قاعدة البيانات
Database

الهدف من الجلسة

في نهاية هذه الجلسة سنكون قد تعرفنا على كيفية إنشاء قاعدة بيانات.

الكلمات المفتاحية

1. قاعدة بيانات Database
2. جدول Table
3. جدول مؤقت Temporary Table
4. منظار View
5. فهرس Index
6. قادح Trigger
7. اجرائيات مخزنة Stored Procedure
8. التوابع Function

التعامل مع قواعد البيانات

تعريف

يمكن تعريف قاعدة البيانات على أنها بنية منطقية **Logical Structure** تتم إدارتها من قبل برنامج متخصص نطلق عليه اسم محرك قواعد البيانات **Database Engine** بحيث يقوم هذا المحرك بإنشاء عدة ملفات فيزيائية موافقة لكل قاعدة ليقوم هذا المحرك بعدها بإدارة عمليات التخزين والاستعلام والحذف والتعديل للبيانات المخزنة في هذه الملفات بشكل كامل.

تحوي كل قاعدة بيانات على العديد من الأغراض **Database Objects** ومن أهم هذه الأغراض:

1. الجداول
2. القيود
3. الفهارس

4. المناظير
5. الجداول المؤقتة
6. الاجرائيات المخزنة
7. التوابع المعرفة من قبل المستخدم
8. القادحات

يتم التواصل مع مختلف محركات قواعد البيانات عبر لغة برمجية موحدة هي لغة الاستعلامات المهيكلة **SQL** وهي اللغة التي تسمح لنا بكتابة التعليمات اللازمة للتعامل مع قواعد البيانات ومختلف أغراضها.

البنية الفيزيائية لقاعدة البيانات

سنقتصر على شرح لمحة موجزة للبنية الفيزيائية لقواعد البيانات المدارة من قبل المحرك Microsoft SQL Server (سنطلق عليه من الآن وصاعدا اسم "محرك مايكروسوفت" أو محرك البيانات اختصارا) حيث تختلف البنية الفيزيائية الموافقة لكل قاعدة ما بين محرك وآخر.

تتكون كل قاعدة بيانات في محرك مايكروسوفت من الملفات التالية:

1. الملفات الأساسية **Primary Files**:

يجب أن تحتوي أي قاعدة بيانات على ملف أساسي واحد وواحد فقط (ملف لاحقته **mdf**) ويحوي هذا الملف البيانات الأساسية لقاعدة البيانات أو ما نطلق عليه اسم البيانات السامية لقاعدة البيانات **Metadata** التي تشتمل على تعاريف مختلف الأغراض المخزنة في قاعدة البيانات. كما يمكن للملف الأساسي أن يحوي أي عدد من أغراض قاعدة البيانات. إن احتواء هذا الملف على البيانات السامية للقاعدة يجعل من سلامة هذا الملف أمرا جوهريا لضمان عمل القاعدة الموافقة وإن تعرض هذا الملف للعطب لأي سبب من الأسباب يجعل القاعدة الموافقة معطوبة ويكون من الصعب، ومن المستحيل أحيانا، إصلاحها.

سمح الإصدار 2012 من محرك مايكروسوفت إمكانية إضافة عدة ملفات أساسية لنفس القاعدة حيث يقوم المحرك عندها بتخزين البيانات السامية في أول ملف منها.

2. الملفات الثانوية **Secondary Files**

يمكن لأي قاعدة بيانات أن تحوي ملف ثانوي واحد أو أكثر وهي ملفات مخصصة لتخزين بيانات أغراض قاعدة البيانات (بيانات الجداول والفهارس على سبيل المثال) ولكنها لا تحوي على بيانات سامية. لاحقة هذه الملفات **ndf**.

3. ملفات المناقلات **Transaction Log Files**

وهي ملفات لتخزين تسلسل العمليات المنفذة على القاعدة ونسخ البيانات قبل وبعد أي عملية. تتميز هذه الملفات بصغر حجمها عموما الأمر الذي يسرع من عمليات القراءة والكتابة فيها مقارنة بالملفات الأساسية والثانوية التي قد تكون ذات حجوم كبيرة تصل حتى مرتبة التيرابايت. يقوم المحرك بتخزين أي عملية نقوم بها على ملف المناقلات (تخزين مؤقت)

ليقوم بنقله تلقائيا إلى ملفات البيانات (تخزين دائم) بشكل دوري (دون أي تدخل من المستخدم). تساعد هذه الملفات بتسريع عمليات التعامل مع قواعد البيانات من جهة ومن جهة أخرى تسمح باسترجاع جميع البيانات المفقودة الناتجة عن توقف المحرك بشكل مفاجئ نتيجة لانقطاع التيار الكهربائي أو أي عطل آخر للمخدم.
تحوي أي قاعدة بيانات على ملف مناقلات واحد على الأقل وفي حال وجود عدة ملفات يتم التخزين فيها تسلسليا أي بعد ملئ الملف الأول يخزن المحرك في الملف الثاني حتى يمتلئ وهكذا حتى يمتلئ آخر ملف ليقوم المحرك بعدها بالتخزين بدءا من بداية الملف الأول (مع حذف أي معلومات سابقة مخزنة فيه).

يتم إنشاء قاعدة بيانات بالتعليمة **Create Database** وحذفها بالتعليمة **Drop Database** وتعديلها بالتعليمة **Alter Database**. وسنعرض فيما يلي أمثلة عن الإنشاء بأشكاله المختلفة إضافة إلى مثال عن الحذف.

إنشاء قاعدة بيانات

1. إنشاء قاعدة بيانات بالإعدادات الافتراضية

أبسط طريقة لإنشاء قاعدة بيانات هي باستخدام الإعدادات الافتراضية للمخدم وفيها لا نحدد أي ملفات ليقوم عندها المخدم تلقائيا بإنشاء الملفات الأساسية الكافية لعمل هذه القاعدة.

لنفترض أننا نريد إنشاء قاعدة اسمها **HelloDB** بالإعدادات الافتراضية. التعليمة الموافقة:

```
Create Database HelloDB
```

يقوم المخدم تلقائيا بإنشاء ملفين موافقين للقاعدة الجديدة بحيث يشتق اسم هذين الملفين من اسم القاعدة وهذين الملفين هما:

1. الملف الرئيسي **HelloDB.mdf** بحجم أولي مقداره 3 ميغابايت وحجم نهائي غير محدود ومعدل نمو مقداره 1 ميغابايت.
2. ملف المناقلات **HelloDB_Log.ldf** بحجم أولي مقداره 1 ميغابايت وحجم نهائي غير محدود ومعدل نمو مقداره 10% من الحجم الحالي لهذا الملف.

يتم إنشاء الملفات السابقة ضمن المجلد **MSSQL/DATA** الواقع ضمن مسار تثبيت محرك قواعد البيانات.

2. إنشاء قاعدة بيانات ضمن مجلد خاص مع تسمية الملفات المكونة لها

لنفترض أننا نريد أن ننشئ قاعدة بيانات منطقية اسمها **Sales** بحيث تكون ملفاتها مخزنة على المسار الفيزيائي **C:\data** . نريد أن تحوي هذه القاعدة على ملف أساسي وحيد اسمه **MySales** وملف مناقلات وحيد اسمه **MySalesLog**.

```
CREATE DATABASE Sales
```

```
ON
( NAME = Sales_dat,
    FILENAME = 'c:\data\MySales.mdf',
    SIZE = 10,
    MAXSIZE = UNLIMITED,
    FILEGROWTH = 5 )
LOG ON
( NAME = 'Sales_log',
    FILENAME = 'c:\data\MySalesLog.ldf',
    SIZE = 5MB,
    MAXSIZE = 25MB,
    FILEGROWTH = 10% )
```

نلاحظ أن لكل ملف العديد من المواصفات أهمها:

1. الاسم المنطقي **Logical Name** الذي تحدده الواصفة **NAME** وهو اسم للاستخدام المنطقي للملف ضمن المحرك بصرف النظر عن اسمه الفيزيائي ومكان تخزينه
2. الاسم الفيزيائي للملف **File Name** وهو ما تحدده الواصفة **FileName** وتحوي المسار الكامل للملف.
3. الحجم البدائي للملف **Size** ويمثل الحجم الأولي المحجوز للملف مقدرة افتراضيا بالميغابايت.
4. الحجم الأعظمي **Max Size** وهو ما تحدده الواصفة **MaxSize** مقدرة بالميغابايت، وإذا أردنا أن يكون حجم هذا الملف غير محدودا نستخدم القيمة الخاصة **Unlimited** لنسمح للملف بالنمو تلقائيا بما تحدده الواصفة **FileGrowth** .
5. معدل النمو **File Growth** وهو ما تحدده الواصفة **FileGrowth** والتي يمكن أن تكون قيمتها إما مطلقة أو نسبية. تتم زيادة حجم الملف بعد امتلاء الحجم الحالي كاملا بالبيانات ويتم حساب الحجم الجديد وفق إحدى القاعدتين:
    i. إذا كان النمو مطلقا: الحجم الجديد = الحجم الحالي + معدل النمو
    ii. إذا كان النمو نسبيا: الحجم الجديد = الحجم الحالي * (1 + معدل النمو)

تنمو جميع ملفات البيانات نموا متجانسا بحيث يتم التخزين بشكل يتناسب مع حجوم كل من هذه الملفات.

3. إنشاء قاعدة بيانات بعدة ملفات بيانات ومناقلات

نريد إنشاء القاعدة **Archive** بحيث تكون جميع ملفاتها ضمن المسار **C:\Data** ولتحوي 3 ملفات رئيسية و 2 ملف مناقلات لتكون بالواصفات التالية:
ملفات البيانات

الاسم المنطقي	الاسم الفيزيائي	الحجم الأولي	الحجم الأعظمي	معدل النمو
Arch1	ArcData1.mdf	MB 10	غير محدود	MB 5
Arch2	ArcData2.ndf	MB 5	MB 15	MB 3
Arch3	ArcData3.ndf	MB 3	MB 12	%20

ملفات المناقلات

الاسم المنطقي	الاسم الفيزيائي	الحجم الأولي	الحجم الأعظمي	معدل النمو
ArcLog1	ArcLog1.ldf	MB 5	MB 10	MB 2
ArchLog2	ArcLog2.ldf	MB 5	MB 20	MB 5

تعليمة الإنشاء الموافقة:

```
CREATE DATABASE Archive
ON
( NAME = Arch1,
      FILENAME = 'c:\data\archdat1.mdf',
      SIZE = 10MB,
      MAXSIZE = UNLIMITED,
      FILEGROWTH = 5
),
( NAME = Arch2,
   FILENAME = 'c:\data\archdat2.ndf',
   SIZE = 5MB,
   MAXSIZE = 15,
   FILEGROWTH = 3
),
( NAME = Arch3,
   FILENAME = 'c:\data\archdat3.ndf',
   SIZE = 3MB,
   MAXSIZE = 12,
   FILEGROWTH = 20%
)
LOG ON
( NAME = Archlog1,
   FILENAME = 'c:\data\archlog1.ldf',
   SIZE = 5MB,
   MAXSIZE = 10,
   FILEGROWTH = 2),
( NAME = Archlog2,
   FILENAME = 'c:\data\archlog2.ldf',
   SIZE = 5MB,
   MAXSIZE = 20,
   FILEGROWTH = 5
)
```

حذف قاعدة بيانات

لحذف أي قاعدة بيانات نستخدم التعليمة **Drop Database**. لحذف قاعدة البيانات السابقة نستخدم الأمر التالي:

```
Drop Database HelloDB
```

التعديل على قاعدة البيانات

1- لإضافة ملف جديد:

ALTER DATABASE <database name>

ADD FILE

([NAME = <'logical file name'>,]

FILENAME = <'file name'>

[, SIZE = <size in KB, MB, GB or TB>]

[, MAXSIZE = < size in KB, MB, GB or TB >]

[, FILEGROWTH = <No of KB, MB, GB or TB |percentage>]) [,...n]

[TO FILEGROUP filegroup_name]

مثال:

ALTER DATABASE mouhannad

ADD FILE

(NAME = 'mAdd',

FILENAME = 'C:\Data\Datasample_database_add.mdf',

SIZE =10)

TO FILEGROUP [sample_database_filegroup1]

2- لتعديل حجم ملف:

ALTER DATABASE mouhannad

MODIFY FILE

(NAME = mAdd,

SIZE = 20MB)

ملاحظة: تعديل الحجم لقيمة أكبر من الحالية أما لقيمة أصغر فيعطي خطأ.

تغيير اسم قاعدة بيانات

نستخدم تابع النظام **SP_RenameDb**. مثلا لتغيير اسم القاعدة **HelloDb** إلى **NewHelloDb** ننفذ الأمر التالي:

```
sp_renamedb 'HelloDb','NewHelloDb'
```

تعليمات الاستعلام على ملفات قواعد البيانات

1. **قائمة بأسماء القواعد المدارة على المحرك الحالي وملفات كل منها**

نستخدم تابع النظام **SP_HelpDB** الذي يعيد قائمة بجميع القواعد المخدمة من المحرك الحالي مع حجم كل منها وللحصول على معلومات خاصة بقاعدة محددة ولتكن **HelloDB** نستخدم نفس التابع السابق ونمرر له اسم القاعدة المطلوبة كما يلي:

```
sp_helpdb HelloDb
```

2. **قائمة ملفات قاعدة البيانات الحالية**

نستخدم تابع النظام **SP_HelpFile** الذي يعرض بيانات مختلف القاعدة الحالية

```
use HelloDb
go
sp_helpfile
sp_helpfile HelloDB_Log
```

يقوم هذا التابع بدون أي متحولات بعرض بيانات ملفات القاعدة الحالية **HelloDB** وإذا أردنا عرض بيانات ملف محدد من ملفات هذه القاعدة فنمرر الاسم المنطقي لهذا الملف كمتحول لتابع النظام.

