

## 1. 트리거(Trigger)란?



트리거(Trigger)란 영어로 방아쇠라는 뜻인데, 방아쇠를 당기면 그로 인해 총기 내부에서 알아서 일련의 작업을 실행하고 총알이 날아갑니다. 이처럼 데이터베이스에서도 트리거(Trigger)는 특정 테이블에 INSERT, DELETE, UPDATE 같은 DML 문이 수행되었을 때, 데이터베이스에서 자동으로 동작하도록 작성된 프로그램입니다. 즉! 사용자가 직접 호출하는 것이 아니라, 데이터베이스에서 자동적으로 호출하는 것이 가장 큰 특징입니다.

트리거(Trigger)는 테이블과 뷰 데이터베이스 작업을 대상으로 정의할 수 있으며, <sup>①</sup>전체 트랜잭션 작업에 대해 발생하는 트리거(Trigger)와 <sup>②</sup>각행에 대해 발생하는 트리거(Trigger)가 있습니다.

## 2. 트리거(Trigger)가 적용되는 예

다음과 같은 상황에서 트리거(Trigger)를 사용할 수 있습니다. 어떤 쇼핑몰에 하루에 수만 건의 주문이 들어옵니다. 주문데이터는 주문일자, 주문상품, 수량, 가격이 있으며, 수천명의 임직원이 일자별, 상품별 총 판매수량과 총 판매가격으로 구성된 주문 실적을 실시간으로 온라인상에 조회를 했을 때, 한사람의 임직원이 조회할 때마다 수만 건의 데이터를 읽고 계산해야합니다. 만약 임직원이 수만명이고, 데이터가 수백만건이라면, 또 거의 동시다발적으로 실시간 조회가 요청된다면 시스템 퍼포먼스가 떨어질 것입니다.

<sup>③</sup>따라서! 트리거(Trigger)를 사용하여 주문한 건이 입력될 때마다, 일자별 상품별로 판매수량과 판매금액을 집계하여 집계자료를 보관하도록 만들어보겠습니다. 먼저 관련된 테이블을 생성해보겠습니다.

주문정보테이블		
필드명	타입	길이
ORDER_DATE	CHAR	8
PRODUCT	VARCHAR2	10
QTY	NUMBER	
AMOUNT	NUMBER	
일자별판매집계테이블		
필드명	타입	길이
SALE_DATE	CHAR	8
PRODUCT	VARCHAR2	10
QTY	NUMBER	
AMOUNT	NUMBER	

테이블은 다음과 같습니다. 주문정보테이블에 실시간으로 데이터가 입력될 때마다 트리거가 발동되어 자동으로 일자별판매집계테이블에 일자별, 상품별 판매수량과 판매금액을 계산해 업데이트 하는 작업을 하도록 하고, 사용자들은 미리 계산된 일자별판매집계테이블을 조회하게 하여 실시간 조회를 지원하게 하는 것입니다.

자, 이제 2개 테이블을 CREATE, DDL을 통해 만들어보겠습니다.

```
CREATE TABLE ORDER_LIST(  
    ORDER_DATE CHAR(8) NOT NULL,  
    PRODUCT    VARCHAR2(10) NOT NULL,  
    QTY        NUMBER NOT NULL,  
    AMOUNT     NUMBER NOT NULL  
);
```

```
CREATE TABLE SALES_PER_DATE(  
    SALE_DATE CHAR(8) NOT NULL,  
    PRODUCT   VARCHAR2(10) NOT NULL,  
    QTY       NUMBER NOT NULL,  
    AMOUNT    NUMBER NOT NULL  
);
```

```
SELECT *  
FROM ORDER_LIST
```

```
SELECT *  
FROM SALES_PER_DATE
```

조회를 해보면 다음과 같습니다.

	ORDER_DATE	PRODUCT	QTY	AMOUNT
--	------------	---------	-----	--------

	SALE_DATE	PRODUCT	QTY	AMOUNT
--	-----------	---------	-----	--------

아직 2개 테이블다 데이터가 없음을 확인할 수 있습니다.

