Mini guide til SQL

Indhold

Opret Databasen

```
CREATE DATABASE POSTDANMARK;
```

Brug Database navn

```
USE DATABASE databasename;
```

Slet Database

```
DROP DATABASE databasename;
```

Opret Tabel

```
CREATE TABLE POST(
 L_ID int IDENTITY(1,1) NOT NULL,
 Land varchar(500) NOT NULL,
 By_ID varchar(500) NOT NULL,
 Postnummer varchar(500) NOT NULL
 )ON [PRIMARY]
```

Slet Tabel

DELETE-sætningen bruges til at slette eksisterende data i en tabel:

```
DELETE FROM table_navn
WHERE condition;
```

Select statement

Vis alt i Tabel POST – (Select statement)

```
SELECT * FROM POST
```

Vis kun By og Postnummer – (Select statement)

```
SELECT By_ID, Postnummer
FROM POST
```

Læs mere i bogen "Introduktion til SQL" på side 12 (Filetering af data) og side 19 (Avanceret filtering)

Hvad betyder NULL?

I databaser er et fælles problem, hvilken værdi eller stedholder du bruger til at repræsentere manglende værdier. I SQL løses dette med NULL. Det bruges til at betegne manglende eller ukendte værdier.

Nøgleordet NULL bruges til at angive disse værdier. NULL er virkelig ikke en bestemt værdi, så meget som det er en indikator. Tænk ikke på NULL som ligner nul eller tomt, det er ikke det samme. NULL (0) og tom er værdier.

NULL Værdi i Sammenligninger:

Når det ikke er muligt at kode dine data specielt med "N/A", kan du bruge det specielle nøgleord NULL til at angive en manglende værdi. NULL er vanskelig. NULL er ikke en værdi i normal forstand. For eksempel er ingen to NULL lig med hinanden. Overraskende NULL = NULL er FALSK!

SQL dækker dog dette dilemma. Du kan bruge IS NULL og IKKE NULL sammenligninger til at teste for NULL værdier.

Eksempel med værdi NULL:

```
SELECT By_ID, Postnummer
FROM POST
WHERE Postnummer IS NULL
```

Læs mere i bogen "Introduktion til SQL" på side 18 (NULL-værdier)

INSERT INTO Syntaks

Det er muligt to skrive INSERT INTO-sætning på to måder.

Den første måde angiver både kolonnenavne og de værdier, der skal indsættes:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Hvis du tilføjer værdier for alle kolonner i tabellen, er det ikke nødvendigt at angive kolonnenavne i SQL-forespørgsel. Men sørg for rækkefølgen af værdierne er i samme rækkefølge som kolonnerne i tabellen. INSERT INTO syntaks ville være som følger:

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

SQL UPDATE Statement

Den UPDATE-sætningen bruges til at ændre de eksisterende poster i en tabel:

```
UPDATE table_name
SET column1 = value1, column2 = value2, …
WHERE condition;
```

SQL JOIN

En JOIN bestemmelse anvendes til at kombinere rækker fra to eller flere tabeller, baseret på beslægtet kolonne mellem dem.

Bemærk, at "CustomerID" kolonnen i "Ordrer" tabel refererer til den "CustomerID" i "Customers Tabelle" bord. Relationer mellem de to tabeller ovenfor er "CustomerID" kolonnen.

Så kan vi skabe følgende SQL-sætning (der indeholder en INNER JOIN), der udvælger poster, der har matchende værdier i begge tabeller:

