UNIT 24

스토어드 프로그램

이 장에서는 DB2가 제공하는 Stored Function 및 Stored Procedure, Trigger 을 통한 Application Logic 작성에 대한 내용을 소개합니다.

DB2 9.7 개발자 가이드

Developer Edition

- Stored Program
- IBM Data Studio Developer를 통해 Stored Program 생성
- IBM Data Studio Developer를 통해 UDF 생성
- IBM Data Studio Developer를 통해 UDF 생성 - 예
- IBM Data Studio Developer를 통해 Table UDF 생성
- Stored Procedure 작성
- Trigger 작성
- 모듈(Module) 작성

TOPIC 03 01 Stored Program

Point

Stored Program은 UDF, Trigger, Stored Procedure등의 작성을 통하여 DBMS내에 응용프로그램 로직을 저장하고 동작하도록 하는 기능입니다.

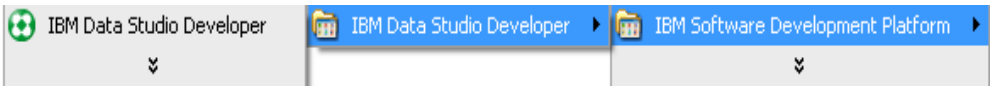
1. Stored Program은 DBMS Application Object 를 사용함으로써 응용프로그램 코드의 단순화, 성능개선, 코드의 재 사용 향상 및 이식성을 증대할 수 있습니다.

2. 일반적은 Editor를 이용하여 작성 및 생성 할수 있습니다.

   그러나 IBM Data Studio Developer를 사용하면 프로그램 디버깅이 가능하여, 작성 시 매우 편리합니다.

3. Data Studio Developer 시작 방법은 아래와 같습니다.

   Windows에서 "시작" → "프로그램" → "IBM Software Development Platform" → "IBM Data Studio Developer " → "IBM Data Studio Developer"를 수행합니다.



Tip

- Data Studio Developer는 9.7에서는 Optim Development Studio로 변경되었습니다.

TOPIC 03 02 Stored Program 생성

Point

IBM Data Studio Developer 실행하여 Stored Program을 생성하는 절차를 소개합니다.

Tip

각 프로젝트는 DB connecti on 이 필요합니다.

1 데이터베이스에 연결합니다.

IBM 에서 "연결" → "새연결" 를 실행합니다.
연결하려는 DB 이름과 호스트 및 포트번호를 그림과 기술합니다.

