

## 8. فصل هشتم: تراکنش ها

### 8.1. مقدمه

در این فصل با مفهوم **تراکنش** در بحث پایگاه داده ها، ویژگی های تراکنش ها، انواع قفل گذاری روی تراکنش ها و تأثیر آن ها بر نحوه ی اجرای تراکنش ها و همچنین مفاهیم مربوط به **بن بست** و **سطوح انزوای تراکنش ها** آشنا می شویم.

### 8.2. مفاهیم اولیه

#### 8.2.1. تعریف تراکنش

تراکنش یک واحد کاری پایگاه داده، متشکل از مجموعه ای از فعالیت ها است که **یا به طور کامل** انجام می شود و یا **اصلاً انجام نمی شود** و به همین دلیل به آن یک واحد کاری می گویند. تراکنش ها دارای 4 ویژگی هستند که به ویژگی های ACID معروف هستند:

**Atomicity, Consistency, Isolation, Durability**

در ادامه به بررسی این ویژگی ها می پردازیم.

#### 8.2.2. ویژگی های ACID

##### • یکپارچگی (Atomicity)

این خاصیت تضمین می کند که **یا تمامی عملیات** مربوط به تراکنش **یا موفقیت** انجام می شود و یا در صورت وجود هرگونه مشکل در اجرای قسمتی از تراکنش، **تمامی تراکنش دچار شکست** می شود کلیه ی تغییرات انجام شده لغو شده و پایگاه داده به حالت قبل از اجرای تراکنش باز می گردد.

برای مثال تراکنشی را در نظر بگیرید که در آن مقداری پول از حساب A کم شده و به حساب B اضافه می شود. در این صورت تراکنش یا به طور کامل انجام می شود و یا این که به کلی لغو می شود. (این امکان وجود ندارد که پول از حساب A کم شده و به حساب B اضافه نشود!)

##### • سازگاری (Consistency)

این خاصیت تضمین می کند که هر تراکنش، خواه موفق خواه ناموفق، پایگاه داده را در یک **وضعیت سازگار** قرار می دهد یعنی **جامعیت داده ها** حفظ می شود و پایگاه داده همواره **سازگار با قیودی** است که برای آن تعریف شده.

به عنوان مثال در بعضی از جداول مقادیر قابل استفاده برای یک خانه از جدول با استفاده از قید check تعریف شده اند. در این صورت برای تغییر این خانه از جدول باید مطابق با قیودی که از قبل تعریف شده است عمل کنیم.

- **انزوا (Isolation)**

هر تراکنش باید کاملاً مستقل و مجزا از سایر تراکنش ها عمل کند و مستقل از اینکه سایر تراکنش ها چه کارهایی انجام می دهند، کار خود را انجام دهد. به بیان دیگر نتیجه ی اجرای چند تراکنش بصورت همزمان باید با نتیجه ی اجرای پشت سر هم همان تراکنش ها برابر باشد.

به عنوان مثال حالتی را در نظر بگیرید که حساب A دارای مقدار 500 دلار موجودی می باشد. فرض کنید قرار است مقدار 400 دلار از حساب A به حساب B منتقل شود و همچنین 300 دلار هم از حساب A به حساب C منتقل شود. در این صورت اگر هر دو تراکنش با هم موجودی حساب A را 500 دلار بخوانند شروع به عملیات انتقال کرده و در پایان برای پایگاه داده مشکل پیش خواهد آمد. در صورتی که اگر به صورت متوالی این کار را انجام دهند عملیات انتقال برای تراکنش دوم به علت کمبود موجودی انجام نمی شود. پایگاه داده از بروز چنین مشکلاتی جلوگیری می کند که در ادامه بیشتر با آن آشنا خواهید شد.

- **ماندگاری (Durability)**

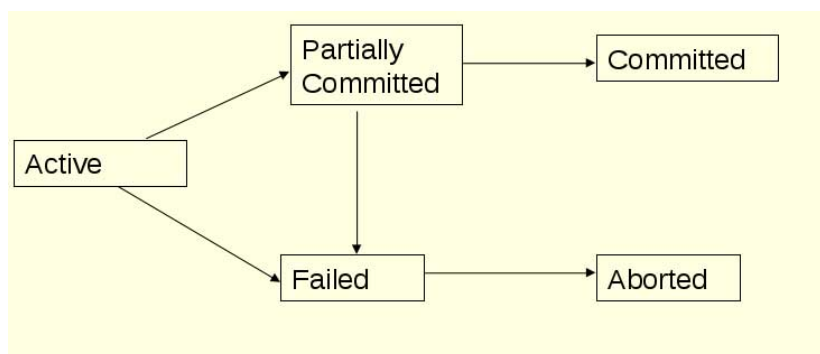
نتایج حاصل از اجرای موفق یک تراکنش، باید در سیستم باقی بماند حتی در صورت وقوع خطای سیستم. اغلب سیستم های مدیریت بانک اطلاعاتی رابطه ای، از طریق ثبت تمام فعالیت های تغییر دهنده ی داده ها در بانک اطلاعاتی، ماندگاری را تضمین می کنند. در صورت خرابی، سیستم قادر است آخرین بهنگام سازی موفق را بهنگام راه اندازی مجدد، بازیابی کند.

### 8.3. وضعیت های مختلف یک تراکنش

تراکنش ها در سیستم همانند یک موجودیت (entity) فعال هستند. همانطور که می دانید ساده ترین موجودیت فعال در سیستم فرآیند ها (process) می باشند که cpu را بعنوان یک ابزار در اختیار گرفته و وظایفی را انجام می دهند. تراکنش نیز یک موجودیت فعال می باشد و همانند سایر موجودیت های فعال دارای وضعیت هایی (state) می باشند که در ادامه هریک شرح داده شده اند :

- **فعال (Active):** تراکنشی که در حالت اجرا است در وضعیت فعال می باشد.
- **کامیت جزئی (Partially Committed):** پس از اجرای آخرین دستور، تراکنش به وضعیت کامیت جزئی می رود.
- **شکست (Failed):** این وضعیت، در روند اجرا خطایی رخ داده و اجرای ادامه تراکنش امکان پذیر نمی باشد.

- خاتمه (Aborted): پس از تشخیص خطا تراکنش می تواند به وضعیت Aborted که در آن جا اجرا متوقف شده و تغییرات ROLLBACK می شوند.
  - Committed: در این وضعیت اجرای تراکنش با موفقیت انجام شده و تراکنش پایان می پذیرد.
- در ادامه نمودار حالت تراکنش ها نشان داده شده است:



شکل 1-8

#### 8.4. دستورات کنترل تراکنش

در ادامه با مهمترین دستورات کنترل یک تراکنش آشنا می شویم که برای مدیریت تراکنش ها لازم هستند.

