SQL

Constraints

Constraints är en ANSI-standard metod för att implementera data- och referensintegritet i en databas.

Följande Constraints-typer finns:

- DEFAULT
- CHECK
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY

Man definierar Constraints genom att använda CREATE TABLE eller ALTER TABLE.

DEFAULT

DEFAULT Constraint sätter in ett värde i en kolumn när man inte specificerar något vid en INSERT.

Exempel:

ALTER TABLE Produkt
ADD CONSTRAINT def_prodnamn DEFAULT ’Saknas’ FOR Namn

CHECK

CHECK Constraints specificerar ett logiskt uttryck som måste uppfyllas för att acceptera inmatad data. Kontrollen utförs både vid INSERT och UPDATE.

Exempel:

```
CREATE TABLE Kund
(Kundid int PRIMARY KEY,
Kundnamn char(50),
Kundadress char(50),
```

```
Kundkredit money,
CONSTRAINT check_kundid CHECK (Kundid BETWEEN 0 and 10000 ) )
```

UNIQUE

UNIQUE Constraints specificerar att två rader i en kolumn inte kan ha samma värde. Ett unikt index skapas automatiskt. UNIQUE Constraints är användbara när man redan har en primärnyckel, men man vill garantera att andra kolumner också är unika.

Exempel:

```
ALTER TABLE Anställda
ADD CONSTRAINT unique_körkortsnr UNIQUE (körkortsnr)
```

Primary key

Lägga till en primary key på en existerande tabell:

```
ALTER TABLE ProdLev ADD CONSTRAINT PK_ProdLev PRIMARY KEY (PId,LevId,Datum)
```

Skapa en tabell med PN

CREATE TABLE Tabellnamn

(Kolumn 1 datatyp [[NOT] NULL],

Kolumn 2 datatyp [[NOT] NULL]…

PRIMARY KEY (Kolunm1, kolumn2…)

Skapa en tabell project enligt datalogisk modell, sätt PN, default samt check.

Projekt(ProjId, KundId,ProjektNamn)

```
CREATE TABLE Projekt
(ProjId char(8) NOT NULL,
KundID char(5) REFERENCES Kund(KundId) NOT NULL ,
ProjektNamn char(40) DEFAULT 'NN',
PRIMARY KEY (ProjID ),
CHECK ((KundId LIKE 'K[0-9][0-9[0-9]')AND (ProjId LIKE 'Proj[0-9][0-9[0-9]')))
```

Lägg till RN efter att tabellen skapats:

```
ALTER Table ProdLev ADD CONSTRAINT fk_pId FOREIGN KEY (PId) REFERENCES
Produkt(PId)
ALTER TABLE ProdLev ADD CONSTRAINT fk_levId FOREIGN KEY (LevId) REFERENCES
Leverantor(LevId)
```

Triggers

Standardvärden och regler är mycket användbara på radnivå, d v s att kontrollera en rad i taget. De kan dock inte användas för att styra vad som händer på tabellnivå. Vi kan t ex inte kontrollera att en person registrerar mer än 24 timmar per dag, även om vi har en regel som säger att varje rad får innehålla max 24 timmar.

Detta och ett flertal andra dataintegritetsproblem löses med triggers i Transact-SQL. En trigger är en slags procedur som kan aktiveras vid INSERT, UPDATE och DELETE.

Med triggers kan vi bl a

- Kontrollera och godkänna värden
- Göra automatiska insättningar, uppdateringar och borttagningar i andra tabeller
- Underhålla referensintegritet

Syntax:

CREATE TRIGGER [<ägare>.]<triggernamn>
ON [<ägare>.]<tabellnamn>
FOR {INSERT, UPDATE, DELETE}
[WITH ENCRYPTION]
AS sqlsatser

eller

CREATE TRIGGER [<ägare>.]<triggernamn>
ON [<ägare>.]<tabell>
FOR {INSERT, UPDATE}
[WITH ENCRYPTION]

```
AS
IF UPDATE (kolumn)
[{AND | OR} UPDATE (kolumn)...] sql_statements
```

Vi kan med triggers dessutom få ut ett egendefinierat felmeddelande. Detta kan vara användbart för att underlätta för användaren. Ett fel orsakat av ett brott mot någon regel ger ett standardmeddelande som inte säger något om godkända värden. Vi får bara reda på att det vi försökte skriva in inte var rätt.