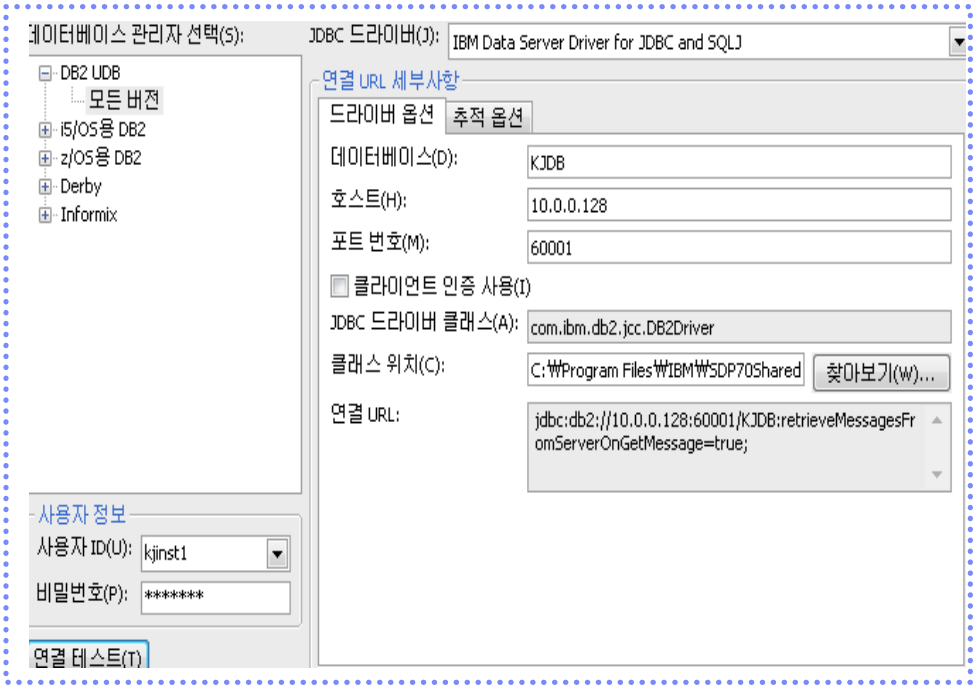


Figure 2402A • • • 데이터베이스 연결

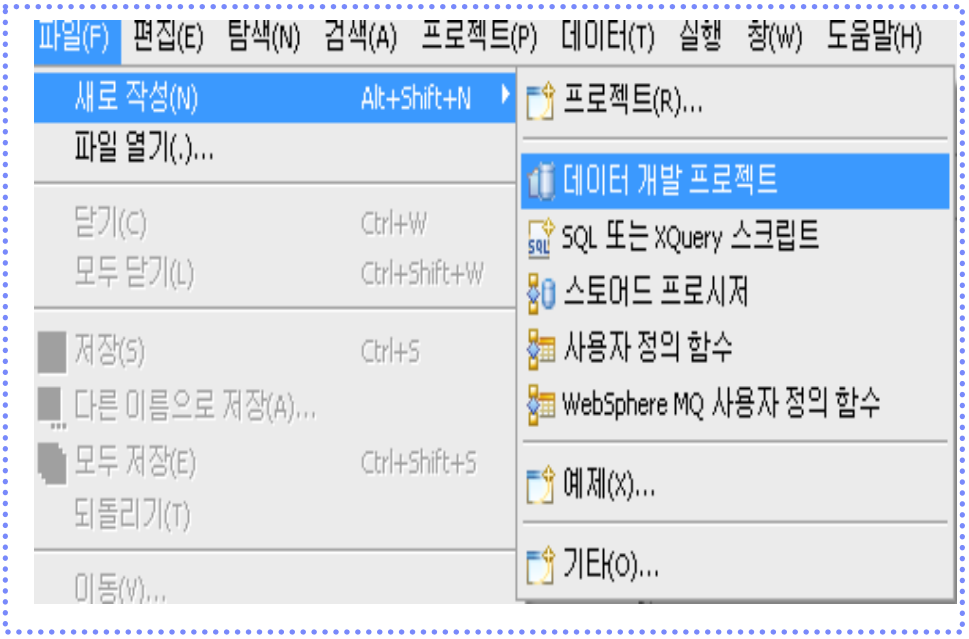


Figure 2402B • • • 데이터 개발 프로젝트를 실행

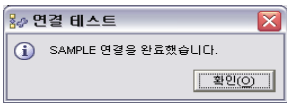
TOPIC 03 02 Stored Program 생성

Point

IBM Data Studio Developer 실행하여 Stored Program을 생성하는 절차를 소개합니다.

Tip

"연결테스트" 가 성공하면 다음과 같은 팝업이 뜹니다.



2 스토어드 프로시저 작성을 선택합니다.

데이터 개발 프로젝트에서 스토어드 프로시저에서 새로 작성을 실행합니다.
이름과 언어 및 개발하려는 상황에 맞게 설정 후 다음 버튼을 클릭합니다.