##### • BEGIN TRANSACTION

این دستور نقطه‌ی شروع یک تراکنش را مشخص می کند. و ساختار این دستور بصورت زیر است.

```
BEGIN TRAN[SACTION] [<transaction name>|<@transaction variable>]
[WITH MARK ['<description>']][;]
```

برای تعریف یک تراکنش، نخستین گام تعریف نقطه‌ی آغاز تراکنش است و این کار با استفاده از عبارت BEGIN TRAN و یا BEGIN TRANSACTION انجام می شود و نامی را هم که برای تراکنش انتخاب می کنیم در قسمت <transaction name> قرار می دهیم.

##### • COMMIT TRANSACTION

این دستور پایان موفق یک تراکنش را مشخص می کند و بعد از اجرای این دستور است که **کلیدی تغییراتی** که تراکنش انجام داده **پایدار** باقی می ماند. (طبق خاصیت ماندگاری (Durability)) و بعد از این تنها راه بازگشت به حالت قبل از اجرای موفق تراکنش، تعریف و اجرای تراکنشی است که از نظر کارایی کاملاً عکس این تراکنش باشد.

ساختار این دستور بصورت زیر است.

```
COMMIT [TRAN[SACTION] [<transaction name>|<@transaction variable>]][];
```

### • ROLLBACK TRANSACTION

به کمک این دستور می توان در هنگام وقوع خطایی در اجرای تراکنش، به حالت قبل از اجرای تراکنش بازگشت در واقع این دستور باعث می شود کلیه کارهای انجام شده توسط تراکنش از دستور BEGIN TRANSACTION فراموش شود (مگر اینکه در بدنه ی تراکنش SAVE POINT تعریف شده باشد که بعدتر به آن خواهیم پرداخت) ساختار کلی این دستور بصورت زیر است.

```
ROLLBACK TRAN[SACTION] [<transaction name>|<save point name>|  
<@transaction variable>|<@savepoint variable>]][];
```

### • SAVE POINT

با استفاده از این دستورات می توان نقاط بازگشتی را تعریف کرد که با استفاده از دستور ROLLBACK به جای بازگشت به ابتدای تراکنش، به این نقاط برگشت. ساختار این دستور بصورت زیر است:

```
SAVE TRAN[SACTION] [<save point name>|<@savepoint variable>]][];
```

نکته ی حائز اهمیت در اینجا این است که پس از اجرای یک دستور ROLLBACK به یکی از SAVEPOINT ها، همه ی SAVEPOINT هایی که تعریف شده اند از بین می روند و در صورت نیاز باید مجدداً SAVEPOINT تعریف کرد.

با یک مثال مرحله به مرحله پیش می رویم و موارد فوق را بررسی می کنیم:  
ابتدا یک جدول با نام MyTranTest در AdventureWorks2012 ایجاد می کنیم:

```
USE AdventureWorks2012  
GO  
CREATE TABLE MyTranTest  
(  
OrderID INT PRIMARY KEY IDENTITY  
);
```

سپس یک تراکنش با نام TranStart1 می سازیم:

```
BEGIN TRAN TranStart1;
```

با استفاده از دستور زیر هر بار یک رکورد به جدول اضافه می کنیم:

```
INSERT INTO MyTranTest  
DEFAULT VALUES;
```

بعد از اضافه شدن رکورد با ID=1 یک SAVEPOINT ایجاد می کنیم:

```
SAVE TRAN FirstPoint;
```

رکورد بعدی با ID=2 را اضافه می کنیم:

```
INSERT INTO MyTranTest  
DEFAULT VALUES;
```

سپس با دستور `ROLLBACK TRAN FirstPoint;` کلیدی تغییرات تا SAVEPOINT اول نادیده گرفته می شوند و این SAVEPOINT هم حذف می شود.

```
ROLLBACK TRAN FirstPoint;
```

رکورد بعدی با ID=3 را اضافه می کنیم و بعد از آن هم یک SAVEPOINT تعریف می کنیم و سپس رکورد با ID=4 را اضافه می کنیم.

```
INSERT INTO MyTranTest  
DEFAULT VALUES;  
SAVE TRAN SecondPoint;  
INSERT INTO MyTranTest  
DEFAULT VALUES;
```

سپس با دستور `ROLLBACK TRAN FirstPoint;` کلیدی تغییرات تا SAVEPOINT دوم نادیده گرفته می شوند و این SAVEPOINT هم حذف می شود. سپس رکورد بعدی با ID=5 را اضافه می کنیم و با دستور COMMIT تراکنش بصورت موفقیت آمیز پایان می یابد.

```
ROLLBACK TRAN SecondPoint;  
INSERT INTO MyTranTest  
DEFAULT VALUES;  
COMMIT TRAN TranStart1;
```

حال 3 رکورد اول از جدول MyTranTest را انتخاب می کنیم و بعد جدول را حذف می کنیم.

```
SELECT TOP 3 OrderID  
FROM MyTranTest  
ORDER BY OrderID DESC;  
DROP TABLE MyTranTest;
```

همانطور که انتظار می رود خروجی بصورت زیر است:

5  
3  
1

در ادامه به بررسی دستوراتی می پردازیم که به کمک آن ها می توانیم وضعیت یک تراکنش و سطح آن تراکنش را در تراکنش های تودرتو بدست آوریم.

#### • @@TRANCOUNT

با استفاده از @@TRANCOUNT می توان پرس وجوهایی برای یافتن سطح تراکنش طراحی کرد. اگر مقدار @@TRANCOUNT برابر با 0 بود یعنی در این نقطه، کد داخل تراکنش نیست. اگر مقدار @@TRANCOUNT بزرگ تر از 0 بود یعنی تراکنش فعال است و اعداد بزرگ تر از 1 هم نمایانگر سطح

تراکنش در تراکنش های تودرتو است. به عبارت دیگر می توان گفت که @@TRANCOUNT تعداد تراکنش های فعال را به ما می دهد. هر دستور BEGIN TRANSACTION یک واحد به @@TRANCOUNT اضافه می کند و هر دستور ROLLBACK TRANSACTION و COMMIT TRANSACTION یک واحد از @@TRANCOUNT کم می کند درحالی که ROLLBACK به یک SAVEPOINT تغییری در @@TRANCOUNT ایجاد نمی کند.

