# SQL-CRUD

En databas är ett program som lagrar ”data”, alltså information i en fil. Ungefär på samma sätt som Excel men med lite smartare funktioner för sökning och indexering.

## CONTENTS

# SQL – HEROES

## DISCLAIMER

Dokumentet är baserad på MySQL men har kommentarer om hur man gör med SQL-Servern också.
MySQL dialekten borde fungera bra även på SQLite och andra databaser.

## CRUDL

CRUD kallas ibland för CRUDL för att man vill kunna se listan på alla rader i tabellen och inte bara en i taget…

| | |
|---|---|
| **C**reate | `INSERT INTO (fält) VALUES (värden),(värden);` |
| **R**ead | `SELECT fält FROM tabell WHERE villkor;` |
| **U**pdate | `UPDATE Tabell SET fält=värde, fält2=värde2;` |
| **D**elete | `DELETE FROM tabell WHERE fält = värde;` |
| **L**ist | `SELECT fält FROM tabell` |

Tänk på att text och strängar ska skrivas med enkelfnuttar ’ ’ och siffor utan fnuttar.

## SKAPA DATABAS

```
CREATE DATABASE DCHeroes;
USE DCHeroes;
```

Man använder USE kommandot för att tala om för scriptet att i fortsättningen ska den köra med den valda databasen.

### SKAPA DEN BARA OM DEN INTE FINNS

För att vara säker på att databasen skapas bara om den inte finns redan kan du skriva

```
CREATE DATABASE IF NOT EXISTS DCHeroes;
```

### SKAPA OM DATABASEN

Ibland vill man vara säker på att allt är nollställt, exempelvis i samband med demonstration av ett projekt eller inlämning *hint, hint* 😊 😊
Då kan man kolla om databasen finns, och i så fall droppa (radera) den, sen skapar en ny fräsch databas.

```
DROP DATABASE IF EXISTS DCHeroes;
CREATE DATABASE DCHeroes;
USE DCHeroes;
```

### TA BORT EN DATABAS

Vill du radera alla spår av din databas, så kan du enkelt skriva

```
DROP DATABASE IF EXISTS DCHeroes;
```

### VÄLJ DATABAS

Efter att ha  skapat en databas, glöm inte att tala om för din Query att du vill använda det. Detta behöver du inte tänka på om du angett databasnamnet vid connection definitionen innan du kopplade in dig till databasen. Men det är bra att använda om man kör en massa queries på samma gång.

## VISA VILKA DATABASER SOM FINNS I SERVER

Det gör man helt enkelt med kommandot

```
SHOW DATABASES;
```

SQL-SERVER VERSION

```
SELECT name, database_id, create_date FROM sys.databases;
```

### VISA VILKA TABELLER SOM FINNS I DATABASEN

Det gör man helt enkelt med kommandot

```
SHOW TABLES;
```

SQL-SERVER VERSION

```
select schema_name(t.schema_id) as schema_name,
       t.name as table_name,
       t.create_date,
       t.modify_date
from sys.tables t
order by schema_name,
         table_name;
```

## SKAPA TABELLER

Primary Key = huvudnyckel för tabellen, auto_increment betyder att räknaren kommer att ökas med ett varje gång en ny rad läggs till i databasen. Även här kan du använda IF NOT EXIST för att kontrollera att du inte försöker skapa en tabell som redan finns.

```
CREATE TABLE IF NOT EXISTS Heroes (
    heroId INTEGER PRIMARY KEY auto_increment,
    name VARCHAR(50) NOT NULL,
    lastName VARCHAR(50) NOT NULL,
    age INTEGER,
    email VARCHAR(50),
    phone VARCHAR(50)
);

CREATE TABLE IF NOT EXIST Pets(
    petId INTEGER PRIMARY KEY auto_increment,
    pet VARCHAR(50)
);
```