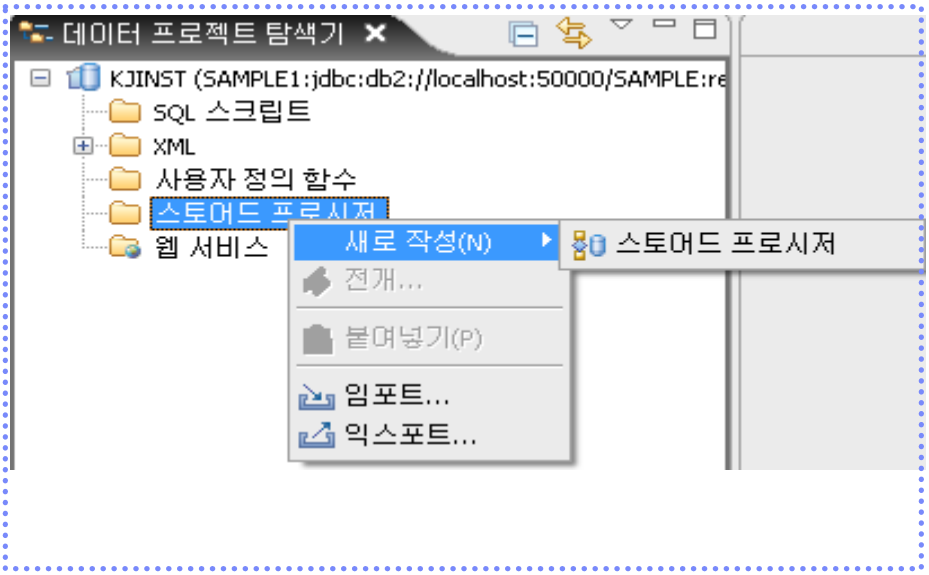


Figure 2402C ••• IBM Data Studio Developer– 스토어드 프로시저작성

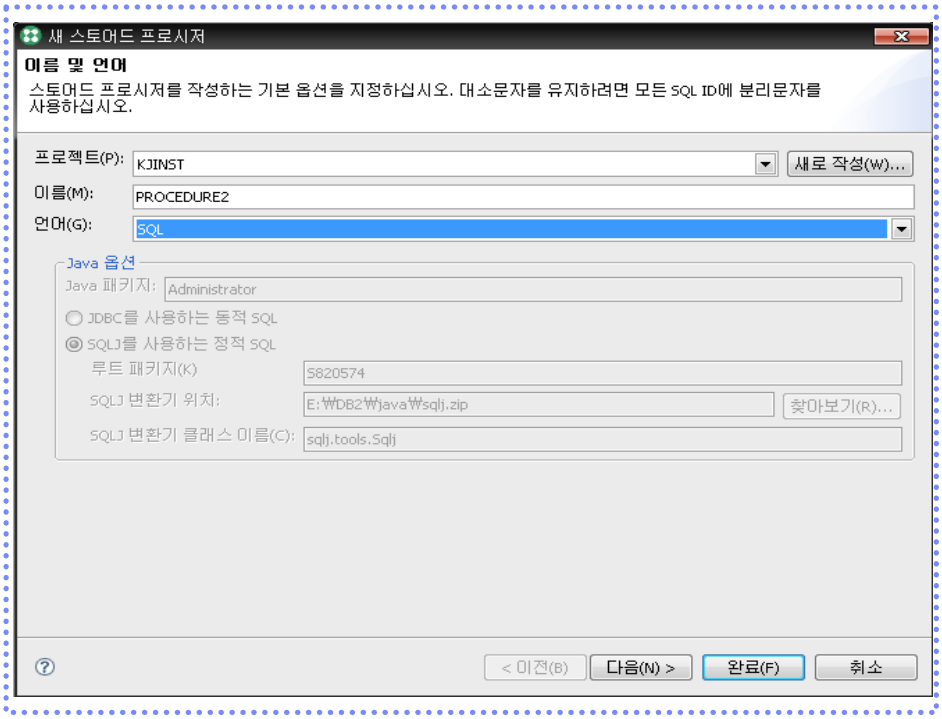


Figure 2402D ••• IBM Data Studio Developer – 스토어드 프로시저 작성

TOPIC 03 02 Stored Program 생성

Point

IBM Data Studio Developer 실행하여 Stored Program을 생성하는 절차를 소개합니다.

