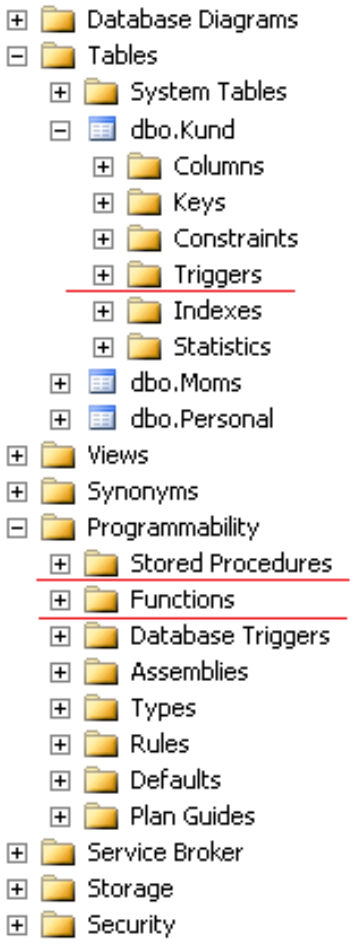
Innehåll Programability del 1

- ✓ Lagrade Procedurer, introduktion
- ✓ Variabler, Lokala och Globala
- ✓ Skapa och ändra en lagrad procedur
- ✓ Indata – parametrar till lagrade procedurer
- ✓ Temporära tabeller
- ✓ Flöden med IF/ELSE, WHILE och CASE
- ✓ Mer om parametrar
- ✓ Felhantering med TRY/CATCH
- ✓ Felmeddelande RAISERROR
- ✓ UDF, User Defined Functions

Chapter 10, 11 och 12.

Beginning SQL Server 2008 for Developers

Programability ?



Avsnittet Programability i SQL Server består av ett flertal underavsnitt.

I detta kursavsnitt behandlas:

- ✓ Stored Procedures / Lagrade Procedurer
- ✓ Functions, UDF (User Defined Functions)
- ✓ Triggers

Koden som används i detta undervisningsmaterial är inte att betrakta som komplett då det vanligtvis har reducerats för att få plats på en sida.
Det exemplifierar hur lösningar kan vara.

Koden bör därför i de flesta fall kompletteras för att utgöra en komplett lösning.

Exempel Programkod T-SQL

```
CREATE PROCEDURE KollaKunderna
@Kundid int=0   /* default värde i Kundid=0 */
AS
SET NOCOUNT ON
IF @Kundid=0
   BEGIN
      SELECT Namn, Adress, Postnr, Ort
      FROM Kund;
   END
ELSE
   BEGIN
      SELECT Namn, Adress, Postnr, Ort
      FROM Kund
      WHERE Kundid=@Kundid;
      IF @@ROWCOUNT=0                          -- inga rader i Retur
         RAISERROR('Finns inte!',16,1)
         RETURN (1)
   END
```

Exempel på en lagrad procedur. Som indata skickas ett Kundid med. Om Kundid=0 visas alla kunder annars sker en kontroll om kunden finns. Finns inte kunden returneras ett meddelande.

Exempel på en UDF, User Define Function. Hämtar artikelpriset från artikel och returnerar detta.

UDF kan inte användas i Default Value som ex: GETDATE()

```
CREATE FUNCTION GetArtikelID(@ArtikelID Int)
RETURNS decimal(6,2)
WITH EXECUTE AS CALLER
AS
Begin
        Declare @Priset Decimal(6,2)
        SELECT @Priset=Artikel.pris
        From Artikel
        WHERE artikelid=@ArtikelID
        RETURN @Priset;
END
GO
```

Vad är en lagrad procedur

En lagrad procedur är ett avsnitt av T-SQL program-kod. Exempelvis:

```
BEGIN
  SELECT Namn, Adress, Postnr, Ort
  FROM Kund;
END
```

Lagrad Procedur
Store Procedure
Sproc, sp
är vanliga namn.

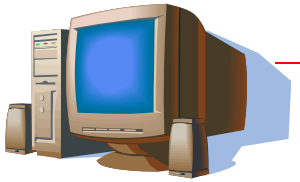
Koden du skriver in kompileras och sparas i databasen. Den kompilerade koden kan du sedan anropa hur många gånger som helst.

En sproc används ofta för att automatisera funktioner i databashanteraren. Där ligger ofta en del affärslogik.

Lagrade procedurer är ett mycket kraftfullt verktyg med vilket du kan lösa många problem.