## SQL-SERVER VERSION

Sql-Server gillar inte att man skapar tabeller med IF NOT EXISTS, istället får man göra såhär

```
IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='Heroes' and xtype='U')
CREATE TABLE Heroes(
    heroId INTEGER PRIMARY KEY IDENTITY(1,1) NOT NULL,
    name VARCHAR(50) NOT NULL,
    lastName VARCHAR(50) NOT NULL,
    age INTEGER,
    email VARCHAR(50),
    phone VARCHAR(50)
);
GO
```

## SQL-SERVER CREATE TABLE

Microsoft har sin egen dialekt, så vår Create table blir såhär istället

Identity(1,1) betyder att den börjar räkna på 1 och ökar med 1 för varje ny rad

```
CREATE TABLE Heroes(
    heroId INTEGER PRIMARY KEY IDENTITY(1,1) NOT NULL,
    name VARCHAR(50) NOT NULL,
    lastName VARCHAR(50) NOT NULL,
    age INTEGER,
    email VARCHAR(50),
    phone VARCHAR(50)
);

CREATE TABLE Pets(
    petId INTEGER PRIMARY KEY IDENTITY(1,1) NOT NULL,
    pet VARCHAR(50),
);
```

## KOPPLINGSTABELL

En kopplingstabell är en tabell som kopplar ihop två eller flera andra tabeller, i sista raderna anger vi vilket fält som ska kopplas till vilken tabell som Foreign Key.

```
CREATE TABLE IF NOT EXISTS Owner(
    ownerId INTEGER PRIMARY KEY auto_increment,
    heroId INTEGER,
    petId INTEGER,
    FOREIGN KEY (heroId) REFERENCES Heroes(heroId),
    FOREIGN KEY (petId) REFERENCES Pets(petId)
);
```

### SQL-SERVER VERSION

```
CREATE TABLE Owner(
    ownerId INTEGER PRIMARY KEY IDENTITY(1,1) NOT NULL,
    heroId INTEGER,
    petId INTEGER
       CONSTRAINT fk_hero FOREIGN KEY (heroId)
    REFERENCES Heroes(heroId),
       CONSTRAINT fk_pet FOREIGN KEY (petId)
    REFERENCES Pets(petId),
);
```

### FOREIGN KEY

Med raden `FOREIGN KEY (heroId) REFERENCES Heroes(heroId)` talar vi om för databasen att tabellen ska ha hänvisning till tabellen Heroes och kolumnen heroId. Detta gör att vi inte kan radera hunden från Pets listan utan att först ha raderat kopplingen till Clark Kent. En sådan regel kallas **Restriction**. Restriction används för att hindra databasanvändarna från att göra dumma saker. Denna restriction gör också att vi inte kan radera tabellen Pets eller Heroes så länge det finns kopplingar i Owner tabellen. Vi kan inte heller radera rader i Heroes eller Pets som är kopplade via sin Primary Key.

### RADERA FÖRST, SKAPA SEN

Du kan använda samma trick här som innan du skapade databasen

```
DROP TABLE IF EXISTS Heroes;
```

Och sedan köra Create Table.

### ÄNDRA I EN TABELL

För att ändra en tabells uppbyggnad skriver du

### LÄGG TILL KOLUMN

```
ALTER TABLE tabellNamn ADD kolumn datatyp;
```

### TA BORT KOLUMN

```
ALTER TABLE tabellNamn DROP COLUMN kolumn;
```

### ÄNDRA TYP AV KOLUMN

MySQL:

```
ALTER TABLE tabellNamn MODIFY COLUMN kolumn datatyp;
```

SQL Server:

```
ALTER TABLE tabellNamn ALTER COLUMN kolumn datatyp;
```

## INMATNING

Mata in en hjälte

```
INSERT INTO Heroes (name, lastName, age, email, phone)
VALUES
   ('Clark','Kent',42, 'clark.kent@dailymirror.com','5555-1212-3434');
```

Insert Into lägger in en rad i tabellen.

