LÄR DIG DATABASER OCH SQL FRÅN GRUNDEN

Adrian Krsmanovic
Robert Westman
William Blennow
Linus Rundberg Streuli
Antonio Prgomet

Första upplagan

Innehållsförteckning

Förord

I detta förord till boken kommer vi presentera bokens *GitHub*-sida, syfte och målgrupp, programvara som används, hur boken är utformad, några kommentarer kopplat till språket samt bokens disposition.

Bokens GitHub

Boken har en tillhörande *GitHub*-sida där material kopplat till boken finns uppladdat. Se följande länk:
https://github.com/AntonioPrgomet/laer_dig_databaser_och_-sql_1uppl

Bokens syfte och målgrupp

Bokens syfte är att lära ut grunderna i databaser och "Structured Query Language" (SQL). Därför kan den användas för självstudier och i kurser av olika slag inom exempelvis yrkeshögskolan, företag eller eller andra organisationer. Boken lär ut databaser och SQL från grunden innebärande att inga förkunskaper inom dessa områden förutsätts. Även de som har vissa förkunskaper kan med fördel läsa boken för att få en gedigen grund.

Programvara som används i boken

Det finns flertalet olika databashanterare där SQL används. Två av de vanligast förekommande är “MySQL” och “Microsoft SQL Server” (MSSQL). I praktiken är skillnaderna små och den läsare som behärskar det ena kan med enkelhet lära sig det andra på kort tid. I boken visar vi kod för både MySQL och MSSQL direkt efter varandra i de fall det finns skillnader. I de fall koden är densamma i båda system så visas endast ett kodexempel som gäller för båda databashanterarna.

Det finns olika grafiska användargränssnitt för MySQL, vi har använt “MySQL Workbench” som är den mest populära. Det kan laddas ned gratis här: https://dev.mysql.com/downloads/workbench/

MSSQL är också gratis att ladda ned och det kan göras här: https://www.microsoft.com/en-us/sql-server/sql-server-downloads

I Kapitel 6 använder vi programmeringsspråket Python för att demonstrera bruket av NoSQL-databaser. I Avsnitt 8.2 använder vi Python för att utveckla en applikation som använder SQL för att lagra och redovisa data. För dessa delar av boken förutsätts det att läsaren behärskar grundläggande Python. Den läsare som inte har erfarenhet av Python kan lära sig det genom att se en video som finns på bokens *GitHub*-sida. Den läsare som vill fördjupa sig ytterligare kan till exempel läsa boken ”Lär dig Python från grunden” som är författad av Linus Rundberg Streuli och Antonio Prgomet.

Bokens utformning

I boken använder vi två sorters informationsrutor som ser ut enligt följande.

> **i** Såhär ser en allmän informationsruta ut. I de här rutorna skriver vi korta fördjupande förklaringar till vissa begrepp och koncept.

> **!** Det här är en viktigt-ruta. De här rutorna använder vi för att sätta fokus på viktiga distinktioner eller detaljer som kan vara svåra att upptäcka men som kan ha stor betydelse.

Ofta är bokens kodexempel annoterade. I högra kanten förekommer siffror som motsvarar en förklarande text under kodexemplet. Efter den förklarande texten följer oftast resultatet av själva koden. Nedan visas ett kodexempel där vi skriver ut två kolumner från en tabell som heter ”Customers”.

```
SELECT CustomerID,          ①
       FirstName            ②
FROM Customers;             ③
```

① Kolumnen “CustomerID” väljs.
② Kolumnen “FirstName” väljs.
③ Kolumnerna väljs från tabellen som heter “Customers”.

CustomerID	FirstName
1	Greta
2	Zara
3	Dragan
4	Lars
5	Greta
6	Karin
7	Sven
8	Johan
9	Hussein
10	Ruth

Varje kapitel avslutas med övningsuppgifter vars syfte är att bidra till att läsaren befäster de kunskaper som gås igenom i kapitlet. Därför är uppgifterna av standard-karaktär snarare än komplex problemlösning.

I boken används tre typer av data när kod demonstreras.

- Enkel exempeldata. Denna typ av data är väldigt enkel eftersom fokus ska vara på att se vad koden gör och det underlättas när datan är enkel. Se exempelvis Avsnitt 3.5.1 där vi skapar tabellerna ”Students” och ”Classes”.

