Grunderna i SQL – del 2

1. Underfrågor
2. Underfrågor som sökvillkor
3. Andra användningsområden för underfrågor

Kap. 6

4. Komplexa underfrågor
5. Tabellkopia
6. Lägga till poster
7. Ändra poster
8. Ta bort poster

Kap. 7
(utom ”How to merge rows”)

9. Datatyper
10. Typomvandling

Kap. 8 och 9
(utom s 284-299 om CASE, IIF, CHOOSE med flera)

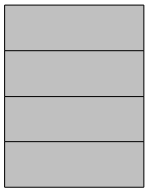
1. Underfrågor

- ***Underfrågor** (**subqueries**)* **är SELECT-*frågor* inuti andra SELECT-*frågor***
  - **Kallas även för en *nästlade frågor* (*nästlade* SELECT-*frågor*)**
- **Returnerar alltid en tabell (kan vara en enda *cell*, en *kolumn* eller en *hel tabell*)**

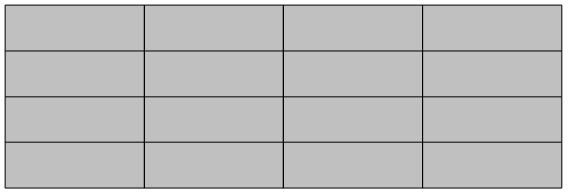
Resultat

Cell

Kolumn



Tabell



Notera

- **En underfråga behöver inte använda `ORDER BY` eller `GROUP BY`-uttryck.**
- ***Underfrågor kan i sin tur innehålla underfrågor***

1.1. Användningsområden

- I **WHERE**- eller **HAVING**-uttrycket som ett *sökvillkor* (kallas även för *predikat*)
  - Resultatet måste då vara en *cell* eller en *kolumn!* Inte flera kolumner.
- I **FROM**-uttrycket som *tabellspecifikation*
- I **SELECT**-uttrycket som *attributspecifikation*

Exempel

- ***Anställda med utbetalning över medelutbetalningen:***

```
SELECT AnställdID, Belopp
FROM Utbetalningar
WHERE Belopp >
     (SELECT AVG(Belopp)
      FROM Utbetalningar)

-- AVG(Belopp): 17300
```

Utbetalningar

AnställdID	Belopp	Månad
5	18700	Juli
5	1600	Juli
10	21200	Augusti
7	NULL	Juli
2	22500	Juli
2	22500	Augusti

Notera

- Underfrågor kan även användas i **GROUP BY**- och **ORDER BY**-satser.

1.2. Underfråga eller join?

- **Både *underfrågor* och *join* kan operera på en eller flera tabeller**
- ***Join-uttryck* går ofta att skriva med *underfrågor* istället (och vice versa)**

Fördel *join*

- **Resultatet kan inkludera attribut från alla tabellerna i *join-uttrycket***
- **Kan ge tydligare syntax vid *koppling* mellan tabeller (främmande nyckel till primärnyckel)**

Fördel *underfråga*

- **Kan använda resultatet från en *aggregatfunktion* som uttryck**
- **Kan ge tydligare syntax när det inte finns någon direkt koppling mellan tabeller**

Vad är snabbast?

- ***Borde* ge samma prestanda, om de verkligen är ekvivalenta, men frågeoptimeraren är inte perfekt.**
- **Murach (2008, 2012) säger att joinar oftast är snabbare.**
- **Gulutzan&Pelzer säger i *SQL Performance Tuning* (2003) att underfrågor är (var) snabbare i SQL Server 2000.**
- **Välj (normalt) det tydligaste. SQL är programmering, och programmering är kommunikation.**

2. Underfrågor som sökvillkor

- **Returneras en *cell* så kan resultatet användas direkt vid jämförelse med en *jämförelseoperator***
- **Returneras en *kolumn* så måste resultatet behandlas som en *lista***
- **En resulterande *kolumn* används lämpligen i samband med `[NOT] IN`-*operatorn***

Syntax

- `{WHERE|HAVING} Attribut operator Underfråga`
- `{WHERE|HAVING} Attribut [NOT] IN (Underfråga)`

Exempel

- ***Alla lärare som tillhör en akademi som börjar på bokstaven 'A' eller 'H':***

```
SELECT *
FROM Lärare
WHERE Akademi IN
     (SELECT AkademiID
      FROM Akademier
      WHERE Name LIKE '[AH]%')
```