이제 트리거(Trigger)를 만들어 보겠습니다. 트리거(Trigger)를 구현하기 위해 우선 절차형 SQL과 PL/SQL을 알아야합니다. 절차형 SQL과 PL/SQL은 따로 글을 올리도록 하고, 오늘은 절차형 SQL과 PL/SQL을 안다는 전제하에 트리거(Trigger)를 구현해보겠습니다.

```
8 ▶ create or replace trigger summary_sales
9   after insert
10  on order_list
11  for each row
12  declare
13    o_date order_list.order_date%type;
14    o_prod order_list.product%type;
15  • begin
16  •   o_date := :new.order_date;
17  •   o_prod := :new.product;
18  •   update sales_per_date
19  •       set qty = qty + :new.qty,
20  •           amount = amount + :new.amount
21  •       where sale_date = o_date
22  •           and product = o_prod;
23  •   if sql%notfound then
24  •       insert into sales_per_date
25  •           values(o_date, o_prod, :new.qty, :new.amount);
26  •   end if;
27  • end;
28  /
```

트리거(Trigger) 처리 절차를 설명하면 다음과 같습니다.

8 ~ 14 Line

Trigger를 선언합니다.

~~Order\_list~~ 테이블에 insert가 발생하면 그 이후 *order-list 테이블의 each row들* each row 즉 각 행에 해당 트리거(Trigger)를 적용한다라는 뜻입니다. 또한, declare 선언문에는 변수를 선언합니다. order\_list 테이블에 있는 order\_date, product Type에 맞게 o\_date, o\_prod 변수를 선언합니다.

15 ~17 Line

~~new~~는 트리거(Trigger)에서 사용하는 구조체입니다. new는 새로 입력된 레코드 값을 담고 있습니다.  
o\_date 에 새로 들어온 order\_date 값을 , o\_prod 에 새로 들어온 product 값을 저장합니다.

18 ~ 26 Line

~~sales\_per\_date~~ 테이블에 update 구문을 실행하는데, 기존에 있는 qty, amount 를 누적합해서 다시 Set 합니다. 여기서 where문을 통해 현재 새로 들어온 날짜와 상품이 일치하는 데이터만 해당 update문을 실행하도록 조건을 걸었습니다.

또한 만약 해당 조건에 모두 해당되지 않는다면, if sql%notfound 구문이 실행됩니다. 기존에 있던 레코드 값이 아니고 전혀 새로운 레코드이기 때문에 insert 구문을 통해 새로 들어온 데이터를 새로 삽입합니다.  
끝으로 / 부분은 트리거(Trigger)를 실행하는 실행명령어입니다.

위 구문을 실행하면, 이제 2개 테이블에 트리거(Trigger)가 적용된 것입니다.

이제 order\_list 테이블에 레코드를 insert 해서 sales\_per\_date 테이블에 트리거(Trigger)가 자동으로 동작하여, 데이터 값을 자동으로 계산하고 반영하는지 확인해보겠습니다!

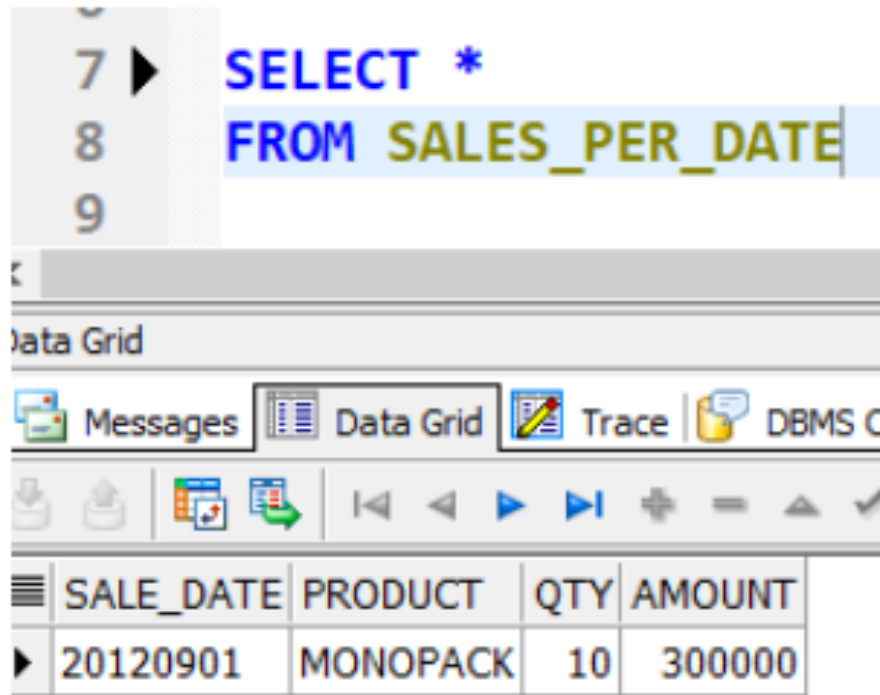
ORDER\_LIST 테이블에 아래와 같이 데이터 값을 삽입해보겠습니다.