- En pedagogiskt utvecklad databas som heter ”Köksglädje”. Det är en exempeldatabas för en fiktiv svensk butikskedja som säljer köksutrustning. Den kan nås på bokens *GitHub*-sida och introduceras i Avsnitt 3.4.

- En exempeldatabas för det fiktiva företaget ”Wide World Importers” som heter ”WideWorldImporters” och är utvecklad av Microsoft. Denna databas är mer komplex och syftet är att läsaren ska se hur det kan se ut i verkligheten. Denna databas används i Avsnitt 8.1. Även denna databas kan nås på bokens *GitHub*-sida.

Språk

Den här boken är skriven på svenska. Bokens kodexempel är däremot på engelska eftersom det är en starkt rådande praxis världen över att kod generellt sett skrivs på engelska. För vissa koncept använder vi de engelska begreppen i den löpande texten då de i praktiken används även när vi talar svenska. Ett sådant exempel är *query*. I de fall engelska begrepp används så är dessa kursiverade med undantag från bokens innehållsförteckning.

I Sverige används kommatecken som decimaltecken och punkt som tusentalavgränsare medan andra länder såsom USA gör tvärtom, det vill säga använder punkt som decimaltecken och komma som tusentalavgränsare. Ett-hundra-två-tusen-komma-ett hade i Sverige skrivits som 100.000,1 medan det i USA hade skrivits som 100,000.1. Vi har i denna bok av konsistensskäl valt att använda den amerikanska standarden eftersom punkt är vad som används som decimalavgränsare i databashanterare såsom MySQL och MSSQL.

Bokens disposition

Denna bok går igenom databaser och SQL från grunden. Det innebär att inga förkunskaper inom databaser eller SQL förutsätts. I Kapitel 1 ges en kontext och introduktion för databaser och SQL. I Kapitel 2 ”Databasteori” går vi igenom grundläggande teori om databaser. I Kapitel 3 ”Grundläggande SQL” gås grunderna för programmeringsspråket “Structured Query Language” (SQL) igenom, vilket används för att hantera data i relationella databaser. Kapitel 4 ”Mer om SQL” är en direkt fortsättning på Kapitel 3 där vi fördjupar oss i mer avancerade tillvägagångssätt inom SQL. Oavsett om vi står inför skapandet av en databas eller om det redan finns en befintlig så behöver den administreras. Det är ämnet för Kapitel 5, ”Databasadministration”. Namnet NoSQL refererar till ”non-SQL” eller ”non-relational” och refererar till den typ av databaser som inte är relationella. Som exempel kan vi tänka på en databas som kan lagra diverse dokument. En överblick över denna typ av databaser ges i Kapitel 6 ”NoSQL”. För många organisationer är data en mycket viktig och värdefull resurs och behöver därför förvaltas. Detta är ämnet för Kapitel 7, “Data Management”. Två större exempel där SQL används presenteras i Kapitel 8 som är bokens avslutande kapitel.

Kapitel 3

Grundläggande SQL

Detta kapitel introducerar programmeringsspråket *Structured Query Language* (SQL) som används för att interagera med relationella databaser. Detta innefattar grundläggande syntax och konventioner, *CRUD*-flödet, *queries* samt funktioner, vyer, lagrade procedurer och index. I Avsnitt 3.4 introducerar vi databasen "Köksglädje" som kommer användas på ett flertal ställen i boken.

> **i** I de flesta fall kommer de kodexempel vi demonstrerar vara likadana för Microsoft SQL Server (MSSQL) och MySQL. I de fall där det skiljer sig kommer vi att demonstrera hur koden ser ut i både MySQL och MSSQL.

3.1 Syntax

Programmeringsspråket SQL är uppdelat i ett antal språkliga element. Dessa är bland annat:

- *Query* är en samling instruktioner som ges till en databas. En *query* består av ett eller flera *statements*.

- *Statement* är den instruktion som ges till databasen. Exempelvis `SELECT`, `DELETE`, `UPDATE`, `CREATE`, `DROP` och `ALTER`.
  Flera *statements* i en *query* separeras med semikolon ";".

- *Clause* är en beståndsdel av ett *statement*. Det kan exempelvis filtrera eller begränsa resultatet. Nedan listas några *clauses*.

`SELECT` används för att specificera vilka kolumner som ska läsas ut.

`FROM` används för att ange vilken tabell kolumner finns i.

`WHERE` används för att styra vilka rader som är en del av resultatet från en *query*, exempelvis vilka rader som ska läsas ut eller vilka rader som ska uppdateras.

