Informations Teknik och Databasutveckling

# Vad är en databas

En databas är en lagringsplats för data

De består utav en eller flera tabeller

Varje tabell består utav en eller flera rader

Varje tabell är en lista utav olika saker. En rad är ett objekt eller en instans utav den typen utav objekt.

Varje kolumn är ett fält.

För att komma åt och hantera databaser används scriptspråk

Finns olika typer utav SQL som alla stöder CRUD, MySQL, MsSQL etc

SQL är ett scriptspråk som används för att hantera databaser. Skriva ny data, hämta data, ta bort data etc.

SQL är standard sedan 1986.

# Att använda en databas

För att använda en databas behöver du:

En databasserver. Tillgänglig lokalt eller via internet.

En RDBMS, Reletion Database Management System ex WorkBench

SQL för att hämta och skriva data

Ett scriptspråk likt PHP eller ASP, dock bara om det är en hemsida.

Det är viktigt att skilja på databaser och databasserver.

En databasserver kan innehålla en eller flera databaser. Det är datorn som håller databasen

En databas är en kollektion utav tabeller.

# Arbeta med databaser, SQL

Databaser i MySQL refereras som ”schema”.

För att skapa en databas så använd kommandot:

### Skapa databas

**Create Schema <namn på databas>;**

Exempel:

**Create Schema GustavsServer;**

Alla databasnamn sparas med småbokstäver så namnet på databasen blir här ”gustavserver”.

Tryck på blixten för att köra alla skrivna kommandon.

Du kan se dina databaser i det högra fältet om du trycker på den nedre fliken till höger.

När du skapat en databas så har den automatiskt fyra underkategorier: Tables, Views, Stored procedures och Functions.

Om du höger trycker på din databas så kan du sätta den som standard så att du inte skriver massor script och data i systemet.

## Schema vs Databas

I programmet MySQL som vi använder så är schema synonymt med databas. Du kan lika gärna använda kommandot ”**create database <databasNamn>**” Det är dock värt att påpeka att i andra program så är schema och databas olika saker.

I dem är ofta ett schema en del utav en databas, ofta en samling utav tabeller. Och en databas är en samling utav scheman.

### CRUD

**C-create**

**R-read**

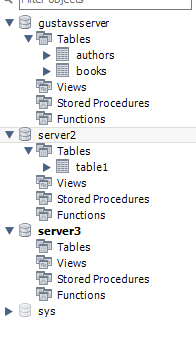
**U-update**

**D-delete/drop**

## Navigera mellan databaser/scheman

När du skriver SQL kommandon så utförs de i den aktiva databasen. Om du till exempel skappar en tabell så hamnar de i den aktiva databasen. För att ändra vilken databas kommandona ska utföras i kan du använda nyckelordet ”**use**” för att ändra den aktiva databasen.

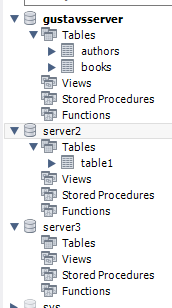
Du ser vilken databas/schema som är aktiv om du kollar i det vänstra navigeringsfältet. Den aktiva databasen är skriven i fet still.



Ovan är ”server3” aktiv.

Jag ändrar databas:

**Use gustavsserver;**



## Skapa en ny tabell

När en databas är skapad kan man skapa en tabell till den

Kommandot är:

**Create table<Dbnamn>.<Tabellnamn>(<fältnamn><databtyp>,<fältnamn><datatyp>);**

Man anger alltså vilken databas man vill skapa en tabell i, följt utav namnet på tabelen. Sedan så beskriver man varje kolumn med två värden inom paranteserna. Vad kolumnen/fältet ska heta samt vilken datatyp den ska ta.

Man måste inte ange databasen. Det räcker att skriva…:

**Create table <Tabellnamn>(**

**<fältnamn><databtyp>,**

**<fältnamn><datatyp>**

**);**

…så länge du är i den databasen du vill använda.

Dent finns några olika giltiga datatyper.

**Int** (heltal)

**Datetime** (datum och tid)

**Varchar(längd)** Textvärden. Tänk dig string. Man måste ange hur lång textbiten får vara inom paranteserna

**Float** Flytvärden/decimaler.

Exempel:

**create table GustavsServer.Books (**

**Title varchar(45),**

**Author varchar(45),**

**PageCount int);**

## Primärnyckel

En primärnyckel är ett värde som alltid är unikt för varje rad. Oftast ett ID.

Man kan använda ett kommando som heter ”**Not Null**” så att fältet alltid måste ha ett värde.

Om man använder ”**Auto\_Increment**” så kommer det automatiskt att skapas ett värde som är ett högre en det senaste om inget annat värde anges.

Ex

**create table gustavsserver.books(**

**Id int not null auto\_increment,** Vi har alltså skapat ett fält som heter ID som måste anges ett värde men också gör det automatiskt.

**Title varchar(45),**

**Author varchar(45),**

**PageCount int,**

**primary key(Id)** Vi anger att fältet Id är huvudnyckeln.

**);**

Man kan ange att en primary key ska bestå utav två andra

## Stoppa in värden i en tabell

Om du använder kommandot:

**select \* from gustavsserver.books;**

Så får du fram ett fönster som visar alla rader i din tabell. Du kan också höger trycka på tabellen i det högra fältet och välja ”select rows”.

Du kan sedan skriva in värden manuellt via den grafiska interfacen.

Sedan kan du trycka på ”apply”(nere till höger). Du kommer då få fram ett fönster som visar vilket kommando som körs. Om du vill köra det manuellt själv så skriver du:

**INSERT INTO `gustavsserver`.`books` (`Id`, `Title`, `Author`, `PageCount`) VALUES ('8', 'Lord of the Rings', 'Tolkien', '500');**

Med nyckelorden **Insert Into** så kan du ange vilken databas och vilken tabell du vill skapa en ny rad i. Du ska sedan ange vilka fält du vill lägga in värden i. Sedan använder du nyckelordet **VALUES** och skriver in vilka värden som ska in.