Exempel: Skapa en trigger för INSERT på tabellen Produkt som ser till att kolumnen ProdId bara innehåller värden av typen PXX.

```
CREATE TRIGGER ins_trigg_produkt ON Produkt
FOR INSERT AS
IF (SELECT COUNT(*) FROM inserted) <>
        (SELECT COUNT(*) FROM inserted WHERE ProdId LIKE 'P[0-9][0-9]')
        BEGIN
                PRINT 'Fel typ av ID för produkter.'
                ROLLBACK TRAN
        END
```

Triggern anropas av servern vid varje insättning i tabellen Produkt. IF-satsen jämför antalet insatta rader med det antalet rader som har ett korrekt ProdId. Om dessa värden inte är lika så är något ProdId felaktigt. Information om detta skrivs ut och insättningen rullas tillbaka.

Tabellerna inserted och deleted

Vid INSERT, UPDATE och DELETE skapas en eller två temporära tabeller. Inserted skapas vid INSERT och UPDATE. Deleted skapas vid DELETE och UPDATE. Tabellerna skapas som en kopia av den tabell som påverkas.

Inserted, nya rader vid INSERT, ändrade rader vid UPDATE

Deleted, borttagna rader vid DELETE, gamla rader vid UPDATE

Ta bort en trigger

Vill man plocka bort en trigger görs detta på följande sätt:

DROP TRIGGER <objnamn>

Constraints

Constraints är en ANSI-standard metod för att implementera data- och referensintegritet i en databas.

Följande Constraints-typer finns:

- DEFAULT
- CHECK
- UNIQUE
- PRIMARY KEY (se flik 9)
- FOREIGN KEY (se flik 9)

Man definierar Constraints genom att använda CREATE TABLE eller ALTER TABLE.

DEFAULT

DEFAULT Constraint sätter in ett värde i en kolumn när man inte specificerar något vid en INSERT.

Exempel:

ALTER TABLE Produkt
ADD CONSTRAINT def_prodnamn DEFAULT 'Saknas' FOR ProdNamn

CHECK

CHECK Constraints specificerar ett logiskt uttryck som måste uppfyllas för att acceptera inmatad data. Kontrollen utförs både vid INSERT och UPDATE.

Exempel:

```
CREATE TABLE Kund
(Kundid int PRIMARY KEY,
Kundnamn char(50),
Kundadress char(50),
```

```
Kundkredit money,
CONSTRAINT check_kundid CHECK (Kundid BETWEEN 0 and 10000 ) )
```

UNIQUE

UNIQUE Constraints specificerar att två rader i en kolumn inte kan ha samma värde. Ett unikt index skapas automatiskt. UNIQUE Constraints är användbara när man redan har en primärnyckel, men man vill garantera att andra kolumner också är unika.

Exempel:

```
ALTER TABLE Anställda
ADD CONSTRAINT unique_körkortsnr UNIQUE (körkortsnr)
```

Referensintegritet

Eftersom en relationsdatabas är uppbyggd av tabeller som är relaterade till varandra via lika värden i kolumner, är det viktigt att referensnycklar alltid har korrekta värden. Ett möjligt användningsområde för triggers är att kontrollera just referensintegriteten.
Med en CONSTRAINT går det att definiera både primärnycklar och referensnycklar.
Denna hanteringen sköttes tidigare enbart i en trigger. Exemplen nedan visar först hur referensintegritet kan lösas med en trigger och sedan hur motsvarande funktion utförs med CONSTRAINT.

Exempel: Se till att varje rad i ProdLev har korrekta värden på ProdId och LevId

Constraint

```
ALTER TABLE Anställda ADD CONSTRAINT PK_AnstNr PRIMARY KEY (Anstnr)

ALTER TABLE Rum ADD CONSTRAINT PK_Rumsnr PRIMARY KEY (Rumsnr)

ALTER TABLE Placering ADD CONSTRAINT FK_Anställda FOREIGN KEY (Anstnr)
                                    REFERENCES Anställda (Anstnr) NOT FOR
REPLICATION

ALTER TABLE Placering ADD CONSTRAINT FK_Rum FOREIGN KEY (Rumsnr)
                                    REFERENCES Rum (Rumsnr) NOT FOR
REPLICATION
```

Trigger

```
CREATE TRIGGER ins_Trigg_Placering ON Placering
FOR INSERT
AS
IF (SELECT COUNT(*) FROM inserted, Anställda ref
        WHERE ref. Anstnr = inserted. Anstnr)<>
        (SELECT COUNT(*) FROM inserted)
BEGIN
        PRINT 'Anställd saknas!'
        ROLLBACK TRAN
        RETURN
END

IF (SELECT COUNT(*) FROM inserted, Rum ref
        WHERE ref. Rumsnr =inserted. Rumsnr)<>
        (SELECT COUNT(*) FROM inserted)
BEGIN
        PRINT 'Rummet finns ej!'
        ROLLBACK TRAN
        RETURN
END
```

Översikt procedurer

En procedur är en namngiven samling av SQL-kommandon som sparas på servern. Proceduren kan sedan köras genom att den anropas med dess namn. En procedur kan returnera värden samt ha in- och ut-parametrar. Det går oftast snabbare att exekvera en procedur än att köra samma SQL-kommando interaktivt.