`ORDER BY` används för att sortera ett resultat.

- *Expressions* är en kombination av operatorer och symboler som returnerar ett värde eller *result set*. Ett *result set* är den data som returneras av en SQL-*query*, exempelvis en tabell från en `SELECT`-*query*. Se Tabell 3.1 för olika operatorer.

- *Predicate* är ett *expression* som returnerar sant, falskt eller okänt. *Predicate* kan användas för att diktera villkoret i en *clause*. Exempelvis `WHERE Amount > 100` eller `WHERE AGE BETWEEN 20 AND 30`.

- *Keywords* är definierade ord i SQL. Det finns reserverade *keywords* som exempelvis `SELECT`, `GROUP` och `CREATE`. Det finns även icke reserverade *keywords* som exempelvis `COLUMN`, `DAY` och `MONTH`.

- *Identifiers* är namn på objekt i en databas, exempelvis tabeller eller kolumner. I de fall en *identifier* skulle ha samma namn som ett reserverat *keyword* behöver det omges av dubbla citationstecken. Generellt sett bör det undvikas att skapa en *identifier* med samma namn som ett *keyword*.

- *Insignificant whitespaces* eller blanka tecken som exempelvis mellanslag ignoreras i SQL-*queries*. Detta gör att SQL-kod kan formateras på ett läsvänligt sätt. I kodblocket nedan ser vi att

raderna är seperererade med tomma rader utan att funktionaliteten påverkas, det påverkar dock läsbarheten negativt i detta fall.

```
UPDATE Products

SET Price = Price - 50

WHERE ProductName = 'Bicycle';

SELECT *
FROM Products;
```

I ovanstående kodblock ser vi några exempel på SQL-element.

- `UPDATE`, `SET` och `WHERE` är alla tre *keywords*, `UPDATE` är även ett *statement* medan `SET` och `WHERE` är *clauses*.
- `UPDATE` följt av "Products" som är en *identifier* specificerar att tabellen "Products" ska uppdateras.
- `SET` följt av "Price" som också är en *identifier* specificerar att kolumnen "Price" ska uppdateras. "Price - 50" är ett *expression* som anger att priset ska sänkas med 50.
- `WHERE` filtrerar vilka rader som ska uppdateras. I detta fall används ett *predicate* för att endast uppdatera rader där "ProductName" är "Bicycle".
- Mellan raderna finns *insignificant whitespaces* som inte påverkar koden. I detta fallet påverkar det dock läsbarheten negativt.
- Sammansatt är all kod ett statement och i detta fall består vår *query* av två *statements*. När vi utför *queryn* sänks priset för "Bicycle" med 50 kr i tabellen "Products". Detta är en följd av det första *statementet*. I det andra *statementet* läser vi ut hela tabellen genom att skriva `SELECT * FROM PRODUCTS;`. Notera att våra två statements separeras genom semikolon ";".

Notera, vi har här skrivit kod för att demonstrera koncepten och använder alltså ingen databas.

Tabell 3.1: Några vanligt förekommande operatorer i SQL.

Kategori	Operator	Beskrivning
Aritmetiska	+	Addition
Aritmetiska	-	Subtraktion
Aritmetiska	*	Multiplikation
Aritmetiska	/	Division
Aritmetiska	%	Modulo
Jämförelser	=	Lika med.
Jämförelser	>	Större än.
Jämförelser	<	Mindre än.
Jämförelser	>=	Större än eller lika med.
Jämförelser	<=	Mindre än eller lika med.
Jämförelser	<>	Inte lika med.
Logiska	`ALL`	SANT om alla jämförelser i en uppsättning är SANT.
Logiska	`AND`	SANT om båda booleska uttrycken är SANT.
Logiska	`ANY`	SANT om någon av jämförelserna i en uppsättning är SANT.
Logiska	`BETWEEN`	SANT om operanden är inom ett visst intervall.
Logiska	`EXISTS`	SANT om en *subquery* innehåller några rader.
Logiska	`IN`	SANT om operanden är lika med ett av uttrycken i en lista.
Logiska	`LIKE`	SANT om operanden matchar ett mönster.
Logiska	`NOT`	Vänder värdet av en annan boolesk operator.
Logiska	`OR`	SANT om något av de booleska uttrycken är SANT.
Logiska	`SOME`	SANT om några av jämförelserna i en uppsättning är SANT.

3.2 Konventioner

Generellt sett bör SQL-koden vi skriver följa konventioner.