-> INSERT INTO ORDER\_LIST VALUES('20120901','MONOPACK',10,300000)

삽입후 ORDER\_LIST, SALES\_PER\_DATE 테이블 조회를 해보겠습니다.



ORDER\_LIST 정상적으로 값이 삽입되었습니다.



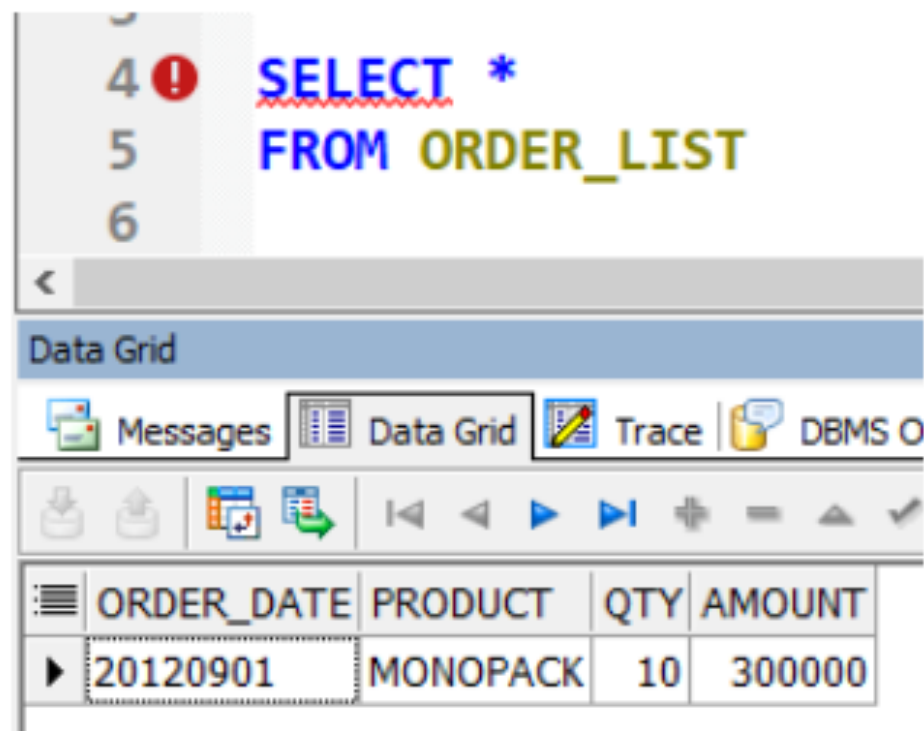
The screenshot shows a SQL query editor with the following SQL statement:

```
7 ► SELECT *  
8 FROM SALES_PER_DATE  
9
```

Below the query editor, the 'Data Grid' tab is active, displaying the results of the query. The grid has four columns: SALE\_DATE, PRODUCT, QTY, and AMOUNT. The first row of data shows a sale on 20120901 for the product MONOPACK with a quantity of 10 and an amount of 300000.

SALE_DATE	PRODUCT	QTY	AMOUNT
20120901	MONOPACK	10	300000

트리거에 의해서 SALES\_PER\_DATE 에도 정상적으로 값이 삽입되어있습니다.



The screenshot shows a SQL query editor with the following SQL statement:

```
4 ! SELECT *  
5 FROM ORDER_LIST  
6
```

Below the query editor, the 'Data Grid' tab is active, displaying the results of the query. The grid has four columns: ORDER\_DATE, PRODUCT, QTY, and AMOUNT. The first row of data shows an order on 20120901 for the product MONOPACK with a quantity of 10 and an amount of 300000.

