# الفصل الثالث عشر
# الإجراءات والأحداث
# Stored procedures & triggers

تعلمنا في الفصول السابقة كل الموضوعات التي تمكننا من إعداد **Queries** أو الجمل التي تقوم بتنفيذ كل ما نريده من قاعدة البيانات سواء بتكوين قاعدة بيانات أو الجداول وغيرها، في هذا الفصل سوف نتعلم بعض الموضوعات المتقدمة ومنها:

- التعامل مع الإجراءات **(Stored Procedures)** .
- التعامل مع الأحداث **(Triggers)**.
- الجداول المؤقتة.

# استخدام الإجراءات (Stored Procedures)

## *تعريف الإجراءات* stored procedures

في الواقع هي عبارة عن برنامج أو مجموعة من الأوامر التي يتم تنفيذها بصفة مستمرة لذلك لا يجب أن يتم كتابتها كل مرة نحتاجها، والميزة التي تحققها أنك عندما تكتب مجموعة من الأوامر في صورة query وتطلب من نظام قواعد البيانات تنفيذها، فأول خطوة يقوم بها قبل التنفيذ هي عملية الترجمة Compilation من الصورة المكتوب بها تلك الأوامر إلى صورة قابلة للتنفيذ، وعملية الترجمة Compilation تبدأ بمراجعة الصيغة أولا Syntax Checking وبعد تصحيح الصيغة أو الشكل العام تتم الترجمة، فإذا قمت كل مرة بتنفيذ نفس الـ Query سيتم ترجمتها أولا، فماذا لو قمنا بتصحيح الصيغة والترجمة ثم تخزينها في إجراء أو (Stored Procedure) على جهاز الخادم ثم بعد ذلك نطلب من النظام التنفيذ فقط، فهل بذلك وفرنا وقت الترجمة واختصرنا بالتالي وقت التنفيذ ؟ أعتقد أن الإجابة واضحة !

كما تعلم عزيزي القارئ أن نظام قواعد البيانات مصمم أساسا لكي يقوم العديد من المستفيدين من خلال الشبكات بتنفيذ أعمالهم، فماذا لو أن عدد من المستفيدين أرسل مجموعة من الـ Queries الكبيرة ليتم ترجمتها ومن ثم تنفيذها ؟ كم سيستغرق ذلك من وقت لإرسال الـQuery إلى النظام والموجود طبعا على خادم Server في مركز رئيسي، وذلك من خلال الشبكات Network وكم سيستغرق التخاطب بين المستفيد والنظام الموجود على الخادم؟ هنا تأتي أهمية الإجراءات أو Stored Procedures فلن يتم التعامل معها إلا وقت التنفيذ، فلن ترسل Query بالكامل لترجمتها وتنفيذها ولكنك ستطلب من النظام التنفيذ فقط، فهل وفرنا وقت الإرسال والترجمة عبر الشبكة ؟ أليس هذا أمرا في غاية الأهمية ؟!

## *إنشاء الإجراءات*

يتم إنشاء الإجراءات بواسطة العبارة CREATE PROCEDURE . في

الإجراءات يمكن استخدام الجداول والاستعلامات أو الدوال المعرفة من قبل المستخدم أو إجراءات أخري . لكي تنشئ إجراء يجب أن تكون المسئول عن قاعدة البيانات ( DBA ) أو مالكا لها ( DB_OWNER ) أو عضوا في الإشراف على النظام ( sysadmin )

*صيغة إنشاء الإجراء*

```
CREATE PROC [ EDURE ] procedure_name [ ; number ]
   [ { @parameter data_type }
      [ VARYING ] [ = default ] [ OUTPUT ]
   ] [ ,...n ]
[ WITH
   { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]
[ FOR REPLICATION ]
AS sql_statement [ ...n ]
```

وفيما يلي شرح المفردات التي يشتمل عليها هذا الأمر

- عبارة **CREATE PROCEDURE** العبارة الأساسية لتخليق إجراء .
- عبارة **procedure_name** اسم الإجراء الجديد .
- الكلمة **number** ; رقم اختياري يستخدم لتمييز الإجراء الجديد عندما نسمي الإجراء بنفس الاسم حتى إذا ما أردنا إلغائهم جميعا مرة واحدة نستطيع إلغائهم بعبارة واحدة .
- المتغير **@parameter** متغير يتم استخدامه في الإجراء .
- العبارة **data_type** نوع المتغير .
- الكلمة **VARYING** تحدد القيمة الخارجة من الإجراء.
- الكلمة **Default** هي القيمة المفترضة للمتغير الخاص بالإجراء.
- الكلمة **OUTPUT** إشارة إلى أن المتغير الخاص بالإجراء راجع من الإجراء.
- المتغير **N** إشارة إلي عدد المتغيرات .