Lärare

Förnamn	Efternamn	E-post	Akademi
Andreas	Persson	andreas.persson@oru.se	*NULL*
Johan	Petersson	johan.petersson@oru.se	3
Kalle	Räisänen	kalle.raisanen@oru.se	3

Akademier

AkademiID	Namn
3	Handelshögskolan
4	Akademin för Naturvetenskap och teknik

2.1. Jämförelse mot en kolumn

- **Vid jämförelse mot en *kolumn* (det kan alltså vara flera värden!) så kan även nyckelorden `ALL` och `SOME` (eller `ANY`) användas**
- **`ALL` och `SOME` gör en jämförelsen mot alla poster i kolumnen (alla *element* i *listan*)**
  - **För all `ALL` så måste uttrycket vara sant för *alla poster i kolumnen***
  - **För `SOME` (eller `ANY`) så måste uttrycket vara sant för *någon av posterna i kolumnen***

Syntax

- `{WHERE|HAVING} Attribut operator [ALL|SOME] (Underfråga)`

Förklaring

- `Attribut > ALL (1, 2, 3)    -- Attribut > 3`
- `Attribut > SOME (1, 2, 3)   -- Attribut > 1`

- `Attribut = ALL (1, 2, 3)    -- Attribut = 1 AND Attribut = 2 AND Attribut = 3`
- `Attribut = SOME (1, 2, 3)   -- Attribut = 1 OR Attribut = 2 OR Attribut = 3`

2.2. Exempel

- ***Alla utbetalningar över högsta medlet för alla månader:***

```
SELECT Belopp
FROM Utbetalningar
WHERE Belopp > ALL (SELECT AVG(Belopp)
                    FROM Utbetalningar
                    GROUP BY Månad)
```

- ***Alla utbetalningar över lägsta medlet för alla månader:***

```
SELECT Belopp
FROM Utbetalningar
WHERE Belopp > SOME (SELECT AVG(Belopp)
                     FROM Utbetalningar
                     GROUP BY Månad)
```

Utbetalningar

AnställdID	Belopp	Månad
5	18700	Juli
5	1600	Juli
10	21200	Augusti
7	NULL	Juli
2	22500	Juli
7	19900	September
2	22500	Augusti
5	5600	September

Notera

- **Jämförelse med `ALL` mot en kolumn utan några poster resulterar alltid i *true***
- **Jämförelse med `ALL` mot en kolumn med endast `NULL`-poster resulterar alltid i *false***
- **Jämförelse med `SOME` (eller `ANY`) mot en kolumn utan poster eller med endast `NULL`-poster resulterar alltid i *false***

2.3. Rekursiva underfrågor

- ***Rekursiv underfrågor*** **utför en underfråga (*inre fråga*) för varje post i den *yttre frågan***
  - ***Underfrågan refererar till attribut i den yttre frågan!***

Exempel

- ***Alla utbetalningar över motsvarande medel för varje månad:***

```
SELECT Belopp, Månad
FROM Utbetalningar AS Yttre
WHERE Belopp >
     (SELECT AVG(Belopp)
      FROM Utbetalningar AS Inre
      WHERE Yttre.Månad = Inre.Månad)
```

Utbetalningar

AnställdID	Belopp	Månad
5	18700	Juli
5	1600	Juli
10	21200	Augusti
7	NULL	Juli
2	22500	Juli
7	19900	September
2	22500	Augusti
5	5600	September

Notera

- **Används samma tabell i underfrågan och den yttre frågan så måste *tabellalias* användas!**
- **Det finns även en EXISTS-*operator* som är användbar i samband med *rekursiva underfrågor***
  - **Returnerar en *indikation* på ifall det finns några poster i underfrågan**
  - `WHERE [NOT] EXISTS (Underfråga)`

3. Andra användningsområden för underfrågor

- **Kan användas i samband med `FROM`-uttrycket som *tabellspecifikation***
- **Kan användas i samband med `SELECT`-uttrycket som *attributspecifikation***

`FROM`-uttrycket

- **En resulterande tabell i `FROM`-uttrycket kallas för en *härledd tabell (CTE, AS)***
- ***Härledda tabeller* är likt *vyer* (senare i denna kurs)**
- **För underfrågor samband med `FROM`-uttrycket så måste ett *alias* skapas**
- **Alla attribut i en *härledd tabell* måste ha tilldelade attributnamn**

