

STORED PROCEDURES

Stored Procedures

- A stored procedure is prepared SQL code that we save so we can reuse the code over and over again. So if we think about a query that we write over and over again, instead of having to write that query each time we would save it as a stored procedure and then just call the stored procedure to execute the SQL code that we saved as part of the stored procedure.
- In addition to running the same SQL code over and over again we also have the ability to pass parameters to the stored procedure.

SYNTAX

CREATE PROCEDURE <procedure_EID>

AS

BEGIN

<SQL Statement>

END

EXECUTE <procedure_EID>

EXEC <procedure_EID>

<procedure_EID>

Stored Procedures

Example 1 : Simple Procedure to get the details of Delhi employees

```
CREATE PROCEDURE SHDELEMP  
AS  
BEGIN  
    SELECT * FROM EMP WHERE CITY = 'DELHI';  
END;
```

Stored Procedures

Example 2 : **Parameterized** Procedure to get the details of employees of the specified city

```
CREATE PROCEDURE SHOWEMP @X VARCHAR(20)
AS
BEGIN
    SELECT * FROM EMP WHERE CITY = @X;
END;
```

Example 3 : **Parameterized** Procedure to get the contents of the specified table

```
CREATE PROCEDURE SHOW @Y VARCHAR(20)
AS
BEGIN
    EXEC('SELECT * FROM ' + @Y );
END;
```

Stored Procedures

Example 4 : **Parameterized** Procedure to insert the data in the emp_sal table

```
CREATE PROCEDURE IN_EMP_SAL
@ID VARCHAR(5), @A VARCHAR(20), @B VARCHAR(20), @X INT
AS
BEGIN
    SET NOCOUNT ON ;
    INSERT INTO EMP_SAL VALUES
    ( @ID, @A, @B, @X );

    SELECT * FROM EMP_SAL
    WHERE EID=@ID;
END;
```

Stored Procedures

A stored procedure with parameters:

SYNTAX

```
CREATE PROCEDURE <procedure_EID>  
@ var1 datatype (size), var2 datatype (size)  
AS  
BEGIN  
[SET NOCOUNT ON/OFF]  
<SQL Statement>  
END
```

```
EXECUTE <procedure_EID> var1 , var2
```

```
DROP PROCEDURE <procedure_EID>
```

ASSIGNMENT



ASSIGNMENT – 8

A-1 : CREATE BELOW PROCEDURES IN THE INVENTORY DATABASE AS SPECIFIED :

ADDSUPPLIER – SHOULD ADD THE SUPPLIER IN THE SUPLIER TABLE AND DISPLAY THE DETAILS OF THE NEW SUPPLIER ADDED.

ADDPRO – SHOULD ADD THE PRODUCT IN THE PRODUCT TABLE AND DISPLAY THE DETAILS OF THE NEW PRODUCT ADDED.

ADDCUST – SHOULD ADD THE CUSTOMER IN THE CUSTOMER TABLE AND DISPLAY THE DETAILS OF THE NEW CUSTOMER ADDED.

ADDORDER – SHOULD ADD THE ORDER IN THE ORDERS TABLE AND DISPLAY THE DETAILS OF THE ORDER. ORDER DATE SHOULD BE CURRENT DATE AND SHOULD COME AUTOMATICALLY.

TRANSACTIONS

Transactions

- A transaction is a unit of work that is performed against a database. For example, if you are creating a record or updating a record or deleting a record from the table, then you are performing a transaction on the table.

Properties of Transactions

Transactions have the following four standard properties, usually referred to by the acronym ACID:

Atomicity: Ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure, and previous operations are rolled back to their former state.

Consistency: Ensures that the database properly changes state upon a successfully committed transaction.

Isolation: Enables transactions to operate independently of and transparent to each other.