Det finns allmänt accepterade konventioner såsom att kommentarer ska användas där det är nödvändigt och i de fallen ska vi kommentera "varför vi gör något" och inte "vad vi gör". Detta eftersom "vad vi gör" kan utläsas från själva koden och blir alltså onödig upprepning. För att kommentera SQL-kod används två bindestreck "`--`" följt av kommentaren. Om kommentaren sträcker sig över flera rader används "`/*`" för att påbörja kommentaren och "`*/`" för att avsluta den.

Alla nyckelord bör skrivas med versaler, trots att nyckelord i SQL inte är skifteslägeskänsliga. Det innebär att även om exempelvis nyckelordet SELECT skrivs som sEleCT så fungerar fortfarande koden, men det följer inte konventionen, försämrar läsbarheten och bör därför inte skrivas på det sättet.

Identifiers ska vara deskriptiva och namngivning ska vara konsekvent.

Insignificant whitespaces ska användas för att formatera koden på ett sätt som gör den enklare att läsa och tyda.

Det existerar även olika konventioner för olika SQL-varianter. I MSSQL används *PascalCase* för namngivning av tabeller och kolumner. Vi skriver exempelvis "EnStorTabell". I MySQL används *snake_case* för namngivning av tabeller och kolumner, vi skriver exempelvis "en_stor_tabell".

Den intresserade läsaren kan söka på internet för att läsa mer om konventioner.

> **!** I kodexemplen kommer vi följa konventionerna för MSSQL. I de fall koden för MSSQL och MySQL skiljer sig åt demonstrar vi kod för båda databashanterarna och följer respektive databashanterares konventioner.

3.3 CRUD

I detta avsnitt kommer vi att demonstrera hur vi utför *Create*, *Read*, *Update* och *Delete*-momenten i *CRUD*-flödet.

3.3.1 Create

Create-momentet innefattar operationen `INSERT INTO`, som används för att lägga till rader i en tabell. Skapandet av databaser och tabeller inkluderas inte i *CRUD*-flödet. Det kommer ändå att demonstreras i detta avsnitt eftersom det ligger nära till hands att utföra det här. Nyckelordet `CREATE` används för att skapa databaser och tabeller. Som vi senare i detta kapitel kommer att demonstrera kan `CREATE` även användas för att skapa andra objekt, exempelvis vyer och lagrade procedurer.

Vi börjar med att skapa en databas genom ett `CREATE`-*statement*. Vi skriver nyckelordet `CREATE` och specificerar att det är en databas med namnet "FruitStand".

```
CREATE DATABASE FruitStand;                    (1)
```

(1) Nyckelorden `CREATE DATABASE` används för att skapa databasen "FruitStand".

> **i** Semikolon ";" används för att separera flera *statements* i en *query*. Har vi endast ett *statement* behöver vi inte använda ett semikolon för att avsluta vårt *statement*, men det är en god vana att göra det.

Vi kan nu ange att det är i vår nyskapade databas vi vill utföra kommande operationer. Detta gör vi genom nyckelordet `USE` följt av namnet på databasen.

```
USE FruitStand;                                (1)
```

① `USE` används för att specificera att kommande *queries* ska exekveras i databasen "FruitStand".

Nyckelordet `CREATE` används även för att skapa tabeller i databaser. I samma `CREATE`-*statement* anges även kolumnerna som ska skapas i tabellen.

Vi skriver nyckelorden `CREATE TABLE` följt av önskat namn på tabellen. I samma *statement* anger vi kolumners namn och kolumners datatyp samt eventuella nycklar och begränsningar.

```
CREATE TABLE ProductsTable(                    ①
    ProductID INT PRIMARY KEY,                 ②
    ProductName VARCHAR(50) UNIQUE,            ③
    Amount INT,                                ④
    Price INT                                  ⑤
);
```

① Nyckelorden `CREATE TABLE` följs av "ProductsTable" för att skapa en tabell med namnet "ProductsTable".
② Kolumnen "ProductID" skapas med datatypen `INTEGER`, och `PRIMARY KEY` anges för att göra kolumnen till tabellens primärnyckel.
③ Kolumnen "ProductName" skapas med datatypen `VARCHAR` där maximalt 50 tecken får användas. Även en *unique constraint* appliceras.
④ Kolumnen "Amount" skapas med datatypen `INTEGER`.
⑤ Kolumnen "Price" skapas med datatypen `INTEGER`.