`SELECT`-uttrycket

- **Underfrågor i samband med `SELECT`-uttrycket måste returnera ett *singelvärde* (en cell)**
- **Ofta kan ett *join-uttryck* användas istället!**

4. Komplexa underfrågor

- **Används *underfrågor* i flera led så kan det totala uttrycket lätt bli komplext!**

Steg för att förenkla komplexa uttryck

1. **Skriv ner problemet i *klartext***
2. **Skissa en lösning på hela problemet med *pseudokod***
3. **Skissa även (ifall det är nödvändigt) en lösning på varje underfråga med *pseudokod***
4. **Skriv och test underfrågorna var för sig**
5. **Skriv det totala uttrycket**

Notera

- ***Att testa en underfråga innan den används i det totala uttrycket är aldrig fel!***

4.1. CTE

- **En *CTE* (Common Table Expression) är ett namngivet delresultat som kan användas sen i frågan**
- **Man kan se det som att det skapas en *temporär tabell.* (Det gör det inte egentligen, dvs man behöver inte vara rädd för att stora tabeller behöver skapas och lagras, men man kan tänka sig det.)**
- ***CTE* används genom nyckelordet `WITH` följt av en *tabelldefinitionen***
  - **SQL-frågan ställs på den temporära tabellen!**
- **Flera CTE-uttryck kan anges genom att separera varje tabell med komma (`,`)**
- **Om man vet vad en *vy* är så är en CTE en vy som är lokal i den här frågan**

Syntax

- ```
  WITH Tabell1 AS (Tabelldefinition)
  [, Tabell2 AS (Tabelldefinition)]
  [...]
  SQL-fråga
  ```

Notera

- **CTE kan användas med alla DML-frågor (dock är `SELECT`-frågor vanligast).**
  ```

4.2. Exempel

- ***Alla utbetalningar över högsta medlet för alla månader (nu med CTE):***

```
WITH Medelbelopp AS (
     SELECT AVG(Belopp) AS Belopp
     FROM Utbetalningar
     GROUP BY Månad
)
SELECT Utbetalningar.Belopp
FROM Utbetalningar
WHERE Utbetalningar.Belopp >
     ALL Medelbelopp.Belopp
```

Utbetalningar

AnställdID	Belopp	Månad
5	18700	Juli
5	1600	Juli
10	21200	Augusti
7	NULL	Juli
2	22500	Juli
7	19900	September
2	22500	Augusti
5	5600	September

- ***Alla utbetalningar över lägsta medlet för alla månader (nu med CTE):***

```
WITH Medelbelopp AS (
     SELECT AVG(Belopp) AS Belopp
     FROM Utbetalningar
     GROUP BY Månad
)
SELECT Utbetalningar.Belopp
FROM Utbetalningar
WHERE Utbetalningar.Belopp >
     SOME Medelbelopp.Belopp
```

5. Tabellkopia med SELECT … INTO

- **Genom att använda ett `INTO`-uttryck i samband med `SELECT`-uttryck så kan *tabellkopior* skapas**
- **Tabellkopian får samma attribut som resultatet av `SELECT`-frågan**
  - **Därför måste även alla attribut namnges (genom *alias*)**
- **`SELECT`-frågan kan som vanligt innehålla *joins*, *grupperingar*, *aggregatfunktioner*, etc.**

Syntax

```
SELECT Attribut
INTO Tabellkopia              -- Anger namnet på tabellkopian
FROM Tabell                   -- Namnet på befintlig(a) tabell(er)
[WHERE Sökvillkor]
[GROUP BY Attribut]
[HAVING Sökvillkor]
[ORDER BY Attribut]
```

Notera

- **Tabellkopian skapas på ”rådata” av resultatet av `SELECT`-frågan (dvs. inga *nyckelförhållanden* etc. kopieras)**

6. Lägga till poster med INSERT

- **Med en `INSERT`-*fråga* kan poster *läggas till* i en tabell**
- **Vid en `INSERT`-fråga så anges *attributen* (*attributlista*) för vilka värden (`VALUES`) som ska *läggas till***
  - **Värdenas typ måste även överensstämma med respektive angivet attribut!**
- ***Attributlistan* kan även utlämnas, men då måste värdena överensstämma med attributen så som dom har definierats i tabellen**

Syntax