### INMATNING AV FLERA PÅ SAMMA GÅNG

```
INSERT INTO Heroes (name, lastName, age, email, phone)
VALUES
   ('Bruce','Wayne',42, 'bruce@waynecorp.com','5555-4567-1212'),
   ('Celina','Kyle',36, 'celina@meow.org','5555-4242-1331'),
   ('Victor','Stone',35, 'cyborg@rus.com','5555-8888-8888');
```

Mata in djur

```
INSERT INTO Pets (pet)
VALUES
       ('cat'), ('dog'), ('bat'), ('dolphin'), ('dragon');
```

Koppla djuren till sina respektive hjältar

```
INSERT INTO Owner (heroId, petId)
VALUES (1,2), (2,3),(3,1);
```

## SELECT - VISA LISTAN PÅ HJÄLTAR

```
SELECT * from Heroes;
```

### VISA LISTAN PÅ DJUR

```
SELECT * FROM Pets;
```

### VISA LISTAN PÅ ÄGARE

```
SELECT * FROM Owner;
```

SELECT Page WHERE Id = 7;

## KOPPLA IHOP INFORMATIONEN FRÅN ALLA TRE TABELLER

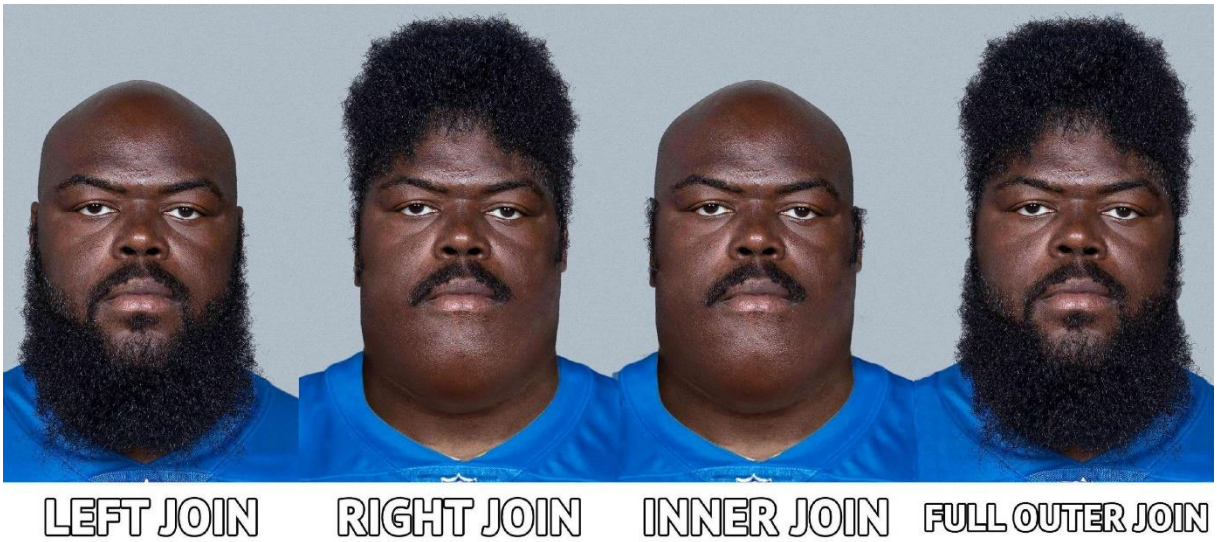
För att kunna se hur tabellerna fungerar tillsammans får vi koppla ihop dem på detta sätt. I SELECT raden talar vi om vilka tabeller som är inblandade. I WHERE raderna förklarar vi för databasen hur den ska koppla ihop informationen.

```
SELECT name, lastName, pet from Heroes, Pets, Owner
WHERE
      Owner.heroId = Heroes.heroId AND
      Owner.petId = Pets.petId;
```

| | name | lastName | pet |
|---|---|---|---|
| 1 | Clark | Kent | dog |
| 2 | Bruce | Wayne | bat |
| 3 | Celina | Kyle | cat |

Som du kan se i resultatet så försvinner Victor från vår lista. Detta på grund av att vi frågade specifikt efter alla hjältar som är kopplade till ett husdjur. Victor har ingen, alltså är han inte med i sökningen. Detta kallas för **Inner Join**. Man skriver sällan filtren på det sätt som vi skrev ovan, det är snyggare och effektivare att använda sig av Joins.