Varför lagrade procedurer

- Ger snabbare, bättre svarstid än att köra dynamisk SQL. Koden är redan kompilerad.
- Enklare installation. Endast en kopia på databasservern – ingen på klienten.
- Reducerad nätverkstrafik. Istället för exempelvis tre SQL satser skickas enbart en instruktion.
- Ökad säkerhet, det går att styra rättigheter för procedurer. Enklare att ta hand om SQL Injektions.
- Ökad säkerhet genom att inte tillåta direktkontakt med tabellerna i databasen. Låt alla data gå via sproc till/från tabellerna.

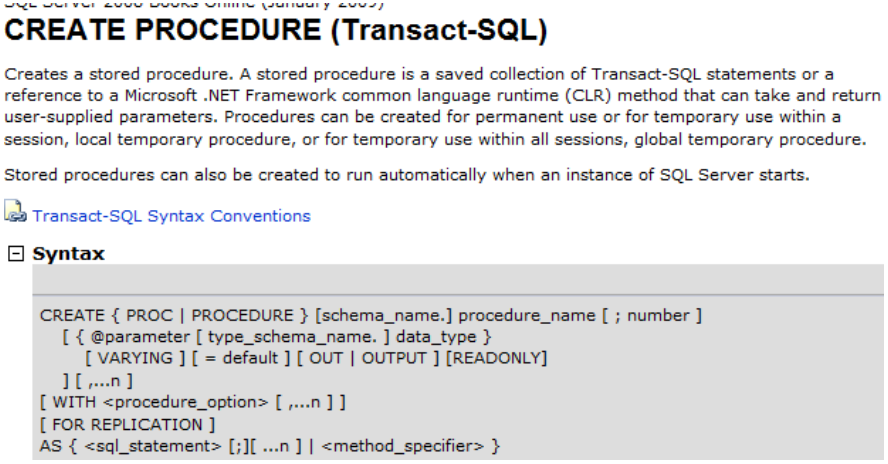


```
BEGIN
   SELECT *
   FROM Artikel;
END
```

Artikelid	Artikelnamn	Antal	Pris
1001	Bildskärm platt 17 tum	25	150,00
1002	Chassi Mini	15	125,00
1003	DVD Skivor	500	2,50
1004	Musmatta	940	7,00

Var finns info om lagrade procedurer

MSSQL se länken som leder till Books OnLine: SQL Server Developer Center



SQL Server 2008 Books Online (January 2009)

CREATE PROCEDURE (Transact-SQL)

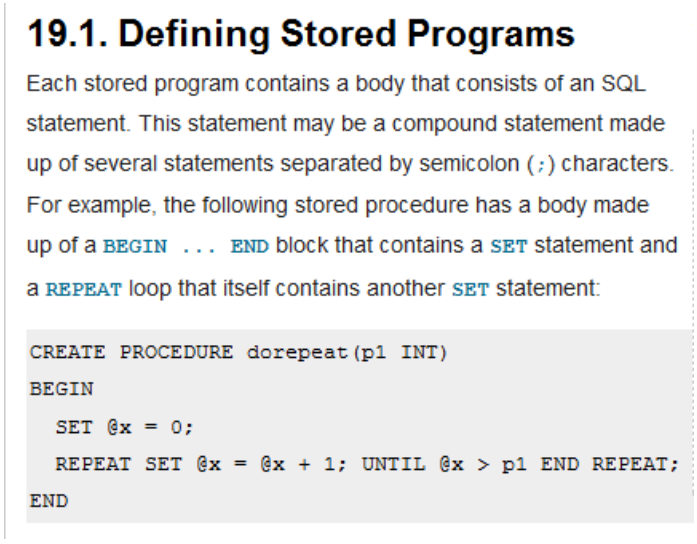
Creates a stored procedure. A stored procedure is a saved collection of Transact-SQL statements or a reference to a Microsoft .NET Framework common language runtime (CLR) method that can take and return user-supplied parameters. Procedures can be created for permanent use or for temporary use within a session, local temporary procedure, or for temporary use within all sessions, global temporary procedure.

Stored procedures can also be created to run automatically when an instance of SQL Server starts.

Transact-SQL Syntax Conventions

Syntax

```
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]
    [ { @parameter [ type_schema_name. ] data_type }
        [ VARYING ] [ = default ] [ OUT | OUTPUT ] [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { <sql_statement> [;][ ...n ] | <method_specifier> }
```



19.1. Defining Stored Programs

Each stored program contains a body that consists of an SQL statement. This statement may be a compound statement made up of several statements separated by semicolon (`;`) characters. For example, the following stored procedure has a body made up of a `BEGIN ... END` block that contains a `SET` statement and a `REPEAT` loop that itself contains another `SET` statement:

```
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
  SET @x = 0;
  REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
END
```

MySQL se hjälpen i Query Browser eller till referensmanualen på mysql.com.