> **i** Används inte `USE` för att specificera vilken databas vi vill arbeta i kan vi skriva följande:
> "`CREATE TABLE` FruitStand.dbo.ProductsTable(…)".
> Vi har specificerat att i databasen "FruitStand" under schemat "dbo" ska tabellen "ProductsTable" skapas. "dbo" är en akronym för "database owner" och är ett schema som automatiskt skapas i MSSQL om inte användaren strukturerar tabeller enligt egna definierade scheman. I MySQL används databasnamn för att separera tabeller snarare än scheman. Notera att databaser i MySQL benämns "schema" vilket kan vara en förvirrande aspekt när man växlar mellan MSSQL och MySQL.

För att lägga till rader i vår nyskapade tabell skriver vi ett `INSERT INTO`-*statement*. Vi använder oss av nyckelorden `INSERT INTO` följt av vilken tabell vi vill lägga till rader i. Därefter används nyckelordet `VALUES` följt av önskade värden för raden.

```
INSERT INTO ProductsTable                          ①
VALUES (1, 'Apples', 10, 20);                      ②
```

① "ProductsTable" anges efter `INSERT INTO` som den tabell där rader ska läggas till.
② Önskade värden för alla kolumner specificeras i en parentes efter nyckelordet `VALUES`.

Vi kan även lägga till fler rader i ett *statement*. I koden nedan lägger vi till två rader.

```
INSERT INTO ProductsTable
VALUES (2, 'Pears', 20, 25),                       ①
       (3, 'Bananas', 25, NULL);
```

① Parenteserna separeras med ett kommatecken.

Eftersom vi inte satte någon *constraint* på "Price"-kolumnen när vi

skapade tabellen kunde värdet sättas till `NULL`. Skulle vi försöka lägga till en ny rad med ett produktnamn som redan existerar i tabellen genereras ett felmeddelande eftersom vi satte begränsningen *unique* för kolumnen "ProductName".

```
INSERT INTO ProductsTable
VALUES (4, 'Apples', 30, 12);                    ①
```

① "Apples" som redan existerar för kolumnen "ProductName" anges som önskat värde.

Ovanstående kodexempel genererar följande felmeddelande:

Msg 2627, Level 14, State 1, Line 1 Violation of UNIQUE KEY constraint 'UQ__Products'. Cannot insert duplicate key in object 'dbo.ProductsTable'. The duplicate key value is (Apples).

Vi gjorde kolumnen "ProductID" till vår *primary key* i tabellen vilket innebär att kolumnen endast får ha unika attribut som är skilda från `NULL`. Försöker vi lägga till en rad där "ProductID" är `NULL` får vi återigen ett felmeddelande.

```
INSERT INTO ProductsTable
VALUES (NULL, 'Bananas', 30, 12);
```

Vi får följande felmeddelande:
Msg 515, Level 16, State 2, Line 1 Cannot insert the value NULL into column 'ProductID', table 'FruitStand.dbo.ProductsTable'; column does not allow nulls. INSERT fails.

Nedanstående kod returnerar ett felmeddelande eftersom värdet "3" redan existerar i kolumnen "ProductID". Felet uppstår eftersom tabellers *primary keys* kräver unika attribut.

```
INSERT INTO ProductsTable
VALUES (3, 'Oranges', 30, 12);
```

Msg 2627, Level 14, State 1, Line 1 Violation of PRIMARY KEY constraint 'PK_Products'. Cannot insert duplicate key in object 'dbo.ProductsTable'. The duplicate key value is (3).

3.3.2 Read

Read-momentet består av operationen `SELECT` som läser ut data från en tabell.

För att utföra operationen skriver vi nyckelordet `SELECT` följt av vilka kolumner vi vill läsa ut och i vilken tabell kolumnerna finns.

```
SELECT * FROM ProductsTable;                    (1)
```

(1) Genom att använda stjärnan "*" så väljs alla kolumner från "ProductsTable"-tabellen.

ProductID	ProductName	Amount	Price
1	Apples	10	20
2	Pears	20	25
3	Bananas	25	NA

Skulle vi endast vara intresserade av att läsa ut specifika kolumner kan vi specificera detta i ett `SELECT`-*statement*.

```
SELECT ProductName,
       Price                                    (1)
FROM ProductsTable;
```

(1) Endast "ProductName" och "Price"-kolumnerna läses ut.

ProductName	Price
Apples	20
Pears	25
Bananas	NA

3.3.3 Update

Update-momentet består av operationen `UPDATE` som används för att uppdatera rader i en tabell.

