Triggery (wyzwalacze)

Triggery

- **Trigger (wyzwalacz)** to fragment kodu SQL składający się z instrukcji deklaratywnych i/lub proceduralnych i przechowywany w katalogu RDBMS
  - specjalne procedury składowane, które są wykonywane automatycznie w odpowiedzi na zdarzenia obiektu bazy danych, bazy danych i serwera
- Triggery mogą również odwoływać się do typów atrybutów w innych tabelach

Triggery i ich rodzaje

- Triggery DML - wywoływane automatycznie w odpowiedzi na zdarzenia INSERT, UPDATE i DELETE w tabelach
- Triggery DDL - uruchamiane w odpowiedzi na instrukcje CREATE, ALTER i DROP; również uruchamiane w odpowiedzi na niektóre procedury składowane w systemie, które wykonują operacje podobne do DDL
- Triggery logowania - uruchamiane w odpowiedzi na zdarzenia LOGON

Tworzenie triggerów

```
CREATE TRIGGER [schema_name.]trigger_name
ON table_name
AFTER  {[INSERT],[UPDATE],[DELETE]}
[NOT FOR REPLICATION]
AS
{sql_statements}
```

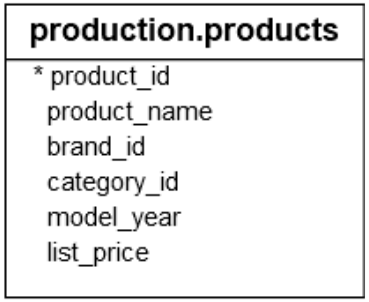
Wirtualne tabele dla triggerów

Automatycznie tworzone tabele: deleted i inserted (temporalne, przechowywane w pamięci)

- nie można bezpośrednio modyfikować danych w tych tablicach
- nie można wykonywać operacji DDL

Zdarzenie DML	Tabela INSERTED	Tabela DELETED
INSERT	wiersze do wstawienia	pusta
UPDATE	nowe wiersze zmodyfikowane	istniejące wiersze zmodyfikowane
DELETE	pusta	wiersze do usunięcia

Przykład triggera



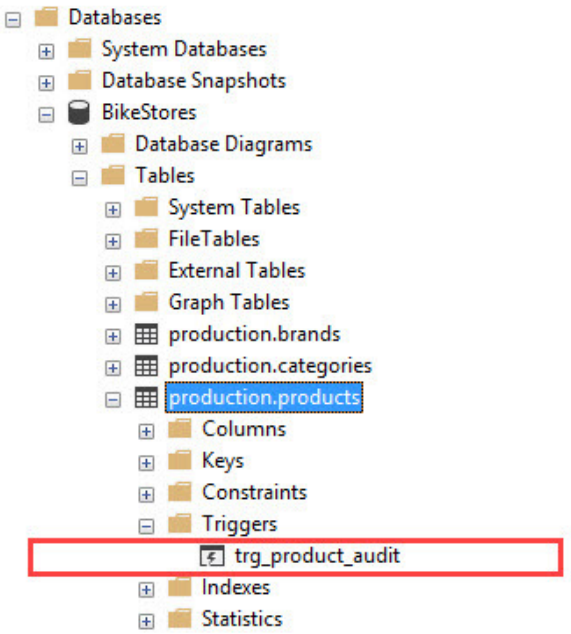

```
CREATE TABLE production.product_audits(
    change_id INT IDENTITY PRIMARY KEY,
    product_id INT NOT NULL,
    product_name VARCHAR(255) NOT NULL,
    brand_id INT NOT NULL,
    category_id INT NOT NULL,
    model_year SMALLINT NOT NULL,
    list_price DEC(10,2) NOT NULL,
    updated_at DATETIME NOT NULL,
    operation CHAR(3) NOT NULL,
    CHECK(operation = 'INS' or operation='DEL')
);
```

Przykład triggera AFTER DML

```
CREATE TRIGGER production.trg_product_audit
ON production.products
AFTER INSERT, DELETE
AS
BEGIN
    SET NOCOUNT ON;
```