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

Forskellige typer af SQL JOIN´s

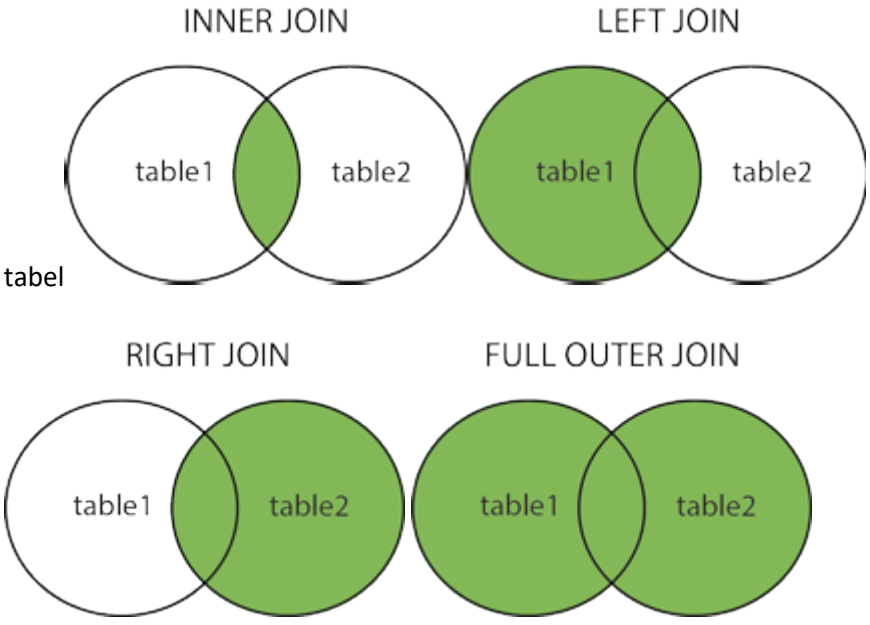
Her er de forskellige typer af samlingerne i SQL:

(INNER) JOIN: Returnerer poster, der har værdier i begge tabeller

LEFT (OUTER) JOIN: Return alle poster fra venstre tabel, og de matchede optegnelser fra det højre tabel

RIGHT (OUTER) JOIN: Return alle poster fra den højre tabel, og de matchede optegnelser fra den venstre tabel

FULL (OUTER) JOIN: Return alle relationer, fra alle værdier fra alle tabelleder i enten venstre eller højre tabel



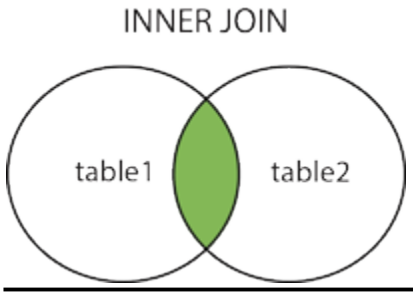
Læs mere i bogen ”Introduktion til SQL” på side 84 (Join-forbindelser)

SQL INNER JOIN Søgeord

Den indre sammenkædning søgeord udvælger poster, der har matchende værdier i begge tabeller.

Syntax for INNER JOIN:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2 ON table1.column_name = table2.column_name;
```

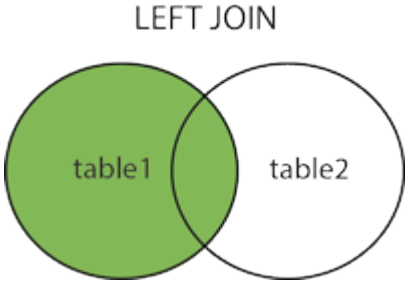


SQL LEFT JOIN Søgeord

Den LEFT JOIN søgeord returnerer alle poster fra venstre tabel (Tabel 1), og de matchede optegnelser fra den højre tabel (Tabel2). Resultatet er NULL fra højre side, når der ikke er match..

Syntax for LEFT JOIN:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2 ON table1.column_name = table2.column_name;
```

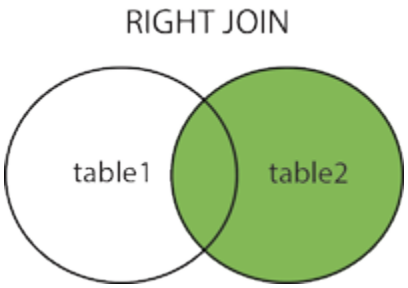


SQL RIGHT JOIN Søgeord

Den RIGHT JOIN søgeord returnerer alle poster fra den højre tabel (Tabel 2), og de matchede poster fra venstre tabel (Tabel 1). Resultatet er NULL fra venstre side, når der ikke er match.