مثال

```
USE SALES
GO
CREATE PROC MYPROC
```

```
AS
SELECT *  From orders
WHERE RequiredData < GetDate( )and
ShippedDate is null
```

في هذا المثال تم إنشاء إجراء باسم MYPROC في قاعدة البيانات SALES يتم فيه اختيار كافة السجلات من جدول orders بشرط أن يكون حقل RequiredDate أقل من تاريخ اليوم ( GetDate ) وبشرط أيضا أن يكون حقل shippedDate يساوي القيمة الخالية NULL .

*تنفيذ الإجراء*

يستخدم أمر EXEC أو EXECUTE لتنفيذ إجراء سبق إنشاؤه والشكل العـــام لجملة EXEC كما يلي :

```
[ [ EXEC [ UTE ] ]
   {
      [ @return_status = ]
         { procedure_name [ ;number ] | @procedure_name_var
   }
   [ [ @parameter = ] { value | @variable [ OUTPUT ] | [ DEFAULT
] ]
      [ ,...n ]
[ WITH RECOMPILE ]
```

حيث

- EXECUTE هي العبارة الأساسية للتنفيذ ( من الممكن استخدام EXEC فقط ) .
- @return_status = هو متغير رقمي يتم فيه توضيح حالة الإجراء .
- Procedure_name اسم الإجراء المراد تنفيذه .
- number ; هو رقم اختياري يستخدم لتمييز الإجراء عندما نسمي الإجراء بـــنفس الاسم حتى إذا ما أردنا إلغائهم جميعا مرة واحدة نستطيع إلغائهم بعبارة واحدة .
- @procedure_name_var متغير يشير لاسم الإجراء .
- @parameter هو متغير يتم استخدامه في الإجراء .

- **Value** قيمة المتغير .
- **OUTPUT** إشارة إلى أن المتغير الخاص بالإجراء راجع من الإجراء .

لتنفيذ الإجراء الذي أنشأناه في المثال السابق استخدم الأمر التالي :

```
EXECUTE MYPROC
GO
```

في المثال التالي سنقوم بإنشاء أجراء ( **procedure** ) باسم ( **print_books** ) لنضع به مجموعة من التعليمات لطباعة أسماء الكتب المخزنة بجدول **store** . وبالتالي نستطيع تنفيذ الإجراء في أي وقت بمجرد استدعائه بدلا من كتابة التعليمات مرة أخرى :

```
Create procedure print_books
as
declare books cursor
for select title  from store
DECLARE     @title varchar(40)
open books
FETCH next from BOOKS
into @title
WHILE @@FETCH_STATUS = 0
BEGIN
        print @title
        FETCH next from BOOKS
        into @title
end
CLOSE BOOKS
DEALLOCATE CURSOR BOOKS
Go
```

نقوم بتنفيذ الإجراء الذي قمنا بتكوينه تحت اسم **print_books** وذلك باستخدام الجملة التالية :

```
Execute print_books
Go
```

إذا أردت تنفيذ الإجراء أي عدد من المرات يلزمك فقط القيام بتنفيذ جملـــة **execute print_books** في كل مرة ، فان كل الأوامر السابقة تم تخزينها بعد ترجمتها وتـــصحيح

الأخطاء وهي جاهزة للتنفيذ تحت هذا الاسم print_books !

*تعديل إجراء*

يتم تعديل الإجراء بالعبارة ALTER PROCEDURE وعند تعديل إجراء يتم استبدال التعريف السابق للإجراء بواسطة ما تم تعديله . لتعديل إجراء استخدم عبارة ALTER PROCEDURE بدلا من عبارة CREATE PROCEDURE واستخدم نفس الخيارات التي سبق شرحها .