Przykład triggera

```
INSERT INTO
   production.product_audits
      (
         product_id,
         product_name,
         brand_id,
         category_id,
         model_year,
         list_price,
         updated_at,
         operation
      )
SELECT
   i.product_id,
   product_name,
   brand_id,
   category_id,
   model_year,
   i.list_price,
   GETDATE(),
   'INS'
FROM
   inserted AS i
UNION ALL
   SELECT
      d.product_id,
      product_name,
      brand_id,
      category_id,
      model_year,
      d.list_price,
      getdate(),
      'DEL'
   FROM
      deleted AS d;
```



Testowanie triggera - insert

```
INSERT INTO production.products(
    product_name,
    brand_id,
    category_id,
    model_year,
    list_price
)
VALUES (
    'Test product',
    1,
    1,
    2018,
    599
);
```

odpalenie triggera production.trg_product_audit

Zawartość tabeli production.product_audits:

change_id	product_id	product_name	brand_id	category_id	model_year	list_price	updated_at	operation
1	322	Test product	1	1	2018	599.00	2018-10-14 15:23:46.837	INS

Testowanie triggera - delete

```
DELETE FROM
    production.products
WHERE
    product_id = 322;
```

odpalenie triggera production.trg_product_audit

Zawartość tabeli production.product_audits:

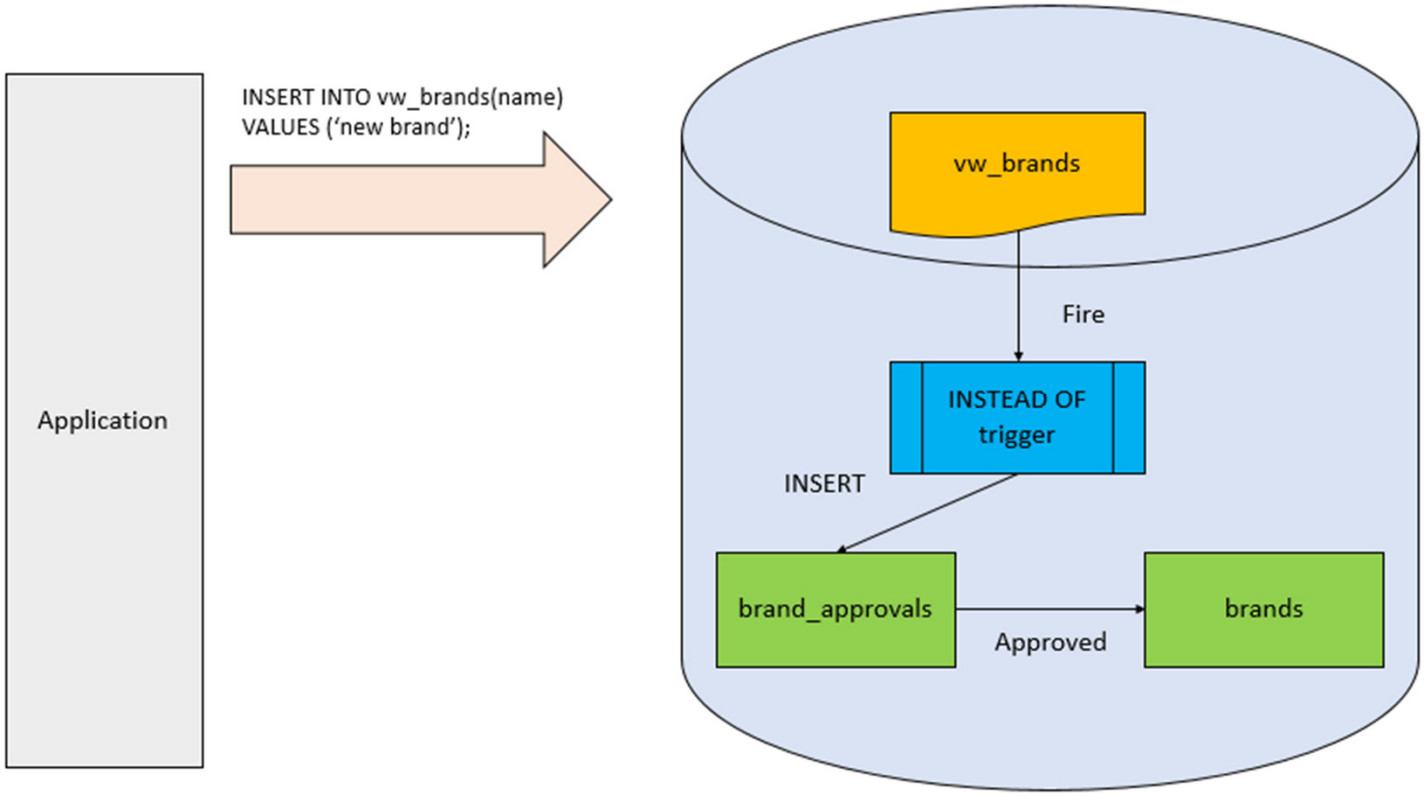
change_id	product_id	product_name	brand_id	category_id	model_year	list_price	updated_at	operation
1	322	Test product	1	1	2018	599.00	2018-10-14 15:23:46.837	INS
2	322	Test product	1	1	2018	599.00	2018-10-14 15:26:34.050	DEL