با بررسی کد زیر و نتیجه ی حاصل از آن مطالب فوق روشن تر خواهد شد:

```
USE AdventureWorks2012
GO
SELECT @@TRANCOUNT
BEGIN TRAN;
SELECT @@TRANCOUNT;
BEGIN TRAN;
SAVE TRAN S1
SELECT @@TRANCOUNT
ROLLBACK TRAN S1
SELECT @@TRANCOUNT
COMMIT
SELECT @@TRANCOUNT
ROLLBACK TRAN;
SELECT @@TRANCOUNT
```

نتیجه:

	(No column name)
1	0
	(No column name)
1	1
	(No column name)
1	2
	(No column name)
1	2
	(No column name)
1	1
	(No column name)
1	0

#### • XACT\_STATE()

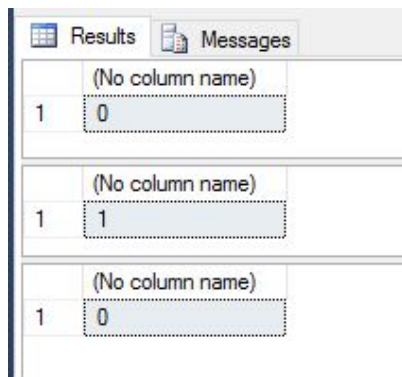
با استفاده از XACT\_STATE() می توان پرس وجوهایی برای یافتن وضعیت تراکنش طراحی کرد. اگر مقداری که XACT\_STATE() برمی گرداند، برابر با 0 بود یعنی تراکنش فعالی وجود ندارد. اگر مقداری که XACT\_STATE() برمی گرداند، برابر با 1 بود یعنی تراکنش COMMIT نشده ای وجود دارد و این تراکنش می تواند COMMIT شود. و اگر مقداری که XACT\_STATE() برمی گرداند، برابر با -1 بود یعنی تراکنش

COMMIT نشده‌ای وجود دارد که این تراکنش بدلیل وجود خطاهای قبلی در سیستم، نمی‌تواند COMMIT شود.

با بررسی کد زیر و نتیجه‌ی حاصل از آن مطالب فوق روشن‌تر خواهد شد:

```
SELECT XACT_STATE()  
BEGIN TRAN  
SELECT XACT_STATE()  
COMMIT  
SELECT XACT_STATE()
```

نتیجه:



(No column name)	
1	0

(No column name)	
1	1

(No column name)	
1	0

## 8.5. انواع تراکنش‌ها در SQL SERVER

تراکنش‌ها را براساس حالتشان در SQL SERVER می‌توان به 3 دسته تقسیم کرد که در ادامه به آن‌ها می‌پردازیم:

### • حالت خودکار (AUTOCOMMIT)

در این حالت با دستورات ساده‌ای از قبیل Insert, Update, Delete همچون تراکنش رفتار می‌شود یعنی اگر موفقیت‌آمیز خاتمه یابند، خودبه‌خود COMMIT می‌شوند و اگر خطایی در اجرای آن‌ها رخ دهد، ROLLBACK می‌شوند. این حالت، حالت پیش‌فرض SQL SERVER برای تراکنش‌ها است. (نیازی به نوشتن هیچ دستور اضافه‌ای نیست)

### • حالت ضمنی (IMPLICIT)

در این حالت وقتی یک دستور DML یا DDL یا SELECT را می‌نویسید، SQL SERVER یک تراکنش ایجاد می‌کند و @@TRANCOUNT یک واحد افزایش می‌دهد اما برخلاف حالت AUTOCOMMIT تراکنش

را بصورت خودکار، COMMIT یا ROLLBACK نمی کند و شما باید COMMIT یا ROLLBACK را به کد اضافه کنید.

این حالت پیش فرض SQL SERVER نیست و بنابراین باید با نوشتن دستور زیر شروع یک تراکنش ضمنی را به SQL SERVER اعلام کرد.

**SET IMPLICIT\_TRANSACTIONS ON;**

برای واضح تر شدن موضوع کد زیر را در سیستم خود بر روی پایگاه داده ی AdventureWorks2012 اجرا کرده و نتیجه را برای مسئول آزمایشگاه توضیح دهید.

```
SET IMPLICIT_TRANSACTIONS ON;
begin tran t1
update Sales.SalesPerson
set TerritoryID=3
where BusinessEntityID=276

begin tran t2
select TerritoryID
from Sales.SalesPerson
where BusinessEntityID=276
commit

rollback
```

### کد تمرین پژوهشی:

مزایا و معایب استفاده از تراکنش های ضمنی را بیان کنید.

#### • حالت صریح (EXPLICIT)

در این حالت آغاز تراکنش با دستور **BEGIN TRAN(SACTION)** و پایان تراکنش با **ROLLUP** **TRAN(SACTION)** و یا **COMMIT TRAN(SACTION)** به طور صریح مشخص می شوند.



## 8.6. قفل گذاری و همزمانی

همزمانی یکی از مسائل مهم در بحث پایگاه داده‌ها است و بیانگر این مفهوم است که دو یا چند کاربر (یا تراکنش) می‌خواهند با یک موجودیت تعامل کنند. البته ماهیت تعامل‌ها می‌تواند متفاوت باشد (update, delete, read, insert) و انتخاب نوع قفل گذاری و برقراری همزمانی هم به ماهیت تعامل کاربران (یا تراکنش‌ها) بستگی دارد. قفل گذاری هم از مسائل مهم در بحث پایگاه داده‌های رابطه‌ای است و با جلوگیری از وقوع عملیات همزمان روی داده، موجب تضمین جامعیت داده‌ها می‌شود. قفل‌ها معمولاً بصورت پویا و توسط lock manager (بخشی از موتور پایگاه داده) کنترل و مدیریت می‌شوند. بطور کلی به کمک قفل‌ها می‌توان از بروز مشکلات زیر جلوگیری کرد:

### • Dirty Read

این مشکل زمانی به وقوع می‌پیوندد که یک تراکنش داده‌ای را بخواند که تراکنش دیگری که هنوز commit نشده، آن داده را تغییر داده باشد. در این صورت اگر تراکنشی که داده را تغییر داده در ادامه مجدداً داده را تغییر دهد یا به جای آنکه commit شود، rollback شود، تراکنش اول داده را اشتباه می‌خواند. برای روشن تر شدن این موضوع به مثال زیر توجه کنید:

Transaction 1 Command	Transaction 2 Command	Logical Database Value	Uncommitted Database Value	What Transaction 2 Shows
BEGIN TRAN		3		
UPDATE col = 5	BEGIN TRAN	3	5	
SELECT anything	SELECT @var = col	3	5	5
ROLLBACK	UPDATE anything SET 3 whatever = @var			5