*حذف الإجراء*

لحذف الإجراء ادخل أمر

```
DROP PROCEDURE procedure_name
```

وسوف يتم حذف الإجراء. لحذف الإجراء السابق استخدم الأمر التالي :

```
DROP PROCEDURE print_books
Go
```

*إدخال قيم عند تنفيذ الإجراء كمدخلات :*

جدول store يحتوي على معلومات عن الكتب مثل اسم الكتاب title الذي قمنا بطباعته في المثال السابق، ولكن عندما نريد أن نطبع اسم الكتب title لناشر معين اليوم وغدا لناشر آخر وهكذا، أي أننا سوف نضيف اسم الناشر عندما نقوم بتنفيذ الإجراء كيف نفعل ذلك ؟

سنقوم بإعداد الإجراء بطريقة يستقبل بها قيم عند التنفيذ هذه القيم تسمى Parameters وبالتالي نستفيد من تخزين الإجراء وأيضا نستفيد من استخدام نفس الإجراء مع إدخال قيم أثناء التنفيذ وهو ما يعرف بتمرير المعاملات Passing Parameters الشكل العام لاستخدام متغيرات كمدخلات كما يلي

```
@parameter data_type
          [=default]
```

وتوضع بعد اسم الإجراء في أمر CREATE PROC

انظر المثال التالي :

```
set statistics time off
drop procedure print_books
go
Create procedure print_books @author varchar(50)
as
declare books cursor
for select title, authors  from store
where authors = @author
DECLARE     @title varchar(40)
open books
FETCH next from BOOKS
WHILE @@FETCH_STATUS = 0
BEGIN
        print @title
        FETCH next from BOOKS
end
CLOSE BOOKS
DEALLOCATE CURSOR BOOKS
Go
```

في هذا المثال استخدمنا **@author** في جملة **CREATE procedure** لتعريف النظام أن هذا البرنامج سوف يستقبل قيمة أثناء التنفيذ، وقمنا بتعريف نوع المتغير **@author** على أنه **varchar(50)** أي سيستقبل حروف، كما قمنا باستخدام **where** للمقارنة فيما بين البيانات الموجودة وقيمة المتغير **@author**، وفيما يلي سنرى كيف نقوم بتنفيذ هذا الإجراء بعد تخزينه وتمرير قيمة له.

```
Execute print_books 'que'
Go
```

بهذه الطريقة يمكن تنفيذ الإجراء **print_books** أي عدد من المرات وفي كل مرة يمكن تمرير قيمة جديدة فمثلاً في المثال السابق قمنا بتمرير قيمة **'que'** كاسم للناشر . ليقوم الإجراء باستخدام هذه القيمة للبحث عن اسم الناشر **author** وطباعة الكتب التي تخصه وهي الكتب التي تساوي **'que'** في حقل **author** .

## التعامل مع الأحداث Triggers

مثلها مثل الإجراء يتم تخزينها ، ولكن طبيعة عمل الـ **Trigger** تشبه طبيعة عمل الحارس ، ولكن حارس على أحد الجداول ، عندما تقوم بتكوين **trigger** فانك تخصصه للعمل مع جدول **Table** معين

يجب أن تأخذ في اعتبارك ما يلي :

- تعرف الأحداث على جدول محدد والذي يشار إليه بـ **Trigger table** حيث يكون هذا الحدث مرتبطا بهذا الجدول .
- يتم تنفيذ الحدث تلقائيا في ٣ حالات: عند إدخال بيانات **INSERT** أو تعديل البيانات **UPDATE** أو عند حذف بيانات **DELETE** بصرف النظر عن الفعل المرتبط بهذا الحدث.
- علي عكس الإجراءات لا يمكن تنفيذ الحدث مباشرة .

إذا كان هناك خطأ ما أثناء تنفيذ **Query** على الجدول مثل **UPDATE** فسيقوم الحدث **TRIGGER** بإلغاء التنفيذ وإعادة بيانات الجدول لأصله **Rollback** .

فيما يلي الشكل العام لإنشاء **Trigger**:

```
CREATE TRIGGER trigger_name
ON { table | view }
{ [ INSERT ] [ , ] [ UPDATE ] [,] [DELETE] }
      AS
Sql statement
```

حيث :

- **CREATE TRIGGER** العبارة الأساسية لإنشاء حدث .
- **Trigger_name** اسم الحدث .
- **table_name ON** يتم في هذا السطر تحديد اسم الجدول الذي سيعمل عليه الحدث .
- **{ BEFOR | AFTER | INSTEAD OF }** زمن تنفيذ الحدث

- BEFOR قبل الحدث .
- AFTER بعد الحدث .
- INSTEAD OF إلغاء الفعل المترتب علي الحدث وتنفيذ فعلا آخر غيره .

- { INSERT | UPDATE | DELETE } أنواع الأحداث.
- AS : الشروط أو التعليمات التي تريد تنفيذها.