انتهت المحاضرة

الجداول
Tables

الهدف من الجلسة

في نهاية هذه الجلسة سوف نكون قد تعرفنا على كيفية إنشاء الجداول.

الكلمات المفتاحية

1. قاعدة بيانات Database
2. جدول Table

التعامل مع الجداول

إنشاء جدول جديد:

CREATE TABLE table_name

(<column name> <data type>

[[DEFAULT <constant expression>]

|[IDENTITY [(seed, increment)

[COLLATE <collation name>]

[NULL|NOT NULL]

[<column constraints>]

|[column_name AS computed_column_expression]

|[<table_constraint>]

[,...n]

)

[ON {<filegroup>|DEFAULT}]

[TEXTIMAGE_ON {<filegroup>|DEFAULT}]

table name:

يعبر عن اسم الجدول

Column name:

اسم الحقل

Data type:

نوع نمط البيانات للحقل.

DEFAULT:

يعبر عن القيمة الافتراضية التي يأخذها الحقل في حال لم ندخل له قيمة.

IDENTITY

يحدد أن الحقل المحدد ذو ترقيم تلقائي، **seed** يعبر عن القيمة الابتدائية للعداد الافتراضي هو 1، **increment** يحدد مقدار التزايد (أو التناقص) للعداد. الافتراضي هو 1.

COLLATE

بشكل مشابه لقاعدة البيانات يحدد نوع الفرز وهل هو حساس للأحرف، ...

NULL/NOT NULL

يحدد هل هذا الحقل يمكن أن يكون ذو قيمة **Null** أم لا؟

Column constraints:

يحدد مجموعة الشروط المطبقة على القيم المدخلة إلى هذا الحقل.

Computed Columns:

يحدد أن قيمة هذا الحقل تشتق من حقول اخرى (بشرط أن تكون من نفس الجدول).

في الاصدارات قبل SQL 2000 لها المشاكل التالية: لا يمكن أن تدرج ضمن الاستعلامات الجزئية، لا يمكن أن نطبق عليها الفهارس أو القيود كالمفتاح الأساسي.

Table Constraints:

تطبق قيود على الادخالات على الجدول ولكن لقيم عدة حقول مع بعضها.

On:

نحدد من خلالها أي ملف بيانات سوف يتم التخزين عليه، بدون تحديد سوف يكون ملف التخزين هو Primary

TEXTIMAGE_ON:

خاص بالجداول التي تحتوي حقول من نوع **nText,image,text** بهدف جعل ملف البيانات لهذه الحقول مختلف عن ملف بيانات باقي الحقول.

مثال:

```
USE Accounting
CREATE TABLE Employees
(
EmployeeID int IDENTITY NOT NULL,
FirstName varchar(25) NOT NULL,
MiddleInitial char(1) NULL,
LastName varchar(25) NOT NULL,
Title varchar(25) NOT NULL,
SSN varchar(11) NOT NULL,
Salary money NOT NULL,
```

PriorSalary money NOT NULL,

LastRaise AS Salary - PriorSalary,

HireDate smalldatetime NOT NULL,

TerminationDate smalldatetime NULL,

ManagerEmpID int NOT NULL,

Department varchar(25) NOT NULL))

تعديل جدول موجود مسبقا:

إما أن نعدل مواصفات حقل موجود مسبقاً

ALTER TABLE Employees

Alter column LastName varchar(35) NOT NULL

شروط تعديل حقل في جدول:

لا يمكن تعديل الحقول التالية:

1. الحقل من نمط **timestamp**.
2. حقل من نمط **computed column** أو يشارك في حساب حقل آخر.
3. للحقل فهرس ما عدا الحقول النصية حيث تسمح بالتعديل بشرط أن يكون الحجم الجديد أكبر أو يساوي الحجم القديم.
4. مشارك في الإحصائيات المولدة (يجب إزالة الإحصائيات ب **DROP STATISTICS**)
5. مستخدم في مفتاح ثانوي أو رئيسي.
6. مطبق عليه قيد **CHECK** or **UNIQUE** لكن يمكن تغيير حجم الحقل.
7. يجب الانتباه في حال تعديل حقل بحيث نغير نمط المعطيات ضمن جدول يحوي بيانات.

أو أن نضيف حقول جديدة للجدول:

ALTER TABLE Employees

```
ADD
DateOfBirth datetime NULL,
LastRaiseDate datetime NOT NULL|
```

أو أن نحذف حقول موجودة:

```
ALTER TABLE Employees
Drop column col-name
```

حذف قاعدة بيانات:

```
DROP DATABASE Accounting
```

حذف جدول أو أكثر:

```
DROP TABLE Customers, Employees
```

انتهت المحاضرة

القيود والتكامل المرجعي
Constraints and Referential Integrity

الهدف من الجلسة

في نهاية هذه الجلسة سوف نكون قد عرضنا مفهوم التكامل المرجعي، كيفية فرض التكامل المرجعي عبر مجموعة من القيود.

الكلمات المفتاحية

Constraints, Primary Key, Foreign Key, Check Constraints, Unique Constraints, Default Constraints.

سنتعرف في هذه الجلسة على:

1. قيد المفتاح الأولي Primary Key Constraint.
2. القيود الفريدة Unique Constraints.
3. قيد المفتاح الثانوي Foreign Key Constraint.
4. قيود التحقق Check Constraints.
5. القيود الافتراضية Default Constraints.
6. قيد Not Null
7. قيد Identity

مقدمة

تمثل القيود منطق عمل Business Logic الذي يقوم مخدم قواعد البيانات بضمان تحقيقه على حقول (أعمدة) جداول قاعدة البيانات. وتشمل هذه القيود مجال القيم التي يمكن إدخالها في عمود إضافة إلى التكامل المرجعي للبيانات.

عند إضافة أو حذف أو تعديل أي بيانات في جدول ما من جداول قاعدة بيانات، فإن مخدم قواعد البيانات يراعي جميع القيود المعرفة على هذه الجدول قبل تنفيذ العملية ولا ينفذ هذه العملية إذا كانت تؤدي إلى أي خلل في القيود المفروضة.

مثال: إذا كان مفتاح فريد **Unique** على جدول ما، فلا يمكنك إضافة أو تعديل سطر ما بحيث يصبح لديك تكرارات في قيمة المفتاح الفريد.

مثال: إذا كان لديك قيد على مجال القيم لعمود ما فإن إضافة أو تعديل قيمة هذا العمود لسطر ما خارج هذا المجال سوف تؤدي إلى توليد خطأ من قبل مخدم قواعد البيانات.

تراعي مخدمات قواعد البيانات التكاملات المرجعية المعرفة بين الجداول، فلا يمكنك **مثلا** إضافة سطر في الجدول الممثل للطرف كثير إن لم يكن هناك سطر موافق في الجدول الممثل للطرف واحد.

وبالمثل: فإنه لا يمكنك حذف سطر من الطرف واحد ما لم تكن جميع الأسطر الموافقة له في الطرف كثير قد حذفت أو أنه لا يوجد أي سطر موافق في الطرف كثير. **(يوجد استثناءات لهذه القاعدة؟)**

تصنيف القيود:

1-Domain Constraints: هي قيود تطبق على حقل معين.

2-Entity Constraints: هي قيود تطبق على الأسطر في الجدول.

3-Referential Integrity Constraint: قيود تطبق على حقول من جدولين مختلفين.

تجعل القاعدة proactive بدلا من أن تكون reactive ضد أخطاء الإدخال.

قيد المفتاح الأولي Primary Key Constraint:

يضمن قيد المفتاح الأولي عدم تكرار القيم في مجموعة الأعمدة المكونة له كما يمنع إدخال القيمة **NULL** في أي من أعمدته. وهو يستخدم لضمان الوحدانية. فمثلا في الجدول Authors من قاعدة البيانات Pubs فإن العمود au_id هو مفتاح أولي يضمن وحدانية رقم كل مؤلف من جهة ومن جهة أخرى يضمن عدم إسناد قيمة **NULL** لهذا العمود من أجل أي مؤلف.

أمثلة:

1. **تحديد المفتاح الأولي عند إنشاء الجدول:** لإنشاء الجدول Department، نكتب التعليمة التالية:

EX 01

```
CREATE TABLE [Department]
CREATE TABLE [Department]
(
  [deptNo] [int] NOT NULL ,
  [deptName] [varchar] (50)  NOT NULL ,
  [mangerSN] [int] NOT NULL ,
  [managerStartDate] [datetime] NULL
```

(التعريف السابق لا يضمن وحدانية وعدم انعدام قيمة رقم القسم وللقيام بذلك نعدل التعريف السابق كما يلي:

```
CREATE TABLE [Department]
(
  [deptNo] [int] NOT NULL PRIMARY KEY,
  [deptName] [varchar] (50)  NOT NULL ,
  [mangerSN] [int] NOT NULL ,
  [managerStartDate] [datetime] NULL
)
```

2. **تحديد المفتاح الأولي بعد إنشاء الجدول:** في المثال السابق تم إنشاء الجدول وتعريف المفتاح الأولي بشكل مباشر. يمكن إنشاء الجدول ومن ثم تعريف المفتاح الأولي على هذا الجدول (إذا احتوى الجدول على بيانات فإن توليد المفتاح الأولي سينجح فقط إذا لم تحتوِ أعمدة المفتاح الأولي على قيم مكررة).

ملاحظة: يمكن أن يحوي الجدول على مفتاح أولي واحد على الأكثر.

EX 02

```
DROP TABLE [Department]
GO
CREATE TABLE [Department]
(
  [deptNo] [int] NOT NULL ,
  [deptName] [varchar] (50)  NOT NULL ,
  [mangerSN] [int] NOT NULL ,
  [managerStartDate] [datetime] NULL )
```

لإضافة القيد بعد إنشاء الجدول نعدل تعريف الجدول كما يلي:

```
ALTER TABLE [Department]
ADD CONSTRAINT [PK_Department] PRIMARY KEY
(
 [deptNo]
)
```

يمكن إلغاء المفتاح الأولي إذا تم إنشاؤه كما في التمرين EX 02 في أي لحظة دون أي ضياع في البيانات. (وذلك لأننا قمنا بتحديد اسم المفتاح بشكل صريح). لحذف قيد معرف على جدول نستخدم التعليمة التالية:

EX 03

```
ALTER TABLE [Department]
DROP CONSTRAINT [PK_Department]
```

القيود الفريدة Unique Constraints:

يضمن القيد الفريد عدم تكرار القيم في الأعمدة المكونة له بين الأسطر المختلفة للجدول. لكنه يسمح بإدخال قيمة NULL في أي من أعمدته.

ملاحظة: يمكن للجدول أن يحتوي على أكثر من قيد فريد، كما يمكن لنفس العمود (الحقل) أن يشارك بعدة مفاتيح فريدة.

مثال

في الجدول السابق نلاحظ أن اسم القسم يجب ألا يتكرر وبالتالي نعدل الجدول السابق بحيث نضيف قيد فريد على اسم القسم.

1. **تحديد قيد فريد أثناء إنشاء الجدول:**

EX 04

```
DROP TABLE [Department]
GO
CREATE TABLE [Department]
(
  [deptNo] [int] NOT NULL ,
  [deptName] [varchar] (50) NOT NULL UNIQUE,
  [mangerSN] [int] NOT NULL ,
  [managerStartDate] [datetime] NULL
)
```

2. **تحديد قيد فريد بعد إنشاء الجدول:**

EX 05

```
DROP TABLE [Department]
GO
CREATE TABLE [Department]
(
```

```
  [deptNo] [int] NOT NULL ,
  [deptName] [varchar] (50) NOT NULL ,
  [mangerSN] [int] NOT NULL ,
  [managerStartDate] [datetime] NULL
)
GO
ALTER TABLE [Department]
ADD CONSTRAINT [UK_deptName] UNIQUE
(
 [deptName]
)
```

يمكننا حذف القيود الفريدة في أي لحظة شريطة معرفة اسم هذا القيد كما تبين التعليمة التالية:

EX 06

```
ALTER TABLE [Department]
DROP CONSTRAINT [UK_deptName]
```

قيد المفتاح الثانوي Foreign Key Constraint:

يعمل المفتاح الثانوي بالتزامن مع مفتاح أولي أو مع مفتاح فريد Unique. يضمن إنشاء مفتاح ثانوي في جدول ما كون القيم الممكن إدخالها في أعمدة هذا المفتاح مطابقة لبيانات المفتاح الأولي أو الفريد الموافق. تسمح هذه التقنية بالحد من تكرار البيانات في قاعدة المعطيات، فمثلا نكتفي بإدخال بيانات الأقسام لمرة واحدة ثم نربط جدول الموظفين بجدول الأقسام لتحديد القسم الذي يتبع له كل موظف عن طريق تحديد رقم القسم مع كل موظف. كما أن المفتاح الثانوي يضمن تكامل المعطيات حيث يتحقق مخدم البيانات من كون الرقم المدخل في حقل رقم القسم موجود فعلا في جدول الأقسام.