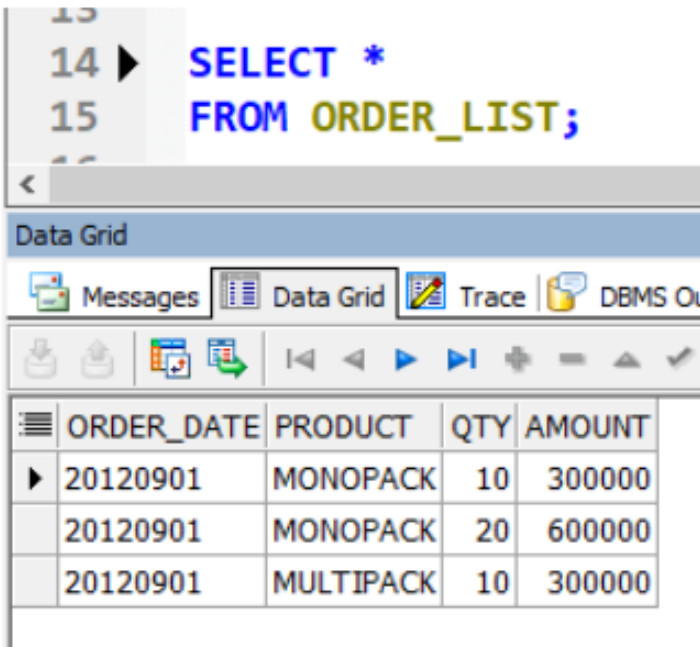
ORDER_DATE	PRODUCT	QTY	AMOUNT
20120901	MONOPACK	10	300000

## 4. 트리거(Trigger)와 트랜잭션(Transaction)의 상관관계

이번에는 다른 상품으로 주문 데이터를 입력한 후 두 테이블의 결과를 조회해보고 트랜잭션을 ROLLBACK 해보겠습니다. 판매 데이터의 입력취소가 일어나면, 주문 정보 테이블(ORDER\_LIST)과 판매 집계테이블(SALES\_PER\_DATE)에 동시에 입력(수정) 취소가 일어나는지 확인해보기 위함입니다.

-> INSERT INTO ORDER\_LIST VALUES('20120901','MULTIPACK',10,300000);

ORDER\_LIST 에 정상적으로 판매 데이터가 삽입되었습니다.



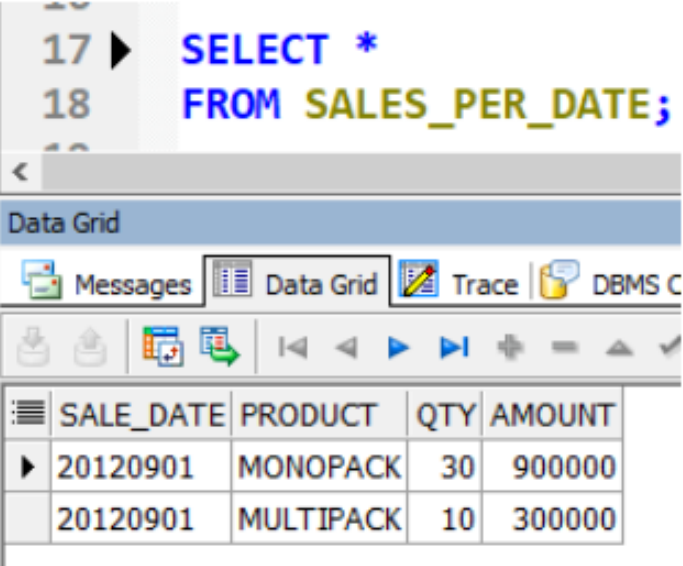
The screenshot shows a SQL IDE with a query editor and a Data Grid. The query editor contains the following SQL statement:

```
14 SELECT *
15 FROM ORDER_LIST;
```

The Data Grid displays the results of the query, showing three rows of data:

ORDER_DATE	PRODUCT	QTY	AMOUNT
20120901	MONOPACK	10	300000
20120901	MONOPACK	20	600000
20120901	MULTIPACK	10	300000

SALES\_PER\_DATE 테이블은 트리거에 의해 판매날짜별, 상품별 누적 물량과 가격이 업데이트 되었습니다.