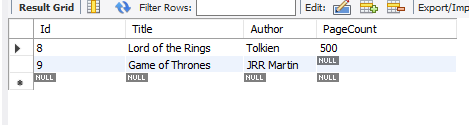
Om vi nu försöker stoppa in en rad utan ID värde så kommer det att automatiskt sätta ett värde högre än det senaste.

Ex:

**insert into gustavsserver.books (Title, Author)**

**Values ("Game of Thrones", "JRR Martin");**

Resultat: Notera Id på ”Game of Thornes”.



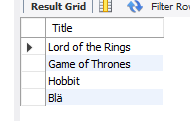
Notera också att ”PageCount” kan vara null.

## Hämta data, Select

Med nyckelordet ”**select”** så kan du hämta data. Du skriver vilket fält du vill komma åt efter select och sedan anger du från vilken databas och tabell du vill hämta.

EX:

**Select Title From gustavsserver.books;**



Om du skriver en stjärna/multiplikationstecken likt tidigare så kan du hämta all data.

**Select \* From gustavsserver.books;**

Om du inte vill ha några dubbletter utav hämtade data så kan du lägga till nyckelordet ”**distinct**”

Ex:

**Select distinct \* From gustavsserver.books;**

### Funktioner när du hämtar data

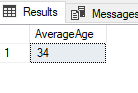
Om du vill hitta medel,max eller minimum värden utav data du hämtar så finns det funktioner du kan använda dig utav i en ”select”-sats.

Man skriver select som vanligt men sedan så skriver du ett funktionsnamn med parenteser. I dessa parenteser så skriver du sedan det fältet som du vill behandla med funktionen. Efter det så ska man skapa ett tillfälligt fält där resultatet utav funktionen placeras. Du skapar detta tillfälliga fält med nyckelordet ”**as**”. Till sist så anger du vilken tabell data ska komma från med ”**from**” som vanligt.

Exempel, vi vill veta medelåldern i en tabell med personer kallad ”**people**”, där personers åldrar ligger i ett fällt kallat ”age”.

**Select avg(age) as AverageAge**

**From people;**



Så här skulle resultatet utav ovan exempel se ut.

Lista på några bra funktioner:

* **Max()** - Hämtar det högsta värdet.
* **Min()** - Hämtar det minsta värdet.
* **Avg()** - Hämtar medelvärdet.
* **Count()** - Hämtar hur många rader det finns. Används ofta med villkor för att se hur många utav en viss typ det kan finnas.
* **Sum()** - Används i ett fält utav en numerisk datatyp (int, float etc) för att hämta den totala summan utav alla fält.

## Filtrerea data, Where

Med nyckelordet ”**Where**” så kan du filtrera vilken data du vill hämta. Det kan också användas för andra kommandon.

**Select \* From gustavsserver.books where Title = "Game of Thrones";**

## 

**Select \* From gustavsserver.books where PageCount = 123;**



Om du vill använda dig utav flera villkor/filter så kan du använda nyckelordet ”**and**” efter where för att lägga till fler villkor.

Ex:

**Select \* from gustavsserver**

**where PageCount = 123**

**and Title = ’Hobbit’;**

### like, filtrera lik data

Om du vill hämta mycket data som alla har liknande värden i något fält men inte exakt samma, så kan man använda ”**like**” operatorn istället för ”**=**” i ett where-uttryck. Säg att vi har en stor tabell med namn och adresser och vi vill få fram alla som bor på Linköpingsvägen. Linköpingsvägen har adresser från Linköpingsvägen1 till Linköpingsvägen2000. Vi kan använda ett SQL kommando som ser ut så här:

**Select \* from personer**

**Where Adress like ’Linköpingsvägen%’;**

Vad vi har skrivit ovan är att alla rader där fältet börjar med ’Linköpingsvägen’ och sedan fortsätter ska hämtas. Man kan se % som en osv.

Man kombinerar alltså en like operator med olika villkor i det värde man letar efter.

**’a%’** = hitta alla värden som börjar med ’a’.

**’%a’** = hitta alla värden som slutar med ’a’.

**’%a%’** = hitta alla värden som har ’a’ i sig.

**’a\_’** = hitta alla värden som börjar med ett a och har en till symbol (bokstav siffra etc ex ’**a1**’, ’**ac**’).

**’a%o’** = hitta alla värden som börjar med ’a’ och slutar med ’o’.

### Flera villkor

Om du vill ha flera villkor i din where-sats så kan du binda ihop olika villkor med nyckelordet ”**and**”.

Ex:

**select \* from books**

**where Title = 'Físh' and PageCount = 2000;**

Hämtar alla böcker med titeln ’Fish’ och en sidmängd på 2000.

## Updatera värden i en tabell, Update

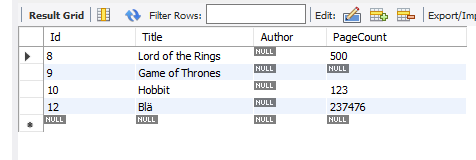
Med ”Update” så kan man uppdatera och ändra värden i en tabell. Om man inte filtrerar vad man vill uppdatera så ändras alla värden. Med ”**set**” så anger man vilket fält som ska ändras.

Ex

**update gustavsserver.books**

**set Author = null;**

Ovan ändrar alla fält i ”Author” kolumnen till null.

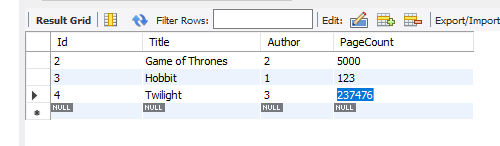


## Radera värden, delete

Med ”**delete**” så kan man ta bort rader i en tabell. Om man inte filtrerar med **”where”** så tas all data i tabellen bort. För att kunna radera rader så måste man vara ute ur ”safe mode”. Detta kan ändras i inställningarna.

Ex:

**Delete From gustavsserver where Title = ”Lord of the Rings”;**



Notera att Id 1 försvann. Om vi hade tagit bort 4 och sedan skapat en ny bok så hade nästa id ändå blivit 5.

## Ändra fält i en tabell, alter

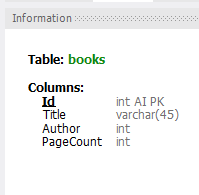
Vi kan ändra själva fälten (kolumnerna) i en tabell, text deras namn, datatyp etc.