Trigger INSTEAD OF

- Pominięcie instrukcji INSERT, DELETE lub UPDATE do tabeli lub widoku i wykonanie innych instrukcji zdefiniowanych w triggerze

```
CREATE TRIGGER [schema_name.] trigger_name
ON {table_name | view_name }
INSTEAD OF {[INSERT] [,] [UPDATE] [,] [DELETE] }
AS
{sql_statements}
```

Trigger INSTEAD OF - przykład




```
CREATE TABLE production.brands (
brand_id INT IDENTITY (1, 1) PRIMARY KEY,
brand_name VARCHAR (255) NOT NULL
);
```

Trigger INSTEAD OF - przykład

- Tworzenie nowej tabeli o nazwie production.brand_approvals do przechowywania marek oczekujących na zatwierdzenie:

```
CREATE TABLE production.brand_approvals(
    brand_id INT IDENTITY PRIMARY KEY,
    brand_name VARCHAR(255) NOT NULL
);
```

Trigger INSTEAD OF - przykład

- Tworzenie nowego widoku o nazwie production.vw_brands na tabelach production.brands i production.brand_approvals:

```
CREATE VIEW production.vw_brands
AS
SELECT
   brand_name,
   'Approved' approval_status
FROM
   production.brands
UNION
SELECT
   brand_name,
   'Pending Approval' approval_status
FROM
   production.brand_approvals;
```

Trigger INSTEAD OF - przykład

- Wstawienie wiersza do widoku production.vw_brands -> wstawienie do tabeli production.brand_approvals za pomocą triggera INSTEAD OF:

```
CREATE TRIGGER production.trg_vw_brands
ON production.vw_brands
INSTEAD OF INSERT
AS
BEGIN
   SET NOCOUNT ON;
   INSERT INTO production.brand_approvals (
      brand_name
   )
   SELECT
      i.brand_name
   FROM
      inserted i
   WHERE
      i.brand_name NOT IN (
         SELECT
            brand_name
         FROM
            production.brands
      );
END
```

Trigger INSTEAD OF - przykład

- Wstawienie nowej marki do widoku production.vw_brands:

```
INSERT INTO production.vw_brands(brand_name)
VALUES('Eddy Merckx’);
```

  - uruchomienie wyzwalacza INSTEAD OF, do wstawienia nowego wiersza do tabeli production.brand_approvals
  - production.vw_brands:

brand_name	approval_status
Eddy Merckx	Pending Approval
Electra	Approved
Haro	Approved
Heller	Approved
Pure Cycles	Approved
Ritchey	Approved
Strider	Approved
Sun Bicycles	Approved
Surly	Approved
Trek	Approved

  - production.brand_approvals:

brand_id	brand_name
1	Eddy Merckx

Trigger DDL

- Odpowiadają na zdarzenia serwera lub bazy danych, a nie na modyfikacje danych w tabeli
  - zdarzenia utworzone przez instrukcję SQL, która zwykle rozpoczyna się od CREATE, ALTER, DROP, GRANT, DENY, REVOKE lub UPDATE STATISTICS
  - np. wyzwalacz DDL, który będzie rejestrował za każdym razem, gdy użytkownik wyda instrukcję CREATE TABLE lub ALTER TABLE.