… och i litteraturen. *SQL Server for Developers sidan 336*.

Skapa en lagrad procedur, CREATE

BEGIN / END motsvarar {} i C#.

En enkel lagrad procedur kan se ut enligt nedan.
Uppgiften för den är att returnera ett resultatset med posterna enligt SQL satsen.

```
CREATE PROCEDURE usp_GetKund                --Skapar sproc
AS
BEGIN
    SELECT Namn, Adress, Postnr, Ort        -- SQL kod
    FROM Kund;
END
GO
```

Först exekverar du koden enligt ovan vilket gör att din lagrade procedur skapas och därefter finns den tillgänglig för exekvering.

Exekvering sker genom anrop med EXEC eller EXECUTE. Exemplet ovan ger följande resultatset:

```
EXEC usp_GetKund
```

	Namn	Adress	Postnr	Ort
1	Adamsssons El	Lingonstigen 13	393 52	KALMAR
2	Berglunds Bilskola	Stigen 71	393 65	KALMAR
3	Erikssons Måleri	Adamsstigen 99	393 64	KALMAR
4	Johansson Kalle	Marsstigen 7B	393 51	KALMAR

Ändra lagrad procedur, ALTER

Högerklicka på den lagrade proceduren och välj Modify.

```
ALTER PROCEDURE usp_GetKund                    --Ändrar sproc
AS
BEGIN
     SELECT Namn, Adress, Postnr, Ort          -- SQL kod
     FROM Kund
     ORDER BY Postnr, Namn;
END
GO
```

```
--  enradig kommentar
/*  flerradig kommentar */
```

När du exekverar koden genomförs ändringen och du får en förändrad lagrad procedur som du kan använda.

Den exekveras sedan på samma sätt som föregående.

```
EXEC usp_GetKund
```

	Namn	Adress	Postnr	Ort
1	Adamsssons El	Lingonstigen 13	393 52	KALMAR
2	Berglunds Bilskola	Stigen 71	393 65	KALMAR
3	Erikssons Måleri	Adamsstigen 99	393 64	KALMAR
4	Johansson Kalle	Marsstigen 7B	393 51	KALMAR

Lokala Variabler

Variabelnamn måste börja med @
Variabeln måste deklareras
Datatypen ska vara en SQL-Server datatyp eller egendefinierad datatyp

```
Declare @Kundid int, @Namn varchar(30)

Declare @Pris Decimal(7,2)
```

Tilldelning av ett värde i en variabel gör du med SET eller SELECT

```
Declare @Antalet Int                          -- variabeln skapas
Declare @Antalet Int = 0                      -- variabeln skapas, default värde = 0

SET @Antalet=23

SET @Antalet= @Antalet+1

SELECT @Antalet=23

SET @Antalet=  (SELECT Antal From Artikel     -- antal hämtas från
               WHERE Artikelid=102);          -- artikel 102

SELECT @Antalet=Antal From Artikel            -- antal hämtas från
WHERE Artikelid=104;                          -- artikel 104

SELECT @Antalet=Antal From Artikel;           -- antal hämtas från sista
                                              -- artikeln
```

Globala Variabler

I SQL Server finns ett antal globala variabler som du har tillgång till. Variablerna finns specificerade i Books Online. Många av de globala variablerna gäller server-informationen eller den aktuella anslutning med den aktuella databasen med sina tabeller.

Namnen på den Globala variablerna börjar alltid med **@@** De globala variablerna är **read-only** och kan inte skapas eller ändras av användaren.

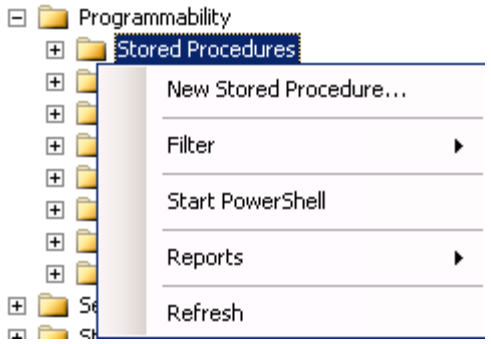
Några variabler är:

```
@@ERROR       -- Error status för sista SQL satsen

@@ROWCOUNT    -- Antalet rader som på verkats

@@IDENTITY    -- Värdet på sista IDENTITY kolumnen

SELECT * From Kund
WHERE Kundid=@@IDENTITY;
```