تجدر الإشارة إلى أنه لا يمكننا بعد تعريف المفتاح الخارجي أن نقوم بحذف المفتاح الرئيسي (أو الفريد) الموافق له ما لم نقم أولا بحذف المفتاح الخارجي.

مثال:

1. **إنشاء مفتاح ثانوي عند إنشاء الجدول**: لإنشاء جدول الموظفين بحيث يشير الحقل deptNo إلى الحقل الذي يحمل نفس الاسم في الجدول Department ننفذ التعليمة التالية:

EX 07

```
DROP TABLE [Department]
GO
CREATE TABLE [Department]
(
```

```
  [deptNo] [int] NOT NULL ,
  [deptName] [varchar] (50)  NOT NULL ,
  [mangerSN] [int] NOT NULL ,
  [managerStartDate] [datetime] NULL
)
GO
ALTER TABLE [Department]
ADD CONSTRAINT [PK_Department] PRIMARY KEY
(
 [deptNo]
)
GO
CREATE TABLE [Employee] (
 [empSN] [int] NOT NULL ,
 [fName] [varchar] (50) ,
 [lName] [varchar] (50) NOT NULL ,
 [birthDate] [datetime] NULL ,
 [hireDate] [datetime] NOT NULL ,
 [address] [varchar] (50) NULL ,
 [sex] [bit] NOT NULL ,
 [salary] [money] NOT NULL ,
 [managerSN] [int] NULL ,
 [deptNo] [int] NULL
  FOREIGN KEY REFERENCES Department(deptNO)
)
```

2. **إنشاء مفتاح ثانوي بعد إنشاء الجدول**: يمكن بنفس الشكل بناء مفتاح ثانوي بعد إنشاء الجدول وفي هذه الحالة يجب أن تكون البيانات الموجودة حاليا في حقول المفتاح الثانوي تكافي بيانات موجودة في حقل (أو حقول) المفتاح الأولي أو الفريد الموافق وإلا فإن عملية الإنشاء تكون فاشلة.

EX 08

```
DROP TABLE [Employee]
DROP TABLE [Department]
GO
CREATE TABLE [Department]
(
  [deptNo] [int] NOT NULL ,
  [deptName] [varchar] (50)  NOT NULL ,
  [mangerSN] [int] NOT NULL ,
  [managerStartDate] [datetime] NULL
)
go
ALTER TABLE [Department]
ADD CONSTRAINT [PK_Department] PRIMARY KEY
```

```
(
 [deptNo]
)
GO
CREATE TABLE [Employee] (
 [empSN] [int] NOT NULL ,
 [fName] [varchar] (50) ,
 [lName] [varchar] (50) NOT NULL ,
 [birthDate] [datetime] NULL ,
 [hireDate] [datetime] NOT NULL ,
 [address] [varchar] (50) NULL ,
 [sex] [bit] NOT NULL ,
 [salary] [money] NOT NULL ,
 [managerSN] [int] NULL ,
 [deptNo] [int] NULL
)
GO
ALTER TABLE Employee
ADD CONSTRAINT [FK_Employee_Departmet]
 FOREIGN KEY (deptNo)
  REFERENCES Department(deptNo)
```

3. **حذف مفتاح ثانوي:**

EX 09

```
ALTER TABLE Employee
 DROP CONSTRAINT [FK_Employee_Departmet]
```

يمكن حذف المفتاح الثانوي في أي وقت بدون أي اعتبارات إضافية.

4. **التحديث التلقائي لبيانات المفتاح الثانوي:**

ماذا يحدث إذا حذفنا قسما وكان هناك موظفين في هذا القسم؟

وماذا يحدث لو عدلنا رقم أحد الأقسام الذي يحوي موظفين؟

مبدئيا لا يمكن حذف قسم يحوي موظفين، كما لا يمكن تعديل رقم قسم يحوي موظفين. إذا للقيام بأي من هاتين العمليتين لابد من حذف جميع الموظفين أولا قبل تنفيذ أي عملية على الأسطر الموافقة من جدول الأقسام.

لحل المشكلتين السابقتين نعرف واصفتين لعملية تعريف المفتاح الثانوي:

1. **ON UPDATE CASCADE** : تعديل بيانات القسم ينتقل بشكل آلي إلى جميع الموظفين المرتبطين بهذا القسم

2. **ON DELETE CASCADE**: حذف أي قسم يؤدي إلى حذف جميع الموظفين في هذا القسم.

مثال:

EX 10

```
ALTER TABLE [Employee]
 DROP CONSTRAINT [FK_Employee_Department]
GO
ALTER TABLE [Employee]
 ADD  CONSTRAINT [FK_Employee_Department]
  FOREIGN KEY([deptNo])
  REFERENCES [Department] ([deptNo])
   ON UPDATE CASCADE
  ON DELETE CASCADE
```

قيود التحقق Check Constraints:

تفرض قيود التحقق شروطا على القيمة value التي يمكن أن توضع في عمود أو مجموعة أعمدة أو تفرض قيودا على تنسيق format القيمة التي توضع في عمود أو مجموعة أعمدة.

مثلا في جدول الموظفين: لكل موظف تاريخ ميلاد birthDate وتاريخ توظيف hireDate. يقبل مخدم قواعد البيانات أي تاريخ للعمودين دون أي قيود. لفرض قيد كون تاريخ الميلاد أكبر من 1950 وكون تاريخ التوظيف أكبر من تاريخ الميلاد + 18 سنة. نعرف القيدين التاليين:

EX 11

```
ALTER TABLE [Employee]
 DROP CONSTRAINT [CK_Employee_birthDate]
GO
ALTER TABLE [Employee]
 DROP CONSTRAINT [CK_Employee_hireDate]
GO
ALTER TABLE [Employee]
 ADD  CONSTRAINT [CK_Employee_birthDate]
 CHECK  ((datepart(year,[birthDate])>(1950)))
GO
ALTER TABLE [Employee]
 ADD  CONSTRAINT [CK_Employee_hireDate]
 CHECK
((datepart(year,[hireDate])>=(datepart(year,[birthDate])+(
18))))
```

تجدر الإشارة إلى أن جميع الأعمدة المشاركة في قيد تحقق يجب أن تكون من نفس الجدول (سنرى لاحقا كيفية فرض قيود تحقق تشمل حقول من عدة جداول في جلسات لاحقة).

مثال:

EX 12

نريد إنشاء جدول Ages يتضمن أسماء Name وأعمار Age مجموعة من الأطفال تتراوح بين سنة و 12سنة. نكتب الصيغة:

```
CREATE TABLE Ages (Name varchar(50) Not Null , Age
INT
Constraint CheckAge CHECK (Age between 1 And 12));
```

ويمكن أن نكتب صيغة تحقق نفس الغرض مع حذف اسم القيد CheckAge إذا كنا لا نريد اسم للقيد.

```
CREATE TABLE Ages (Name varchar(50) Not Null ,
Age INT CHECK(Age between 1 And 12));
```

يمكن لشرط القيد Check أن يتألف من تعبيرات منطقية تحتوي على عمليات منطقية مثل And أو Or فمثلاً إذا أردنا السماح بإدخال عمر بين 1و12 أو يساوي 15نكتب الصيغة:

```
CREATE TABLE Ages
(Name varchar(50) Not Null ,
Age INT CHECK(Age = 15 OR Age between 1 And 12));
```

يمكن وضع أكثر من قيد Check على حقل وحيد فمثلاً إذا أردنا السماح بالقيم للعمر بين 1و12 عدا العمر 3 نكتب الصيغة:

```
CREATE TABLE Ages
(Name varchar(50) Not Null , Age INT,
Constraint CheckAge1 CHECK (Age between 1 And 12),
Constraint CheckAge2 CHECK (Age <> 3));
```

انتبه:

لا يطبق القيد Check عندما لا يتلقى الحقل أي إدخالات (أي عندما تكون قيمته Null) .

القيود الافتراضية Default Constraints:

تحديد قيمة تلقائية لحقل ما. يأخذ هذا الحقل تلك القيمة حين لا يتم إسناد أية قيمة بديلة.

EX 13

```
ALTER TABLE [Departmenttt]

   ADD CONSTRAINT [def_Departmentt] DEFAULT 5 for [mangerSN]
```

مثال:

لنفترض أننا ندخل بيانات عدد كبير من الموظفين في اليوم الواحد. سيكون إدخال تاريخ التوظيف موحدا لجميع هؤلاء الموظفين وبالتالي سيكون عملا تكراريا. يمكن افتراض أن جميع الموظفين المضافين في هذا اليوم لهم تاريخ توظيف موافق لتاريخ اليوم الحالي وبالتالي نضع قيد قيمة افتراضية على العمود hireDate.

EX 14

```
DROP TABLE [Employee]
GO
CREATE TABLE [Employee]
(
 [empSN] [int] NOT NULL, CONSTRAINT PK_Employee primary key,

 [fName] [varchar](50)  NULL,
 [lName] [varchar](50) NOT NULL,
 [birthDate] [datetime] NULL,
 [hireDate] [datetime] NOT NULL CONSTRAINT
[DF_Employee_hireDate]  DEFAULT (getdate()),
 [address] [varchar](50)  NULL,
 [sex] [bit] NOT NULL,
 [salary] [money] NOT NULL,
 [managerSN] [int] NULL,
 [deptNo] [int] NULL,
 CONSTRAINT [PK_Employee] PRIMARY KEY
 (
  [empSN] ASC
 )
)
GO
ALTER TABLE [Employee]
 ADD  CONSTRAINT [FK_Employee_Department]
 FOREIGN KEY([deptNo])
  REFERENCES [Department] ([deptNo])
```

```
 ON UPDATE SET NULL
 ON DELETE CASCADE
GO
ALTER TABLE [Employee]
 ADD  CONSTRAINT [CK_Employee_birthDate]
 CHECK  ((datepart(year,[birthDate])>(1950)))
GO
ALTER TABLE [Employee]
 ADD  CONSTRAINT [CK_Employee_hireDate]
 CHECK
((datepart(year,[hireDate])>=(datepart(year,[birthDate])+(18
))))
```

القيد NOT NULL:

يستخدم هذا القيد في حال أردنا أن نمنع إدخال القيمة Null في الحقل المراد تقييده. يكفي لتطبيق هذا القيد إضافة التعبير Not NULL عند إنشاء الجدول.

EX 15

إذا أردنا إنشاء جدول بأسماء الموظفين وتوصيف عملهم بحيث نمنع إعطاء القيمة Null لأي حقل من حقول الجدول نستخدم الصيغة:

```
CREATE TABLE Employees (name varchar(40) NOT NULL ,
Job varchar(50) NOT NULL);
```

إذا حاولنا تنفيذ الاستعلام التالي بغرض إضافة سجل إلى جدول الموظفين:

Insert into Employees(name) values('Adel')

سنلاحظ هذا التعبير لن يعمل وسيولد رسالة خطأ تفيد بضرورة إعطاء قيمة للحقل Job، كون القيمة التلقائية التي سيأخذها الحقل Job في حال عدم إدخال أي قيمة له هي القيمة Null.

القيد IDENTITY:

توفر قواعد البيانات آلية لتوليد قيم عددية بصورة آلية كقيم لحقل ما عند إضافة سجل جديد إلى الجدول. يمكن استخدام هذه التقنية بالاشتراك مع القيد PRIMARY KEY لتوليد قيم تسلسلية تلقائية في حقل يكون هو حقل المفتاح الرئيسي للجدول.

نستخدم في SQL Server التعبير IDENTITY.

مثال:

نود إنشاء جدول بأرقام تسلسلية للطلاب وأسمائهم:

الأرقام تبدأ من 100 وتتزايد بمقدار 1:

EX 16

```
CREATE TABLE Students (Name varchar(50) ,
ID INT IDENTITY (100,1) PRIMARY KEY NOT NULL);
```

في حال لم نحدد البذرة لبدء العد (100) والتزايد (1) تكون القيمة التلقائية للبذرة والتزايد هي (1).