Skapa en procedur

Man skapar procedurer i en databas med hjälp av kommandot CREATE PROCEDURE.

Syntax:

```
CREATE PROCEDURE [ägare].procedurnamnn [;nummer]
        [
                {@parameter  datatyp} [=default] [OUTPUT]
        ]
        [,…n]

AS
        sql-satser
```

Exempel:

```
CREATE PROCEDURE Vikt10
AS
        SELECT *
        FROM Produkt
        WHERE Vikt > 10



CREATE PROCEDURE Minsta
AS
DECLARE @minsta int, @meddelande varchar(255)
   SELECT @minsta = MIN(Vikt)
   FROM Produkt
   SELECT @meddelande = 'Den minsta vikten är' + @minsta + 'kg'
   PRINT @meddelande
RETURN

CREATE PROCEDURE update_kvantitet
AS
        IF(SELECT AVG(Kvantitet) FROM ProdLev) > 200
        BEGIN
```

```
                    UPDATE ProdLev
                    SET Kvantitet = Kvantitet * 2
                    WHERE Kvantitet < 200
                    PRINT 'Kvantiteten har uppdaterats till följande värden'
                    SELECT  *
                    FROM ProdLev
          END
RETURN
```

Anropa en procedur

Syntax: EXEC *procedurnamn*

Exempel:

EXEC Vikt10

EXEC Minsta

EXEC update_kvantitet

Skapa en procedur med inparametrar

Inparametrar tillåter att information skickas in till en procedur.

I detta exempel använder vi en inparameter för att ställa ett villkor i proceduren:

```
CREATE PROCEDURE vikt
          @vikt int
AS
          SELECT *
          FROM Produkt
          WHERE Vikt > @vikt
RETURN
```

Anropa en procedur med parametrar

Parametrar kan skickas in i en procedur via *position* eller *namn*.

Syntax: **Position**

EXEC procedurnamn värde1, värde2, värde3, …

Syntax: **Namn**

EXEC procedurnamn parameter1 = värde1, parameter2 = värde2, parameter3 = värde3 …

Exempel:

Alt 1: EXEC vikt 10

Alt 2: EXEC vikt @vikt = 10

Skapa en procedur med standardvärde till parametrar

Man kan ge parametrar standardvärden när man skapar en procedur. Om man vid anrop av proceduren inte anger något värde för parametern erhåller den standardvärdet istället.

Exempel:

```
CREATE PROCEDURE Hämta_Produkt
        @prodid char(5) = 'P01'
AS
        SELECT  Prodid, ProdNamn, Kategori
        FROM Produkt
        WHERE ProdId = @prodid
```

Skapa en procedur med outputparameter

En procedur kan returnera information till den anropande proceduren eller klienten med output parametrar. För att använda en output parameter måste man specificera nyckelordet OUTPUT i både CREATE PROCEDURE och EXECUTE.

Exempel:

```
CREATE PROCEDURE räknare
                @mult1 int, @mult2 int, @resultat int OUTPUT
AS
                SELECT @resultat = @mult1*@mult2

RETURN
```

Anropa en procedur med outputparameter

För att kunna anropa en procedur som har outputparameter måste man deklarera en variabel som ska erhålla värdet från proceduren.

Exempel:

```
DECLARE @res int

EXEC räknare 10, 4, @res OUTPUT

SELECT @res
```

Ta bort en procedur

Man tar bort en procedur genom att använda kommandot DROP PROCEDURE.

Exempel:

```
DROP PROCEDURE räknare
```

Titta på information om procedurer

Man kan använda systemprocedurer för att titta på information om en procedur.

Systemprocedur	**Information**
sp_help procedurnamn	Visar en lista över parametrar och dess datatyper för en specifik procedur
sp_helptext procedurnamn	Visar texten för en specificerad procedur om den inte är krypterad
sp_stored_procedures	Returnerar en lista över alla procedurer som finns i den aktuella databasen