- @@CONNECTIONS (Transact-SQL)
- @@CPU_BUSY (Transact-SQL)
- @@CURSOR_ROWS (Transact-SQL)
- @@DATEFIRST (Transact-SQL)
- @@DBTS (Transact-SQL)
- @@ERROR (Transact-SQL)
- @@FETCH_STATUS (Transact-SQL)
- @@IDENTITY (Transact-SQL)
- @@IDLE (Transact-SQL)
- @@IO_BUSY (Transact-SQL)
- @@LANGID (Transact-SQL)
- @@LANGUAGE (Transact-SQL)
- @@LOCK_TIMEOUT (Transact-SQL)
- @@MAX_CONNECTIONS (Transact-SQL)
- @@MAX_PRECISION (Transact-SQL)
- @@NESTLEVEL (Transact-SQL)
- @@OPTIONS (Transact-SQL)
- @@PACK_RECEIVED (Transact-SQL)
- @@PACK_SENT (Transact-SQL)
- @@PACKET_ERRORS (Transact-SQL)
- @@PROCID (Transact-SQL)
- @@REMSERVER (Transact-SQL)
- @@ROWCOUNT (Transact-SQL)
- @@SERVERNAME (Transact-SQL)
- @@SERVICENAME (Transact-SQL)
- @@SPID (Transact-SQL)
- @@TEXTSIZE (Transact-SQL)
- @@TIMETICKS (Transact-SQL)
- @@TOTAL_ERRORS (Transact-SQL)

Skapa en lagrad procedur i SSMS (1 av 2)

1 Högerklicka på **Stored Procedures** som du hittar under Programmability i din databas.



2 Välj **New Stored Procedure...**

```
-- ============================================
-- Template generated from Template Explorer using:
-- Create Procedure (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- This block of comments will not be included in
-- the definition of the procedure.
-- ============================================
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =============================================
CREATE PROCEDURE <Procedure_Name, sysname, ProcedureName>
    -- Add the parameters for the stored procedure here
    <@Param1, sysname, @p1> <Datatype_For_Param1, , int> = <Default_Value_Fo
    <@Param2, sysname, @p2> <Datatype_For_Param2, , int> = <Default_Value_Fo
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT <@Param1, sysname, @p1>, <@Param2, sysname, @p2>
END
GO
```

SET ANSI_NULLS ON/OFF
Sätter att SQL Server ska arbeta enligt ansi standard med null-värden

SET QUOTED_IDENTIFIER ON /OFF
Du kan använda ” (citationstecken) istället för [] då reserverade namn används .
SELECT namn, ”select” From ”From”

SET NOCOUNT ON/OFF
Radräknarens resultat visas inte,
Se @@ROWCOUNT

Skapa en lagrad procedur i SSMS (2 av 2)

3 Du får en del färdig kod.
Anpassa innehållet.
Tryck exempelvis Ctrl+Shift+M eller skriv direkt.

Du kan också skapa en sproc genom att skriva koden direkt i en query.

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- ============================================
-- Author:      Sven Åke
-- Create date: 2009-02-01
-- Description: Undervisningsexempel
-- ============================================
CREATE PROCEDURE GetKund
AS
BEGIN
    SET NOCOUNT ON;
    SELECT *
    FROM Kund;
END
GO
```

GO avslutar batch
; avslutar SQL sats

4 Exekvera koden så skapas den lagrade proceduren och är sedan färdig att användas.

Messages

```
Command(s) completed successfully.
```

5

```
EXEC usp_GetKund
```

Results | Messages

	Kundid	Namn	Adress	Postnr	Ort
1	1	Svenssons Mekaniska	Storgatan 23	39351	KALMAR
2	2	Anderssons El	Ekstigen 12	39364	KALMAR
3	3	Bilar Uthyres AB	Ringvägen 98	39362	KALMAR
4	4	MatBosse	Torget 1a	39365	NYBRO
5	5	Knytet	Storgatan 32	39352	KALMAR
6	7	Bolaget	Lingonkråkan 1	38156	NYBRO
7	9	Bo Svensson	Storgatan 23	39364	Kalmar

Exempel TelefonLista

Exempel på en lagrad procedur som resulterar i en telefonlista för de kunder som har telefon (inner join).