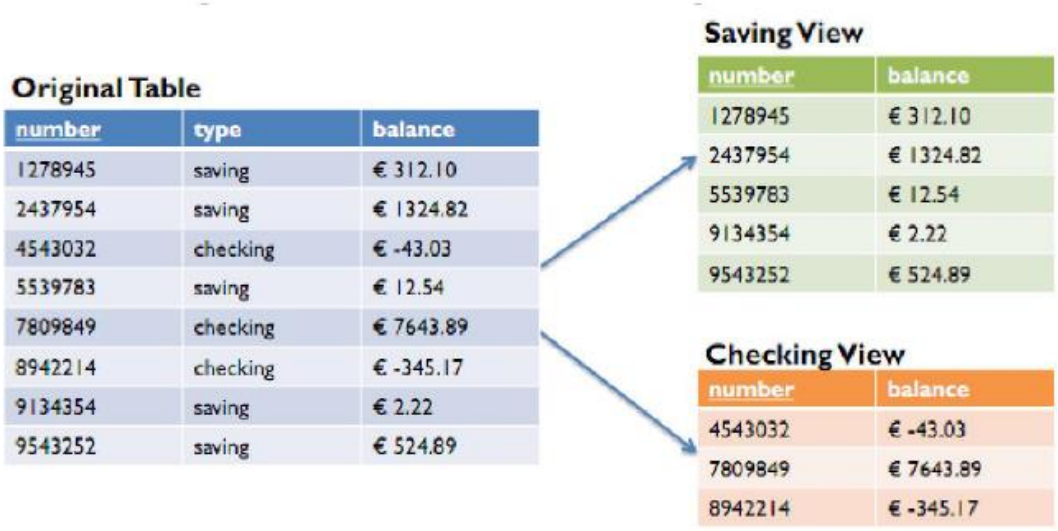
انتهت المحاضرة

المناظير
VIEWS

مقدمة:

استخدمنا الكثير من صيغ SQL المعقدة نسبياً للاستعلام عن البيانات . لكن، سيصبح من المرهق كتابة هذه الصيغ في كل مرة نريد استخدامها إذا كانت هذه العمليات تستخدم بكثرة . لهذا السبب، تتضمن SQL تعابير خاصة تمكننا من توليد جدول افتراضي (وهو عبارة عن استعلام يعود لنا بجزء من جدول أو عدة جداول) ندعوه منظار.

- معظم التطبيقات تستخدم جدول افتراضي (ندعوه منظر View) على قاعدة البيانات بدلا من استخدام كامل بيانات الجدول
- هناك نوعين من المناظير
  - بشكل عام المناظير فقط للقراءة
  - ولكن يمكن ان تكون قابلة للتعديل



خواص المنظار:

- يعتبر المنظار أحد أغراض قاعدة البيانات ، بالتالي يمكن إنشاء العمليات الأساسية عليها من إنشاء وحذف وتعديل (Create, Drop ,Alter).
- تساعد المناظير في تجميع البيانات التي يرجعها استعلام ما من جدول أو عدة جداول ، فهي توفر جداول افتراضية تحتوي على مجموعة البيانات المطلوبةمن الجداول الأساسية.
- يعامل المنظار كأي جدول من جدا ول قاعدة البيانات. فالفرق الأساسي بينه وبين الجدول هو أن البيانات التي تحتويه مخزنة فيزيائياً في الجداول ، بالتالي عند تعديل بيانات المنظار وكونها بيانات افتراضية مخزنة فيزيائياً في الجداول الأساسية للقاعدة سيؤدي ذلك إلى تعديل بيانات الجدول الأساسي (ويمكن تعديل بنية المنظار كما سنرى لاحقاً). هذا يقودنا إلى نتيجة أنه يمكن تعديل (إدخال) بيانات الجدول من خلال المنظار (ولكن بشروط).
- من ناحية الأمان:

- تمنع المناظير المستخدمين من الوصول إلى الجداول الأصلية في قواعد البيانات ، كون البيانات التي يحتويها هي جزء من البيانات الموجودة في الجداول الأصلية (أي أنها ناتجة عن استعلام).
- يمكن منع المستخدم من تعديل بيانات الجدول الأساسي من خلال منظار ، حيث يمكن أن يشكل نسخة بيانات قابلة للقراءة فقط يمكن تخصيصها للمستخدمين ذوي الصلاحيات المنخفضة فيما إذا قمنا بتحديد صلاحيات هذا الغرض.

إنشاء منظار: Create View:

الصيغة العامة لإنشاء منظار في SQLServer:

```
CREATE VIEW view_name AS query;
```

حيث:

view_name: اسم المنظار.

query: الاستعلام الذي يعيد لنا مجموعة البيانات المطلوبة (يمكن أن يكون من جدول أو عدة جداول).

مثال:

students

matNr	firstName	lastName	sex
1005	Clark	Kent	m
2832	Louise	Lane	f
4512	Lex	Luther	m
5119	Charles	Xavier	m

exams

matNr	crsNr	result
1005	100	3.7
2832	102	2.0
1005	101	4.0
2832	100	1.3

resultsCrs100

student	result
Louise Lane	1.3
Clark Kent	3.7

```
CREATE VIEW resultsCrs100 (student, result) AS
  SELECT (firstName || ' ' || lastName), result
    FROM exams e, students s
    WHERE crsNr = 100 AND s.matNr = e.matNr
```

مثال:

نفترض أنه لدينا استعلام معقد يعيد قيم من جدولين مرتبطين بأسلوب Inner Join، قمنا بإنشاء المنظار ليحتوي على بيانات الحقول المعادة من هذا الاستعلام، والصيغة كالتالي:

```
CREATE VIEW MySimpleView Projects.projectName ,
count (Tasks.taskID) AS TasksNumber
From Tasks Inner Join Projects
ON Tasks.projectID = Projects.projectID
Group by projectName;
```

حيث قمنا بإنشاء المنظار MySimpleView ويحتوي على الحقول projectName و TasksNumber.

وبعد إنشاء هذا المنظار يمكننا ببساطة إنشاء أي استعلام والتعامل معه على انه جدول (لكنه افتراضي) من خلال الصيغة التالية:

```
Select projectName from MySimpleView;
```

تعديل البيانات من خلال المنظار:

يمكن تعديل بيانات المنظار الافتراضية كما تحدثنا سابقاً ولكن أي تعديل في هذه البيانات سيعدل ببيانات الجدول الأساسي بالتالي، يمكن أيضاً تعديل بيانات الجدول من خلال المنظار. ولكن لا يمكن تطبيق التعابير Insert، Update و Delete على المنظار كما في الجداول لتعديل بياناته إلا في حال توافرت الشروط التالية:

- يجب أن لا يحتوي الاستعلام الخاص بالمنظار أي تابع تجميعي كما يجب ألا يستخدم تعبير Group By.
- يجب أن لا يحتوي الاستعلام الخاص بالمنظار التعبير Top أو Distinct.
- يجب أن لا يحتوي الاستعلام الخاص بالمنظار حقول تم إجراء عمليات حسابية عليها أو تم حسابها.

تعديل بنية المنظار: Alter View:

لتعديل بنية منظار ما في SQLServer يمكننا تغيير توصيفه باستخدام الصيغة التالية:

```
ALTER VIEW view_name AS newQuery;
```

مثال:

إذا كان لدينا الجدول Studio الحاوي على أسماء وأرقام الاستديوهات studioNumber, studioName وأرقام للبرامج ProgramNumber التي ستصور في مركز تلفزيوني. وإذا كان لدينا الجدول Actors الحاوي على أسماء الممثلين ActorName وأسماء البرامج التي يشتركون في تمثيلها programeName. وإذا أردنا تعديل بنية المنظار ActorsStudios تصبح الصيغة:

```
ALTER VIEW ActorsStudios AS
Select actorName , studioName from Actors Inner Join Studios
ON Studios.studioNumber = Actors.studioNumber;
```

Drop View: حذف منظار:

يمكننا حذف منظار بصورة كاملة من قاعدة البيانات، ولكن حذف منظار لا يعني حذف الأصول أو الجداول الأساسية التي دخلت في الاستعلام الذي أنشأ المنظار.

يستخدم التعبير DROP VIEW في SQLSERVER بالصيغة التالية:

```
DROP VIEW view_name;
```

مثال:

فإذا كان لدينا المنظار MyView الذي أنشأناه بالصيغة:

```
CREATE VIEW MyView AS
Select * from MyTable;
```

لحذف المنظار MyView نستخدم:

```
DROP VIEW MyView;
```

بعد تنفيذ هذه الصيغة سيتم حذف المنظار MyView وستفشل الاستعلامات على هذا المنظار مثل الاستعلام:

```
Select* from MyView;
```

كما ذكرنا سابقاً أن حذف منظار لا يعني حذف الجداول الأصلية التي دخلت في الاستعلام الذي أنشأ المنظار ففي مثالنا، لا يتأثر الجدول MyTabl e بحذف المنظار MyView.

استخدام الكائن View:

إدخال البيانات باستخدام View:

تخيل أن لديك جدول الموظفين وتريد إنشاء قائمة بأسمائهم فقط. لأجل ذلك، يمكنك إنشاء View على النحو التالي:

```
CREATE VIEW dbo.EmployeesNames
AS
SELECT FirstName,
       LastName,
       LastName + ', ' + FirstName AS FullName FROM Persons;
GO
```

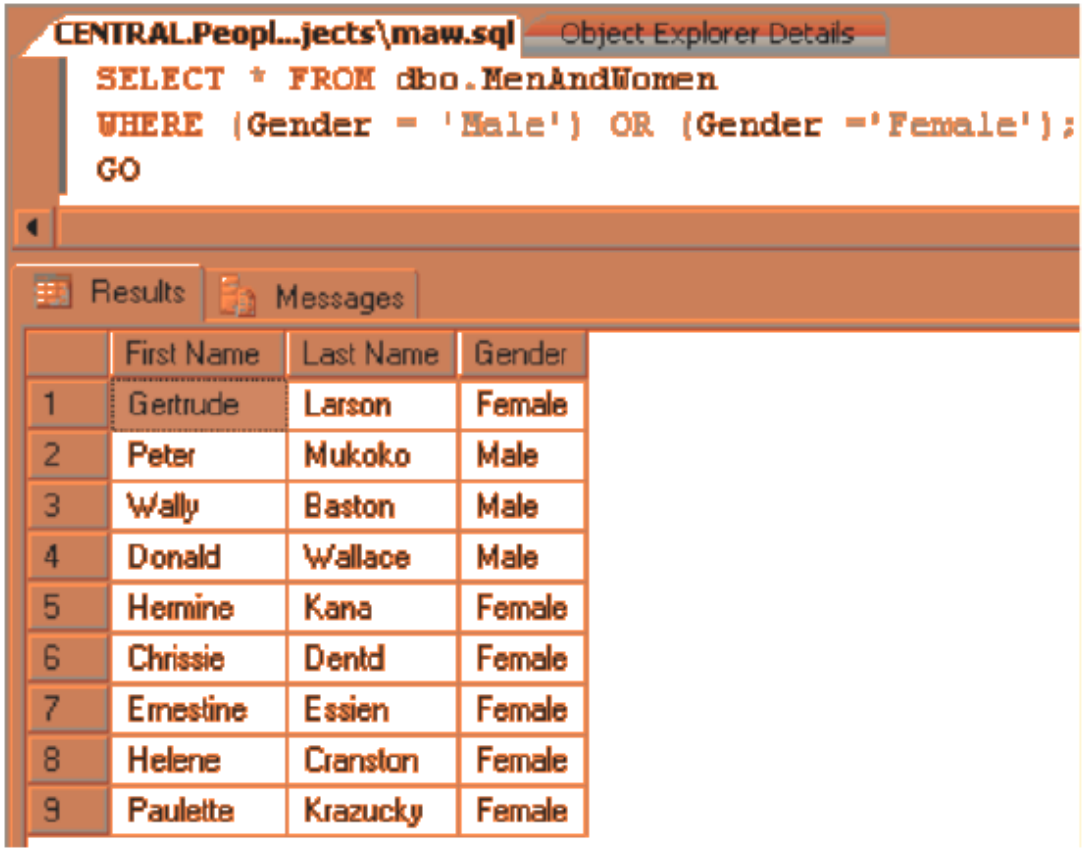
يستند كائن View السابق إلى جدول، حيث يمكنك القيام بإدخال البيانات إلى هذا الجدول من خلال كائن View، عوضا عن إدخال البيانات مباشرة إلى الجدول. للقيام بذلك، يمكنك إتباع نفس القواعد التي استعرضنا في درس سابق. مثال:

```
INSERT INTO dbo.EmployeesNames(FirstName, LastName)
VALUES('Peter', 'Justice');
```

إدخال البيانات باستخدام View يعني إدخال البيانات في الجدول الأصل، وهذا يعني أن الجدول سيتم تحديثه تلقائيا. استنادا إلى هذه الميزة، يمكنك إنشاء View خصيصا لتحديث البيانات في الجدول، بحيث تقوم بعرض الحقول التي يسمح بتحديثها فقط.

كائن View والمعايير:

من اهم خصائص الاستعلام انه يتميز بالمرونة، حيث يمكنك الوصول إلى اية نتيجة من خلال بيانات معقدة عن طريق معالجة البيانات باستخدام الشروط. يمكن استخدام الشروط لعرض محتوى View بدلا من الجداول التي من شأنها تعقيد الاستعلام أكثر. يمكنك عند إنشاء View ضمن عبارة SELECT، تحديد الحقول وفق ترتيب معين، وإضافة معايير لاستبعاد بعض السجلات. مثال:



	First Name	Last Name	Gender
1	Gertrude	Larson	Female
2	Peter	Mukoko	Male
3	Wally	Baston	Male
4	Donald	Wallace	Male
5	Hermine	Kana	Female
6	Chrissie	Dentd	Female
7	Ernestine	Essien	Female
8	Helene	Cranston	Female
9	Paulette	Krazucky	Female

كائن View والتوابع:

يمكن إنشاء View أكثر تعقيدا أو ذات وظائف متقدمة، ويمكن إشراك التوابع المبرمجة كما يمكن استخدام التوابع المضمنة مع SQL Server.