Durability: Ensures that the result or effect of a committed transaction persists in case of a system failure

Transactions

There are following commands used to control transactions:

- **COMMIT:** To save the changes.
- **ROLLBACK:** To roll back the changes.
- **SAVEPOINT:** Creates points within groups of transactions in which to ROLLBACK.

AUTO INCREMENT FIELD

Auto Increment

Auto Increment allows a unique number to be generated automatically when a new record is added in to the table.

- Identity (START, INCREMENT)

Example :

```
create table emp2  
(id int identity (1,1) primary key,  
EID varchar (30),  
age int);
```

SEQUENCES

Sequences

Sequences are the objects in SQL Server that is used to generate a number sequence. These are normally used to create a unique number.

▪ Syntax

```
CREATE SEQUENCE sequence_EID  
[ AS datatype ]  
[ START WITH value ]  
[ INCREMENT BY value ]  
[ MINVALUE value | NO MINVALUE ]  
[ MAXVALUE value | NO MAXVALUE ]  
[ CYCLE | NO CYCLE ]  
[ CACHE value | NO CACHE ];
```

Sequences

- **Example 1:**

```
Create sequence MYSEQ  
AS INT  
START WITH 1  
INCREMENT BY 1  
MINVALUE 1  
MAXVALUE 1000  
No CYCLE  
CACHE 5;
```

- **Example 2:**

```
Create sequence MYSEQ  
START WITH 1  
INCREMENT BY 1
```

```
Drop Sequence MYSEQ;
```

NOTE: Sequences are the global objects, however, auto increment works on the table level

Sequences

- Using Sequences

```
SELECT NEXT VALUE FOR MYSEQ;
```

- Using sequence in the insert statement.

```
INSERT INTO CANDIDATE VALUES (NEXT VALUE FOR MYSEQ, 'AJAY');
```

- Procedure using sequence to generate the candidate ID and insert the data in table.

```
CREATE PROCEDURE ADDCANDIDATE (@N AS VARCHAR(50))
```

```
AS
```

```
BEGIN
```

```
    DECLARE @A AS INT;
```

```
    DECLARE @C AS CHAR(5);
```

```
    SET @A = ( NEXT VALUE FOR MYSEQ);
```

```
    IF @A <10
```

```
        SET @C = CONCAT('C00' , @A);
```

```
    ELSE IF @A<100
```

```
        SET @C = CONCAT('C0' , @A);
```

```
    ELSE IF @A<1000
```

```
        SET @C = CONCAT('C' , @A);
```

```
    INSERT INTO CANDIDATE VALUES (@C, @N);
```

```
END;
```


Auto Generation of ID Using Sequence

Function to generate a Alpha Numeric ID

```
CREATE FUNCTION GENID (@C CHAR (1) , @I INT)
RETURNS CHAR(5)
AS
BEGIN
    DECLARE @r CHAR(5);
    DECLARE @ID CHAR(5);
    SELECT @R = CASE
        WHEN @I < 10 THEN CONCAT(@C,'000')
        WHEN @I < 100 THEN CONCAT(@C,'00')
        WHEN @I < 1000 THEN CONCAT(@C,'0')
        WHEN @I < 10000 THEN @C
        ELSE 'NULL'
    END;
    SET @ID= RTRIM(@R) + LTRIM(CONVERT(CHAR(4),@I));
    RETURN @ID;
END;
```

Auto Generation of ID Using Sequence

Using user defined function with a sequence in a procedure to add an student in to the table:

```
CREATE PROCEDURE ADDSTU @X CHAR(20)
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO STU
    VALUES(DBO.GENID('S',NEXT VALUE FOR MYSEQ),@X);

    SELECT * FROM STU;

END;
```



ASSIGNMENT – 9

A-1 : CREATE A FUNCTION FOR AUTOGENERATION OF 5 CHARACTERS ALPHA NUMERIC ID. IT SHOULD ACCEPT 2 PARAMETERS A CHARACTER AND THE NUMBER AND RETURN THE ID BY CONCANATING THE CHARACTER , REQUIRED ZEROS AND THE SPECIFIED NUMBER.

