med statistik.

Medlem					
medlemId	**medlemNamn**	**medlemAdress**	**medlemFödd**	**medlemAntalmål**	**medlemMatchId**
1	Kalle Karlsson	Vägen 1 421 34 Göteborg	2001	4	1
3	Sara Persson	Vägen 3 421 36 Göteborg	2004	2	2
4	Pelle Persson	Vägen 8 423 45 Göteborg	2001	1	1

fotbollsMatch			
fotbollsMatchId	**fotbollsMatchMotstandarlag**	**fotbollsMatchResultat**	**fotbollsMatchPlan**
1	Höjdarna BK	4-1	Ruddalen
2	Solen BK	3-2	Ullevi

1. Visa alla medlemmar som är födda 2001.

2. Visa alla medlemmar som har gjort mer än 2 mål.

3. Visa alla medlemmar som är födda senare än 2002 och gjort mer än ett mål.

4. Visa alla medlemmar kopplat till matcher.

5. Visa alla medlemmar som har varit med på matchen mot Höjdarna BK. Man ska kunna visa namn, matchresultat och motståndarlag.

6. Visa medelvärdet för antal mål för medlemmarna.

7. Visa alla medlemmar som är födda mellan 1999 och 2002.

8. Visa medelvärdet för antal mål grupperat på matcher.

Index och prestanda

I en relationsdatabas kan man skriva sökningar med SELECT-frågor och så länge man har mindre mängder data fungerar det bra. Men om mängden data ökar eller om man har komplicerade frågor kommer sökningarna att ta lång tid.

Man brukar då prata om att databasen har dåliga prestanda och att man behöver förbättra prestandan. Det gör man först och främst genom att skapa index till tabellerna.
Ett index kan man se som ett register i en bok. Om man t ex söker efter gitarr i ett musiklexikon så är det enklare att leta i registret först och få reda på numret för sidan där ordet finns istället för att bläddra genom boken.
Anledningen är att registret bara är några sidor och sorterat i bokstavsordning till skillnad från lexikonet som har flera hundra sidor.

Ett index i en relationsdatabas fungerar som en extra tabell med en pekare in i huvudtabellen. Detta istället för sidhänvisningar i registret i en bok.

bok		
bok_id	**bok_forfattare**	**Bok_titel**
2	Strindberg, August	Röda rummet
5	Carroll, Lewis	Alice i underlandet
9	Lindgren, Astrid	Vi på saltkråkan
10	Tolkien, J.R.R	Sagan om ringen

Här är en tabell med böcker. Databashanteraren har sorterat på bokid och det är praktiskt när vi behöver veta ett id för att t ex skapa en relation till en beställning av boken. Men när vi ska söka efter en viss bok då behöver vi läsa genom tabellen rad för rad.
En sökning efter ”Röda rummet” skulle se ut så här.

```
SELECT bok_forfattare, bok_titel FROM bok WHERE bok_titel = ´Röda rummet’
```

Är det bara några rader i tabellen som i detta exempel går det bra men är det en stor databas så kommer sökningen att ta för lång tid.

Vi skapar då ett index för titel. Det kan man göra med kommandot **create index.**

```
CREATE INDEX titelindex ON bok(bok_titel)
```

Titelindex ovan är bara ett namn och indexet kan döpas till vad som helst.
Nu bygger databashanteraren en extra tabell som används internt för att söka efter titeln för en bok. Den nya tabellen med indexet är sorterat i bokstavsordning och det gör att det går snabbare att hitta ”Röda rummet” och därefter referensen till den vanliga tabellen.

Så här kommer indextabellen att fungera som skapas för bokens titel. Bok_id är referensen till huvudtabellen. Indexet är sorterat i bokstavsordning efter titel.

titelindex	bok_id
Alice i underlandet	5
Röda rummet	2
Sagan om ringen	10
Vi på saltkråkan	9

Varför kan man då inte skapa index för samtliga kolumner i en tabell så att man inte behöver tänka på att skapa index i efterhand. Nackdelen med index är att indextabellerna blir stora när man hanterar en stor mängd data. Indexet måste också uppdateras när den indexerade tabellen har förändrats. Det tar också tid att uppdatera ett index så tabellens innehåll blir långsammare ju fler index man har.

Det här bör man tänka på när man ska skapa index för en tabell:

- Vilka sökningar kommer man att göra?
- I en tabell med böcker kommer man att söka på författare och titel men behöver man söka på beskrivning för boken eller pris.
- Man skapar bara index för de kolumner man behöver göra sökningar på, inte det som ska visas i en sökning som t ex pris och beskrivning.

Här är ett exempel på att skapa två index för titel och författare. Sedan ett exempel på en sökning som visar författare, titel, pris och beskrivning.

```
CREATE INDEX titelindex ON bok(bok_titel)
CREATE INDEX forfattareindex ON bok(bok_forfattare)

SELECT bok_forfattare, bok_titel, bok_pris, bok_beskrivning FROM bok
WHERE bok_titel = ´Röda rummet’
```

Man brukar dela in index i olika klasser. Här är två av de vanligaste.