- ```
  INSERT [INTO] Tabell [(Kolumnlista)]
  [DEFAULT] VALUES (Värde1, Värde2, ...)
  [,(Värde1, Värde2, ...)] ...
  ```

Notera

- **Nyckelordet `INTO` efter `INSERT` är valfritt, men ökar läsbarheten.**
- **Fr.o.m *SQL Server 2008* så kan flera poster läggas till med samma `INSERT` -uttryck**
  - **Flera listor med värden separeras med komma (`,`).**
  ```

6.1. Exempel

- ***Lägger till en lärare i tabellen lärare (med angiven attributlista):***

```
INSERT INTO Lärare (Akademi, E-post, Efternamn, Förnamn)
VALUES (3, 'klaus.klasson@oru.se', 'Klausson', 'Klaus')
```

- ***Lägger till en lärare i tabellen lärare (utan angiven attributlista):***

```
INSERT INTO Lärare
VALUES ('Klaus', 'Klausson', 'klaus.klasson@oru.se', 3)
```

Lärare (original)

Förnamn	Efternamn	E-post	Akademi
Andreas	Persson	andreas.persson@oru.se	*NULL*
Johan	Petersson	johan.petersson@oru.se	3
Kalle	Räisänen	kalle.raisanen@oru.se	3

Default: 3

Lärare (efter tillägg)

Förnamn	Efternamn	E-post	Akademi
Andreas	Persson	andreas.persson@oru.se	*NULL*
Johan	Petersson	johan.petersson@oru.se	3
Kalle	Räisänen	kalle.raisanen@oru.se	3
Klaus	Klausson	klaus.klasson@oru.se	3

Default: 3

Notera

- ***Strängvärden*** **måste anges inom** ***dubbla apostroftecken*** **(' ')!**

6.2. Poster med NULL eller DEFALUT -värden

- Tillåter ett attribut `NULL`-värden så kan nyckelordet `NULL` användas istället för ett värde
- Om ett attribut har ett `DEFAULT`-värde så kan nyckelordet `DEFALUT` användas istället för ett värde
- Attribut som tillåter `NULL` och/eller `DEFAULT`-värden kan utlämnas från *attributlistan*
- Tillåter alla attribut `NULL` och/eller `DEFAULT`-värden så kan hela listan av värden utlämnas
  - Nyckelordskombinationen `DEFAULT VALUES` används istället

Exempel

- *Föregående exempel men utan ingivna* `NULL` *eller* `DEFAULT` *-attribut:*

```
INSERT INTO Lärare (Efternamn, Förnamn, E-post)
VALUES ('Klaussson', 'Klaus', 'klaus.klasson@oru.se')
```

- *Ännu ett skrivsätt för föregående exempel:*

```
INSERT INTO Lärare (Förnamn, Efternamn, E-post, Akademi)
VALUES ('Klaussson', 'Klaus', 'klaus.klasson@oru.se', DEFAULT)
```

6.3. Poster från andra tabeller med INSERT INTO … SELECT

- **Genom *underfrågor* istället för nyckelordet `VALUES` så kan poster från andra tabeller *läggas till***
- **Anges *attributlista* för `INSERT`-uttrycket så måste underfrågan returnera värden som överensstämmer med *attributlistan***
- **Utelämnas *attributlistan* så måste underfrågan returnera värden som överensstämma med attributen så som dom har definierats i tabellen**

Syntax

```
INSERT [INTO] Tabell_1 [(Attribut)]
SELECT Attribut
FROM Tabell_2
[WHERE Sökvillkor]
```

Notera

- **Avancerade underfrågor går självklart att använda!**
  - **Resultatet av underfrågan måste dock överensstämmer med *attributlistan*.**

7. Ändra poster med UPDATE

- **Med en `UPDATE`-*fråga* så kan poster *ändras* i en tabell**
- **Attribut och respektive nytt värde anges i ett `SET`-uttryck**
- **Antalet poster poster som ska påverkas *begränsas* genom `WHERE`-uttrycket**
  - **Genom *härledd tabeller* i `FROM`-uttrycket så kan sökvillkoret ytterligare begränsas!**
- **Attribut kan även *ändras* till `NULL` eller `DEFAULT`-värde (ifall attributet tillåter detta)**