Man gör detta med ”**alter table**” och ”**modify column**”. När man gör detta så måste man först radera eller ändra alla värden till null.

Vi vill ändra på ”**Author**” fältet och göra data typen till int.

**alter table gustavsserver.books**

**modify column Author int;**



Notera ”Author”. Den är nu int.

Om man vill ta bort en kolumn helt och hålet så använder man nyckelordet **drop column**.

**Alter table gustavsserver.books**

**Drop column Title;**

Om du har en tabell skapad men den har ingen **primary key** så kan du lägga till det med kommandot**:**

**ALTER TABLE <tabell>  
ADD PRIMARY KEY (<fält>);**

Ex:

**Alter table gustavsserver.books**

**Add primary key(Id);**

Om du vill ta bort primary key egenskapen från ett fält så kan du skriva:

**Alter table gustavsserver**

**Drop primary key**

## Hämta data från 2 tabeller med relation, join

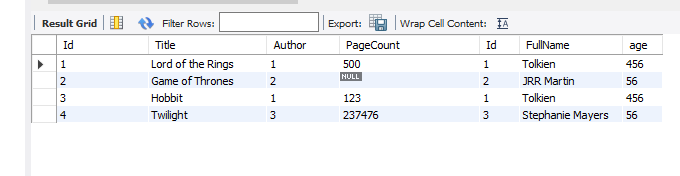
För at hämta data från databaser med ett fält som är relaterade så använder man nyckelordet ”**join**”

I vårt fall så är Author i books lika med Id i authors. För att ange vilka två fält som ska matcha så använder man nyckelordet ”**on**”.

Ex:

**Select \* from gustavsserver.books**

**join authors on books.Author = authors.Id;**



## Ordna data, Order By

Om du ska hämta data men vill ha det ordnat efter ett visst fält så kan du använda ”**order by**” kommandot.

Ex:

**Select \* from books**

**Order by books.PageCount;**

Om du vill ordna tabellen efter flera fält så lägger du till de fälten med ett kommatecken.

**Select \* from books**

**Order by books.PageCount,books.Author;**

## Kommentarer i SQL.

Om du vill skriva en rad kommentarer i SQL så använder du ”**--**”

Ex:

**--detta är en kommentar.**

Om du vill kommentera en sektion så använder du ”**/\*\*/**” likt c#.

**/\***

**Detta är en kommentar**

**\*/**

## Foregin keys

Om du har två olika tabeller med två fält som representerar samma sak så är det smart om des värden kan vara samma. Till exempel så kanske du har en tabell med personer och en tabell med order från din internetshop. Du vill kunna se på ordern vem det är som har lagt den så du vill kunna se personens unika Id i orderlistan. Med ”**foregin key**” så kan du referara till ett fält i en annan tabell och ange det som ett värde.

Nedan har jag en tabell med kunder och en tabell med lagda order.

**create table Customers (**

**FirstName varchar(45),**

**LastName varchar(45),**

**Age int,**

**BirthDate date,**

**ID int not null auto\_increment,**

**primary key(ID)**

**);**

**create table Orders (**

**OrderDate datetime,**

**OrderID not null,**

**Customer int not null,**

**PriceTotal float,**

**foreign key(Customer) references Customers(ID)** Se här.

**Primary key(OrderID)**

**);**

Genom att göra så här så har vi sett till att det inte går att lägga till några ”Customer” som inte finns i ”Customers” tabellen.

Syftet med en foregin key är alltså att kunna länka ihop olika värden i olika tabeller så att de representerar samma sak.

När du skapar en foregin key så måste den referera till en primary key.

Tabellen med foregin key kallas för ”**child tablle**” och tabellen med värdet som den refererar till kallas ”**parent table**”.

# Git, GitHub

Git är ett verison Control system som, tillåter oss att spara olika verisioner och filer utav ett program.

## Repository

En repository är en samling utav filer som är tillsammans i Git. Man lägger till filer med ”Add”. Och man sparar filer med ”Commit”.

## Använda Git

Börja med att skapa en ny mapp på skrivbordet.

Om du har GIT installerat så kan du öppna mappen med GIT. Antagligen med en GUI eller via kommando prompten.

Vanliga kommandon:

**LS** – visar innehållet i mappen

**Touch** – skapar en fil man måste ange namnet med en fil ändelse tex index.html.

**Touch index.html**

**Git init** – ”git initialise”. Skapar en repository.

Använda ”Git init” för att skapa en repository så du kan börja jobba med GIT.

För att kunna använda kommandon som push så måste du ha dit namn sparat i Global config.

Använd kommandot:

**Git config –global user.name ’Gustav Höglund’**

Och

**Git config –global user.email ’gustav.hoglund.syne20lin@tucsweden.com’**

Den lokala filen kan enbart innehålla ett namn och en epost.

Lite mer kommando. Taget från Marcus PowerPoint

**git init** - Skapar en ny Lokal Repository

**git add** **<file>** - Lägg till en fil till Repository.

**Git rm <file>** - ta bort en fil.

**git status** - Se statusen av din Repository

**git commit** - Bekräfta och gör en ny version i din Repository.

**git push** - Skicka din Repository till Git Hub

**git pull** - Hämta Repository från Git Hub

**git clone** - Skapa en ny Lokal Repository baserat på en online Repository

Om du använder git status så kan du se vilka filer som är sparade i repository. Om de är röda så använd ”git add”. Den filen är då cashad och du kan använda ”commit” för att spara den sedan.

Om du vill lägga till alla filer utav en viss typ så kan du använda ”add” tillsammans med en ”\*” och filändelsen

**Git add \*.html**

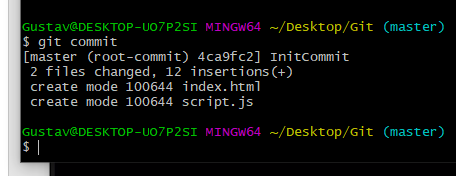
Om du vill lägga till alla filer i mappen så använder du enbart en ”.”.