Klustrade index (clustering index), är sorterat i samma ordning som huvudtabellen. Ett primärindex blir automatiskt ett klustrat index. Den här typen av index är praktiskt när man ska söka på t ex ett kundnummer men inte när man ska söka efter ett speciellt namn eller titel på en bok som i exemplet ovan.

Sekundärindex (secondary index), eller *oklustrade index (unclustred index)* som det också kallas. Ett sekundärindex är sorterat i en annan ordning än huvudtabellen. Som i exemplet ovan när vi gjorde ett index för boktiteln. Det skapas då en indextabell med en referens till huvudtabellen. Den här typen av index är inte lika snabb som ett klustrat index men eftersom det sorteras i bokstavsordning i indextabellen går det ändå snabbt att söka genom.

Frågor:

1. Om man ska söka på kundnummer och man har skapat en primärnyckel för den kolumnen har man behov av att skapa ett nytt index för att få en snabb sökning? Förklara också hur du kom fram till det.

2. Ge något exempel på vad man ska tänka på när man skapar ett index och i vilket sammanhang indexet är skapat.

3. Varför är det inte bra att skapa index för samtliga kolumner i en databas?

4. Förklara vad ett klustrat index är.

5. Förklara vad ett sekundärindex (oklustrade index) är.

Övning:

- Du ska skapa en tabell för hyresgäster som en hyresvärd använder för att kunna söka efter hyresgäster efter t ex lägenhetsnummer, namn, adress.
- Tänk ut vilka index som behövs utifrån vilka sökningar du tror man kommer att behöva göra.
- Skriv också några SQL-frågor för sökningar i tabellen och hur indexen har skapats med SQL.

Huvudtabell

Indextabeller (rita hur indextabellerna fungerar)

Skriv SQL-frågor för att skapa index och några sökningar

Databas – övningar

Välj ut och arbeta med ett tema. Hinner du kan du göra mer än en databas.

Bokutlåning

Ett datakonsultföretag vill hålla reda på vilka böcker de anställda lånar från företagets interna referensbibliotek. Företaget vill kunna hålla reda på vilka böcker som för tillfället är utlånade och vid senare tillfälle föra statistik över de mest utlånade böckerna, mest frekvente boklånare m.m. Det är inte relevant med ISBN-nummer utan företaget märker upp och numrerar böckerna med ett löpnummer.

Det kan bl.a. finnas med uppgifter om:

- De anställda.
- På företaget finns det fem avdelningar: administration, ekonomi, sälj, utveckling och marknadsföring.
- Alla Böcker med titel, inköpsdatum, pris och ett boknummer.
- Böckerna ska kunna delas in i olika ämnesområdena, Databas, Webb, Programmering, Ordbehandling o.s.v.
- Det ska finnas uppgift om vilket datum en bok lånas ut och lämnas tillbaka.

Skapa SQL-frågor:

Det ska gå att plocka fram listor på vilka böcker som finns för utlåning, vilka som är utlånade och vilka som finns inne för tillfället.

Det ska gå att skriva ut listor på vilka böcker som finns i ett visst ämne och vilka böcker som är mest utlånade.

Den här ordningsföljden är en hjälp för att komma igång med arbetet:

1. Planera databasen med ER-modellen.
2. Tänk genom att du har normaliserat databasen.
3. Ta fram de logiska tabellerna.
4. Kom fram till vilka kolumner (attribut) som kommer att behövas.
5. Definiera relationer mellan tabeller.
6. Skapa eventuella nycklar för koppling/relationer mellan tabeller.
7. Skapa databasen.
8. Skapa SQL-frågor för att skapa tabellerna.
9. Skapa relevanta frågor, både för att lista, sortera, lägga till, uppdatera och ta bort data.
10. Tänk ut vilka index som ev. behövs.
11. Skapa vyer där det underlättar och ev. Stored Procedures och/eller Triggers.

Fiskeförening

En fiskeförening ska registrera medlemmar och deras medlemsavgifter. Fiskeklubben har återkommande klubbtävlingar där det kan vara behövligt att skriva ut tävlings- och resultatlistor (medlemmarna behöver inte vara med i alla tävlingar). Det ska gå att föra statistik över hur det har gått för en medlem i de olika tävlingarna under året.

Det ska bl.a. finnas med uppgifter om:

- Medlemmarna och om de betalt sin medlemsavgift.

- Årets olika tävlingar.
- Tävlingsresultat som mäts i antal kilo fisk.

Skapa SQL-frågor:

En lista över medlemmar som inte har betalt sin avgift.

En lista över start och resultatlistor till tävlingarna.

En lista över stormästartiteln som blir den som lyckats fånga mest fisk under årets samtliga tävlingar.