شکل 8-2

### • Non-repeatable reads

این مشکل زمانی به وقوع می‌پیوندد که یک تراکنش داده‌ای را دو مرتبه بخواند و در بین این دو مرتبه یک تراکنش دیگر داده را تغییر دهد. در این صورت خروجی تراکنشی که دو مرتبه یک داده را با دو مقدار متفاوت خوانده، نامعتبر خواهد بود. برای روشن تر شدن بحث به مثال زیر توجه کنید:

Transaction 1	Transaction 2	@Var	What Transaction 1 Thinks Is in The Table	Value in Table
BEGIN TRAN		NULL		125
SELECT @Var = value FROM table	BEGIN TRAN	125	125	125
	UPDATE value, SET value = value - 50			75
IF @Var >=100	END TRAN	125	125	75
UPDATE value, SET value = value - 100		125	125 (waiting for lock to clear)	75
(Finish, wait for lock to clear, then continue)		125	75	Either: -25 (If there isn't a CHECK constraint enforcing > 0) Or: Error 547 (If there is a CHECK)

شکل 3-8

#### • **Phantoms**

Phantom به رکوردهایی گفته می‌شود که به‌طور غیرمنتظره‌ای در نتایج ظاهر می‌شوند و این غیرمنتظره بودن ناشی از آن است که مثلاً برخلاف انتظار ما تحت تأثیر یک update یا delete قرار نگرفته‌اند. مثلاً حالتی را در نظر بگیرید که یک تراکش همه‌ی رکورد های یک جدول را پاک (delete) کند و در همین حین تراکش دیگری چند رکورد به آن اضافه کند. در این صورت انتظار ما حذف همه‌ی رکورد ها بوده در صورتی که هنوز چندین رکورد در جدول باقی مانده است.

#### • **Lost updates**

این مشکل زمانی به وقوع می‌پیوندد که دو update روی یک رکورد بصورت تقریباً همزمان رخ دهند بنابراین عملیاتی که لحظه‌ای زودتر تمام شده، نتیجه‌اش روی رکورد باقی نمی‌ماند برای روشن شدن موضوع به مثال زیر توجه کنید:

فرض کنید اعتبار حساب مشتری X برابر 5000 است. شما حساب او را باز می‌کنید تا اعتبار را افزایش دهید. همزمان همکار شما هم همین حساب را باز می‌کند تا آدرس مشتری را تغییر دهد پس او هم دقیقاً اطلاعاتی را می‌بیند که شما دیده‌اید. حال ابتدا شما اعتبار حساب این مشتری را به 7500 تغییر می‌دهید و عملیات بروز رسانی اطلاعات حساب توسط شما به پایان می‌رسد. لحظاتی بعد هم همکار شما آدرس جدید مشتری را وارد می‌کند و عملیات بروز رسانی اطلاعات حساب توسط همکار شما به پایان می‌رسد حال نتیجه این گونه است که چون عملیات همکار شما قبل از

پایان عملیات شما شروع شده و بعد از آن خاتمه یافته، نتایج حاصل از عملیات شما بر روی اطلاعات حساب گم شده! و فقط آدرس مشتری X تغییر کرده.

قفل‌ها را می‌توان برحسب نیاز روی منابع مختلف (پایگاه داده، جدول، سطر و ...) قرار داد و همچنین می‌توان از انواع قفل‌ها استفاده کرد که مهمترین آن‌ها را در ادامه معرفی می‌کنیم:

#### • قفل اشتراکی<sup>۲</sup>

از این قفل‌ها هنگام انجام عملیات خواندن استفاده می‌کنیم تا حین انجام عملیات خواندن، داده‌ها تغییر نکنند. قفل‌های اشتراکی با یکدیگر سازگار هستند و بنابراین می‌توان از چندین قفل اشتراکی بصورت همزمان استفاده کرد.

#### • قفل انحصاری<sup>۳</sup>

همانطور که از اسمشان مشخص است این قفل‌ها وقتی روی یک منبع قرار می‌گیرند، آن منبع را منحصراً در اختیار یک کاربر (تراکنش) قرار می‌دهند و با هیچ نوع قفل دیگری سازگاری ندارند و نمی‌توانند همزمان با قفل دیگری فعال شوند. با وجود این قفل‌ها هیچ دو کاربری نمی‌توانند همزمان عملیات delete, update یا هر عملیات دیگری را روی منبعی که این قفل را دارد انجام دهند.

#### • قفل به‌روزرسانی<sup>۴</sup>

می‌توان عملیات به‌روزرسانی را متشکل از دو مرحله دانست:  
مرحله اول: جستجو توسط عبارت where برای یافتن داده‌ای که باید به‌روز شود.  
مرحله دوم: انجام عملیات به‌روزرسانی روی داده یافته شده در مرحله اول.  
می‌توان گفت قفل‌های به‌روزرسانی در مرحله اول به‌روزرسانی از اشتراکی هستند و در مرحله دوم به‌روزرسانی به قفل انحصاری تبدیل می‌شوند. و این از بروز بن‌بست جلوگیری می‌کند. (در ادامه به بحث بن‌بست وارد می‌شویم).

### 8.7. بن‌بست<sup>۵</sup>

بن‌بست زمانی رخ می‌دهد که دو تراکنش هرکدام یک منبع در اختیار دارند و آن را قفل کرده‌اند و هر یک منتظر دیگری هستند تا منبعی که در دست آن تراکنش است را بگیرند و قفل کنند بنابراین این دو تراکنش باید تا بینهایت

<sup>2</sup> Shared Lock

<sup>3</sup> Exclusive Lock

<sup>4</sup> Update Lock

<sup>5</sup> Deadlock

منتظر یکدیگر باشند و هیچ‌یک نمی‌توانند کار خود را به اتمام برسانند. برای روشن شدن بحث به مثال زیر که وقوع یک بن‌بست را نشان می‌دهد، توجه کنید:

Session 1	Session 2
USE TSQL2012; BEGIN TRAN;	USE TSQL2012; BEGIN TRAN;
UPDATE HR.Employees SET Region = N'10004' WHERE empid = 1	
	UPDATE Production.Suppliers SET Fax = N'555-1212' WHERE supplierid = 1
UPDATE Production.Suppliers SET Fax = N'555-1212' WHERE supplierid = 1	
<blocked>	UPDATE HR.Employees SET phone = N'555-9999' WHERE empid = 1
	<blocked>

شکل 4-8

به‌طور معمول در sql server تراکنش‌ها مدت تعریف‌شده‌ای برای آزاد شدن یک منبع و استفاده از آن منتظر می‌مانند مگر آنکه sql server تشخیص وقوع بن‌بست دهد. در این صورت sql server بر اساس تخمینی از هزینه‌ی rollback کردن هر تراکنش، یکی از تراکنش‌ها را به عنوان قربانی انتخاب کرده و آن را rollback می‌کند تا تراکنش دیگر بتواند کار خود را ادامه دهد.