3 SQL 작성을 클릭하여 스토어드 프로시저를 작성합니다.

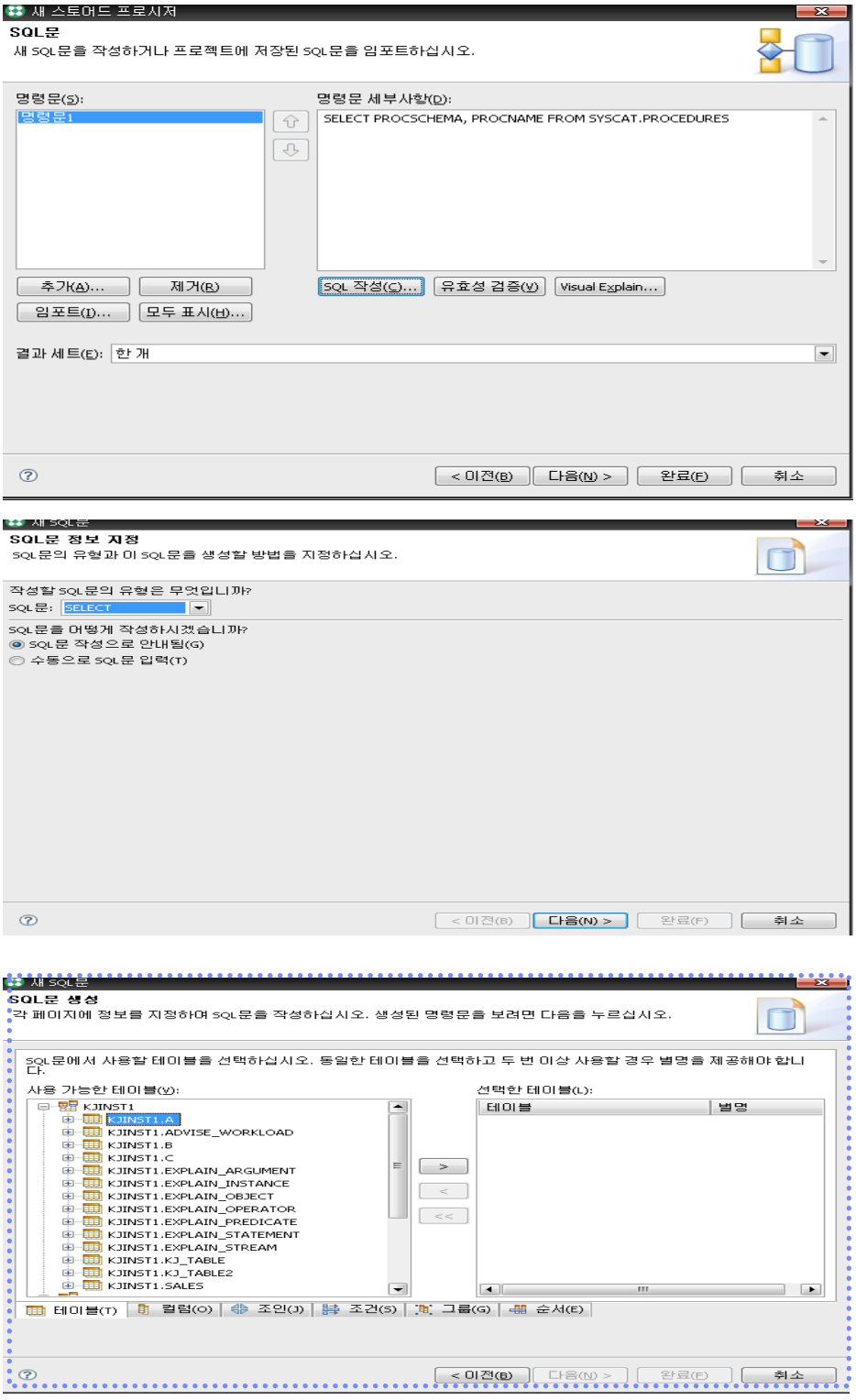


Figure 2402E IBM Data Studio Developer – 스토어드 프로시저 작성

TOPIC 03 03 UDF 생성

Point

IBM Data Studio Developer를 실행하여 User-Defined Function (UDF) 을 작성하는 방법입니다.

Tip

SQL UDF를 단순한 과정을 거쳐 생성합니다. 다음 예는 세금을 계산 하는 간단한 "TAX" function을 생성합니다.