Syntax

```
UPDATE tabell
SET Attribut1 = Värde1 [, Attribut2 = Värde2]...
[FROM Tabell AS Alias]
[WHERE Sökvillkor]
```

Notera

- **Ifall `WHERE`-uttrycket utelämnas så kommer alla poster i tabellen att påverkas!**
- **En *identitetskolumn* går inte att ändra!!**

7.1. Exempel

- ***Ändrar tillhörande akademi för ”Klaus Klaussson”:***

```
UPDATE Lärare
SET Akademi = 4
WHERE Förnamn = 'Klaus' AND Efternamn = 'Klausson'
```

- ***Ändrar så att alla lärare tillhör akademi nr. 3:***

```
UPDATE Lärare
SET Akademi = DEFAULT
```

Lärare

Förnamn	Efternamn	E-post	Akademi
Andreas	Persson	andreas.persson@oru.se	*NULL*
Johan	Petersson	johan.petersson@oru.se	3
Kalle	Räisänen	kalle.raisanen@oru.se	3
Klaus	Klausson	klaus.klasson@oru.se	4

Default: 3

7.2. Underfrågor i UPDATE-frågor

- **Underfrågor går att använda inom `SET`, `FROM` och `WHERE`-uttrycken i en `UPDATE`-fråga**
  - **I `SET`-uttrycket så kan underfrågan returnera ett *singelvärde***
  - **I `WHERE`-uttrycket så kan en underfråga begränsa *sökvillkoret***
  - **I `FROM`-uttrycket så kan en underfråga användas för att identifiera *tillgängliga attribut***

Exempel

- ***”Klaus” tillhör nu den akademi som har det högsta akademi ID:***

```
UPDATE Lärare
SET Akademi =
     (SELECT MAX(AkademiID)
      FROM Akademier)
```

Akademier

AkademiID	Namn
3	Handelshögskolan
4	Akademin för naturvetenskap och teknik
5	Musikhögskolan

Lärare

Förnamn	Efternamn	E-post	Akademi
Andreas	Persson	andreas.persson@oru.se	*NULL*
Johan	Petersson	johan.petersson@oru.se	3
Kalle	Räisänen	kalle.raisanen@oru.se	3
Klaus	Klausson	klaus.klasson@oru.se	5

- ***Alla som tillhör en akademi som börjar på ”M” ska nu tillhöra ”Handelshögskolan”:***

```
UPDATE Lärare
SET Akademi = 3
WHERE Akademi =
     (SELECT AkademiID FROM Akademier WHERE Name LIKE 'M%')
```

7.3. Join i UPDATE -frågor

- **Är attribut av andra tabeller av intresse för en `UPDATE`-fråga så kan tabeller sammanfogas med en *join* i `FROM`-uttrycket**
  - **Attribut från alla sammanfogade tabellerna kan användas i `SET`- och `WHERE`-uttrycken!**

Exempel

- ***Alla som tillhör en akademi som börjar på ”M” ska nu tillhöra ”Handelshögskolan” (nu med ett join-uttryck):***

```
UPDATE Lärare
SET Lärare.Akademi = 3
FROM Akademier JOIN Lärare
    ON Lärare.Akademi = Akademier.AkademiID
WHERE Akademier.Namn LIKE 'M%'
```

Akademier

AkademiID	Namn
3	Handelshögskolan
4	Akademin för naturvetenskap och teknik
5	Musikhögskolan

Lärare

Förnamn	Efternamn	E-post	Akademi
Andreas	Persson	andreas.persson@oru.se	*NULL*
Johan	Petersson	johan.petersson@oru.se	3
Kalle	Räisänen	kalle.raisanen@oru.se	3
Klaus	Klausson	klaus.klasson@oru.se	3

8. Ta bort poster med DELETE

- **Med en `DELETE-frågor` så kan poster *tas bort* från en tabell**
- **Villkoret för att en post ska tas bort anges i `WHERE`-uttrycket**

Syntax

- ```
  DELETE [FROM] tabell
  [FROM tabell]
  [WHERE sökvillkor]
  ```

Exempel

- ***Ta bort alla lärare som har ett efternamn som börjar på ”Pe”:***

```
DELETE FROM Lärare
WHERE Efternamn LIKE 'Pe%'
```