The screenshot shows a SQL IDE with a query editor and a Data Grid. The query editor contains the following SQL statement:

```
17 SELECT *
18 FROM SALES_PER_DATE;
```

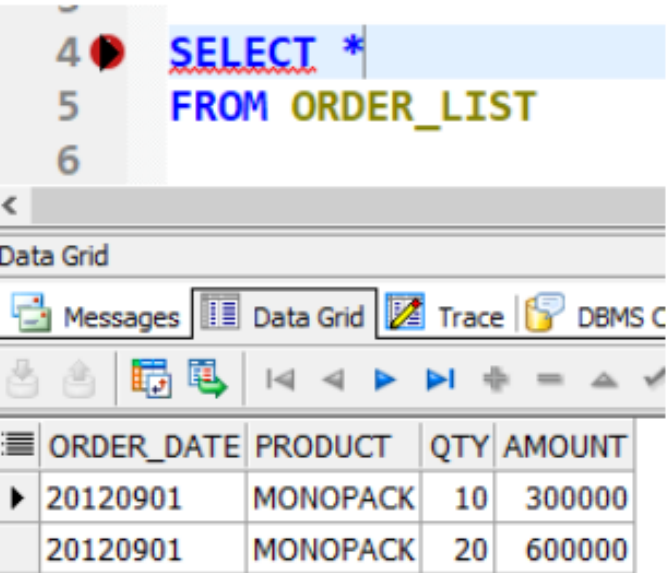
The Data Grid displays the results of the query, showing two rows of data:

SALE_DATE	PRODUCT	QTY	AMOUNT
20120901	MONOPACK	30	900000
20120901	MULTIPACK	10	300000



이제 ROLLBACK; 명령어를 실행해보겠습니다.

ORDER\_LIST 테이블에 방금 삽입한 판매데이터가 취소되었습니다.

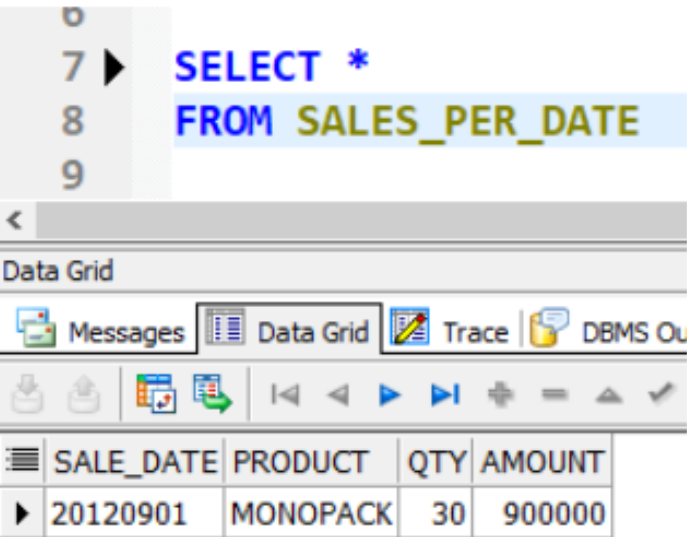


The screenshot shows a SQL Developer window with a query editor containing the following SQL statement:

```
SELECT *  
FROM ORDER_LIST
```

Below the query editor is a 'Data Grid' tab. The grid is empty, indicating that no data was returned from the query, which is consistent with the text stating that the data was rolled back.

SALES\_PER\_DATE 테이블에도 똑같이 트리거로 입력된 데이터 정보까지 하나의 트랜잭션으로 인식하여 입력 취소가 되었습니다.



The screenshot shows a SQL Developer window with a query editor containing the following SQL statement:

```
SELECT *  
FROM SALES_PER_DATE
```

Below the query editor is a 'Data Grid' tab. The grid contains one row of data:

SALE_DATE	PRODUCT	QTY	AMOUNT
20120901	MONOPACK	30	900000

즉, 트리거는 데이터베이스에 의해 자동 호출되지만 결국 INSERT, UPDATE, DELETE 구문과 하나의 트랜잭션 안에서 일어나는 일련의 작업들이라 할 수 있습니다.

(추가로 트리거는 Begin ~ End 절에서 COMMIT , ROLLBACK 을 사용할 수 없습니다.)