```
CREATE PROCEDURE usp_GetTelefonLista
AS
BEGIN
    SELECT k.Namn, ko.Kontakt, kt.Kontakttyp
    FROM Kund AS k
    INNER JOIN Kontakt AS ko
       ON k.Kundid = ko.Kundid
    INNER JOIN Kontakttyp AS kt
       ON ko.Kontakttypid = kt.Kontakttypid;
END
GO
```

Du kan också använda vyer i en lagrad procedur.

```
BEGIN
    SELECT *
    FROM TelefonLista;
END
```

Results | Messages

	Namn	Kontakt	Kontakttyp
1	Svenssons Mekaniska	070-54715879	Mobil
2	Svenssons Mekaniska	0480-987654	Kontor
3	Svenssons Mekaniska	ek@svnsmek.se	Epost
4	Anderssons El	070-987652	Mobil
5	Anderssons El	0480-982354	Kontor

```
EXEC usp_GetTelefonLista;
```

Exempel Fakturalista

Exempel på en lagrad procedur som resulterar i en lista för de artiklar som har sålts. Alla fakturarader kommer med i resultatsetet.

```
CREATE PROCEDURE usp_GetFakturaLista
AS
BEGIN
   SELECT f.fakturaid,f.Datum,fr.Artikelid, a.namn,
   fr.antal,fr.pris, fr.rabatt, m.moms,
   fr.antal*fr.pris*(1-fr.rabatt) as Nettopris,
   fr.antal*fr.pris*(1-fr.rabatt)*m.Moms as Moms,
   fr.antal*fr.pris*(1-fr.rabatt)*(1+m.Moms as Radsumma
   FROM Faktura AS f
   INNER JOIN Fakturarad as fr
      ON f.fakturaid=fr.fakturaid
   INNER JOIN Artikel as a
      ON fr.artikelid=a.artikelid
   INNER JOIN Moms as m
      ON fr.momsid=m.momsid;
END
GO
```

```
EXEC usp_GetFakturaLista;
```

IN Parameter

Antag att du vill skicka med data så att endast en viss kund visas med sina tillhörande data.

Komplettera med parametern @Kundid och vilken datatyp den har. Parametern kan heta något annat än Kundid.

OBS! om du ska ändra en befintlig lagrad procedur – högerklicka på den och välj Modify. Om den är öppen – ändra CREATE till ALTER.

```
CREATE PROCEDURE usp_GetKundSpecifik
@Kundid int
AS
BEGIN
   SELECT Namn, Adress, Postnr, Ort
   FROM Kund
   WHERE Kundid=@Kundid;
END
GO
```

Lagras i första variabeln

Kundid tillämpas

Parametervärde

```
EXEC usp_GetKundSpecifik 1
```

OBS! Tänk på datatypen. Ett textfält/sträng omgärdas av apostrof (’)

Exempel Fakturalista

```
CREATE PROCEDURE usp_GetFakturaLista
@Fakturaid int
AS
BEGIN
   SELECT f.fakturaid, f.Datum,fr.Artikelid, a.namn,
   fr.antal,fr.pris, fr.rabatt, m.moms,
   fr.antal*fr.pris*(1-fr.rabatt) as Nettopris,
   fr.antal*fr.pris*(1-fr.rabatt)*m.Moms as Moms,
   fr.antal*fr.pris*(1-fr.rabatt)*(1+m.Moms as Radsumma
   FROM Faktura AS f
   INNER JOIN Fakturarad as fr
      ON f.fakturaid=fr.fakturaid
   INNER JOIN Artikel as a
      ON fr.artikelid=a.artikelid
   INNER JOIN Moms as m
      ON fr.momsid=m.momsid
   WHERE F.fakturaid=@Fakturaid;
END
GO
```

Vad händer om IN parameter är NULL?

```
EXEC usp_GetFakturaLista 12;
```

Exempel Ny Kund

Exempel på en lagrad procedur som resulterar i att en ny kund skrivs in i Kund-tabellen.

Validering bör göras på indata så indata inte är Null.

```
CREATE PROCEDURE usp_NewKund
@Namn varchar(50),
@Adress varchar(25),
@Postnr int,
@Ort varchar(25)
AS
BEGIN
   INSERT INTO KUND (Namn, Adress, Postnr, Ort)
   VALUES (@namn, @Adress, @Postnr,@Ort);
END
GO
```

```
EXEC usp_NewKund 'EL AB','Ekstigen 99',39364,'KALMAR'
```

Exempel ÄndraKund

Exempel på en lagrad procedur som resulterar i att uppgifterna på en befintlig Kund uppdateras, ändras.