Det finns fyra olika joins som vi ska kolla på.

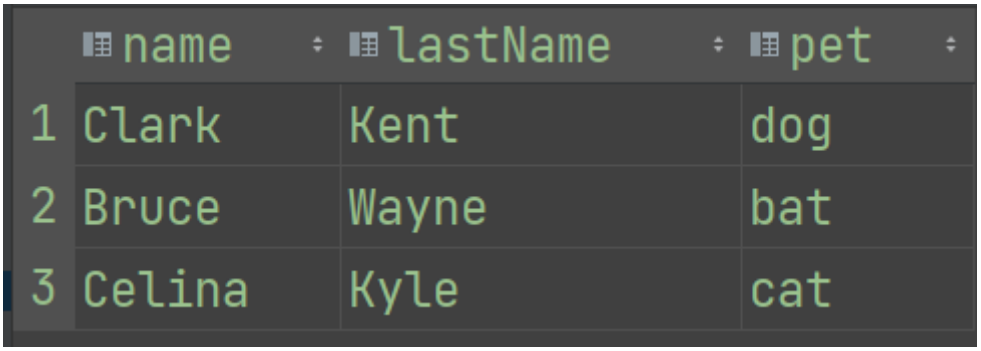


## INNER JOIN

Inner join väljer alla som matchar exakt på båda sidorna om frågan. Då vi har tre tabeller får vi skapa en **INNER JOIN** från första tabellen till mittentabellen, sen från andra tabellen till mittentabellen.

```
SELECT Heroes.name, Heroes.lastName, Pets.pet
FROM   Heroes
    INNER JOIN Owner ON Heroes.heroId = Owner.heroId
    INNER JOIN Pets ON Owner.petId = Pets.petId
```

Första INNER JOIN kopplar ihop Heores med Owner och plockar fram alla matchingar där, därefter tar den andra INNER JOIN vid och kopplar ihop resultatet med Pets tabellen. Sen presenteras allt till användaren.



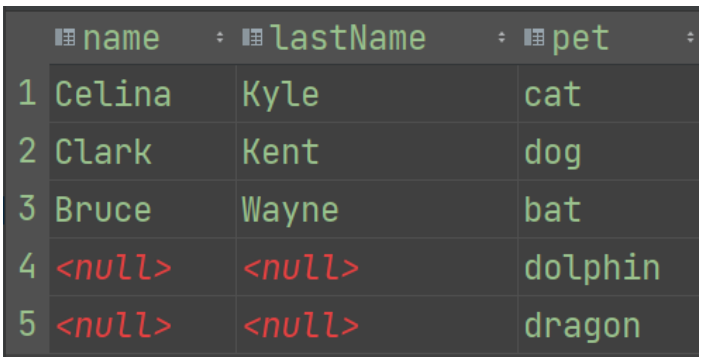
| | name | lastName | pet |
|---|---|---|---|
| 1 | Clark | Kent | dog |
| 2 | Bruce | Wayne | bat |
| 3 | Celina | Kyle | cat |

## RIGHT JOIN

Right Join kopplar ihop tabellerna men tar också med allt i den högra kolumnen.

```
SELECT Heroes.name, Heroes.lastName, Pets.pet
FROM   Heroes
    RIGHT JOIN Owner ON Heroes.heroId = Owner.heroId
    RIGHT JOIN Pets ON Owner.petId = Pets.petId
```

Så vi får en lista på alla djur och alla ägare, om ingen ägare finns så får vi NULL i ägarens plats.
(Kanske inte det man förväntade sig, men det är all-right 😊)



| | name | lastName | pet |
|---|---|---|---|
| 1 | Celina | Kyle | cat |
| 2 | Clark | Kent | dog |
| 3 | Bruce | Wayne | bat |
| 4 | *<null>* | *<null>* | dolphin |
| 5 | *<null>* | *<null>* | dragon |