Lärare

Förnamn	Efternamn	E-post	Akademi
Andreas	Persson	andreas.persson@oru.se	*NULL*
Johan	Petersson	johan.petersson@oru.se	3
Kalle	Räisänen	kalle.raisanen@oru.se	3
Klaus	Klausson	klaus.klasson@oru.se	3

Notera

- ***Ifall `WHERE`-uttrycket utlämnas så kommer alla poster att tas bort!!!***
- **Nyckelordet `FROM` efter `DELETE` är valfritt, men ger en tydligare syntax.**
```

8.1. Underfrågor och join i DELETE-frågor

- **Genom *underfrågor* eller *join-uttryck* i samband med `FROM`-uttrycket så kan `DELETE`-frågor baseras på attribut i andra tabeller**
- **Används *underfrågor* i samband med `WHERE`-uttrycket så kan resultatet begränsas ytterligare**

Exempel

- ***Ta bort alla akademier som inte har några lärare:***

```
DELETE FROM Akademier
WHERE AkademiID NOT IN
     (SELECT DISTINT Akademi
      FROM Lärare)
```

Akademier

AkademiID	Namn
3	Handelshögskolan
4	Akademin för naturvetenskap och teknik
5	Musikhögskolan

- ***Ta bort alla lärare som tillhör ”Musikhögskolan”:***

```
DELETE Lärare
FROM Akademier JOIN Lärare
    ON Lärare.Akademi = Akademier.AkademiID
WHERE Akademier.Name = 'Musikhögskolan'
```

Lärare

Förnamn	Efternamn	E-post	Akademi
Andreas	Persson	andreas.persson@oru.se	*NULL*
Johan	Petersson	johan.petersson@oru.se	3
Kalle	Räisänen	kalle.raisanen@oru.se	3
Klaus	Klausson	klaus.klasson@oru.se	5

9. Datatyper

- **Datatyperna i SQL delas in i *strängtyper*, *numeriska typer*, *tidstyper* (och *andra* typer)**

Datatyper (repetition)

- `bit` `-- En bit som kan anta värdet 0 eller 1`
- `int, bigint, smallint, tinyint` `-- Heltal i olika storlekar`
- `money, smallmoney` `-- Används för ekonomiattribut`
- `decimal, numeric` `-- Decimaltal`
- `float, real` `-- Flyttal`
- `datetime, smalldatetime` `-- Används för datumattribut`
- `char, varchar` `-- Text och strängar (original ASCII)`
- `nchar, nvarchar` `-- Text och strängar (Unicode)`
- `rowversion (timestamp)` `-- Används för att sätta en tidsstämpel`

Notera

- **SQL Server stödjer de flesta *ANSI-standard-datatyperna. A*nvänd hellre SQL Server-typerna!**
- **`rowversion` rekommenderas då Microsoft planerar att byta ut standardrnd `timestamp`.**

9.1. Numeriska datatyper

- ***Heltalstyper* används till att lagra *heltal***
- ***Decimaltalstyper* används för att lagra *decimaltal* (exakt precision)**
- ***Flyttalstyper* används för att lagra *approximerade decimaltal* (flytande precision)**
- **Det finns funktioner att använda i samband med *numeriska datatyper***

Funktioner

- **ROUND**(decimaltal,längd[,funktion]) -- Avrundar ett decimaltal
- **ISNUMERIC**(uttryck) -- Returnerar ifall ett uttryck är numeriskt
- **ABS**(tal) -- Ger det absoluta värdet
- **CEILING**(decimaltal) -- Avrundar uppåt
- **FLOOR**(decimaltal) -- Avrundar nedåt
- **SQUARE**(flyttal) -- Returnerar ett flyttal upphöjt till två
- **SQRT**(flyttal) -- Returnerar roten ur ett flyttal
- **RAND**([heltal]) -- Returnerar ett slumptal

Notera

- **SQUARE och SQRT kräver ett flyttal (`float` eller `real`).**

9.2. Exempel och problem

- **Flyttal är approximationer. Det är inte lämpligt att ha ett exakt sökvillkor (som = eller <>) på ett flyttal**
  - **Använd istället ett intervall i sökvillkoret!**

Exempel

- `ROUND(12.5,0)` `-- Avrundas till 13.0`
- `CEILING(12.5)` `-- Avrundas även den upp till 13`
- `FLOOR(12.5)` `-- Avrundas ner till 12`

- **SELECT-*fråga som användes för att söka alla poster som är ungefär lika med* `1.00`:**