**Git add .**

Om du ändrar en fil som är i staging/cashad (du har använt ”add” och sedan gjort något med den) så kommer de ändringarna inte följa med. Om du kollar status så kommer du se att den har ändrats. Du måste lägga till den igen för att korrekt variant ska sparas.

För att spara filen i repositoryn så använda commit. När du gör det så kommer en texteditor att hoppa upp.

Praxis när du gör den första commit:en så ska man skriva en kommentar ”InitComit”. För att kunna skriva en kommentar så ska man trycka på **I**. Sedan så trycker du på **Escape** för att hoppa ut texteditorn. Och till sist så skriver du ”**:wq**” för att hoppa ut.



Efter att du gjort din ”commit” så kan det vara bra att kolla status.

Ett snabbt och enkelt sätt att göra allt detta är med kommandot:

**Git commit -m ’meddelande’**

Vi kan alltså skriva meddelanden med ”-m”.

## .gitignore

Om du vill ha några filer som ignoreras utav GIT så ska du skapa en fil som kallas **”.gitignore**”

Alltså:

**Touch .gitignore**

Detta skapar en text fil. I textfilen så skriver du namnet på filerna du vill att GIT ignorerar.

Det går att ignorera mappar och hela deras innehåll.

För att skapa en mapp i GIT bash så skriv:

**Mkdir mapp2**

Så skapas en mapp.

Skriv sedan ”**/mapp2**” i .gitignore så ignoreras den mappen och allt dess innehåll.

Om du vill ignorera alla filer utav en viss typ så skriver du en ”**\***” och följer det med filändelsen utav typen du vill ignorera. Som exempel så vill jag att GIT ignorerar alla ”.txt” filer så jag skriver följande i ”.gitignore”

**\*.txt**

När du vill ignorera flera filer så ska de alla skrivas på varsin ny rad.

Ex

**\*.txt**

**/mapp2**

## Branches

Om du arbetar på ett projekt och ska lägga till någon form utav funktionalitet så är det inte särskilt smart att göra ändringar direkt i de filer som finns. Det vore smartare att skapa en kopia utav hela projektet och arbeta med detta. Detta förförande kallas att göra en ny gren, ”**branch**”, utav projektet. Det gör så att du kan göra ändringar på kodbasen utan att det stör pågående operationer eller andra personers arbete. När du sedan är färdig så kan du implementera dina ändringar i huvud grenen som kallas ”**master**”.

För att skapa en ny gren använd kommandot:”**Git branch <namn>”**

Exempel:

**Git branch GustavBranch**

Detta skapar då en gren kallad ”GustavBranch”.

Du kan se vilken gren du är på i GIT bash längst till höger i raden som också visar dit directory (vart du är).

Gusta@LAPTOP-GOHBUROR MINGW64 ~/Desktop/GIT (master)

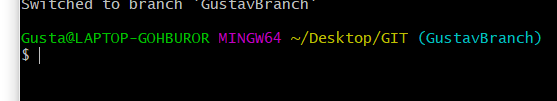
Här ser vi att jag är i master även efter att jag skapat en ny gren.

Du ser också vilken gren du arbetar i om du kollar status med ”git status”.

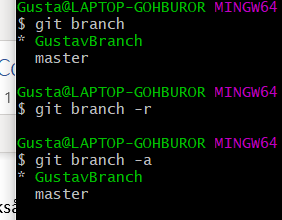
För att hoppa till din nya gren, eller någon annan gren, använd kommandot: ”**git checkout <grennamn>**”

Ex:

**Git checkout GustavBranch**



Om du använder kommandot ”**git branch**” så kan du se vilka grenar som finns tillgängliga lokalt. Altså på datorn du använder. Du kan lägga till ”**-r**” för att se de som finns på en annan dator (även över internet). Lägger du till ”**-a**” så kan du se alla tillgängliga grenar.



När du är i din egna gren så kan du göra ditt arbete utan att det påverkar master-repository:n. Du kommer att se att filer och sådant som du skapar i projektet dycker upp som vanligt i den mappen du har projektet. Glöm inte att lägga till (”git add”) och ”commit:a” de ändringar du gör.

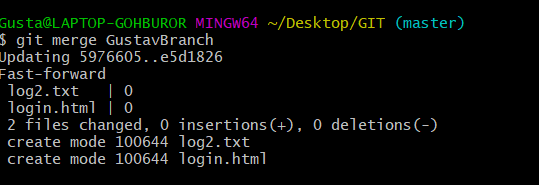
När du har gjort det du ska och testat att det fungerar så kan du slå ihop grenarna för att föra över ditt arbete till master-grenen. Först måste du då gå tillbaka till huvud-grenen.

**Git checkout master**

När du hoppar över till master-grenen så bör du se att de filer du skapade i din gren försvinner från mappen.

För att sedan slå ihop grenarna och föra över det arbete du gjorde så använd kommandot ”**git merge <grennamn>”.**

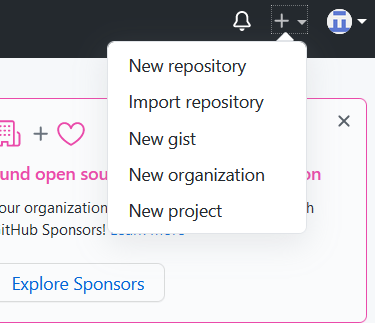
**Ex:**



## GITHub, Repository över internet/remote repository

Om du samarbetar med flera personer eller arbetar på flera olika datorer kan det vara smidigt att lägga en repoitory på internet. Detta kan du göra med GitHub.com.

På github.com så kan du skapa en repository om du klicakr på + tecknet uppe till höger.



Du kommer då få en länk som du ska använda ”**git remote add <förvarNamn> <länk>**” med.

Ex:

**git remote add origin https://github.com/Gurra-byte/rep1.git**

Om du sedan använder ”git remote” för att se vilka icke lokala repository du har som kommer du se att det finns en som heter ”origin”.

Efter det så behöver du använda:

**git push -u origin master**

Detta lägger upp din master branch till förvaret/repoitory på github.com.