إذا لم تكن التوابع المضمنة تفي بالغرض، يمكنك إنشاء تابع خاص بك. مثال:

```
USE People;
GO
CREATE FUNCTION dbo.GetFullName
(
        @FName varchar(20),
        @LName varchar(20)
)
RETURNS varchar(41)
AS
BEGIN
        RETURN @LName + ', ' + @FName;
END
GO
```

بعد إنشاء التابع الذي تريد، يمكنك إدراجه في جسم استعلام View إذا لزم الأمر. مثال:

```
CREATE VIEW dbo.MyPeople
AS
SELECT dbo.GetFullName(FirstName, LastName) AS [Full Name],
       dbo.Genders.Gender
FROM   Genders INNER JOIN dbo.Persons
ON     dbo.Genders.GenderID = dbo.Persons.GenderID;
```

CENTRAL.Peopl... dbo.MyPeople | Object Explorer Details

	Full Name	Gender
▶	Larson, Gertrude	Female
	Mukoko, Peter	Male
	Baston, Wally	Male
	Wallace, Donald	Male
	Kana, Hermine	Female
	Dentd, Chrissie	Female
	Essien, Ernestine	Female
	Cranston, Helene	Female
	Palau, Robert	Unknown
	Krazucky, Paulette	Female
*	*NULL*	*NULL*

انتهت المحاضرة

الإجراءات المخزنة
StoredProcedure

مقدمة:

تعتبر الإجرائيات المخزنة من المواضيع الهامة كونها تعطي العديد من الميزات لـSQL .

تغطي هذه الجلسة بعض النقاط المتعلقة بإدارة الإجرائيات المخزنة وكيفية التعامل معها في قواعد البيانات المختلفة.

سنتعرف في هذا القسم على:

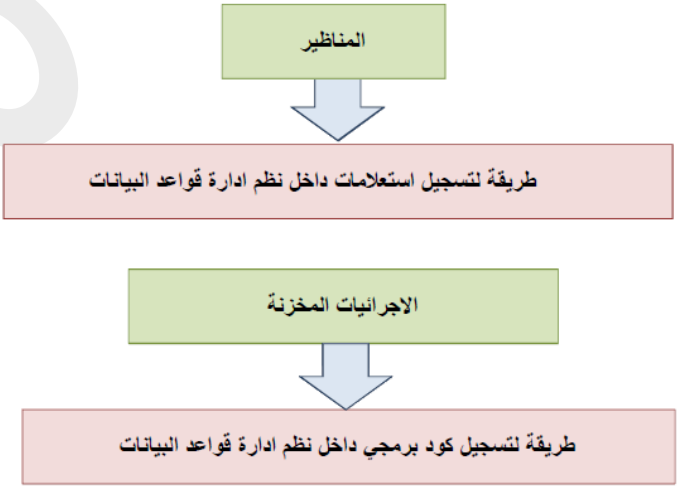
- مفهوم الإجرائيات المخزنة وفائدتها
- إدارة الإجرائيات المخزنة (إنشاؤها، حذفها وتعديلها)
- المتغيرات وتعيينها
- معاملات الدخل والخرج

خواص الإجراء المخزن:

تكلمنا فيما سبق عن المناظير وذكرنا أنها تسهل تمثيل الاستعلامات المعقدة دون الحاجة إلى إعادة كتابة هذه الاستعلامات.

الإجرائيات المخزنة عبارة عن آلية تلعب دور مشابه للدور الذي تلعبه المناظير ولكن بصورة موسعة.

- إذ لا تنحصر قدرة الإجرائيات المخزنة على استعلام واحد بل يمكن لإجرائية مخزنة واحدة أن تقوم بمجموعة من الأعمال كإضافة سجل، ثم تعديل البيانات في سجل آخر، والقيام بالحسابات على مجموعة أخرى من السجلات وذلك اعتبارًا من أمر واحد من المستخدم.
- عند استخدام الإجرائيات المخزنة نقوم بإنشاء إجرائية مع معامِلات دخل وخرج بحيث يمكننا استدعاؤها بالصورة نفسها المستخدمة في تعبيرات SQL المعيارية.
- تحتوي الإجراءات المخزنة تعبير أو مجموعة من تعبيرات SQL والتي تنفذ كجزء من تنفيذ الإجرائية مع إمكانيات التحكم بتدفق التنفيذ أو إضافة حلقات أو التنفيذ الشرطي، مما يعطي الإجرائيات المخزنة تميزاً كبيراً.
- تختلف الطريقة التي تطبق فيها قواعد البيانات الإجرائيات المخزنة بالرغم من أن الفكرة الأساسية موحدة.



استدعاء الإجراء المخزن:

يتم استدعاء الإجرائيات المخزنة باستخدام تعبيرات تعتمد على قاعدة البيانات المستخدمة فهي تستخدم التعبير EXECUTE أو EXEC في SQLServer.

مثال:

إذا أردنا استدعاء إجرائية مخزنة باسم UpdateRec مع تحديد رقم 3 كمعامل دخل تصبح صيغة الاستدعاء في SQLServer كما يلي:

```
EXECUTE UpdateRec 3;
```

نلاحظ أننا لم نستخدم الأقواس مع معامِلات الدخل والخرج في الصيغة الخاصة بـSQLServer أما إذا استخدمنا أكثر من معامل، فيتم القسم فيما بينها بفاصلة.

إنشاء إجراء مخزن: Create StoredProcedure:

يمكننا إنشاء إجرائية مخزنة باستخدام التعبير CREATE PROCEDURE وذلك حسب الصيغة التالية:

```
CREATE PROCEDURE sp_name (parameter_list)

AS sp_body;
```

حيث:

sp_name: اسم الإجرائية المخزنة.

Parameter_list: قائمة بمعامِلات الدخل والخرج لهذه الإجرائية مفصولة بفواصل.

sp_body: محتوى أو تعليمات الإجرائية المخزنة.

مثال:

إذا أردنا إنشاء إجرائية مخزنة لتقوم بحذف سجل من جدول Products اعتماداً على رقم السجل الذي نمرره لتلك الإجرائية، كمعامل دخل، والذي يمثل قيمة الحقل ID لهذا السجل، نستخدم الصيغة:

```
CREATE PROCEDURE deleteProduct (@ProductID INT)
AS
      BEGIN
Delete from Products where ID = @ProductID;
END;
```

ما نلاحظه في هذه الصيغة أنه قد تم احتواء جسم الإجرائية المخزنة في التركيب BEGIN، END وتم استخدام معامل الدخل ProductID كجزء من الاستعلام.

استخدمنا الإشارة @ قبل اسم المعامل لأن الصيغة السابقة هي صيغة الإجرائية في SQL Server.

الآن إذا أردنا استخدام الإجرائية المخزنة التي قمنا بكتابتها، يمكن ببساطة استدعاءها وتنفيذها لحذف السجل ذو قيمة الحقل ID المساوية لـ8 باستخدام الصيغة:

```
EXECUTE deleteProduct (8);
```

حذف إجراء مخزن: Drop StoredProcedure:

تستخدم أغلب قواعد البيانات الصيغة المستخدمة لحذف إجرائية مخزنة هي:

```
DROP PROCEDURE procedure_name;
```

تعديل إجراء مخزن: Alter StoredProcedure:

لتعديل إجرائية مخزنة نستخدم التعبير ALTER PROCEDURE وهو مشابه للتعبير CREATE PROCEDURE مع فارق واحد يكمن في أنه يعيد إنشاء الإجرائية المخزنة، ولها الصيغة التالية:

```
ALTER PROCEDURE sp_name (parameter_list)

AS sp_body;
```

مثال:

إذا أردنا تعديل الإجرائية المخزنة myStoredProcedure التي تقوم بإدراج سجل في جدول myTable لتقوم بحذف سجل من هذا الجدول نكتب الصيغة:

```
ALTER PROCEDURE myStoredProcedure (myRecordID INT)
AS
Delete from myTable where ID = myRecordID;
```

إنشاء واستخدام المتحولات والمعاملات:

هناك ثلاثة أنواع من المعاملات في الإجرائيات المخزنة هي :معاملات الدخل، معاملات الخرج ومعاملات الدخل/خرج.

تستخدم معاملات الدخل لتمرير البيانات إلى الإجرائية، ومعاملات الخرج لإعادة قيم من الإجرائية، ومعاملات الدخل خرج لتأمين الغرضين معاً.

قبل الدخول في تفاصيل معاملات الدخل والخرج لابد من أن نتكلم عن المتحولات في SQL بصورة عامة.

المتحولات في SQL :

المتحولات في SQL مشابهة للمتحولات في اللغات الأخرى غرضها الأساسي تخزين قيم في الذاكرة يمكن استرجاعها باستخدام اسم المتحول.

للتصريح عن متحول نستخدم الصيغة:

```
DECLARE var_name var_type (length);
```

يمكن استخدام عبارة تصريح واحدة لأكثر من متحول في حال كانت جميع المتحولات المراد التصريح عنها من نفس النوع ونفس الحجم. تأخذ العبارة الصيغة:

```
DECLARE var1_name, var2_name, var3_name var_type (length);
```

مثال:

إذا أردنا التصريح عن متحول صحيح وثلاثة متحولات من نوع varchar نكتب الصيغة:

```
DECLARE var1 INT;
DECLARE var2, var3, var4 varchar(50);
```

إسناد قيمة للمتحول:

بعد التصريح عن المتحولات قد نحتاج إلى إسناد قيمة لها أحياناً. تختلف الصيغة الخاصة بإسناد قيمة لمتحول من قاعدة بيانات إلى أخرى. ففي حالة SQLServer هي من الصيغة:

```
SET @var_name = value;
```

أو

```
SELECT @var_name = value;
```

ما إن يصبح للمتحول قيمة تستطيع استخدامه بحرية ضمن الاستعلام.

لنعد مجددًا إلى المعاملات الخاصة بالإجرائيات المخزنة:

ماذا لو أردنا أن نعطي قيم تلقائية للمعاملات الخاصة بإجرائية مخزنة؟

لإسناد قيم تلقائية للمعاملات نستخدم إشارة المساواة بعد تعريف المعامل ونضع القيمة التلقائية كما في حالة SQLServer.

مثال:

إذا أردنا إنشاء الإجرائية المخزنة التي تقوم بإدراج اسم وعنوان كل شخص في جدول Contacts بحيث يأخذ العنوان القيمة التلقائية ' Unknown ' في حال عدم إعطاء قيمة للمتغير نكتب الصيغة:

```
CREATE PROCEDURE Insert Contacts
(@myName varchar(50), @myAddress varchar(50) = 'Unknown')
AS Insert Into Contacts (contactName, contactAddress)
Values (@myName, @myAddress);
```

استخدام معاملات الخرج:

كما سبق ذكرنا فإن معاملات الخرج مخصصة لإعادة قيم من الإجرائية المخزنة إلى التطبيق الذي نفذ الإجرائية.

استخدام معاملات الخرج في SQLServer:

لتحديد كون أحد المعاملات في إجرائية مخزنة في SQServer كمعامل خرج، نضيف التعبير OUTPUTو تكون الصيغة من الشكل:

```
CREATE PROCEDURE procedure_name
(@output_parameter_name INT OUTPUT)
…;
```

مثال:

إذا أردنا إنشاء إجرائية مخزنة تقوم بإعادة اسم شخص صاحب رقم هاتف معين ووضع القيمة المعادة في متحول الخرج theName نكتب الصيغة:

```
CREATE PROCEDURE getName
(@theNumber varchar(15) , @theName varchar(50) OUTPUT)
AS
BEGIN
SET @theName = (SELECT Name from Phonebook where Number =
@theNumber)
END;
```

نلاحظ أننا استخدمنا التعبيرSET لنعطي المتحول theName القيمة التي أعادها الاستعلام.

ويمكننا استخدام هذه الإجرائية المخزنة مباشرةً بالشكل:

```
DECLARE @theName varchar(50);
EXECUTE getName ‘4445467’ , @theName OUTPUT;
PRINT @theName;
```

قمنا هنا:

- بالتصريح عن المتحول theName,
- ثم بتنفيذ الإجرائية المخزنة مستخدمين theName كمتحول خرج لاستلام القيمة التي تعيدها الإجرائية،
- ثم بإخراج قيمة هذا المتحول.

حول الإجرائيات المخزنة

رأينا قوة الإجرائيات المخزنة كما أنها تتضمن آليات للتنفيذ الشرطي والحلقات والمؤشرات.

ترى بعض النظريات ضرورة التركيز على تنفيذ جميع الاستعلامات عن طريق الإجرائيات المخزنة وذلك للأسباب التالية:

- غالبًا ما تتم عملية ترجمة مسبقة للإجرائيات المخزنة في قواعد البيانات العلائقية مما يجعل استخدامها فرصة لتحسين الأداء.
- تعتبر الإجرائيات المخزنة وسيلة جيدة لرفع درجة الأمان كونها توقف مجموعة من الهجومات الأمنية المحتملة على قاعدة البيانات.
- تضمين منطق العمل في قاعدة البيانات نفسها وجعل التطبيقات تلعب دور واجهة تعامل مع البيانات فقط.