```
SELECT Talen
FROM Flyttalstabell
WHERE Talen = 1.00                -- Ej säkert att det blir exakta träffar

SELECT Talen
FROM Flyttalstabell
WHERE Talen BETWEEN 0.99 AND 1.01  -- Bättre att använda ett intervall
```

- ***(Ja, `1.00` lagras exakt som ett flyttal, men inte `0.40` och `0.60`. Så vad blir `0.40 + 0.60`?)***

9.3. Strängtyper

- ***Strängar* med *fast antal tecken* använder lika stort utrymme för varje cell**
- ***Strängar* med *varierande antal tecken* använder olika stort utrymme för varje cell**
- ***Normala strängar* använder *1 byte (8 bitar, tal mellan 0 och 255)* för varje tecken**
- ***Unicode strängar* använder *2 byte* för varje tecken**
  - **Kan representerar de flesta tecken från de flesta språken (inklusive svenska *å*, *ä* och *ö*)**

Syntax

```
char[(n)], nchar(n)          -- Strängar med ett fast antal tecken
varchar[(n)], nvarchar(n)    -- Strängar med varierande antal tecken
```

Notera

- **Längden på en sträng kan variera mellan 1 och 8000 tecken (eller 4000 tecken för Unicode).**
- **Äldre versioner av SQL Server använde `text` och `ntext`**
- **Har ersätts genom att använda nyckelordet `max` tillsammans med *strängtyperna:***
  - `nvarchar(max)`
  - **Ger varierande strängar upp till *2 GB* i storlek.**

9.4. Funktioner för att arbeta med strängtyper

- **Det finns ett stort antal funktioner att använda i samband med *strängtyper***

Funktioner

- **LEN**(sträng) -- Längden av en sträng
- **LTRIM**(sträng) -- Tar bort mellanrum i början av en sträng
- **RTRIM**(sträng) -- Tar bort mellanrum i slutet av en sträng
- **LEFT**(sträng, längd) -- Returnerar tecken från början av en sträng
- **RIGHT**(sträng, längd) -- Returnerar tecken från slutet av en sträng
- **SUBSTRING**(sträng, start, längd) -- Returnerar en delsträng
- **REPLACE**(sök, träffar, ersätt) -- Returnerar en ersättningssträng
- **CHARINDEX**(tecken, sträng) -- Returnerar index för angivet tecken
- **PATINDEX**(söksträng, sträng) -- Returnerar index för angiven sträng
- **REVERSE**(sträng) -- Returnerar en omvänd sträng
- **LOWER**(sträng) -- Omvandlar en sträng till endast gemener
- **UPPER**(sträng) -- Omvandlar en sträng till endast versaler

9.5. Exempel och problem

- **Sortering på attribut innehållande nummer av *strängtyp* kan ge oväntade resultat**
  - **Genom att *typomvandla* strängar till heltal så kommer sorteringen att ge förväntat resultat!**

Exempel

- ***För- och efternamn ska sammanfogas till ett fullständigt namn, men förnamnet ska endast skrivas med en initial:***

```
SELECT LEFT(Förnamn, 1) + '. ' + Efternamn AS Fullnamn
FROM Lärare
```

- ***Skapa ett ”användarnamn” för lärare:***

```
SELECT LEFT(Förnamn, 3) + '. ' +
    RIGHT(Efternamn, 2) AS Användare
FROM Lärare
```

Lärare

Förnamn	Efternamn	E-post	Akademi
Andreas	Persson	andreas.persson@oru.se	*NULL*
Johan	Petersson	johan.petersson@oru.se	3
Kalle	Räisänen	kalle.raisanen@oru.se	3
Klaus	Klausson	klaus.klasson@oru.se	5

Notera

- ***I SQL så börjar index på 1 och går till strängens totala längd!! Inte 0 till längden minus ett, som i många programmeringsspråk!***

9.6. Tidstyper

- ***Tidstyper* används för att representera *datum* och *tider***
- **Tidstyperna anges med en *formatering*, ex:**
  - **`yyyy-mm-dd` för datum (`2011-11-15`)**
  - **`hh:mi` för tid (`13:15`)**
- ***Default* är `1900-01-01` respektive `12:00`**

Syntax