### تمرین پژوهشی:

برای کاهش امکان وقوع بن‌بست چه کارهایی می‌توان انجام داد؟

## 8.8. سطوح انزوا<sup>6</sup>

تاکنون با مشکلاتی که در نبود قفل‌ها مکن است رخ دهد و همچنین برخی از قفل‌ها آشنا شدیم اما ذکر این نکته ضروری است که بین تراکنش‌ها و قفل‌ها رابطه‌ای ناگسستگی وجود دارد. در sql server وقتی تراکنشی در حال تغییر یک داده یا در حال نوشتن است، هیچ تراکنش دیگری نمی‌تواند به آن داده دسترسی داشته باشد و بنابراین وقوع بن‌بست در این شرایط گاهی اجتناب‌ناپذیر است. اما میزان وقوع بن‌بست می‌تواند با توجه به سطح انزوای انتخاب شده برای تراکنش‌ها کم یا زیاد شود. در ادامه به بررسی مهمترین و پراستفاده‌ترین سطوح انزوای تراکنش‌ها آشنا می‌شویم.

### • Read Uncommitted

این سطح انزوا اجازه‌ی خواندن رکوردهایی که توسط یک تراکنش Commit نشده، تغییر کرده‌اند را می‌دهد. بنابراین درگیری تراکنش‌ها را بسیار کم می‌کند اما ممکن است dirty read, phantoms, non-repeatable reads رخ دهند.

### • Read Committed

این سطح انزوا، سطح پیش‌فرض برای تمام تراکنش‌ها در sql server است و اجازه‌ی خواندن داده‌هایی که در تراکنش دیگری تغییر کرده‌اند را فقط به شرطی می‌دهد که آن تراکنش commit شده باشد. بنابراین dirty read رخ نخواهد داد اما ممکن است phantoms, non-repeatable reads رخ دهند.

### • Repeatable Read

این سطح انزوا تضمین می‌کند که هر رکوردی که در تراکنش خوانده شود، بعداً هم مجدداً می‌تواند خوانده شود و اجازه‌ی تغییر یا حذف رکوردهای خوانده‌شده را نمی‌دهد. بنابراین قفل‌های اشتراکی را تا پایان تراکنش نگه می‌دارد.

### • Snapshot

در این سطح، تصویری (به عبارت بهتر یک کپی) از داده‌ها به تراکنش داده می‌شود که هیچ قفلی روی آن نیست و تراکنش می‌تواند هر عملیاتی را روی آن انجام دهد و تغییرات این تراکنش روی آن کپی از داده‌ها توسط سایر تراکنش‌ها قابل‌رؤیت نیست همان‌طور که این تراکنش تغییراتی را که سایر تراکنش‌ها روی داده‌ها انجام می‌دهند را نمی‌بیند.

---

<sup>6</sup> Isolation levels

- **Serializable**

این سطح از انزوا که بیشترین درگیری را بین تراکنش‌ها بوجود می‌آورد اجازه‌ی تغییر در داده‌ای را که توسط یک تراکنش Commit نشده، خوانده شده است، نمی‌دهد تا زمانی که آن تراکنش خاتمه پذیرد و مثل آن است که تراکنش‌های وابسته به یک منبع مشترک کاملاً پشت سرهم (و نه بصورت همزمان) اجرا می‌شوند.

پس از اینکه به کمک این اطلاعات تصمیم گرفتیم از چه سطح انزوایی باید استفاده شود، با استفاده از دستور زیر سطح انزوای موردنظر را تنظیم می‌کنیم که برای همه‌ی تراکنش‌های یک ارتباط (connection) تنظیم می‌گردد.

```
SET TRANSACTION ISOLATION LEVEL {READ COMMITTED|READ UNCOMMITTED|SNAPSHOT|SERIALIZABLE|REPEATABLE READ}
```

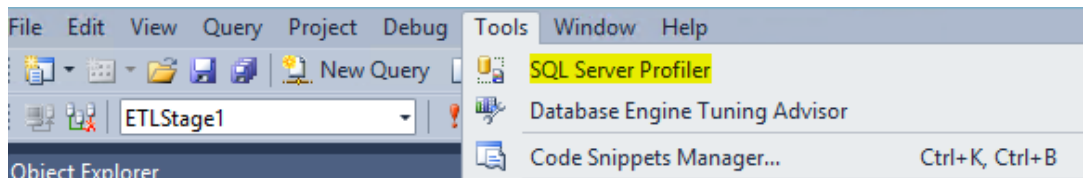
## 8.9. آشنایی با SQL Profiler

SQL Server ابزار بسیار خوبی ارائه می‌کند که اجازه می‌دهد تا وضعیت‌های در حال اجرا را روی SQL Server خود مشاهده کنید، و همچنین در جمع‌آوری مقیاس‌هایی مانند استمرار، تعداد خواندن‌ها، تعداد نوشتن‌ها، ماشینی که query را اجرا می‌کند و غیره کمک می‌کند که این ابزار Profiler نامیده می‌شود. SQL Server Profiler در واقع یک ابزار گرافیکی برای نظارت بر ردیابی و کارایی SQL Server است. با استفاده صحیح از این برنامه شما می‌توانید کارایی و Performance بانک اطلاعاتی خود را بالا ببرید. این برنامه کلیه Command‌های اجرایی را که به سمت SQL Server ارسال می‌شود را با توجه به شرایطی که تعیین کرده‌اید می‌گیرد. این برنامه به صورت پیش فرض روی SQL Server نصب است.