Syntax for RIGHT JOIN:

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2 ON table1.column_name = table2.column_name;
```



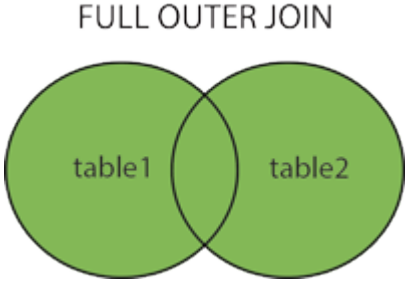
SQL FULL OUTER JOIN Søgeord

Den FULL OUTER JOIN søgeord giver alle relationer, når der er et match i mellem højre og venstre tabel.

Bemærk: FULL OUTER JOIN potentielt kan vende tilbage meget store resultatsæt-sæt!

Syntax for FULL OUTER JOIN:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2 ON table1.column_name = table2.column_name;
```



SQL GROUP BY Statement

Den GROUP BY erklæring bruges ofte med samlede funktioner (COUNT, MAX, MIN, SUM, AVG) til gruppere resultat-sæt af en eller flere kolonner. Grupperne + Sortering = ORDER BY

Syntax for GROUP BY:

```
SELECT column_name(s)
FROM  table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

SQL (PRIMARY KEY) Primærnøgle

En primær nøgle, også kaldet en primær søgeord, er en nøgle i en relationel database, der er unik for hver post. Det er en unik identifikator, såsom et kørekort, telefonnummer (inklusive områdekode) eller køretøjets identifikationsnummer (VIN). En relationel database skal altid have én og kun én primær nøgle. Primære nøgler forekommer typisk som kolonner i relationelle database tabeller.

PRIMARY KEY skal indeholde unikke værdier, og kan ikke indeholde NULL-værdier.

En tabel kan kun have én PRIMARY KEY, der kan bestå af enkelte eller flere felter.

Eksempel på syntax med PRIMARY KEY:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age  int,
    PRIMARY KEY (ID)
);
```

Se video om PRIMARY KEY:

https://www.lynda.com/SQL-Server-tutorials/Understanding-composite-primary-keys/385694/443444-4.html

---

SQL (FOREIGN KEY) fremmednøglen

I forbindelse med relationelle databaser, FOREIGN KEY også kendt som fremmed nøgle er et felt (eller en samling af felter) i én tabel, som entydigt identificerer en række anden tabel eller den samme tabel.

I enklere ord, er den fremmede nøgle (FOREIGN KEY) er defineret i en anden tabel, men det henviser til den primære nøgle i den første tabel.

For eksempel kan en tabel kaldet Medarbejder har en primær nøgle kaldet employee_id. En anden tabel kaldet Medarbejderdetaljer har en fremmed nøgle som refererer employee_id for at entydigt at identificere forholdet mellem de to tabeller.

En FOREIGN KEY i en tabel peger på en primær nøgle i en anden tabel.

Example på syntax med FOREIGN KEY:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

Se video om FOREIGN KEY:

https://www.lynda.com/MySQL-tutorials/Understanding-foreign-key-%20constraints/418255/441669-4.html

---

Se også videoen om Database Design 30 - Simple Key, Composite Key, Compound Key for at får at få en grund forståles for PRIMARY KEY og FOREIGN KEY

Video link: https://www.youtube.com/watch?v=vsGDtnBCwgg

Normalisering af Database

Database Normalisering er en teknik til at organisere dataene i databasen. Normalisering er en systematisk tilgang til dekomponerende tabeller for at eliminere dataredundans og uønskede egenskaber som Insertion, Update og Deletion Anamolies. Det er en proces i flere trin, der sætter data i tabular form ved at fjerne dublet data fra relationstabellerne.

Normalisering anvendes til hovedsagelig to formål:

1. Eliminere reduntant (ubrugelig) data.

2. Sikring af dataafhængigheder giver mening, dvs. data lagres logisk.

Normaliseringsregel

Normaliseringsregel er opdelt i følgende normale form.

1. Første Normalform (1NF)

2. Anden Normalform (2NF)

3. Tredje Normalform (3NF)

4. Sammenfatning af de 3 normalformer (BCNF)

Læs mere om normalisering af database:

http://www.studytonight.com/dbms/database-normalization.php

http://llk.dk/rtvnc0

eller i bogen "Introduktion til SQL" side 127 (Normalformer)

Video om normalisering af database:

https://www.youtube.com/watch?v=fg7r3DgS3rA

https://www.youtube.com/watch?v=UrYLYV7WSHM

https://www.youtube.com/watch?v=JjwEhK4QxRo

https://www.lynda.com/SQL-Server-tutorials/Reviewing-database-normalization/383047/450339-4.html

https://www.lynda.com/MySQL-tutorials/Normalizing-table/418255/441694-4.html