Trigger DDL - zastosowanie

- Rejestrowanie zmian w schemacie bazy danych
- Zapobieganie pewnym zmianom w schemacie bazy danych
- Odpowiedź na zmiany w schemacie bazy danych

```
CREATE TRIGGER trigger_name
ON { DATABASE |  ALL SERVER}
[WITH ddl_trigger_option]
FOR {event_type | event_group }
AS {sql_statement}
```

Trigger DDL - przykład

- Przechwytywanie wszystkich modyfikacji wprowadzanych do indeksu bazy danych, aby lepiej monitorować wydajność serwera bazy danych w odniesieniu do tych zmian indeksu.
- Tworzenie tabeli index_logs:

```
CREATE TABLE index_logs (
    log_id INT IDENTITY PRIMARY KEY,
    event_data XML NOT NULL,
    changed_by SYSNAME NOT NULL
);
```

Trigger DDL - przykład

- Trigger do śledzenia zmian indeksu

```
CREATE TRIGGER trg_index_changes
ON DATABASE
FOR
   CREATE_INDEX,
   ALTER_INDEX,
   DROP_INDEX
AS
BEGIN
   SET NOCOUNT ON;

   INSERT INTO index_logs (
      event_data,
      changed_by
   )
   VALUES (
      EVENTDATA(),
      USER
   );
END;
GO
```

Trigger DDL - przykład

- Tworzenia indeksów na imię i nazwisko:

```
CREATE NONCLUSTERED INDEX nidx_fname
ON sales.customers(first_name);
CREATE NONCLUSTERED INDEX nidx_lname
ON sales.customers(last_name);
```

- czy zdarzenie zostało prawidłowo przechwycone przez trigger:
  - SELECT * FROM index_logs;

log_id	event_data	changed_by
1	<EVENT_INSTANCE><EventType>CREATE_INDEX</EventType><PostTime>2018-...	dbo
2	<EVENT_INSTANCE><EventType>CREATE_INDEX</EventType><PostTime>2018-...	dbo

```xml
<EVENT_INSTANCE>
  <EventType>CREATE_INDEX</EventType>
  <PostTime>2018-10-16T07:48:21.417</PostTime>
  <SPID>52</SPID>
  <ServerName>DESKTOP-57TKHNJ</ServerName>
  <LoginName>sa</LoginName>
  <UserName>dbo</UserName>
  <DatabaseName>BikeStores</DatabaseName>
  <SchemaName>sales</SchemaName>
  <ObjectName>nidx_fname</ObjectName>
  <ObjectType>INDEX</ObjectType>
  <TargetObjectName>customers</TargetObjectName>
  <TargetObjectType>TABLE</TargetObjectType>
  <TSQLCommand>
    <SetOptions ANSI_NULLS="ON" ANSI_NULL_DEFAULT="ON"
    <CommandText>CREATE NONCLUSTERED INDEX nidx_fname
ON sales.customers(first_name)</CommandText>
  </TSQLCommand>
</EVENT_INSTANCE>
```

Wyłączanie/włączanie triggera

- Czasami, w celu rozwiązywania problemów lub odzyskiwania danych, można tymczasowo wyłączyć trigger

```
DISABLE TRIGGER [schema_name.][trigger_name]
ON [object_name | DATABASE | ALL SERVER]

ENABLE TRIGGER [schema_name.][trigger_name]
ON [object_name | DATABASE | ALL SERVER]
```

Wyświetlanie definicji triggera

- Widok sys.sql_modules:

```
SELECT
   definition
FROM
   sys.sql_modules
WHERE
   object_id = OBJECT_ID('sales.trg_members_delete’);
```

- Procedura składowana sp_helptext stored

```
EXEC sp_helptext 'sales.trg_members_delete'
```

Triggery

- Zalety
  - Automatyczne monitorowanie i weryfikacja w przypadku określonych zdarzeń lub sytuacji
  - Modelowanie dodatkowej semantyki i/lub reguł integralności bez zmiany interfejsu użytkownika lub kodu aplikacji
  - Przypisywanie wartości domyślnych do typów atrybutów dla nowych krotek
  - Synchroniczne aktualizacje w przypadku replikacji danych
  - Automatyczne audyty i logowania, które mogą być trudne do wykonania w dowolnej innej warstwie aplikacji
  - Automatyczne eksportowanie danych

Triggery

- Wady
  - Ukryta funkcjonalność, która może być trudna do śledzenia i zarządzania
  - Efekty kaskadowe prowadzące do nieskończonej pętli triggera wyzwalającego inny wyzwalacz itp.
  - Niepewne wyniki, jeśli zdefiniowano wiele triggerów dla tego samego obiektu bazy danych i zdarzenia
  - Zakleszczenia
  - Złożoność debugowania
  - Problemy z utrzymaniem i wydajnością