## LEFT JOIN

Left Join kopplar ihop tabellerna men tar också med allt i den vänstra kolumnen.

```
SELECT Heroes.name, Heroes.lastName, Pets.pet
FROM   Heroes
    LEFT JOIN Owner ON Heroes.heroId = Owner.heroId
    LEFT JOIN Pets ON Owner.petId = Pets.petId
```

Så vi får en lista på alla ägare och dess djur, om inget djur finns så får vi NULL i djurets plats.

| | name | lastName | pet |
|---|---|---|---|
| 1 | Clark | Kent | dog |
| 2 | Bruce | Wayne | bat |
| 3 | Celina | Kyle | cat |
| 4 | Victor | Stone | <null> |

## FULL OUTER JOIN

Knasigt nog så får jag inte detta till att fungera i MySQL även om det står i Mysql-manualen att det ska funka, men det funkar i SQL-Server.

| | name | lastName | pet |
|---|---|---|---|
| 1 | Clark | Kent | dog |
| 2 | Bruce | Wayne | bat |
| 3 | Celina | Kyle | cat |
| 4 | Victor | Stone | NULL |
| 5 | NULL | NULL | dolphin |
| 6 | NULL | NULL | dragon |

```
SELECT Heroes.name, Heroes.lastName, Pets.pet
FROM   Heroes
    FULL JOIN Owner ON Heroes.heroId = Owner.heroId
    FULL JOIN Pets ON Owner.petId = Pets.petId;
```

Detta JOIN tar alla poster som matchar på båda sidorna, plus alla som inte matchar. Varför man nu skulle vilja ha den listan - det är en bra fråga.

## TRANSACTIONS – SKYDD MOT TABBAR

En transaktion skyddar databasen från felaktiga inmatningar eller raderingar.

```
START TRANSACTION;
DELETE FROM Heroes WHERE heroId>0;
SELECT * from Heroes;
```

---

MYSQL-VERSION

```
BEGIN TRANSACTION;
DELETE FROM Heroes WHERE heroId>0;
SELECT * from Heroes;
```

### ROLLBACK ÅTERSTÄLLER ALLT

```
ROLLBACK;
```

### COMMIT SPARAR ÄNDRINGAR

```
COMMIT;
```

## UPDATE – ÄNDRA DATA I TABELLEN

Har man råkat mata in något fel kan man uppdatera det lätt och enkelt

```
UPDATE Heroes
SET
      email = 'MrCyborg@cyborg.rus'
WHERE email = 'cyborg@rus.com'
```

## ORDER BY - SORTERING

Man kan sortera resultatet av sin sökning direkt i frågan.

### SORTERA PÅ NAMN

```
SELECT name, lastName from Heroes ORDER BY name;
```

### SORTERA PÅ EFTERNAMN

```
SELECT name, lastName from Heroes ORDER BY lastName;
```

### SORTERA PÅ ÅLDER

```
SELECT name, lastName from Heroes ORDER BY age;
```

### DESC - SORTERA PÅ ÅLDER I OMVÄND ORDNING (ÄLDST FÖRST)

```
SELECT name, lastName from Heroes ORDER BY age DESC;
```

### DESC - SORTERA PÅ ÅLDER I OMVÄND ORDNING (YNGST FÖRST)

```
SELECT name, lastName from Heroes ORDER BY age ASC;
```

ASC behöver dock inte skrivas, för det är standardvalet.