Följande kodexempel demonstrerar hur `UPDATE` används för att uppdatera rader i en tabell.

```
UPDATE ProductsTable                                    ①
SET Price = 25                                          ②
WHERE ProductName = 'Apples';                           ③
```

① `UPDATE` används för att specificera att det är i "ProductsTable"-tabellen som operationen ska utföras.
② `SET` uppdaterar "Price"-kolumnen till 25 för alla rader som matchar villkoret i kommande `WHERE`-*clause*.
③ `WHERE` används för att specificera att enbart rader där "ProductName"-kolumnen har värdet "Apples" ska uppdateras.

Vi skriver ett `SELECT`-*statement* för att läsa ut raderna och verifiera att uppdateringen har utförts enligt förväntan.

```
SELECT Price,
       ProductName
FROM ProductsTable
WHERE ProductName = 'Apples';
```

Price	ProductName
25	Apples

Vi ser att produkten "Apples" nu har priset 25 precis som vi önskade.

3.3.4 Delete

Delete-momentet består av en operation, `DELETE`, som används för att radera rader i en tabell. Vi kommer i detta avsnitt även demonstrera `DROP` eftersom det ligger nära till hands. `DROP` används för att radera bland annat tabeller och databaser.

Vi använder nyckelordet `DELETE` följt av den tabell vi vill radera rader i. Vi behöver också specificera vilken eller vilka rader vi vill radera genom en `WHERE`-*clause*. Använder vi inte en `WHERE`-*clause* raderas alla rader i tabellen.

```
DELETE FROM ProductsTable                    (1)
WHERE ProductName = 'Apples';                (2)
```

(1) "ProductsTable" anges som tabellen där rader ska raderas.
(2) En `WHERE`-*clause* används för att specificera att rader där kolumnen "ProductName" har värdet "Apples" ska raderas.

För att verifiera att raden tagits bort kan vi skriva ett `SELECT`-*statement*.

```
SELECT * FROM ProductsTable
WHERE ProductName = 'Apples';
```

ProductID	ProductName	Amount	Price

Vi ser att raden med produktnamnet "Apples" har raderats precis som förväntat.

För att radera en hel tabell skriver vi ett `DROP`-*statement*. Vi använder nyckelorden `DROP TABLE` följt av namnet på den tabell vi vill radera.

```
DROP TABLE ProductsTable;                    (1)
```

(1) "ProductsTable" specificeras efter `DROP TABLE` för att raderas.

Vi kan också radera en hel databas genom ett `DROP`-*statement*. Vi specificerar att det är en databas med nyckelorden `DROP DATABASE` följt av namnet på den databas vi vill radera.

```
DROP DATABASE FruitStand;
```

3.4 Introduktion av databasen Köksglädje

I detta avsnitt introduceras exempeldatabasen "Köksglädje" som vi kommer använda ett flertal gånger i boken. Databasen finns tillgänglig på bokens *GitHub*-sida. Vi börjar med att beskriva databasen i ord och genom ett *Entity-Relationship*-diagram (ER-diagram). Därefter läser vi in databasen och gör några utskrifter från den.

3.4.1 Beskrivning av databasen Köksglädje

Databasen "Köksglädje" är en exempeldatabas för en fiktiv svensk butikskedja som säljer köksutrustning. All data som representerar priser och transaktioner är i svenska kronor, SEK. I databasen "Köksglädje" finns det totalt 8 tabeller.

"CustomerContactLog"-tabellen har information relaterad till kampanjutskick mot kund och innehåller kolumnerna:

- ContactLogID som är tabellens primärnyckel.
- CustomerID som är varje kunds unika kundnummer.
- CampaignID som är varje kampanjs unika nummer.
- ContactDate som innehåller datum för eventuella kampanjutskick.

"Customers"-tabellen har information relaterat till specifika kunder och innehåller kolumnerna:

- CustomerID som är tabellens primärnyckel.
- FirstName
- LastName

- Email
- ApprovedToContact som visar om en kund får kontaktas.
- ActiveMember som visar om en kund just nu är medlem.
- JoinDate

”MarketingCampaigns”-tabellen har information om existerande kampanjer och innehåller kolumnerna:

- CampaignID som är tabellens primärnyckel.
- CampaignName som visar en kampanjs namn.
- StartDate
- EndDate
- DiscountPercentage
- CategoryID

”ProductCategories”-tabellen har information om de olika produktkategorier som finns och innehåller kolumnerna:

- CategoryID som är tabellens primärnyckel.
- CategoryName
- Description som beskriver de olika produktkategorierna.