Du kommer att behöva logga in på github igen. Om du laddar om sidan kommer du då att se att du har filerna tillgängliga på github.se. Om du sedan gör några ändringar så behöver du bara använda ”**git push**” för att skicka det till github.se. Om någon annan gör ändringar som du vill hämta så använder du ”**git pull**”. Om det finns flera icke-lokala förvar så skulle det hämta/skicka alla. Det är då bra att vara specifik och använda namnet eller <url:en> på det förvaret du vill hämta/skicka.

Ex: **git pull origin** hämtar från ”origin”.

Om du vill dela med dig utav projektet eller använda samma repository för ett annat projekt, så kan du klicka på ”code” (på github.com). Du kommer där få en HTTPS länk till din repository.

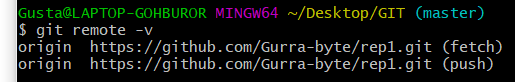
Du använder då kommandot ”**git clone <länk>”** för att hämta ned hela projektet. Detta kommer laddas ned i en mapp.

Ex:

**git clone https://github.com/Gurra-byte/rep1.git**

Om du vill se vilka icke-lokala förvar/repositories du har i ett projekt och deras <url:er> så kan du använda ”**git remote -v**”.

Ex:



# Skapa en databas med tabeller med en grafisk UI

Det är kan vara lite jobbigt att skapa en databas med enbart skriptkommandon men smidigt nog så finns det ett sätt att med en GUI bygga en modell för en databas som du sedan kan använda som mall för en databas.

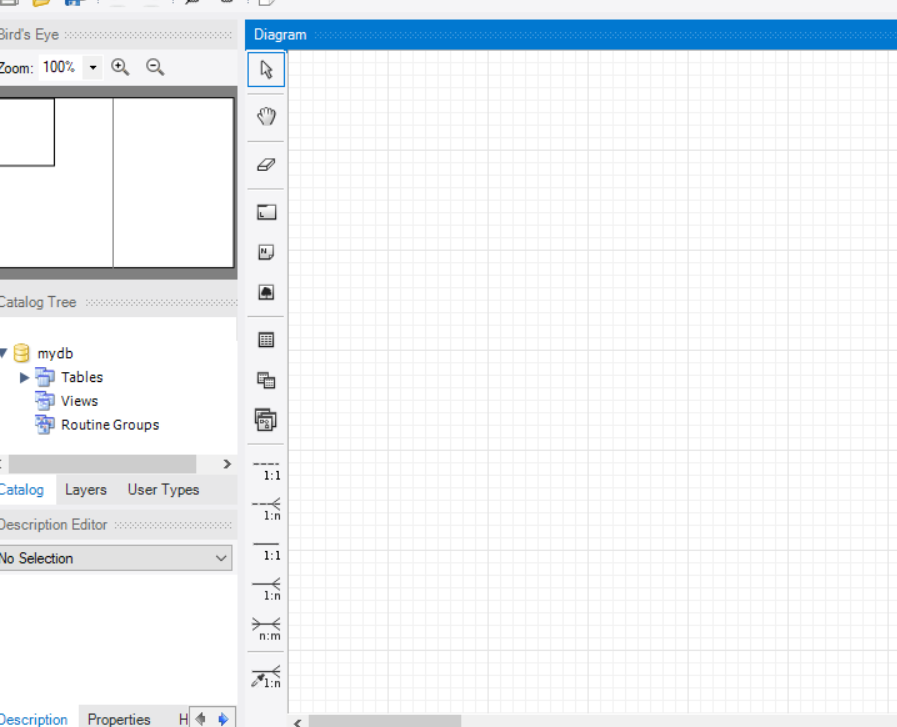
Tryck på:

**File** (uppe till höger) **-> New model**

Du får då upp en ruta där du kan skapa tabeller, vyer, rutiner, och mera.

För att göra allt detta i ett mer övergripande grafiskt diagram så kan du trycka på ”**add diagram**” längst upp. Det diagram som skapas kommer att ge dig möjligheten att arbeta med hela databasen i ett enkelt att förstå gränssnitt.

Dubbeltryck på diagramet som skapas så kommer det att se ut så här:



Rutnätet till höger är din databas och dess innehåll. Till vänster så har du dina redigerings verktyg.

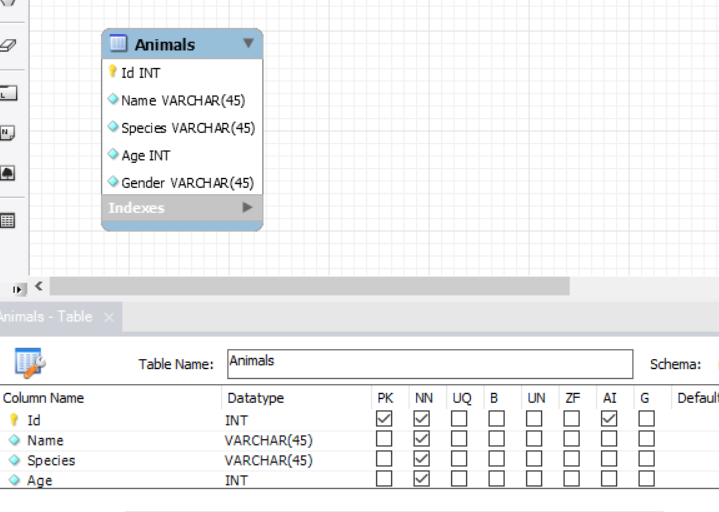
Med de i mitten så kan du skapa tabeller, vyer och rutin grupper.

En vy visar resultat utav ett givet SQL kommando, tänk vad du ser när du använder ”select \* from <tabell>”. Resultatrutan som hoppar upp är en vy.

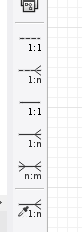
För att skapa ett objekt tryck på vad du vill ha och tryck sedan vart i rutnätet du vill ha det.

När du har skapat ett objekt som en tabell så kan du dubbel trycka på det för att editera det. Du kan här ändra namn på tabellen och skapa fält/kolumner till den. Du kan också ange egenskaper på fälten som primary key, bot null, auto increment etc.

Ex:



Om du vill skapa foregin keys, dvs fält som är referenser till en annan tabells fält, så använder du dig utav linjerna som finns under tabell, vy och rutin grupp knapparna.