انتهت المحاضرة

التعابير الشرطية والحلقات
Conditionals & Loops

مقدمة:

نتعرف في هذه الجلسة على مفهوم التعابير الشرطية والحلقات وكيفية استخدامها في قواعد البيانات المختلفة.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- مفهوم التعابير الشرطية وكيفية استخدامها
- مفهوم الحلقات واستخدامها

كتابة النصوص البرمجية في الإجرائيات المخزنة:

ذكرنا مسبقًا أن الإجرائيات المخزنة تتألف بصورة أساسية من تعابير SQL بالإضافة إلى لغة برمجة بنيوية تتضمن تعابير خاصة للتنفيذ الشرطي والحلقات وإمكانيات أخرى. تختلف التعابير البرمجية بين أنظمة قواعد البيانات ولكنها تتشابه في التعليمات الأساسية.

التنفيذ الشرطي:

في الكثير من الأحيان نجد أنه من الضروري ألا يتم تنفيذ استعلام ما ضمن إجرائية مخزنة إلا عند تحقق شرط معين كإدخال معامل بقيمة صحيحة مثلاً.

قد يكون أحد السيناريوهات المحتملة أيضاً الرغبة في عدم تنفيذ استعلام ما لم يعيد استعلام سابق نتيجة متوقعة.

أو أن نحتاج إلى تنفيذ استعلامات مختلفة بحسب قيمة معامل محدد مثلاً.

تحتوي أغلب أنظمة قواعد البيانات العلائقية على نوعين أساسين من التعابير الشرطية:

- ا لتعبير IF.....ELSE وهو بالصيغة:

```
IF condition
conditionTrueBody;
ELSE
conditionFalseBody;
```

حيث تمثل condition الشرط الواجب تحققه لتنفيذ العبارات التي تمثلها conditionTrueBody وإلا سيتم تنفيذ العبارات التي تمثلها conditionFalseBody.

- التعبير CASE.....WHEN ويكتب بالصيغة:

```
CASE expression
WHEN value1 THEN result1
WHEN value2 THEN result2
.....
WHEN valueN THEN resultN
ELSE resultElse
END;
```

في هذه الصيغة سيتم تقييم التعبير expression ومقارنته مع القيم Value1...ValueN وستكون نتيجة هذه الصيغة هي النتيجة result المقابلة لأول عملية مطابقة، وإلا سيتم تنفيذ النتيجة resultElse.

سنقوم فيما يلي باستعراض عمل هذه البنى.

التعابير الشرطية

سنرى كيفية استثمار التعابير الشرطية IF.... ELSE و WHEN....CASE في قواعد بيانات SQL Server.

التركيب IF.... ELSE:

تأخذ بنية IF.... ELSE الشرطية الصيغة التالية في SQL Server:

```
IF condition
BEGIN
trueStatments
END
ELSE
BEGIN
falseStatments
END
```

يمكن حذف BEGIN و END من الصيغة السابقة في حال كان المطلوب تنفيذ تعبير واحد في كتلة BEGIN END.كما يمكن كذلك الاستغناء عن ELSE مع الكتلة الخاصة بها في حال عدم الرغبة بتنفيذ أي تعبير عند عدم تحقق الشرط.

مثال:

إذا أردنا أن نكتب الإجرائية المخزنة التي تقوم بحساب كمية المبيعات من مادة ما، وكتابة تعليق في حقل خاص في جدول المواد عن ضرورة إيقاف هذه السلعة في حال عدم بيع أي كمية منها، نكتب الصيغة:

```
CREATE PROCEDURE getSalesAndComment
(@myProductID INT, @mySales INT OUTPUT,@myComment
VARCHAR(40) OUTPUT)
AS
BEGIN
SET @mySales= (Select sum(Quantity) from sales
where productID=@myProductID);
IF @mySlales=0
SET @myComment='STOP THIS PRODUCT';
ELSE
```

```
SET @myComment='KEEP THIS PRODUCT';
Update products set ProductComment=@myComment
Where productID=@myProductID ;
END
```

نلاحظ أننا لم نستخدم كتلة BEGIN END لأننا لم نطلب تنفيذ سوى أمر واحد ضمن كتلة IF وELSE، لاستخدام هذه الإجرائية يمكننا كتابة الصيغة:

```
DECLARE @theComment VARCHAR(40);
DECLARE @theSales INT
EXECUTE getSalesAndComment(3, @theSales OUTPUT,
@theComment OUTPUT);
PRINT @theSales;
PRINT @theComment;
```

نلاحظ أننا صرحنا عن المتغيرات التي سيتم إعادتها من الإجرائية وقمنا بطباعتها.

التركيب CASE…. WHEN:

تأخذ بنية CASE WHEN الشرطية الصيغة التالية في SQL Server:

```
SET @aVariable= CASE expression
WHEN value1 THEN result1
WHEN value2 THEN result2
    …..
WHEN valueN THEN resultN
ELSE resultElse
END;
```

لا نستطيع استخدام التركيب CASE WHEN في SQL Server إلا ضمن عبارة SELECT أو عبارة إسناد قيمة لمتغير.

يمكن استخدام التعبير CASE دون وضع أي تعبير بعدها حيث يمكن هنا استخدام value1… valueN لوضع شروط مختلفة.

مثال:

إذا أردنا أن نكتب الإجرائية المخزنة التي تقوم بعرض عبارة تعتمد على علامة طالب وتعبر عن مستواه بعد أن تأخذ على الدرجة التي حصل عليها:

```
CREATE PROCEDURE getStudentLevel
(@myStudentName VARCHAR(50), @myStudentLevel
VARCHAR(40) OUTPUT)
AS
BEGIN
DECLARE studentGrade INT;
SET @studentGrade= (Select studentGrade from students
where studentName=@myStudentName);
SET @myStudentLevel= CASE
WHEN @studentGrade>80 THEN 'VERY GOOD'
WHEN @studentGrade>70 THEN 'GOOD'
WHEN @studentGrade>60 THEN 'NOT BAD'
WHEN @studentGrade>50 THEN 'PASS'
WHEN @studentGrade<50 THEN 'FAIL'
END;
END;
```

ولاستخدام هذه الإجرائية يمكننا كتابة الصيغة:

```
DECLARE @theStudentLevel VARCHAR(40);
EXECUTE getSalesAndComment('sami', @ theStudentLevel OUTPUT);
PRINT @ theStudentLevel;
```

هنا كما نلاحظ صرحنا عن المتغيرات التي سيتم إعادتها من الإجرائية وقمنا بطباعتها.

مثال:

لنكتب الإجرائية المخزنة التي تقوم بإدراج حجز جديد ضمن جدول الحجوزات Reservations وحساب كلفة الحجز بناء على عدد الأيام ونوع الغرفة:

```
CREATE  PROCEDURE Reservations
(@myRoomType varchar(10), @myDays INT)
AS
BEGIN
IF @myRoomType= 'Single'
Insert Into Reservations (roomType, Fee)
Values(@myRoomType,20*@myDays);
IF @myRoomType='Double'
Insert Into Reservations (roomType, Fee)
Values(@myRoomType,35*@myDays);
IF @myRoomType='Sweet'
Insert Into Reservations (roomType, Fee)
Values(@myRoomType,45*@myDays);
END;
```

ولاستخدام هذه الإجرائية يمكننا كتابة الصيغة:

```
EXECUTE  Reservations ('Single',5);
```

الحلقات والتكرار

تدعم قواعد البيانات تعابير مختلفة لتأمين تنفيذ كتلة من التعليمات لأكثر من مرة اعتماداً على استمرار تحقق شرط ما.

ففي SQL Server يستخدم التعبير WHILE مع BEGIN و END بالصيغة:

```
WHILE Condition
BEGIN
loopBody
END;
```

مثال:

إذا أردنا كتابة الإجرائية المخزنة التي تقوم بتوليد عشر أرقام عشوائية بين 1 و100 ووضعها في الجدول Numbers وذلك في قواعد بيانات SQL Server نستخدم الصيغة:

```
CREATE PROCEDURE tenRandoms()
AS
BEGIN
DECLARE @myNumber INT;
WHILE @myNumber<100
BEGIN
Insert Into Numbers(number) Values(Round(rand)*100);
SET @myNumber=@myNumber+1;
END;
END;
```

انتهت المحاضرة

المؤشرات
Cursors

مقدمة:

نتعرف في هذه الجلسة على مفهوم المؤشرات والمؤشرات الضمنية وكيفية إدارتها والاستفادة منها.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- مفهوم المؤشرات والمؤشرات الضمنية
- استخدام المؤشرات في إعادة قائمة نتائج

ما هي المؤشرات؟

المؤشرات هي عبارة عن طريقة لتمثيل مجموعة سجلات ضمن نص SQL بحيث تسمح بالدوران على هذه السجلات.

بصورة عامة، تؤثر المؤشرات على أداء قاعدة البيانات وينصح عادة بعدم استخدامها إلا في حال الضرورة، وبالأخص عند الحاجة إلى القدرة على التحرك بين كل سجل من مجموعة من السجلات بصورة منفصلة.

التصريح عن مؤشر:

للتصريح عن مؤشر نستخدم الصيغة:

```
DECLARE cursorName CURSOR FOR cursorSpecification;
```

حيث تعبر cursorSpecification عن الاستعلام الذي سوف يستخدم معه المؤشر.

في حال أردنا تعديل البيانات عن طريق المؤشر توجب علينا إضافة FOR UPDATE إلى نهاية الصيغة حيث تصبح الصيغة:

```
DECLARE cursorName CURSOR IS cursorSpecification FOR
UPDATE;
```

تقوم هذه العملية بإقفال الجداول التي يستخدمها المؤشر بحيث لا يتمكن مستخدم آخر من تعديلها.
يمكننا تحديد الجداول التي نود إقفالها بإضافة OF columnList حيث columnList هي قائمة بأسماء الحقول وفي هذه الحالة سيتم فقط إقفال الجداول التي تنتمي إليها تلك الحقول.
يمكننا أيضًا تحديد الخيار FOR READ ONLY عوضًا عن FOR UPDATE إذا كنا لا نريد للمؤشر أن يكون قابلاً للتعديل.

استخدام المؤشرات:

قبل استخدام مؤشر علينا أوً لا أن نقوم بفتح هذا المؤشر وذلك بالصيغة:

```
OPEN cursorName;
```

تستطيع الآن بعد فتح المؤشر استعادة سجلات منه وتخزينها في متحولات محلية.
للقيام بهذا العمل سنستخدم التعبير FETCH وصيغته من الشكل:

```
FETCH cursorName INTO var1,var2,var3,…,varN;
```

تقوم هذه العملية بإعطاء المتحولات var1,var2,var3,…varN قيم حقول السجل الحالي للمؤشر. يجب أن تتطابق أنواع هذه المتحولات مع أنواع الحقول الموافقة لها.
يمكننا التنقل للحصول على قيم الحقول للسجل الأول باستخدام FETCH FIRST، وللسجل السابق باستخدام FETCH PRIOR، وللسجل الأخير باستخدام FETCH LAST عوضًا عن FETCH NEXT.
كما يمكننا تحديد رقم السجل الذي نود استعادة معلوماته وذلك باستخدام FETCH

.FETCH ABSOLUTE N أو RELATIVE N

تصبح المرحلة الأخيرة بعد فتح المؤشر ونقل معلومات السجل إلى المتحولات الخاصة به، هي إغلاق المؤشر وذلك باستخدام الصيغة:

```
CLOSE cursorName;
```

انتبه:

قد تحتاج في SQL Server إلى تحرير الموارد التي استخدمها مؤشرك.

مثال:

نريد كتابة الصيغة التي تقوم بالدوران على مجموعة قيم لأسماء وأرقام هواتف أشخاص وطباعة هذه المعلومات عن طريق استخدام المؤشرات.

```
DECLARE @myNumber varchar(15);
DECLARE @myName varchar(50);
DECLARE myCursor CURSOR FOR
Select contactNumber, contactName from Contacts;
OPEN myCursor;
FETCH NEXT from myCursor INTO @myNumber, @myName;
WHILE @@FETCH_STATUS = 0
PRINT @myName, @myNumber;
FETCH NEXT from myCursor INTO @myNumber, @myName;
END;
```

هنا قمنا بالتصريح عن المؤشر myCursor وتحديد الاستعلام الذي يمثل مجموعة السجلات التي يعبر عنها المؤشر، ثم قمنا بفتح المؤشر والمرور على السجلات واحداً تلو الآخر باستخدام التعبير .FETCH NEXT

استخدمنا هنا @@FETCH_STATUS لتحديد فيما إذا تم الوصول إلى نهاية المؤشر حيث ستكون قيمتها -1مقابل 0 في الحالة العادية.