”Products”-tabellen har information relaterad till de produkter som säljs och innehåller kolumnerna:

- ProductID som är tabellens primärnyckel.
- ProductName
- Description
- Price som visar produktens försäljningspris.
- CostPrice som visar produktens inköpspris.
- CategoryID

“Stores”-tabellen som har information relaterad till varje butik och innehåller kolumnerna:

- StoreID som är tabellens primärnyckel.
- StoreName
- Location
- ManagerName
- ContactNumber

”TransactionDetails”-tabellen har detaljerad information om varje transaktion och innehåller kolumnerna:

- TransactionDetailID som är tabellens primärnyckel.
- TransactionID
- ProductID
- Quantity
- PriceAtPurchase
- CampaignID
- TotalPrice

”Transactions”-tabellen har generell information om varje transaktion och innehåller kolumnerna:

- TransactionID som är tabellens primärnyckel.
- CustomerID
- StoreID
- TransactionDate
- TotalAmount

Notera att exempelvis ”TotalPrice” från tabellen ”TransactionDetails” representerar den totala summan per produkt ifall en kund köpt flera produkter. Det vill säga ”PriceAtPurchase” multiplicerat med ”Quantity”. Detta kan jämföras med ”TotalAmount” från ”Transactions”-tabellen, som i stället visar totalbeloppet för en hel transaktion som kan innehålla flera produkter. ”TransactionDetails”-tabellen innehåller alltså detaljerad information, medan ”Transactions”-tabellen innehåller övergripande information. Skulle en transaktion från ”Transactions”-tabellen innehålla fyra olika produkter så beskrivs dessa i detalj i ”TransactionDetails”-tabellen. Denna distinktion är viktig att göra då varje produkt kan ha varierande pris, beroende på om en kampanj är aktiv eller inte. Denna information hämtas då från ”MarketingCampaigns”-tabellen.

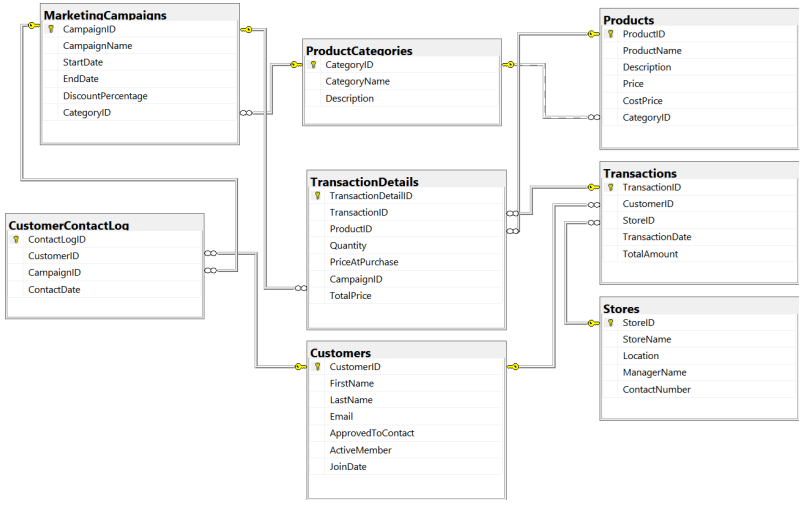
3.4.2 Entity-Relationship-diagram

Entity-Relationship-diagram benämns ofta som “ER-diagram” och används för att visa strukturen i en databas. I diagrammet visas tabeller (*entity*) och deras tillhörande kolumner samt relationerna (*relations-*

hip) mellan tabellerna.

För att generera ER-diagram finns det inbyggda verktyg i både MySQL och MSSQL. I MySQL klickar vi på “Database” och sedan “Reverse engineer” för att lägga till de tabeller vi önskar i vårt diagram. I MSSQL använder vi “Object Explorer” och högerklickar på “Database Diagrams” för att sedan välja “New Database Diagram”. Andra verktyg som kan användas för att skapa ER-digram är “Lucidchart”, “SQL Database Modeler” eller “Quick Database Diagrams”. Du kan snabbt lära dig hur du till exempel använder “Quick Database Diagrams” genom att gå till deras hemsida: https://www.quickdatabasediagrams.com/ . De flesta av de ER-diagram som visades i Kapitel 2 skapades med hjälp av “Quick Database Diagrams”. Exempelvis Figur 2.6.