För att skapa referenser så klickar du på en utav knapparna. Sedan trycker du på den tabellen som ska ha en referens. Och sen så trycker du på den tabellen som ska refereras till. Det fältet som skapas i den första tabellen kommer att anta värdet utav huvudnyckeln/”primary key” i den andra tabellen. Notera att det måste var just en ”primary key” eller en ”unique key” som den refererar till.

Du kan också se att det finns två alternativ som är streckade. Dessa är non identifierande referenser. Vad detta innebär är att det är ok om de lämnas tomma när ett objekt/en rad skapas.

De hela linjerna är identifierande och när en rad skapas så måste ett värde som är en referens till en annan tabell anges.

## Skapa en databas efter din mall.

När du är nöjd med hur din modell ser ut så kan du skapa en databas efter den. Tryck på fliken ”**database**” och sedan ”**forward enginer**”. Detta kommer då att auto generera ett SQL-script som skapar din databas med allt innehåll enligt modell-filen.

Om du sedan lägger till något i din databas eller har en annan databas vars struktur du vill dela med dig av eller spara så kan du skapa en modellfil utav en existerande databas med raden under ”**reverse enginer**” som du kan hitta under ”forward enginer”.

# Views och stored procedures

Om man har ett sett med kommandon som man kör regelbundet som att vissa viss data eller att ändra på något värde så kan det vara ineffektivt att skriva dem manuellt varje gång de ska användas. Om SQL kommandona också är stora och komplicerade så blir det väldigt tids effektivt. Man skulle kunna spara dem som en egen fil och sedan öpnna filen och köra kommandot men det finns ett smidigare sätt.

”**Views och stored procedures**” (vyer och procedurer) är ett sätt att skapa metoder med unika identifierare likt dem i C# eller andra. Dessa kan spara SQL kommandon som kan köras genom att anropa identifieraren och på så viss så undviker du att behöva skriva repetitiv kod.

## Views

Views är specifikt et SQL ”select” kommando som sparas och anropas. Det ska inte vara något annat som ”create” eller ”alter”.

Det är ett bra sätt att spara kod för data som ska visas ofta.

Vyer används för slipa skriva stora select kommandon som ska användas ofta. Istället för att skriva hela satsen om och om igen så kan vi bara skriva vy-namnet.

Syntax för att skriva en vy:

**Create view <namn> AS**

**<selectstatment>;**

Sedan så använder man ett ”select”-kommando för att anropa vyn som du skapade.

**Select \* from <namn>;**

Exempel, vi har en databas över ett zoo och vi vill se en lista över alla djur plus deras skötare och deras inhägnad. Vi skapar en vy så vi sliper skriva koden varje gång vi vill se all data.

**Create view `V\_zoo`**

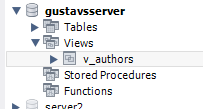
**Select \* from animals**

**Join keepers on keepers.Id = animals.animalKeeperId**

**Join pen on pen.Id = animmals.animalPenId;**

Obs notera accenttecknet ”``” på vy-namnet. Det måste vara en sådan accent på namn utav vyer när du arbetar med MySQL. Du skapar dem genom att trycka shift-´ (bredvid backspace).

När du kör koden som kommer du se att det skapas en ny vy under ”views” fliken i navigeringsfältet för din databas.



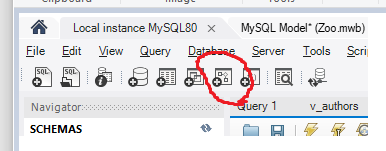
## Stored procedures

Om vyer är enbart select-kommandon som kan sparas så att man slipper att skriva dem om och om igen så är ”**stored procedures**”/sparade procedurer vilken typ utav kommando som helst. Det kan vara, likt vyer, ett kommando för att hämta data. Men det kan också vara kommandon som sorterar, skapar, ändrar, raderar data eller allt på en gång.

Sparade procedurer kan användas för att göra en automatisk handling vid en given tidpunkt varje dag. Kanske för att verifiera en samlings datas integritet. Det är också vanligt att de fungerar som metoder för att ändra på data i en databas kopplad till en applikation. Om en användare vill ändra sitt användarnamn i en applikation så går inte applikationen in i databasen och gör det själv. Den anropar istället en fil med en sparad SQL-procedur som ändrar på data på förfrågning.

### Skapa en Stored Procedure i MySql

För att skapa en lagrad procedur i MySQL workbench så trycker du på knappen ”**create new stored procedure in the active schema in the connected server**”.



Då auto genereras följande kod åt dig:

**CREATE PROCEDURE `new\_procedure` ()**

**BEGIN**

**END**

**”`new\_procedure`”** detta är namnet på proceduren du skapar så se till att bytta den till något passande. I parenteserna så skriver du eventuella parametrar (data som ska skickas in vi användning utav metoden). Sedan skriver du det kommandot som ska göras mellan ”**begin**” och ”**end**”.

Ett exempel som hämtar data från en tabell med personer utav en vis ålder.

**CREATE PROCEDURE `GetAllOfAge` (age int)**

**BEGIN**

**select \* from people**

**where people.Age = age;**

**END**

Tryck sedan på ”apply” så skapas proceduren åt dig. Du kan då hitta den under fliken ”stored procedures” under databasen i navigatorn.

### Skapa en Stored Procedure mauellt, Delimiter

Om du använder MySQL-Workbench för att skapa en procedurs så görs mycket utav jobbet åt dig. Det är dock inte särskilt svårt att göra en egen procedur manuellt.

Innan du kan göra det så behöver du förstå vad en delimiter är.

#### Delimiter

En delimiter är ett tecken som markerar slutet på ett kommando. Du har använt det tidigare och det är ”**;**”(semikolon). Det används när man ska skilja på flera olika kommandon som ska köras. Ex:

**Select \* from people;**

**Select \* from books;**

Ovan så används ”;” för att skilja på de olika kommandona. Om du inte har dem där så blir kompilatorn förvirrad och tror att allt du skriver ska vara ett långt kommando och det blir då syntax fel.

Men vad händer om vi har ett kommando i ett annat kommando, som när vi sparar ett kommando i en procedur som vi skapar.