RECREATE BELOW PROCEDURES IN THE INVENTORY DATABASE AS SPECIFIED (ALL THE ID s SHOULD BE AUTOMATICALLY GENERATED USING ABOVE CREATED FUNCTION AND SEQUENCES) :

ADDSUPPLIER – SHOULD ADD THE SUPPLIER IN THE SUPLIER TABLE AND DISPLAY THE DETAILS OF THE NEW SUPPLIER ADDED.

ADDPRO – SHOULD ADD THE PRODUCT IN THE PRODUCT TABLE AND DISPLAY THE DETAILS OF THE NEW PRODUCT ADDED.

ADDCUST – SHOULD ADD THE CUSTOMER IN THE CUSTOMER TABLE AND DISPLAY THE DETAILS OF THE NEW CUSTOMER ADDED.

ADDORDER – SHOULD ADD THE ORDER IN THE ORDERS TABLE AND DISPLAY THE DETAILS OF THE ORDER. ORDER DATE SHOULD BE CURRENT DATE AND SHOULD COME AUTOMATICALLY.

TRIGGERS

Triggers

Triggers are the *special kind* of stored procedures

which get automatically executed

whenever the event



on the under ling table occurs.

Triggers

A trigger is a database object that is attached to a table. Triggers are often referred to as a "special kind of stored procedure". The main difference between a trigger and a stored procedure is that the trigger is attached to a table and is only fired when an INSERT, UPDATE or DELETE occurs. You specify the modification action(s) that fire the trigger when it is created.

- **Syntax**

```
CREATE TRIGGER trigger_EID  
ON table_EID  
FOR INSERT|UPDATE |DELETE  
AS  
BEGIN  
    SQL Statements;  
END;
```

Triggers

Example 1: Trigger to update the stock when product is sold.

```
CREATE TRIGGER TR_INVENT_UPDATE
ON SALES
FOR INSERT
AS
BEGIN
UPDATE INVENT SET StockQty = StockQty- (SELECT QTY FROM INSERTED
)
WHERE PID = (SELECT PID FROM INSERTED);
END;
```

Triggers

Example 2: Trigger to delete the order if the product is deleted from the inventory.

```
CREATE TRIGGER TR_SALE_DELETE
ON INVENT
FOR DELETE
AS
BEGIN
DELETE FROM SALES WHERE PID = (SELECT PID FROM DELETED);
END;
```


Triggers

Example 3: Trigger to update the stock when the order quantity has been updated.

```
CREATE TRIGGER TR_STOCK_UPDATE2
ON SALES
FOR UPDATE
AS
BEGIN
    UPDATE Stock SET SQty = SQty + (SELECT QTY FROM DELETED)
    WHERE PID = (SELECT PID FROM DELETED);

    UPDATE Stock SET SQty = SQty - (SELECT QTY FROM INSERTED)
    WHERE PID = (SELECT PID FROM INSERTED);
END;
```



Triggers

Example 4: Trigger to check & update the stock when the order is placed

```
CREATE TRIGGER TR_INVENT_CHECK
ON SALES
FOR INSERT
AS
BEGIN
    DECLARE @QS AS INT;
    DECLARE @QR AS INT;
    SET @QR= ( SELECT QTY FROM INSERTED);
    SET @QS = (SELECT StockQty FROM INVENT WHERE PID=(SELECT PID FROM inserted));
    IF @QS >= @QR
        Begin
            UPDATE INVENT SET StockQty = StockQty- (SELECT QTY FROM INSERTED )
            WHERE PID = (SELECT PID FROM INSERTED);
            COMMIT;
        end
    ELSE
        ROLLBACK;
    );
```





Thanks!

**EVERY ENDING
IS REALLY JUST A
NEW BEGINNING**

Rajeev Garg
Data Analytics Trainer