I Figur 3.1 ser vi ett ER-diagram för databasen “Köksglädje”. Vi kan se vilka tabeller som finns i databasen och respektive tabells kolumner. Vi ser även relationerna mellan tabellerna.



Figur 3.1: ER-diagram för “Köksglädje” som skapats i MSSQL.

3.4.3 Inläsning av databasen Köksglädje

För att kunna använda databaser i databashanterare såsom *MySQL* eller MSSQL behöver vi först importera dem. I bokens tillhörande *GitHub*-sida hittar vi databasfilerna och vi förklarar nu hur de importeras för respektive databashanterare.

I MySQL utgår vi från "Local instance"-fliken, klickar på "Server" och väljer "Data Import". Vi väljer "Import from Self-Contained File" och anger sökvägen till filen. Sedan klickar vi på "Start Import", när importen är slutförd klickar vi på uppdatera-knappen och vi kan se vår databas "Köksglädje" i programmet.

I MSSQL högerklickar vi på "Databases", trycker på "Restore Database", "Device" och därefter på de tre punkterna "…", "Add" och lokaliserar den plats vi har sparat *.bak* filen på. Slutligen trycker vi på OK och slutför inläsningen av databasen.

När vi har läst in databasen "Köksglädje" i vår databashanterare kan vi göra några utskrifter från den för att inspektera datan.

Vi börjar med att ange att det är i databasen "Köksglädje" vi vill utföra kommande operationer. Detta gör vi genom nyckelordet `USE` följt av "Köksglädje".

```
USE Köksglädje;
```

I nedanstående kodexempel skriver vi ett `SELECT`-*statement* för att se vilka de olika produktkategorierna är.

```
SELECT CategoryID,
       CategoryName                                    ①
FROM ProductCategories;
```

① Kolumnerna "CategoryID" och "CategoryName" kolumnerna läses ut.

CategoryID	CategoryName
1	Kökskniver
2	Köksmaskiner
3	Grytor/Kastruller
4	Bakredskap
5	Köksredskap
6	Förvaring
7	Stekpannor
8	Övrigt

I nedanstående kodexempel använder vi en `WHERE`-*clause* på "CategoryID"-kolumnen för att se namn och pris för alla produkter i kategorin "Kökskniver".

```
SELECT ProductName,
       Price
FROM Products
WHERE CategoryID = '1';                                    (1)
```

(1) `WHERE` används för att filtrera datan genom villkoret "CategoryID = 1".

ProductName	Price
Kockkniv	999
Brödkniv	749
Santokukniv	1199
Skalkniv	299
Grönsakskniv	499

3.5 Queries

I detta avsnitt kommer vi att demonstrera olika typer av *queries*.

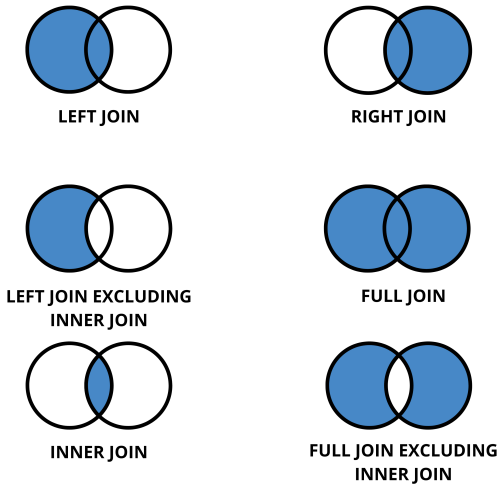
3.5.1 Joins

`JOIN`-*clauses* i SQL används för att kombinera kolumner från en eller flera tabeller till en ny tabell. Oftast används två eller flera tabeller men det är även möjligt att göra det som kallas en `SELF JOIN`. Detta innebär att en tabell utför en *join* på sig själv. Vi går inte in på detta utan nämner endast det för den intresserade läsaren.

I detta avsnitt kommer följande sex *joins* att demonstreras:

- `INNER JOIN`
- `LEFT JOIN`
- `RIGHT JOIN`
- `FULL JOIN`
- `LEFT JOIN EXCLUDING INNER JOIN`
- `FULL JOIN EXCLUDING INNER JOIN`

Se Figur 3.2 för en grafisk representation över dessa *joins*.



Figur 3.2: De sex nämnda joins.