```
date                -- Representera ett datum
time(n)             -- Representera en tid (n anger precisionen på sek)
datetime2(n)        -- Representera både ett datum och en tid
datetimeoffset(n)   -- Offset för tidszoner
```

Notera

- **Innan SQL Server 2008 så användes `datetime` och `smalldatetime`.**
- **Anges endast 2 siffror för år i `date`, så betyder `00–49` *detta* sekel och `50–99` är *föregående* sekel.**

9.7. Funktioner för att arbeta med tidstyper

- **Det finns självklart även funktioner att använda i samband med *tidstyper***
  - **Som *del-argument* nedan används: `year`, `month`, `day`, `week`, `hour`, `minute`, etc.**

Funktioner

- **SYSDATETIME**() -- Returnerar nuvarande datum (systemklockan)
- **DAY**(datum), **MONTH**(datum), **YEAR**(datum) -- Returnerar dag, månad eller år
- **DATENAME**(del, datum) -- Returnerar angiven delen som en sträng
- **DATEPART**(del, datum) -- Returnerar angiven delen som ett heltal
- **DATEADD**(del, tal, datum) -- Addera tal till den angivna delen
- **DATEDIFF**(del, start, slut) -- Skillnaden i del mellan start- och stoppdatum
- **ISDATE**(uttryck) -- Returnerar ifall ett uttrycket är tidstyp

Exempel

- ***Vilka utbetalning har gjorts för september månad:***

```
SELECT Belopp
FROM Utbetalningar
WHERE MONTH(Datum) = 9
```

Utbetalningar

AnställdID	Belopp	Datum
5	18700	2010-07-02
10	21200	2010-08-15
2	22500	2010-07-21
7	19900	2010-09-09
2	22500	2010-08-18
5	5600	2010-09-12

10. Typomvandling

- **Många *typomvandlingar* sker *implicit* i SQL Server**
  - **Används olika datatyper i samma uttryck så omvandlar SQL Server hela uttryck till den ”*lägsta” gemensamma typen***
- **Vissa typomvandlingar sker dock inte implicit utan måste göras *explicit***
- **För *explicit typomvandling* så finns funktionerna `CAST` och `CONVERT`**
  - **I samband med en `CONVERT` så anges även en *formatkod* (sid. 251 i kursboken).**

Syntax

- `CAST(uttryck AS datatyp)  -- Omvandlar explicit ett uttryck till önskad datatyp`
- `CONVERT(datatyp, uttryck [,formatering.])  -- Explicit omvandling med angiven formatering`

Notera

- **Självklart kan explicita omvandling även användas för att öka läsbarheten i syntaxen!**
- **`CAST` är *ANSI-standard* medan `CONVERT` endast finns i SQL Server.**

10.1. Exempel

- ***Utbetalda beloppen ska även anges med en enheten ”kr”:***

```
SELECT AnställdID, CAST(Belopp AS VARCHAR) + ' kr' AS BeloppMedEnhet, Månad
FROM Utbetalningar
```

- ***Säkerställer att sorteringen sker korrekt (även ifall AnställdID skulle vara en sträng):***

```
SELECT *
FROM Utbetalningar
ORDER BY CAST(AnställdID AS INT)
```

Utbetalningar

AnställdID	Belopp	Månad
5	18700	Juli
5	1600	Juli
10	21200	Augusti
7	NULL	Juli
2	22500	Juli
7	19900	September
2	22500	Augusti
5	5600	September

10.2. Andra typer av omvandlingar

- **Nedan följer ytterligare ett par funktioner för att omvandla mellan *tecken och tal***

Syntax

- `STR(float[,längd,[decimaler]])  -- Omvandlar ett flyttal till en sträng`
- `CHAR(heltal)    -- Omvandlar heltal till motsvarande tecken (ASCII)`
- `ASCII(sträng)   -- Omvandlar första tecknet till motsvarande heltal (0-255)`
- `NCHAR(heltal)   -- Omvandlar heltal till motsvarande tecken (Unicode)`
- `UNICODE(sträng) -- Omvandlar första tecknet till motsvarande heltal (0-65535)`

Exempel

- **CHAR** ***och*** **NCHAR** ***är användbara till att lägga till kontrolltecken till en sträng:***

```
SELECT Förnamn + CHAR(13) + Efternamn AS Fullnamn   -- Lägger till ett ”radbryt”
FROM Lärare
```