مثال

نريد أن نضع ضوابط على جدول Books_out على سبيل المثال بحيث نمنع إضافة Insert أي سجل record ليس له مقابل في الجدول الرئيسي store . انظر المثال التالي:

```
CREATE TRIGGER CHECK_BOOKS_OUT
ON BOOKS_OUT
FOR INSERT AS
        IF NOT EXISTS (SELECT * FROM BOOKS_OUT O, STORE
S
        WHERE O.BOOK_ID = S.BOOK_ID)
        BEGIN
                PRINT 'INVALID BOOK_ID'
                ROLLBACK
        END
Go
```

في هذا المثال اخترنا اسم check_books_out للحدث ثم ذكرنا اسم الجدول الذي نضع عليه الضوابط وهو في هذه الحالة books_out في السطر الثاني من الجملة بعد كلمة ON ، ثم حددنا الأوامر التي يشملها الحدث trigger وهي في هذا المثال INSERT بعد كلمة FOR وتشير كلمة AS إلى الشروط أو التعليمات التي تريد تنفيذها إذا حاول أحد المستفيدين تنفيذ جملة INSERT على جدول BOOKS_OUT .

فإذا تحقق الشرط أي إذا لم يوجد مقابل للسجل في الجدول الرئيسي ستنفذ التعليمات الموجودة بين BEGIN وEND وهي طباعة رسالة. وإعادة الجدول إلى حالته الأصلية.

حاولنا أن نستخدم الأمر SELECT وحاولنا إدخال سجل و به حقل BOOK_ID ليس

له مقابل في جدول STORE كما يلي :

```
INSERT BOOKS_OUT
VALUES (99,'ELNEEL',444,20)
```

والنتيجة كما ظهرت في واجهة Query Analyzer كما يلي :

```
SQL Server Execution Times:
  CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
  CPU time = 0 ms, elapsed time = 0 ms.
INVALID BOOK_ID
```

هناك عدد من الضوابط أو القيود التي يجب أن تضعها في اعتبارك عند استخدامك للأحداث TRIGGERS ومنها :

- لا تستخدم TRIGGERS مع الجداول المؤقتة TEMPORARY TABLES
- يتم استخدام TRIGGERS مع جداول في قاعدة البيانات الحالية، لذلك يجب استخدام أمر USE database_name اذا أردت التحول الى قاعدة بيانات أخرى.
- لا يمكن إنشاء أحداث TRIGGERS مع الجداول التخيلية VIEWS.
- عند حذف جدول DROP TABLE فان كل الأحداث TRIGGERS المتعلقة بهذا الجدول يتم حذفها تلقائيا.

*تعديل الحدث وإلغائه*

يتم تعديل الحدث بأمر مشابه لأمر CREATE مع استبدال أمر CREATE TRIGGER بأمر ALTER TRRIGGER . أما حذف الحدث فيتم بأمر

```
DROP TRIGGER trigger_name
```

## الجداول المؤقتة Temporary Tables

هي جداول Tables يتم التعامل معها بكل الأوامر التي تعلمناها من قبل مثل create و select وdrop وغيرها، ولكنها مؤقتة . النوع الأول منها يتم حذفها تلقائيا بمجرد انتهاء

تنفيذ الجملة **Query** التي كونتها، والنوع الآخر يتم حذفه تلقائيا عندما ينتهي جميع المستفيدين من التعامل مع هذا الجدول أو انتهاء النظام الذي يشير إليها .

سنقوم الآن بتكوين هذا الجدول مع ملاحظة أن الجدول المؤقت يمكن التعامل معه من خلال المستفيد الذي يقوم بتكوينه وهو الذي يحدد استخدامه :

```
create table #store (
book_id int not null,
title varchar(40) not null,
authors varchar(50) not null,
copies int not null)
go
```

يستطيع أي عدد من المستفيدين إنشاء أو تنفيذ الجملة السابقة ويكون لكل منهم حق استخدام الجدول وتنفيذ جمل مثل **update, insert, delete** على الجدول الذي قاموا بتكوينه وبنفس الاسم الذي يشترك فيه الآخرون ولكن كل يعمل بمعزل عن المستفيدين الآخرين، لأن كل منهم سيكون له جدول مؤقت مثل **#store** خاص به ولا علاقة له بباقي المستفيدين حتى لو كان لديهم جدول مؤقت بنفس الاسم **#store** ، ويمكن لأي منهم حذف الجدول الخاص به باستخدام الأمر :

```
drop table #store
go
```

بينما سيتم حذف هذه الجداول تلقائيا بمجرد أن يقوم المستفيد بالخروج من النظام **Logout** أو **sign-out** .