Om du enbart skriver:

**create procedure GetAllPeople()**

**Begin**

**Select \* from people;**

**end;**

Så kommer du att få kompilerings fel då kompilatorn tror att det är två olika kommandon.

Ett som ser ut så här:

**create procedure GetAllPeople() Begin Select \* from people;**

och det andra:

**end;**

Och det är ju inte vad vi vill göra. För att lösa detta så måste vi ändra vilket tecken ”Delimitern” är.

Detta gör man med nyckelordet ”Delimiter”. Ett exempel där vi behöver använda en delimiter, när vi skapar en procedur manuellt, skulle se ut så här:

**DELIMITER //**  Vi ändrar delimiter/avslutningstecknet från ”**;**” ti ”**//**”

**use gustavsserver//** anger vilken databas som proceduren ska skapas i. Inte nödvändigt.

**CREATE PROCEDURE GetAllProducts()**

**BEGIN**

**SELECT \* FROM books;** Vi kan nu använda ”;” utan att det blir kompilerings fel då kompilatorn inte ser det som ett avslut.

**END //** Här skriver vi vårt nya avslutnings tecken. Kompilatorn tolkar då allt innan detta tecken som ett enda kommando.

**DELIMITER ;** När vi är klara så ändrar vi tillbaka avslutningstecknet till ”;” så att det inte blir något krångel efteråt.

Kodbiten ovan är alltså hur vi skapar procedurer manuellt.

#### Använda en stored procedure

För att använda ett sparat kommando så använder man nyckelordet ”**call**” i MySQL. I andra program så kan det vara ”**exec**” eller ”**execute**”.

# Normaliseringsformeln

När man skapar en databas så är det viktigt att tänka på dess struktur. Man vill ha en databas som är enkel att hantera. Dvs att lägga till, ta bort eller ändra på data. Det är också viktigt att data integriteten är god. Alltså att parad data, som efternamn och förnamn till exempel, inte hamnar på fel plats och refererar till fel objekt. Det ska också inte finnas några dubbletter utav samma data.

För att underlätta med detta så har några standardpraxis för att skapa databaser skrivits som kallas normaliseringsformel.

## Första Normalformen, 1NF

Första normaliserings formeln har följande bestämmelser

* Det får enbart finnas ett värde per cell.
* Det ska inte finnas några repeterade fältvärden.
* Varje rad ska vara unik och ha ett unikt Id-värde.

## Andra normalformen, 2NF

Andra normalformen har följande bestämmelser:

* Tabellen måste uppfylla första normaliserings formen.
* Fält som inte är primärnycklar måste vara beroende utav alla primärnycklar.

Om du till exempel har en tabell utav en kundorder:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Date | Item | Total Price | Address | Maunfacturer |
| Gustav | 210822 | Dator | 1000 | Linköpingsvägen1 | Dell |

Det är namnet på kunden (Gustav), datumet på ordern (210822), priset och innehållet i ordern (dator) som gör ordern/raden unik. För att uppnå andra normaliserings formen så måste alla rader som inte är dessa, vara beroende utav allihop. Annars så ska det stå i en separat tabell.

Adressfältet är enbart beroende utav namnet på kunden. Det har inget med resten utav ordern att göra. Lika så är tillverkaren (Manufacturer) enbart beroende utav produkten (Item). Dessa värden skulle således vara bättre placerade i två egna tabeller.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Date | Item | Price |
| Gustav | 200822 | Computer | 200 |
|  |  |  |  |
|  |  |  |  |
| Name | Adress |  |  |
| Gustav | Linköping |  |  |
|  |  |  |  |
| Item | Manufacturer |  |  |
| Computer | Dell |  |  |

## Tredje Normalformen, 3NF

* Tabellen måste uppfylla andra normaliserings formen.
* Et värde i en tabell får inte vara beroende utav ett annat värde som inte är en huvudnyckel.

Som exempel så har jag expanderat produkt (item) med lite mer info. Här är tillverkaren beroende utav produkten. Och adressen är i sin tur beroende utav tillverkaren.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Item | Manufacturer | Details | Cost | Adress |
| Computer | Dell | White | 1000 | Norköpingsvägen1 |
|  |  |  |  |  |

Detta är inte enligt tredje normalformen för att lösa detta så placerar vi adressen i en annan tabell.

|  |  |  |  |
| --- | --- | --- | --- |
| Item | Manufacturer | Details | Cost |
| Computer | Dell | White | 1000 |
|  |  |  |  |
|  |  |  |  |
| Manufacturer | Adress |  |  |
| Dell | Norköpingsvägen1 |  |  |

## Boyce-Codd Normalform,BCNF

BCNF är en variant utav tredje formen och den säger att alla värden ska vara referenser till huvudnycklar i andra tabeller. Så du får inte skapa några egna rader utan du måste enbart använda referenser.

## Att inte normalisera

Normaliserings reglerna är skrivna främst för att se till att man inte ska få några dubbletter i sin databas. Men ibland så kan man argumentera att vissa data bör finnas i samma tabell även om det strider mot normalisering. Vidare om man ska följa normaliserings formerna till punkt och pricka så kan det skapa en situation där du har väldigt många tabeller. Att söka igenom en databas med många tabeller tar också längre tid så det kan påverka prestanda om du lägger för stor vikt på att följa normal formlerna.

Använd sunt förnuft. En tabell med personers adresser där namn, gatuadress, postnummer och stad står med, bryter kanske mot tredje formen men det är ändå en vettig lösning.

# JSON och NoSQL

JSON, JavaScriptObjektNotation, är ett format för att hantera data i javascript Data som sparas i JSON skriv på ett sådant sätt att det är självbeskrivande. Vanligt vis så sparas data i textfiler där data är sparade likt ett Diktionary i key/value par. Dvs att du har två värden per rad. Ett som används för att identifiera (key) det andra (value).

När man använder JSON istället för SQL så hämtas inte lite data i taget per förfrågningar utan all data hämtas på en gång och det filtreras sedan i JavaScript.