### 8,9,1. گام‌های استفاده از SQL Profiler

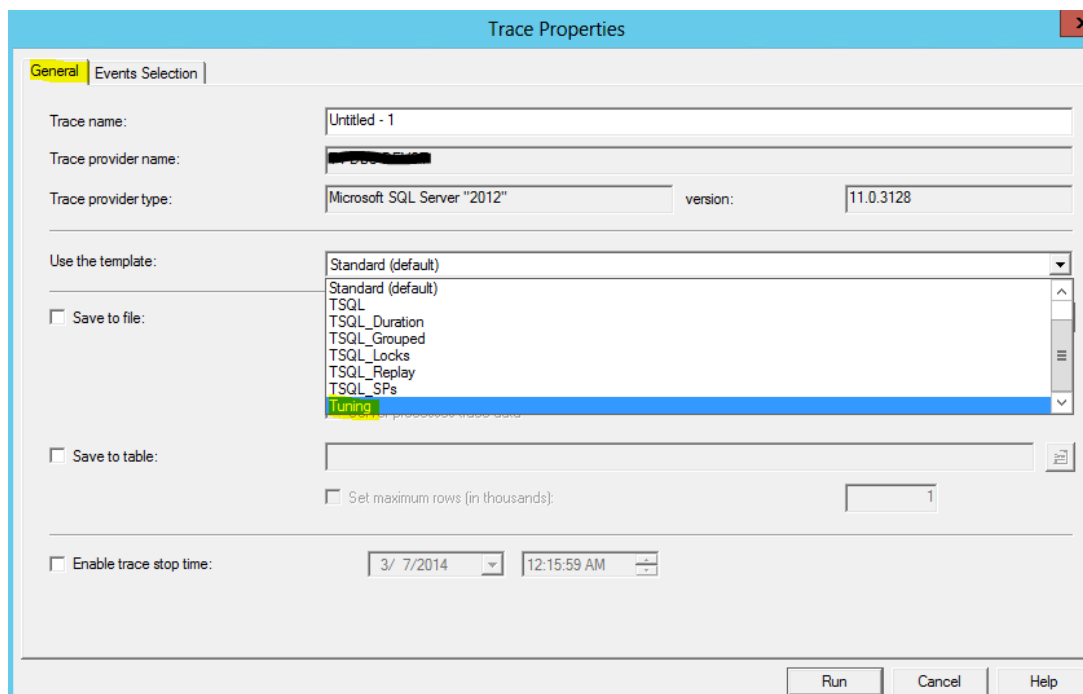
- گام اول: ایجاد یک فایل TRACE/LOAD برای پایگاه داده

ابتدا SQL Server Profiler را باز کنید.



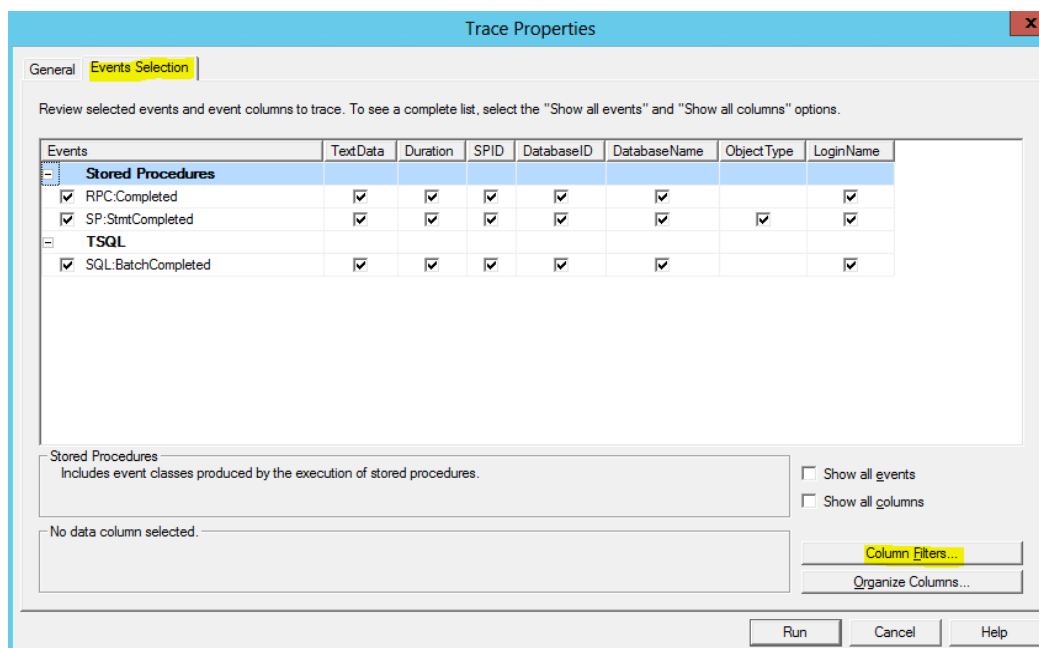
شکل 5-8

گزینه Connect to the server را انتخاب کنید سپس گزینه‌ی Tuning را از General، tab انتخاب کنید.



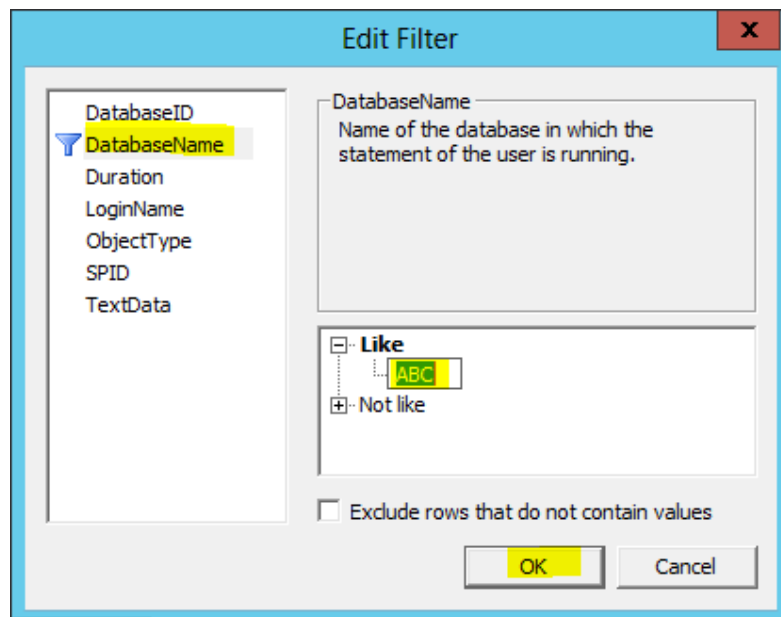
شکل 6-8

گزینه Column Filters را از tab Events Selection انتخاب کنید.