```
CREATE PROCEDURE usp_UpdKund
@Kundid Int =0,
@Namn varchar(50),
@Adress varchar(25),
@Postnr int,
@Ort varchar(25)
AS
BEGIN
   UPDATE KUND
   SET Namn = @Namn,
   Adress = @Adress,
   Postnr = @Postnr,
   Ort = @Ort
   WHERE Kundid=@Kundid;
END
GO
```

Hur testa om ändringen genomfördes?

```
EXEC usp_UpdKund 12, 'EL AB','Ekstigen 98',39364,'KALMAR'
```

Exempel RaderaKund

Exempel på en lagrad procedur som resulterar i att posten med den befintliga kunden raderas ur Kundtabellen.

```
CREATE PROCEDURE usp_DelKund
@Kundid Int =0
AS
BEGIN
   DELETE FROM Kund
   WHERE Kundid=@Kundid;
END
GO
```

```
Kontroll:

Vill du verkligen
radera kunden?
```

Vad händer med Kundens kontaktuppgifter. Det beror på hur du har satt RI.

```
EXEC usp_DelKund 12
```

Programflöde IF / ELSE (1)

Behöver flödet ytterligare beskrivas? BEGIN – END har samma betydelse som { } i C#. Om inte BEGIN END anges följer endas en rad på IF.

```
CREATE PROCEDURE usp_GetKunderna (@Kundid int=0)
AS
IF @Kundid=0
    SELECT Namn, Adress, Postnr, Ort
    FROM Kund;
ELSE
    BEGIN
       SELECT Namn, Adress, Postnr, Ort FROM Kund
       WHERE Kundid=@Kundid;
       IF @@ROWCOUNT=0                  -- inga rader i Retur
          RAISERROR('Kunden som du efterfrågat saknas!',16,1)
    END

SELECT 'Denna rad ligger utanför IF / ELSE' as Info
```

Programflöde IF EXISTS / ELSE (2)

Ett exempel där kontrollen sker om en kund finns i Kund. Detta med hjälp av IF EXISTS.

```
CREATE PROCEDURE usp_GetKunderna (@Kundid int=0)
AS
IF EXISTS (SELECT * FROM Kund WHERE Kundid=@Kundid)

    SELECT Namn, Adress, Postnr, Ort  FROM Kund
    WHERE Kundid=@Kundid;
ELSE
    SELECT Namn, Adress, Postnr,
    Ort FROM Kund;

SELECT 'Denna rad ligger utanför IF / ELSE !' AS Info
```

Programflöde WHILE

Du kan använda CONTINUE och BREAK tillsammans med WHILE

1

```
DECLARE @iNr int
SET @iNr=0
WHILE @iNr<6
    SELECT @iNr as Nummer
    SET @iNr=@iNr+1

-- @iNr uppdateras aldrig
```

2

```
DECLARE @iNr int
SET @iNr=0
WHILE @iNr<6
    BEGIN
        SELECT @iNr as Nummer
        SET @iNr=@iNr+1
    END
```

3

```
DECLARE @iNr int
SET @iNr=0
WHILE @iNr<6
    BEGIN
        SELECT @iNr as Nummer
        SET @iNr=@iNr+1
        IF @iNr>3
           BREAK
    END
-- Hur stor blir @iNr ??
```

4

```
DECLARE @iNr int
SET @iNr=0
WHILE @iNr<6
    BEGIN
        SELECT @iNr as Nummer
        SET @iNr=@iNr+1
        IF @iNr>3
           SET @iNr=@iNr-1
           BREAK
    END
```

?

Oändlig loop bryts med Alt+Break

CASE

CASE är inget specifik för programmability verktyg utan det är mer ett SQL verktyg.

```
CASE Siffra
      WHEN 10 THEN 'Tio'
      WHEN 20 THEN 'Tjugo
      WHEN 30 THEN 'Trettio'
      ELSE 'Noll'
   END AS Betalstatus
```

```
CASE
   WHEN Siffra = 10 THEN 'Tio'
   WHEN Siffra = 20 THEN 'Tjugo
   WHEN Siffra = 30 THEN 'Trettio'
   ELSE 'Noll'
END AS Betalstatus
```

```
SELECT Fakturaid, Datum,
   CASE
      WHEN BetalDatum Is Null  THEN 'Obetald'
      ELSE 'Betald'
   END AS Betalstatus
FROM Faktura
```

ISNULL(fält,värde)
Kan också användas!

Exempel TelefonLista

```
CREATE PROCEDURE usp_GetTelefonListan
@Kundid int = 0
AS
BEGIN
    If @Kundid=0
       SELECT k.Namn, k.Ort, t.telnr, ty.teltyp
       FROM Kund as k
       INNER JOIN Telefon AS t
         ON k.kundid=t.Kundid
       INNER JOIN Telefontyp AS ty
         ON t.Telefontypid=ty.Telefontypid;
    ELSE
       SELECT k.Namn, k.Ort, t.telnr, ty.teltyp
       FROM Kund as k
       INNER JOIN Telefon AS t
         ON k.kundid=t.Kundid
       INNER JOIN Telefontyp AS ty
         ON t.Telefontypid=ty.Telefontypid
       WHERE Kundid=@Kundid;
END
GO
```

IN parameter>0 ger en viss kund. Om 0 ger det alla kunder som har minst en telefon. Vad händer om in parameter är Null?