إعادة جدول نتائج من إجرائية:

حتى الآن استعرضنا كيفية استعادة قيم إفرادية من إجرائيات مخزنة باستخدام معاملات الخرج، ولكن يمكن لنا إعادة كامل جدول القيم من إجرائية. تعتبر هذه العملية بسيطة جداً في SQL Server فهي تأخذ الصيغة:

```
CREATE PROCEDURE procedureName AS query;
```

ويمكننا تنفيذها باستخدام EXECUTE أو EXEC بالصيغة:

```
EXEC procedureName;
```

مثال:

لإنشاء إجرائية باسم myProcedure تقوم بإعادة أسماء المنتجات من جدول المنتجات نكتب الصيغة:

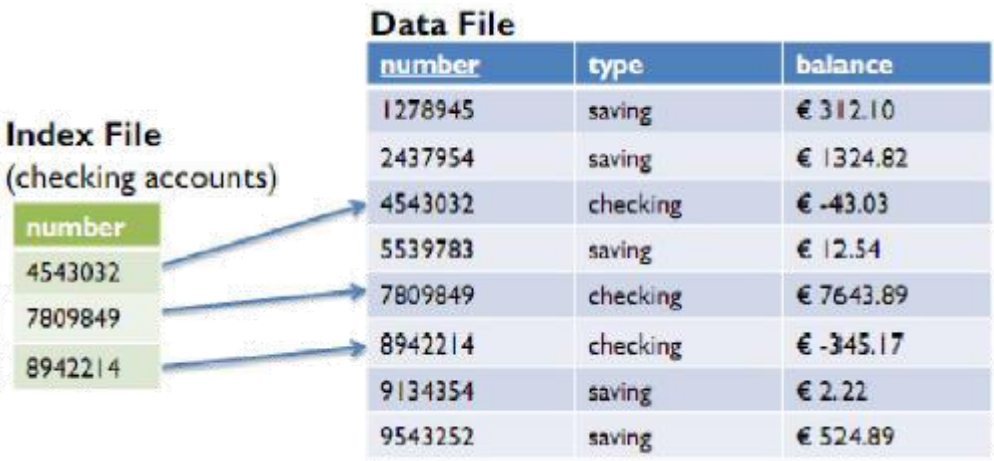
```
CREATE PROCEDURE myProcedure
AS select productName from Products;
```

انتهت المحاضرة

الفهارس
Indexes

مقدمة:

تعتبر الفهارس أحد أغراض قاعدة البيانات التي صممت لزيادة سرعة العمليات وذلك عبر توليد جداول بحث داخلية. إذ يشبه فهرس قاعدة المعطيات، فهرس كتاب.



خواص الفهرس:

- يحسِّن استخدام الفهارس الأداء في حال كانت الاستعلامات المستخدمة من النوع Select، ويضر بالأداء في حال كانت العمليات هي عمليات حذف وإدراج وتعديل (Insert,Update,Delete) لأنه سيكون مطلوب من تلك العمليات أن تطبّق على الجداول الأساسية وعلى الفهارس التابعة للجدول. ولكن يمكننا القول أنّ استخدام الفهارس يحسن الأداء بشكل يتجاوز ما قد تسببه عمليات التعديل والحذف والإضافة من تراجع في الأداء.
- يمكننا إنشاء فهرس أو مجموعة من الفهارس على جدول معين بحيث يمكن لكل فهرس أن يعمل على حقل أو على مجموعة من الحقول.
- عندما تتم فهرسة جدول حسب حقل معين، تتم فهرسة سجلات هذا الجدول حسب قيمة الحقل ونمطه مما يسمح بإيجاد هذه القيمة سريعاً عندما نجري عملية بحث معتمدة على الحقل المفهرس.
- يؤثر استخدام عدد كبير من الفهارس سلبياً على الأداء.
- عادة ما يقوم محرك البيانات بإنشاء فهارس تلقائية للمفاتيح الرئيسية.

مثال:

إذا كان لدينا جدول كبير يحتوي على أسماء الزبائن ومعلوماتهم فإن وجود فهرس على حقل اسم الزبون سيسرع عمليات البحث والاستعلام التي تستخدم الاسم. لأن عملية البحث هنا لا تتم بمسح كامل للجدول الأصلي بل بالبحث ضمن جدول الفهرس أولاً قبل الوصول إلى القيم المطابقة في الجدول الأصلي.

Create Indexes:
إنشاء الفهارس:

- تطبق الفهارس عادةً على الأعمدة المستخدمة في التعابير Where و Order By وتلك المستخدمة مع التعبير ON الخاص بالتعبير Join.
- يفيد استخدام الفهارس مع الحقول التي يقل فيها تكرار القيم حيث تساعد الفهارس على الحصول على اصطفائية عالية.
- للفهارس نوعين:
  - فهارس فريدة أي لا تسمح بتكرار قيم في الحقل المفهرس.
  - فهارس غير فريدة تسمح بتكرار القيم ضمن الحقل المفهرس.
- لسنا بحاجة إلى فهرسة المفتاح الرئيسي أو الحقول ذات القيد Unique لأن نظام قاعدة البيانات يولد تلك الفهارس تلقائياً ويقوم بإزالتها حال إزالة صفة المفتاح الرئيسي أو قيد Unique عن هذه الحقول.

لإنشاء فهرس فريد (لا يسمح بتكرار قيم الحقل المفهرس) في SQLServer نستخدم الصيغة:

```
CREATE UNIQUE INDEX index_name ON tableName (FieldName);
```

ولإنشاء فهرس غير فريد (يسمح بتكرار قيم الحقل المفهرس) في SQLServer نستخدم الصيغة:

```
CREATE INDEX index_name ON tableName (FieldName);
```

مثال:

لدينا الجدول Phonebook الذي يحتوي أسماء الأشخاص Name، وأرقام هواتفهم Number وتصنيفهم Category: أصدقاء، عائلة، الخ...فإذا أردنا إنشاء فهرس فريد على حقل فسيكون الاختيار الصحيح هو الحقل Number حيث علمنا أن حقل الاسم هو حقل مفتاح رئيسي مفهرس

تلقائياً، وحقل التصنيف Category لا يصلح ليكون فهرس فريد بسبب تكرار قيمته. لذلك ننشئ الفهرس بالصيغة:

```
CREATE UNIQUE INDEX myIndex ON Phonebook (Number);
```

هذه الصيغة ستسرع إعادة نتائج البحث في استعلام من الشكل:

```
Select * from Phonebook where Number = '5437268';
```

ملاحظة:

في حالة الفهارس على حقول سلاسل المحارف، لا يتم البحث ضمن جدول الفهرس في حال تم الاستعلام عن سلسلة محرفية جزئية (من السلسلة المحرفية الأساسية) لا تبدأ من بداية السلسلة الأصلية ففي استعلام من الشكل:

```
Select * from tableName where strColumn like '%substr';
```

لا يستخدم الفهرس المنشأ على الحقل strColumn وستتم عملية مسح كاملة للجدول الأساسي لإتمام عملية البحث.

حذف الفهارس: Drop Indexes:

لحذف فهرس نستخدم في SQL التعبير DROP INDEX. لكن يحتاج تطبيقه على قواعد البيانات المختلفة، تعديلات طفيفة في الصيغة ففي SQL Server نحتاج إلى تحديد اسم الجدول والفهرس، أي تكون الصيغة بالشكل:

```
DROP INDEX myTable.myIndex;
```

مثال:

إذا أردنا حذف الفهرس numberIndex من الجدول Phonebook نستخدم الصيغة في SQL Server:

```
DROP INDEX Phonebook.numberIndex;
```

انتهت المحاضرة

التوابع المعرفة من قبل المستخدم
Functions

الكلمات المفتاحية:

تابع، معامِل، دخل، خرج، إنشاء، صيغة.

ملخص:

تعرفنا في مراحل سابقة على التوابع المسبقة التعريف وسنتعرف في هذه الجلسة على التوابع المعرَّفة من قبل المستخدم في قواعد البيانات المختلفة.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- التوابع وإنشاؤها في قواعد بيانات SQL Server

إنشاء التوابع

استعرضنا في الجلسات الماضية التوابع وأنواعها وميزنا التوابع التجميعية والتوابع الدرجية بأنواعها المختلفة ولكن ماذا لو أردنا إنشاء توابعنا الخاصة؟

يُعرِّف المعيار ANSI SQL-99 التعبير CREATE FUNCTION الذي يساعد على إنشاء التوابع.

تسمى التوابع التي يتم إنشاؤها بالتعبير CREATE FUNCTION، بالتوابع المعرفة من قبل المستخدم أو User Defined Functions.

تأخذ الصيغة البسيطة العامة في أغلب قواعد البيانات الشكل:

```
CREATE FUNCTION function name [(parameter list)]
RETURNS data type
-- SQL Statements
```

للخوض في التفاصيل لا بد لنا من الاطلاع على كيفية تعامل كل قاعدة بيانات مع التوابع.

استدعاء تابع:

يتم استدعاء التوابع المعرَّفة من قبل المستخدم بنفس الطريقة المستخدمة مع التوابع الأخرى أي يمكننا بعد إنشاء التابع استدعاؤه

بصورة فورية بالشكل:

```
SELECT function_name (parameter_list) AS Test;
```

التوابع في SQL Server:

يستخدم SQL Server لإنشاء تابع من قبل مستخدم الصيغة التالية:

```
CREATE FUNCTION function name [(parameter list)]
RETURNS data_type
AS
BEGIN
SQL Statements
RETURN expression
END;
```

يمكن للتوابع في SQL Server أن تحتوي أكثر من تعبير ولكن يجب ألا تسبب أي تغيير على البيانات في قاعدة البيانات.

إنشاء التوابع

مثال:

إذا أردنا إنشاء تابع باسم formatName() الذي يقوم بإعادة الاسم الكامل مرتبًا، بوضع الاسم الثاني أولاً والاسم الأول ثانيًا مفصولاً بفاصلة نكتب الصيغة:

```
CREATE FUNCTION formatName (@fullName varchar(50))
RETURNS varchar(50)
AS
BEGIN
RETURN WRITE (@fullName, LEN (@fullName) - CHARINDEX (' ', @fullName)
+1) + ', ' +
LEFT (@fullName, CHARINDEX (' ', @fullName) - 1)
END;
```

نلاحظ في البداية أننا قمنا:

-بتعريف معاملات الدخل لهذا التابع، ففي حالتنا هناك معامل واحد باسم fullName،

-باستخدام التعبير RETURNS

-بتحديد أن هذا التابع سيعيد قيمة من النوع المحدد بعد RETURNS

-باستخدام التعبير RETURN لإعادة قيمة من النوع المحدد بعد RETURNS.

الآن لاستعمال هذا التابع يمكننا كتابة الصيغة:

```
SELECT formatName ('Majd Amer');
```

انتهت المحاضرة

القادحات
TRIGGERS

مقدمة:

تعتبر القادحات من الآليات التي تزود المبرمج بتقنيات تساعده على التحكم بشكل أفضل بالأحداث وبالعمليات في قواعد البيانات.

خواص القادحات:

- تعتبر القادحات وسيلة لتأمين تطبيق قواعد العمل وتكامل البيانات واتساقها وتماسكها داخل قاعدة البيانات. فهي نوع خاص من الإجرائيات المخزنة المرتبطة بجدول معين، يتم تنفيذها بصورة آلية عند أي تعديل على هذا الجدول.
- يمكننا باستخدام معالجات الأحداث أن نؤقمت بدء تنفيذ نص SQL برمجي بجعله يتجاوب مع أحداث معينة.
- من الضروري عند التعامل مع القادحات تحديد الجدول المراد ربط القادح معه وتحديد الحدث الذي سيتم إطلاقه. فهل نريد مثلاً للقادح أن يعمل حين نضيف سجل ما إلى جدول، أم عند حذف سجل ما من الجدول، أم عند تعديله، أم عند ظهور تركيبة معينة من هذه الأحداث.
- كما نحتاج إلى اتخاذ قرار بشأن لحظة الإطلاق، وفيما إذا كانت قبل أو بعد حدوث. فهل سيتم مثلاً تحفيز القادح قبل عملية إضافة السجل أم بعدها.
- بإمكاننا تنفيذ قادحات تعاكس الأحداث التي سببت إطلاقها، كأن ننفذ قادح ما قبل تنفيذ حذف سجل بحيث نلغي عملية تنفيذ الحذف كلياً.
- تشبه القادحات الإجرائيات المخزنة من حيث النص البرمجي الذي بإمكانها احتوائه ومن حيث قدرتها على تمرير معاملات دخل وخرج.

- نستطيع إنشاء أكثر من قادح على جدول واحد في قاعدة البيانات ويمكن لقادح أن يطلق قادح آخر أو أن يطلق نفسه بصورة عودية.

أنواع القادحات:

تصنف القادحات إلى نوعين أساسيين وذلك تبعًا لعدد مرات تنفيذ نص القادح عندما يؤثر الحدث على مجموعة من السجلات:

- **القادحات على مستوى التعابير:**

حيث يتم إطلاق هذا النوع من القادحات مرة واحدة لكل تعبير INSERT أو DELETE أو UPDATE حتى ولو قام هذا الأخير بالتأثير على أكثر من سجل.

فإذا كان لدينا قادح يرتبط بتعبير يسبب حذف مئة سجل، سيتم تنفيذ النص البرمجي للقادح مرة واحدة.

- **القادحات على مستوى السجل:**

في هذا النوع من القادحات يتم تنفيذ النص البرمجي للقادح مع كل سجل تؤثر فيه إحدى عبارات INSERT أو DELETE أو UPDATE. فإذا كان لدينا قادح من هذا النوع مرتبط بتعبير UPDATE يقوم بتعديل مئة سجل، سيتم تنفيذ النص البرمجي للقادح مئة مرة.

يمكننا جعل أي من نوعي القادحات ينفذ قبل أو بعد أو عوضاً عن الحدث الذي قام بإطلاقها.

يجب أن نأخذ بالحسبان أن القادحات التي تنفذ قبل الحدث، تنفذ دون تحقق الحدث نفسه، بينما القادحات التي تنفذ بعد الحدث، لن تنفذ ما لم يتم تنفيذ التعبير الممثل للحدث بصورة صحيحة. لذلك غالباً ما تستخدم هذه القادحات في تقييم نجاح التعابير والتعليمات.