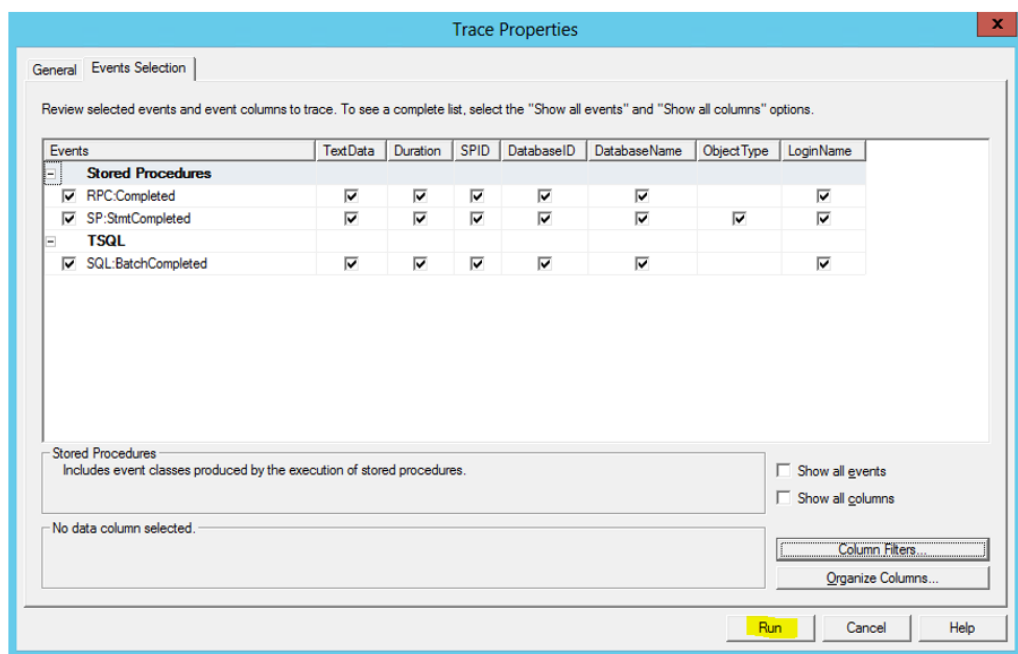
شکل 7-8

DatabaseName را از منو انتخاب کنید و سپس نام پایگاه داده را وارد کنید به عنوان مثال در اینجا نام پایگاه داده ABC انتخاب شده است که بر اساس آن فایل trace/load برای بهینه سازی پرس و جو ایجاد خواهد شد.



شکل 8-8

روی گزینه‌ی ok کلیک کنید.



شکل 9-8

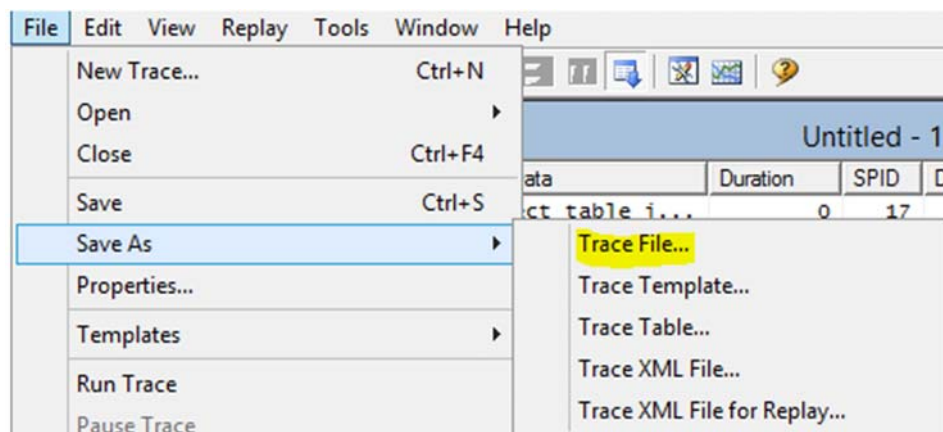
روی گزینه‌ی Run کلیک کنید.



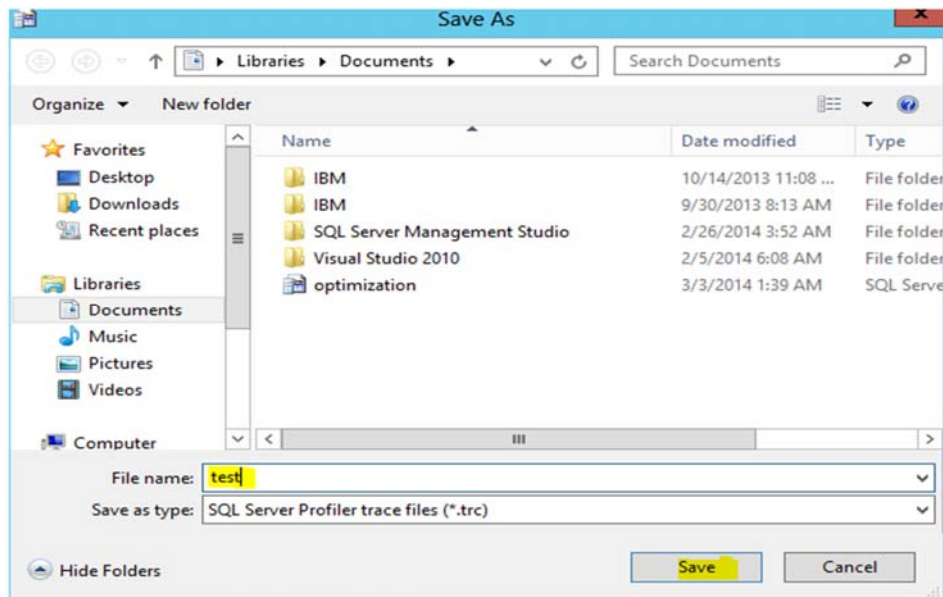
EventClass	TextData	Duration	SPID	DatabaseID	Object Type
SP:StmtCompleted	select table_i...	0	22	7	20816 - PQ
SP:StmtCompleted	select table_i...	0	9	7	20816 - PQ
SP:StmtCompleted	select table_i...	0	27	7	20816 - PQ
SP:StmtCompleted	select table_i...	0	20	7	20816 - PQ
SP:StmtCompleted	select table_i...	0	9	7	20816 - PQ
SP:StmtCompleted	select table_i...	0	19	7	20816 - PQ
SP:StmtCompleted	select table_i...	0	22	7	20816 - PQ
SP:StmtCompleted	select table_i...	0	30	7	20816 - PQ
SP:StmtCompleted	select table_i...	0	22	7	20816 - PQ
SP:StmtCompleted	select table_i...	0	24	7	20816 - PQ
SP:StmtCompleted	select table_i...	0	27	7	20816 - PQ
SP:StmtCompleted	select table_i...	0	30	7	20816 - PQ
SP:StmtCompleted	SELECT StatMan...	9	56	7	20801 - AQ
SP:StmtCompleted		15	56	7	20801 - AQ
SQL:BatchCompleted	select * from ...	21	56	7	
SQL:BatchCompleted	select * from ...	4	56	7	

شکل 10-8

Trace file را ذخیره کنید.