```
EXEC usp_GetTelefonListan 23  -- 0 som parameter ger alla
```

Ett exempel

Det är ofta som man vill göra en simulering av värden från flera tabeller. Då skapar du en ny tabell, temporär, och för över data från andra tabeller.

Därefter behandlar man data i den nya tabellen för att till sist kasta tabellen.

Exempel: ***Vi vill ha en lista på hur mycket våra kunder har handlat och när de har handlat. Vilken kund har handlat vad och hur mycket.***

Följande steg gör man då med en lagrad procedur:

1. Skapar den nya tabellen (CREATE TABLE….
2. Överför data från andra tabeller (INSERT INTO…..
3. Behandlar data (SELECT …..
4. Slutligen raderar tabellen (DROP TABLE……

En kortversion av den lagrade proceduren finns på nästa sida……

sproc usp_GetKundStatistikLista

```
CREATE PROCEDURE usp_GetKundStatistikAlla
AS
BEGIN
    CREATE Table #Temp                          -- # anger lokal temptabell
    (                                           -- ## anger global temptabell
        Eid int Primary Key IDENTITY (1,1),
        Namn varchar(50) NOT NULL,
        Ort varchar(30) NOT NULL,
        Datum smalldatetime NOT NULL,
        ArtikelID int NOT NULL,
        Antal smallint,
        Pris Decimal(8,2)
    )

    INSERT INTO #Temp (namn,ort,datum,artikelid, antal, pris)
    SELECT k.namn, k.ort, f.datum, fa.artikelid, fa.antal, fa.pris
    FROM Kund as k INNER JOIN Faktura as f ON k.kundid=f.kundid
    INNER JOIN Fakturarad as fa ON f.fakturaid=fa.fakturaid;

    SELECT * From #Temp;                        -- bearbetar temporära data

    DROP Table #Temp;                           -- Behövs inte men är god sed!!!!
END
GO
```

Med CTE, Common Table Expression

Sätter namnet på tabellen

```
CREATE PROCEDURE usp_GetKundStatistikAllaCTE
AS
BEGIN
    WITH KundStatistikLista AS
    (
        SELECT k.namn, k.ort, f.datum, fa.artikelid, fa.antal, fa.pris
        FROM Kund as k INNER JOIN Faktura as f ON k.kundid=f.kundid
        INNER JOIN Fakturarad as fa ON f.fakturaid=fa.fakturaid
    )
    SELECT *
    From KundStatistikLista
    ORDER BY namn;

END
GO
```

SQL satsen skapar den temporära tabellen automatiskt.

Fälten i den nya tabellen får samma namn som fälten i SELECT satsen.

```
EXEC usp_GetKundStatistikAllaCTE
```

CTE är en ny typ av temporär tabell. Tabellens fält beror på vad du tar ut med den första selectsatsen. SQL satsen utanför parenteserna () är den som presenterar urvalet.

Ordning för Parameter

Parameter kan skickas i den ordningen som vi anger, dvs variablernas ordning. De kan också skickas i namnordning eller blandat.

Det är användbart när man har många parametrar. När man väl skickat namnet så **måste alla efterföljande** parametrar skickas med namn.

```
CREATE PROCEDURE usp_DelFaktRad
@Faktradid int,
@Artid int,
@Ant int = 1
```

Vid **by position** lagras första parametern i första variabeln.