## COUNT - TA REDA PÅ ANTAL HJÄLTAR

```
SELECT COUNT (heroId) FROM Heroes;
```

## DISTINCT - TA REDA PÅ ANTAL ÅLDERSGRUPPER

Distinct ser till att vi aldrig får dubbletter i sökningarna.

```
SELECT COUNT (DISTINCT age) FROM Heroes;
```

## TA REDA PÅ HUR MÅNGA SOM HETER BRUCE

(Det kan vara Bruce Banner och Bruce Wayne)

```
SELECT COUNT (heroId) FROM Heroes Where Name='Bruce';
```

## TA REDA PÅ ANTAL UNIKA NAMN I LISTAN

Nu räknar vi alla Bruce som en, oavsett hur många det är

```
SELECT COUNT(DISTINCT name) FROM Heroes;
```

## FLER HJÄLTAR ATT LEKA MED

```
INSERT INTO Heroes (name, lastName, age, email, phone)
VALUES
('Diana', 'Prince', 28, 'diana@amazon.com', '555-555-5552'),
('Peter', 'Parker', 28, 'peter@dailybugle.com', '555-155-5155'),
('Bruce', 'Banner', 28, 'bruce@culvertUni.org', '555-545-5755'),
('Selina', 'Kyle', 25, 'selina@meow.org', '555-575-5559'),
('Wade', 'Wilson',  30, 'dead@pool.org', '555-585-1555'),
('John', 'Wick', 30, 'babayaga@contineltal.org', '555-755-5254'),
('Arthur', 'Curry', 42, 'arthur@atlante.an', '555-545-5255'),
('Barry', 'Allen', 21, 'barry.allen@centralcity.pd', '555-558-8888'),
('James', 'Gordon', 45, 'james.gorgon@gothamcity.pd', '555-565-5655'),
('Alfred', 'Pennyworth', 52, 'alfred.pennyworth@waymansion.com', '555-535-
3555'),
('Amanda', 'Waller', 43, 'boss@argus.org', '555-455-5554');
```

## DELETE FROM - RADERA

Efter att ha matat in alla så ser vi att några av dem inte tillhör DC utan snarare Marvel. Så vi får ta bort dem ur listan. Man kan antingen ta bort dem genom att söka på deras namn, men det är inte alltid bra att göra så, då det kan råka finnas flera personer som heter så. Om vi ska ta bort Hulk så skulle vi kunna radera Bruce

Bruce Banner, Peter Parker, Wade Wilson och John Wick hör inte hemma i DC världen.

Vi plockar fram en lista på dem:

```
SELECT heroId, name, lastName FROM Heroes Where lastName
IN('Parker','Banner','Wick','Wilson');
```

| | heroId | name | lastName |
|---|---|---|---|
| 1 | 6 | Peter | Parker |
| 2 | 7 | Bruce | Banner |
| 3 | 9 | Wade | Wilson |
| 4 | 10 | John | Wick |

Dem kan du radera med gott samvete.

Kör inte: DELETE FROM Heroes WHERE name='Bruce';

Det skulle även ta Bruce Wayne, alltså Batman och det får inte ske. Vi gör snarare så att vi letar upp ID på den hjälten vi vill radera från listan ovan.

Vi kan ta bort Spindelmannen genom att skriva:

```
DELETE FROM Heroes WHERE heroId=6;
```

Men vi har fler personer att radera, för att slippa en massa DELETE rader kan vi skriva

```
DELETE FROM Heroes Where heroId IN(6,7,9,10);
```

Vi kan även använda NOT IN eller BETWEEN för att radera hur vi vill, på exakt samma sätt som vi söker.

Då vi raderar baserat på indexet så är vi 100% säkra på att rätt rad raderas.

## MER SÖKNING

Som så såg kan man använda specialord för sökning.

- `SELECT * FROM Heroes Where HeroId IN (1,2,3,4,5)`
- `SELECT * FROM Heroes Where HeroId NOT IN (1,3,3,7)`
- `SELECT * FROM Heroes Where HeroId BETWEEN 1 AND 5`

## UTMANING!

När vi nu kollar på listan ser vi att vi har två Cat woman, men den senare har namnet rätt stavat. Ta bort den senare Selina och ändra första till att heta Selina.

Du ska använda SELECT, DELETE och UPDATE.

## BRA LÄNKAR

- https://www.mysqltutorial.org/
- https://www.w3schools.com/mysql/mysql_sql.asp
- https://sqlzoo.net/wiki/SQL_Tutorial