استخدام القادحات:

تستخدم القادحات في تطبيقات مختلفة ولكن أهم استخداماتها هي:

1- تأمين التكاملية المرجعية والتي عادةً ما تتم باستخدام المفاتيح الأجنبية (التي ندعوها أيضاً المفاتيح الثانوية). ففي بعض الحالات، نستخدم القادحات إذا لم تكن هذه المفاتيح قوية بالدرجة الكافية لتنفيذ هذا العمل، كحالة التكاملية المرجعية بين الجداول في أكثر من قاعدة بيانات، أو على أكثر من مخدم

قاعدة بيانات.

2- تطبيق قواعد العمل المعقدة لضمان عدم وجود أية عملية تقوم بكسر قواعد العمل أو تكاملية البيانات.

3- متابعة وتسجيل كل العمليات التي تتم على الجداول في قاعدة البيانات.

4- محاكاة عمل القيد CHECK بين الجداول وقواعد البيانات ومخدمات قواعد البيانات.

5- اعتراض أوامر المستخدم واستبدالها، حيث يمكننا مقاطعة الأوامر أو الأفعال التي يقوم بها المستخدم واستبدالها بأفعال أخرى.

استخدام القادحات في قواعد بيانات SQLServer:

تدعم قواعد بيانات SQLServer القادحات على مستوى التعبير فقط ولا تدعم القادحات على مستوى السجل. فإذا استخدمنا تعبير يقوم بحذف عشرة سجلات، سوف يتم تنفيذ القادح مرة واحدة لكامل العبارة.

لا يدعم SQLServer القادحات التي تسبق الحدث أي التي تستخدم التعبير BEFORE، ويدعم فقط القادحات اللاحقة. وابتداءً من النسخة SQLServer 2000 بات يدعم القادحات البديلة أي التي تستخدم التعبير INSTEAD .

إنشاء قادح: Create Trigger:

لإنشاء قادح في SQLServer:

```
CREATE TRIGGER triggerName ON tableName FOR [|INSTEAD]
action AS -- procedureBody;
```

مثال:

إذا أردنا إنشاء قادح مرتبط بحدث إضافة سجل إلى الجدول Test الذي يحتوي القيمة value وحقل الرقم التسلسلي ID, بحيث يقوم هذا القادح بتسجيل عملية إضافة السجل إلى الجدول Audit، نكتب الصيغة:

```
CREATE TRIGGER myTrigger ON Test FOR Insert
AS
DECALRE @newValue varchar(50)
SELECT @newValue = value FROM Inserted
Insert Into Audit (newValue) Values (@newValue);
```

نلاحظ في الصيغة السابقة أننا قمنا باستخدام القادح بعد إضافة سجل إلى الجدول Test كما نلاحظ أننا استفدنا من قيم الجدول Inserted وهو جدول تم إنشاؤه تلقائياً يحتوي السجل الذي يتم العمل عليه وله بنية مطابقة لبنية الجدول الذي تم ربط القادح به.

عند ربط قادح بعملية حذف سجل، يتم أيضًا إنشاء جدول باسم Deleted يحتوي السجل الذي تم حذفه له نفس بنية الجدول الذي تم ربط القادح به.

تكمن مشكلة هذا الحل في أننا لن نستطيع تسجيل عملية إدراج مجموعة من السجلات باستخدام تعبير واحد. فإذا تم استخدام التعبير INSERT INTO.... SELECT ، لن يتم تنفيذ القادح سوى مرة واحدة. إذ سبق وذكرنا أن SQLServer لا تدعم القادحات على مستوى السجل.

مثال:

إذا أردنا إنشاء قادح ليقوم بتسجيل عمليات الحذف من جدول Test، تكون الصيغة:

```
CREATE TRIGGER logDelete
ON Test FOR DELETE AS
DECALRE @newValue varchar(50)
SELECT @deletedValue = value FROM Deleted
Insert Into Audit (newValue) Values (@deletedValue);
```

تعديل قادح: Alter Trigger:

لتعديل قادح ما في SQLServer نستخدم الصيغة التالية:

```
ALTER TRIGGER triggerName ON tableName FOR [|INSTEAD]
action AS -- procedureBody;
```

مثال:

إذا أردنا دمج عمليتي الحذف والإدراج في قادح واحد يمكن تعديل القادح السابق myTrigger بحيث نحصل على الصيغة:

```
ALTER TRIGGER myTrigger ON Test
FOR INSERT, DELETE
AS
IF EXISTS (SELECT 1 FROM Inserted)
BEGIN
INSERT INTO Audit (newValue) SELECT Inserted.Value FROM Inserted
END
ELSE IF EXISTS (SELECT 1 FROM Deleted)
BEGIN
INSERT INTO Audit (newValue) SELECT Deleted.Value FROM Deleted
END;
```

ستعمل الصيغة السابقة حتى في حالة إدراج أكثر من سجل في الجدول Test، ولكن عملها لن يكون بفضل القادحات من مستوى (لأنها غير مدعومة أصلاً)، بل بفضل التعبير الذي تم اختياره في عملية إدراج القيم ضمن الجدول Audit، حيث يقوم التعبير بإدراج كامل السجلات المدخلة والتي تم نسخها إلى الجدول Deleted أو Inserted إلى الجدول Audit.

حذف قادح: Drop Trigger:

لحذف قادح في SQLSERVER نستخدم الصيغة التالية:

```
DROP TRIGGER triggerName;
```

مثال:

لحذف القادح باسم myTrigger الذي أنشأناه مسبقاً نستخدم الصيغة:

```
DROP TRIGGER myTrigger;
```

حالة Update:

لم نذكر حالة ربط القادح في SQLServer بتعديل سجل، ولم نذكر شيئاً عن وجود ما يسمى Updated على غرار Deleted وInserted وذلك لعدم وجوده أصلاً.

فعلياً عند ربط قادح بحدث UPDATE، يتم ملء كل من الجدولين Inserted وDeleted.

فإذا أردنا تعديل المثال السابق ليشمل حالات الإدراج والحذف والتعديل يمكننا كتابة الصيغة:

```
ALTER TRIGGER myTrigger ON Test FOR INSERT, DELETE, UPDATE AS
IF EXISTS (SELECT 1 FROM Inserted) AND EXISTS (SELECT 1 FROM Deleted)
BEGIN
INSERT INTO Audit (newValue) SELECT D.Value FROM Deleted D JOIN
Inserted I on D.Value = I.Value
END
ELSE IF EXISTS (SELECT 1 FROM Inserted)
BEGIN
INSERT INTO Audit (newValue) SELECT Inserted.Value FROM Inserted
END
ELSE IF EXISTS (SELECT 1 FROM Deleted)
BEGIN
INSERT INTO Audit (newValue) SELECT Deleted.Value FROM Deleted
END;
```

نلاحظ هنا أننا اختبرنا وجود سجلات في كلا الجدولين Inserted وDeleted لنتحقق بأن العملية هي عملية تعديل، ثم قمنا بربط السجلات في الجدولين بJoin لاستخلاص القيم التي نريد إدراجها.

تفعيل وتعطيل القادحات:

قد يلزمنا في بعض الأحيان ولأسباب خاصة بتحسين الأداء، تعطيل عمل بعض القادحات بشكل مؤقت، وبالأخص تلك التي تؤدي مهام المتابعة وتسجيل الحركة.

يتم تفعيل القادحات تلقائيًا بعد إنشائها، ولكنك تستطيع تعطيلها باستخدام التعبير ALTER

TABLE وذلك اعتماداً على الصيغة:

```
ALTER TABLE table_name DISABLE TRIGGER trigger_name;
```

ولإعادة تفعيل قادح تم تعطيله يمكننا استخدام الصيغة:

```
ALTER TABLE table_name ENABLE TRIGGER trigger_name;
```

أما إذا أردنا تعطيل أو تفعيل جميع القادحات الخاصة بجدول ما، فنستخدم التعبير ALL بدلاً من اسم القادح. تصبح الصيغة عندها:

```
ALTER TABLE table_name [ENABLE | DISABLE] TRIGGER ALL;
```

انتهت المحاضرة

المادة	قواعد معطيات /2/
عنوان الوظيفة	تصميم قاعدة معطيات متقدمة وربطها بواجهة برمجية
تاريخ التوزيع	
تاريخ الإعادة	
الموارد المطلوبة	MS Word, MS Visio, MS Visual Studio, MS SQLServer
اللغات المستخدمة	T-SQL ,C#

مشاريع مقترحة

المشروع الأول: نظام محاكاة بنك
المشروع الثاني: نظام الحجوزات الفندقية
المشروع الثالث: نظام الاستعارة من مكتبة
المشروع الرابع: نظام سجلات عقارية
المشروع الخامس: نظام تأجير سيارات
المشروع السادس: توصيف أجهزة (حاسوبية، موبايلات، أجهزة تلفاز)

المطلوب:

1. اختيار أحد المشاريع السابقة، ويمكان للطالب اختيار مشروع خارجي بعد تقديم توصيف نصي له والموافقة عليه.
2. وضع نص تفصيلي للمشروع.
3. وضع مخطط قاعدة البيانات Database Schema باستخدام Visio.
4. إنشاء قاعدة البيانات Database مع تحديد ملفات قاعدة البيانات اللازمة والحجم الأولي لقاعدة البيانات باستخدام تعليمات SQL.
5. توصيف جداول قاعدة البيانات باستخدام القالب التالي:

جدول (هنا نضع اسم الجدول باللغة العربية)	**(هنا نضع اسم الجدول)**
(هنا نضع توصيف لمهمة الجدول)	
مفتاح أساسي (هنا نحدد مهمة كل حقل في الجدول)	ID
تخزين اسم الشخص	Name
تحديد اختصاص الشخص	Specification
	....

6. إنشاء 3 جداول Tables على الأقل باستخدام تعليمات SQL.
7. إضافة المفاتيح الأساسية PK والأجنبية FK، وكذلك إضافة القيود Constraint المناسبة على حقول الجداول السابقة باستخدام تعليمات SQL.

8. اختبار قاعدة البيانات بإدخال 3 سجلات في كل جدول، وكذلك اجراء عملية تعديل وعملية حذف على كل جدول.
9. وضع مخطط قاعدة البيانات باستخدام Diagram المتاح في SQLServer.
10. إنشاء الفهارس Indexes المناسبة لكل جدول.
11. توصيف كل جدول باستخدام بطاقة الجدول التالية:

بطاقة جدول			
الرقم المتسلسل	**اسم الجدول**		
(اختر التسلسل الذي تراه مناسباً للجداول**)**	**(**اسم الجدول**)**		
الحقول			
اسم الحقل	**النمط**	**يقبل NULL**	**القيمة الافتراضية**
القيود			
اسم القيد	**نوع القيد**	**الحقول المشاركة**	**الحقول الخارجية المشاركة**
الفهارس			
اسم الفهرس	**نوع الفهرس**	**الحقول المشاركة**	**هل الفهرس عنقودي Clustered**
T-SQL			
إنشاء الجدول			
Create Table ... **(…) ON …**			
إنشاء القيود (غير المضمنة في إنشاء الجدول)			
Alter Table … Add Constraint xx …			
إنشاء الفهارس			
Create Index …			
ملاحظات			
ضع هنا أي تعليق تراه مناسباً لتبرير إنشاء القيود أو الفهارس وكذلك لتبرير أنماط معطيات الحقول إذا كان ذلك ضروريا برأيك.			

12. إنشاء التوابع Functions اللازمة لتعديل الخرج المعطى للنتائج (مثلاً دمج الاسم والكنية- تعديل قيمة عددية...).
13. إنشاء المناظير Views التالية:
    a. View لاستعلام يحوي ربط داخلي بين الجداول.
    b. View يمكن من خلاله إدخال بيانات إلى جدول أو التعديل عليه.
14. إنشاء الإجرائيات المخزنة Stored Procedures التالية، مع أخذ لقطة للشاشة لنتيجة التنفيذ عند الاستدعاء:
    c. إجرائية مخزنة لإضافة بيانات إلى كل جدول.
    d. إجرائية مخزنة لإضافة بيانات إلى أحد الجداول من خلال المنظار View.
    e. إجرائية مخزنة لتعديل بيانات أحد الجداول بشرط معين.
    f. إجرائية مخزنة لحذف سجلات أحد الجداول بشرط معين.
    g. إجرائية مخزنة للاستعلام من الجداول تعود بمتحولات خرج لسجل واحد فقط.
    h. إجرائية مخزنة للاستعلام من الجداول تعود بجدول نتائج.
    i. إجرائية مخزنة تقوم بطباعة عدة سجلات باستخدام المؤشرات Cursors.
    j. إجرائية مخزنة للاستعلام من الجداول تعود بنتائج انتقائية باستخدام التعليمات الشرطية والحلقات.
15. إنشاء القوادح Triggers اللازمة.
16. إنشاء مستخدم User، وإنشاء دور Role فقط للاستعلام من الجداول، ومنح المستخدم هذا الدور.
17. بناء واجهات بسيطة للنظام لإدخال القيم اللازمة وعرض النتائج باستخدام تقنية Ado.NET.
18. توثيق خطوات العمل على ملف Word.
19. توثيق تعليمات T-SQL على ملف بلاحقة *.sql.