```
EXEC usp_DelFaktRad 6,104,2                              -- By position

EXEC usp_DelFaktRad @Faktradid=6,@Artid=104,@Ant=2 -- By name

EXEC usp_DelFaktRad 6,@Artid=104                         -- By position
                                                         -- and name.
                                                         -- @Ant = default
@Artid=104
EXEC usp_GetArtikel @Artid                               -- med variabel
```

Felhantering TRY / CATCH

Ett fel /misslyckande kan bero på många olika orsaker. Om ett misslyckande uppstår så fortsätter T-SQL med nästa sats. Ett misslyckandet kan exempelvis bero av valideringen på en constraints för ett fält eller som i exemplet:

```
BEGIN TRY
   UPDATE Telefon
   SET TeltypID=14     -- 14 finns inte!”
   WHERE Telefonid=2;
END  TRY
BEGIN CATCH
   SELECT   ERROR_NUMBER() As ErrorNumber,
            ERROR_SEVERITY() As ErrorSeverity,
            ERROR_STATE() As ErrorState,
            ERROR_PROCEDURE() As Errorprocedure,
            ERROR_LINE() As ErrorLine,
            ERROR_MESSAGE() As ErrorMessage
END CATCH
```

Om operation i TRY misslyckas utförs koden i CATCH

Severity-nivåer

Results | Messages

	ErrorNumber	ErrorSeverity	ErrorState	Errorprocedure	ErrorLine	ErrorMessage
1	547	16	0	NULL	2	The UPDATE statement conflicted with the FOREIGN KEY constraint "FK_Telefon_Telefontyp". .

Felmeddelande RAISERROR

RAISERROR (msg_str, serverity, state…..

RAISERROR (’Kunden saknas!’,16,1)

Syntax

```
RAISERROR ( { msg_id | msg_str | @local_variable }
   { ,severity ,state }
   [ ,argument [ ,...n ] ] )
   [ WITH option [ ,...n ] ]
```

STATE	Ett tal 1-127 som indikerar var felet uppstått
WITH LOG	Skriver också felet i SQL Server/Windows eventlogg
arg1..n	Valfria värden placeras i ”placeholders” i felmeddelanden

Severity-nivåer

Severity	Beskrivning
0 och 10	Varning
1-9, 11-16	Applikationsfel
17 och 18	Systemfel, även hårdvara
19-25	Systemfel, även hårdvara, kräver WITH LOG

Den globala variabeln **@@ERROR** sätts till msg_id om severity>10

Inbyggda RAISERROR o sys.messages

Det finns en mängd med system felmeddelande. De har gemensamt att msg_id är upp tom 50000.

```
SELECT *
FROM sys.messages
```

Results | Messages

	message_id	language_id	severity	is_event_logged	text
1	21	1033	20	0	Warning: Fatal error %d occurred at %S_DATE. Note t...
2	101	1033	15	0	Query not allowed in WAITFOR.
3	102	1033	15	0	Incorrect syntax near '%.*ls'.
4	103	1033	15	0	The %S_MSG that starts with '%.*ls' is too long. Maxim...
5	104	1033	15	0	ORDER BY items must appear in the select list if the st...

Du kan lägga till och ta bort egna meddelanden som du vill använda dig av. Kravet är att msg_id ska vara större än 50000.

```
EXEC sp_addmessage 50001, 16, 'Saknar Kundid!', us_english -- en ny

EXEC sp_dropmessage 50001, us_english                      -- raderar
```

RAISERROR (50001,16,1)

annars

RAISERROR ('Saknar Kundid',16,1)

```
RAISERROR(50001,16,1)
```

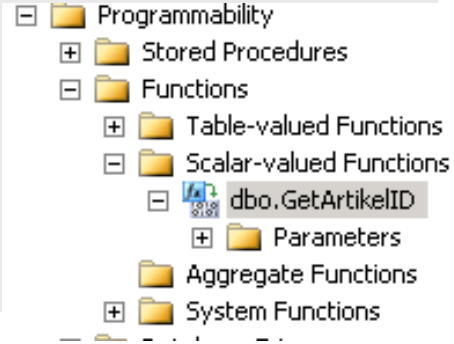
Messages

```
Msg 50001, Level 16, State 1, Line 1
Saknar Kundid!!!!
```

Egendefinierad skalärfunktion UDF

UDF = User Defined Function, egen definierad funktion. En UDF är en variant av en lagrad procedur.

```
CREATE FUNCTION GetArtikelPris(@ArtikelID Int)
RETURNS decimal(6,2)
AS
BEGIN
        Declare @Priset Decimal(6,2)
        SELECT @Priset=Artikel.pris
        From Artikel
        WHERE artikelid=@ArtikelID
        RETURN @Priset;
END
GO
```



När du kör koden skapas en funktion som återfinns i den databas du går utifrån. Ska du ändra den? Modify i menyval eller ALTER.

En UDF kan inte användas till Default value i en tabell!

Så här kan du använda den:

```
SELECT ArtikelID, Pris, dbo.GetArtikelPris(ArtikelID)
From Fakturarad;
```