Den här ordningsföljden är en hjälp för att komma igång med arbetet:

1. Planera databasen med ER-modellen.
2. Tänk genom att du har normaliserat databasen.
3. Ta fram de logiska tabellerna.
4. Kom fram till vilka kolumner (attribut) som kommer att behövas.
5. Definiera relationer mellan tabeller.
6. Skapa eventuella nycklar för koppling/relationer mellan tabeller.
7. Skapa databasen.
8. Skapa SQL-frågor för att skapa tabellerna.
9. Skapa relevanta frågor, både för att lista, sortera, lägga till, uppdatera och ta bort data.
10. Tänk ut vilka index som ev. behövs.
11. Skapa vyer där det underlättar och ev. Stored Procedures och/eller Triggers.

Resebolag

Ett resebolag som arrangerar resor till olika resmål, har fått problem med att hålla översikt över vilka inkvarteringsplatser som gäller för de olika resmålen. Det finns nämligen många olika inkvarteringsplatser på de olika platserna. Det är också viktigt att ha ständig dialog med kontaktpersoner på resmålen, det kan finnas fler kontakter per resmål.

Skapa SQL-frågor:

Listning av alla inkvarteringsplatser på ett resmål.
Listning av alla resmål i ett givet land.

Den här ordningsföljden är en hjälp för att komma igång med arbetet:

1. Planera databasen med ER-modellen.
2. Tänk genom att du har normaliserat databasen.
3. Ta fram de logiska tabellerna.
4. Kom fram till vilka kolumner (attribut) som kommer att behövas.
5. Definiera relationer mellan tabeller.
6. Skapa eventuella nycklar för koppling/relationer mellan tabeller.
7. Skapa databasen.
8. Skapa SQL-frågor för att skapa tabellerna.
9. Skapa relevanta frågor, både för att lista, sortera, lägga till, uppdatera och ta bort data.
10. Tänk ut vilka index som ev. behövs.

11. Skapa vyer där det underlättar och ev. Stored Procedures och/eller Triggers.

Skivsamling

Du vill skapa en databas som ska hålla ordning på din skivsamling. Du vill i första hand registrera uppgifter om skivor och artister.

- Uppgifter om skivornas titel, utgivningsår, skivbolag och artist. För artisterna kan du registrera namn, nationalitet o.s.v.
- Artisterna kan tillhör olika kategorier pop, rock, jazz o.s.v.

Skapa SQL-frågor:

Databasen ska klara av att söka fram artister och registrera ny skivor och det ska gå att skriva ut listor över en artists alla skivor.

Den här ordningsföljden är en hjälp för att komma igång med arbetet:

1. Planera databasen med ER-modellen.
2. Tänk genom att du har normaliserat databasen.
3. Ta fram de logiska tabellerna.
4. Kom fram till vilka kolumner (attribut) som kommer att behövas.
5. Definiera relationer mellan tabeller.
6. Skapa eventuella nycklar för koppling/relationer mellan tabeller.
7. Skapa databasen.
8. Skapa SQL-frågor för att skapa tabellerna.
9. Skapa relevanta frågor, både för att lista, sortera, lägga till, uppdatera och ta bort data.
10. Tänk ut vilka index som ev. behövs.
11. Skapa vyer där det underlättar och ev. Stored Procedures och/eller Triggers.

Begrepp och ämnesrelaterad terminologi

Ett databassystems funktioner och uppbyggnad

Data

Databas (engelska: **database**).

Databashanterare, databashanteringssystem, DBHS (engelska: **database management system, DBMS**).

Schema, **databasschema** (engelska: **schema**, **database schema**).

Datamodell (engelska: **data model**).

SQL. Av **Structured Query Language**.

Fråga (engelska: **query**).

Frågespråk (engelska: **query language**).

DDL, **datadefinitionsspråk**

DML, **datamanipuleringsspråk**

Logiskt dataoberoende (engelska: **logical data independence**).

Fysiskt dataoberoende (engelska: **physical data independence**).

Tre-schema-arkitekturen (engelska: **the three-schema architecture**).

Gränssnitt (engelska: **interface**).

Vy (engelska: **view**).

Konceptuell datamodell eller **begreppsmässig datamodell** (engelska: **conceptual data model**).

Implementationsmodell eller **implementeringsmodell** (engelska: **implementation data model**).

Relationsmodellen (engelska: **the relational model**).

DBA, **databasadministratör** (engelska: **database administrator**).

Applikationsprogram (engelska: **application program**) eller **tillämpningsprogram**.

Relationsmodellen

Relationsmodellen (engelska: **the relational model**).

Relationsdatabas (engelska: **relational database**).

Relation

Tabell (engelska: **table**).

Tupel (engelska: **tuple**).

Rad (engelska: **row**).

Attribut (engelska: **attribute**).

Kolumn (engelska: **column**).

Primärnyckel (engelska: **primary key**).

Referensattribut (engelska: **reference attribute**).

Främmande nyckel (engelska: **foreign key**).

Referensintegritet (engelska: **referential integrity**).

Normalformer

Normalform (engelska: **normal form**).

Normalisering (engelska: **normalisation**).

Redundans (engelska: "redundancy").

Index och prestanda

Index

Klustrade index

Sekundärindex (oklustrade index)

Primärnyckel

Att tänka på vid skapande av index