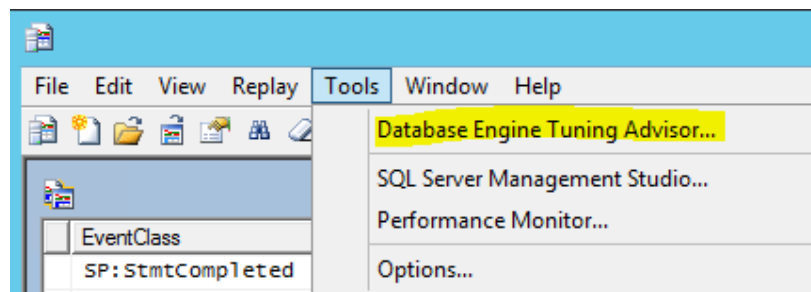
شکل 11-8



شکل 8-12

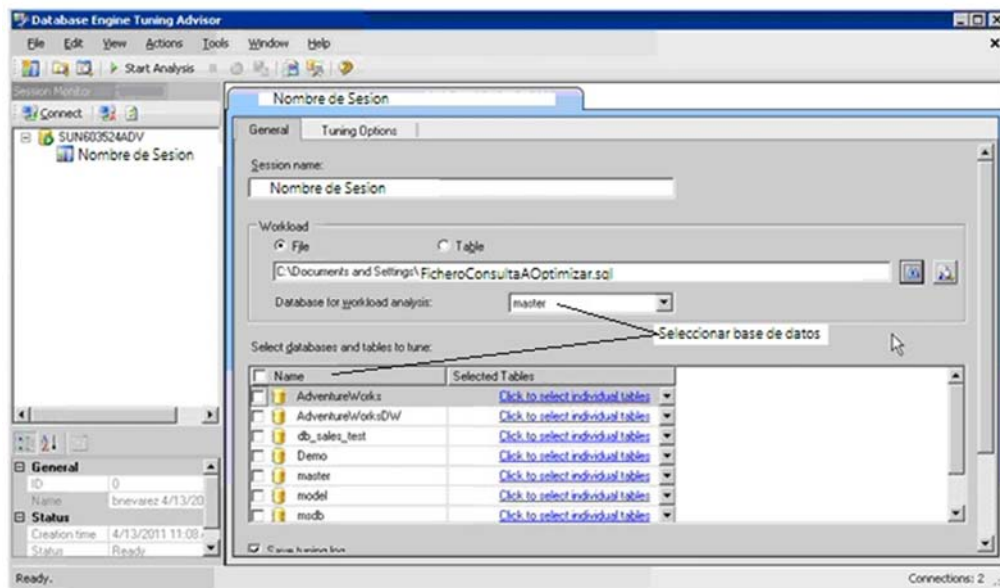
- گام دوم: فایل **LOAD** را در برنامه **Tuning Wizard** قرار دهید

Database Engine Tuning Wizard را باز کنید.



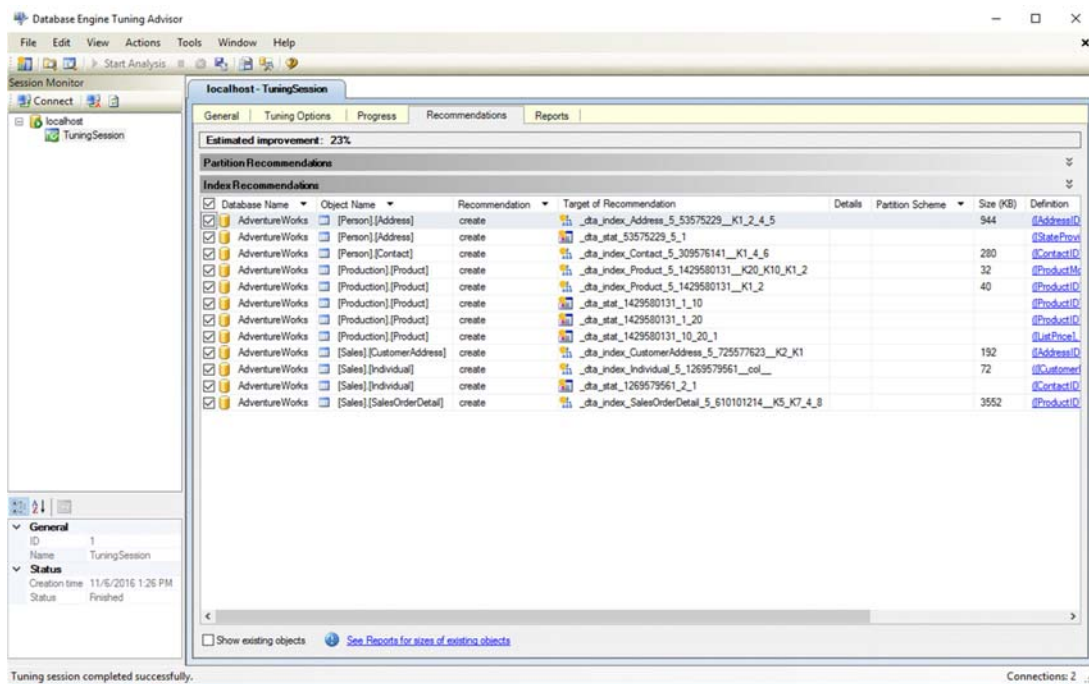
شکل 8-13

گزینه **File** و نام پایگاه داده را از قسمت **General** انتخاب کنید و سپس گزینه **Start Analysis** را انتخاب کنید.



شکل 8-14

- گام سوم: قسمت Suggestion / definition ایجاد شده توسط Tuning Wizard را بررسی کنید



شکل 8-15

## 8.10. تمرین

1- با یک مثال برای Adventure works (شییه مثالی که برای بن بست آورده شد) نشان دهید که قفل

انحصاری با قفل اشتراکی سازگاری ندارد.

2- سناریوهایی برای نشان دادن Dirty Read و Non Repeatable Read طراحی و اجرا کنید.

## 8.11. منابع

MCTS\_70-433\_SQLServer\_2008DatabaseDevelopment

Wiley.Microsoft.SQL.Server.2008.Bible.Aug.2009

sql\_tutorial from [www.tutorialspoint.com](http://www.tutorialspoint.com)

Professional Microsoft SQL Server 2008 Programming

sql\_server\_2012\_t-sql\_recipes\_3rd\_edition

Microsoft.Press.Training.Kit.Querying.Microsoft.SQL.Server.2012.Exam.70-461

Training Kit (Exam 70-462)\_Administering Microsoft SQL Server 2012 Databases