NoSQL är ett sätt att spara data på, då man använder sig utav andra sätt än relaterade databaser som SQL-tabeller. Ofta så sparas denna data i textfiler och det är även vanligt att det sparas i JSON format eller i XML. Dessa textfiler kan också innehålla kommentarer, nummeriska värden och funktioner

JSON kan även liknas med XML. De är båda självbeskrivande, hierarkiska och kan avläsas utav många program men JSON anses bättre då:

* det inte behövs några avslutstaggar för varje fält
* det är kortare och snabbare att läsa/skriva
* det går att använda array:er
* det finns inbyggda funktioner i JavaScript för att behandla data (parsa/konvertera etc) medans XML behöver externa funktioner.

Föra att skriva data i JSON format så ska man skriva dem som objekt där varje fält har två värden. En nyckel som används som identifierare/namn/nyckel på fältet och ett som anger dess värde. Identifieraren skrivs som ett string värde och det värdet fältet (tror de brukar kallas egenskaper/properties i javascript ibland) ska hålla skrivs som vanligt.

Ett objekt:

**Let json = {**

**”firstname”:”Gustav”,**

**”lastname””Höglund”**

**”kurser”:[ ”HTML”, ”Programmering grund”, ”Molnbaserad Arkitektur”]**

**};**

En array med objekt:

**Let json2 = [**

**{**

**”color” : ”blue”,**

**”weight” : 200**

**”shape” : ”round”**

**},**

**{**

**”color” : ”red”,**

**”weight” : 100,**

**”shape” : ”oval”**

**}**

**];**

Json tillåter följande datatyper:

* string (måste skrivas med **”**, inte ’)
* nummer (int, float etc)
* objekt
* array
* boolean
* null

Funktioner, datum och undefined är inte tillåtet.

För att konvertera ett JSON formaterat strängvärde till ett javascript objektanvänder du ”**Json.Parse(<Jsonsträng>)**”.

**Var jsonString = ’{”color”:”blue”, ”shape”:”round”}’**

**Var objekt2 = Json.parse(jsonString);**

För att skriva ut objektegenskaper:

**Document.write(objekt.firstname);**

**Document.Write(objekt2.color);**

Konvertera JSON objekt till ett JSON formaterat strängvärde:

**Var jsonObjekt = {**

**Color : ”blue”,**

**Shape : ”Square”,**

**Nummber : ”9”,**

**Size : ”large”**

**};**

**Var string = JSON.stringify(jsonObjekt);**

**Document.write(string);**

Om du vill rader fält i ett JSON objekt så kan du använda nyckelordet delete.

Ex

**Delete jsonObjekt.Color;**

Json filer brukar vara sparade i webbervers online och hämtas med serverscript.

# Microsoft SQL

## Skapa databas

För att skapa en databas i Microsoft SQL så är det enklaste att högertrycka på ”databases” till höger och välja ”**new database**”.

Du kommer då att få fram en ruta där du kan namnge databasen och ange hur mycket minne du vill tilldela den. Du kan också se databasens lokala sökväg.

När du skapat en databas så kan du expandera den för att se allt dess innehåll i navigatorn till vänster.

## SQL i MicrosoftSQL

För att få fram en textruta att skriva SQL-script i så klickar du på ”**new query**”.

För att välja vilken databas scripten körs i så har du en ”dropdown-flik” precis över navigatorn där du väljer den aktiva databasen.

Om du vill skapa en tabell så skriver du:

**create table employees (**

**Id\_Employees int,**

**FirstName\_Employees varchar(50),**

**LastName\_Employees varchar(50),**

**Age\_Employees int,**

**Gender**\_Employees varchar(10)

**)**

Du behöver inte skriva några delimitertecken “**;**” I Microsoft sql.

För att köra ett script så klickar du på execute eller F5.

Om du vill skapa en tabell med ett grafiskt verktyg så högerklickar du på ”**tables**” i navigatorn.

Du får då fram en ruta där du kan skapa de fälten du vill ha.

Om du vill kunna ändra på en skapad tabell så kan du högertrycka på den och välja ”**design**”.’

För att ange ett fält som primärnyckel så högertrycker du på det fältet och väljer ”**set primary key**”.

### Auto increment

För att skapa ett auto increment så kan du gå in i design-läget i en tabell. Välj fältet du vill auto incrementera. Se den nedre rutan “**Column properties**”. Scrolla ned till “**Identity specifikation**”, öppna tabben och tryck på “**(is identity)**”. Du ser nedan hur mycket värdet incrementeras per element **“identity increment”** och vart den börjar “**identity seed**”.

Du kan också göra det när du skapar tabellen och des kolumner med “**Identity()**”.

**create table test (**

**Id int identity(1,1)** det vänstra värdet i parenteserna är “seed” (vart den börjar) och den högra är incrementet (ökningen).

**\_Name varchar**

**)**

## Stoppa in värden

**insert into employees**

**values(**

**1,'Gustav', 'Höglund', 25, 'Male'**

**);**

Ovan så har jag inte angett vilka fält som värdena ska tilldelas. Om du inte skriver något så antas det att du ska skriva in ett värde för varje fält. Det är då viktigt att du ser till att skriva in fälten I korrekt ordning.

Om du vill stoppa in flera rader på en gång så kan du skriva rader inom parenteser och separera dem med ett kommatecken.

Ex:

**insert into employees**

**values**

**(1,'Gustav', 'Höglund', 25, 'Male'),**

**(2, ‘Sara’, ‘Sarasson’, 45, ‘Female’)**

**;**

Om du får felmeddelande på namnet på databasen:

“**invalid object name**” men du har skapat databasen så är det intelisense som inte är uppdaterad.

Tryck på **“edit”** uppe till höger.

Välj

**Intelisense -> refresh local cache.**

Eller tryck **ctrl** + **shift** + **R**

## Lägga till columner/fält

För att lägga till kolumner kan du antagligen använda det grafiska verktyget genom att högertrycka på tabellen och välja “**design**”.

Om du vill göra det med kod så gör du så här:

**alter table test**

**add**

**\_name varchar(45),**

**price decimal;**

Alltså:

**Alter table <tabellnamn>**

**Add**

**<kolumnnamn> <datatyp>,**

**<kolumnnamn> <datatyp>,**

**<kolumnnamn> <datatyp>;**