1 UDF 생성하기

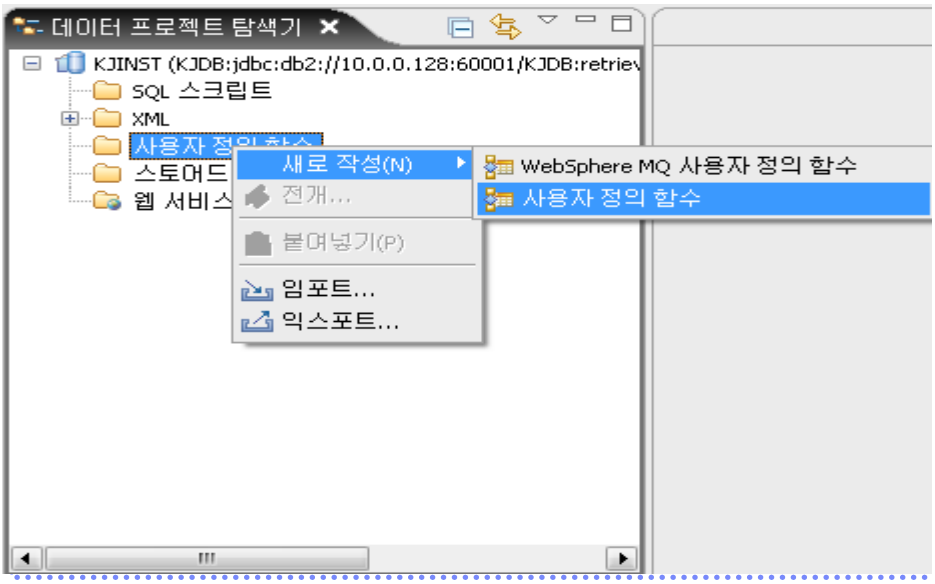
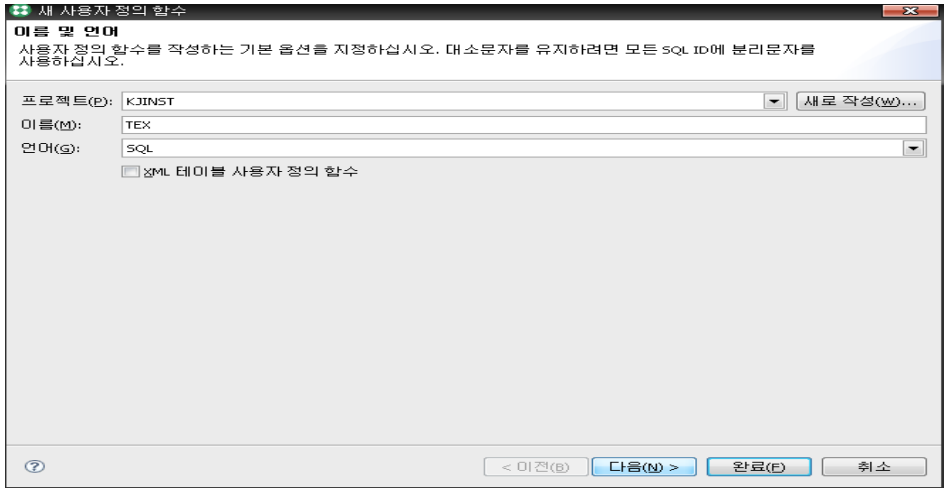


Figure 2403A IBM Data Studio Developer– UDF 새로 작성-1

➲ 다음 그림에서 완료 버튼을 클릭하고 아래와 같이 내용을 수정합니다.



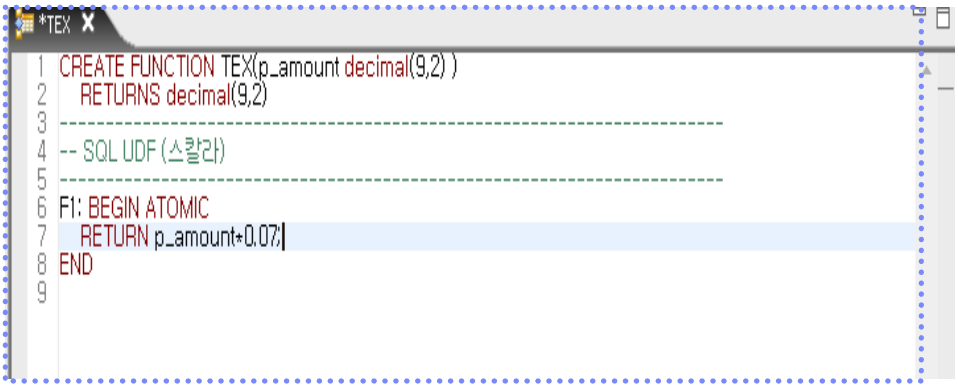


Figure 2403B IBM Data Studio Developer– UDF 새로 작성-2

TOPIC 03 03 UDF 생성

Point

IBM Data Studio Developer를 실행하여 User-Defined Function (UDF) 을 작성하는 방법입니다.

Tip

CREATE FUNCTION 다음에는 함수 이름이 반드시 기술되어야 하며, 괄호내부에 매개변수와 데이터 타입을 여러 개 지정할 수 있습니다.

2 UDF 의 필수구문 및 선택구문 구성

"CREATE FUNCTION"문장은 몇 가지 필수구문 및 선택 구문으로 구성됩니다.

예제 에서는 p_amount DECIMAL(9,2) 하나의 Parameter를 정의하였습니다. Parameter p_amount는 판매금액을 나타냅니다. Application에서 함수를 call하면, 함수는 판매금액에 대한 세금을 반환합니다. "RETURN" 구문은 DECIMAL(9,2) 반환을 기술합니다.

함수 본문은 "BEGIN ATOMIC ... END SQL" block 내부에 기술합니다.
함수는 다음 한 문장을 기술합니다.
IBM Data Studio Developer에서 "전개" 버튼을 눌러 실행합니다.

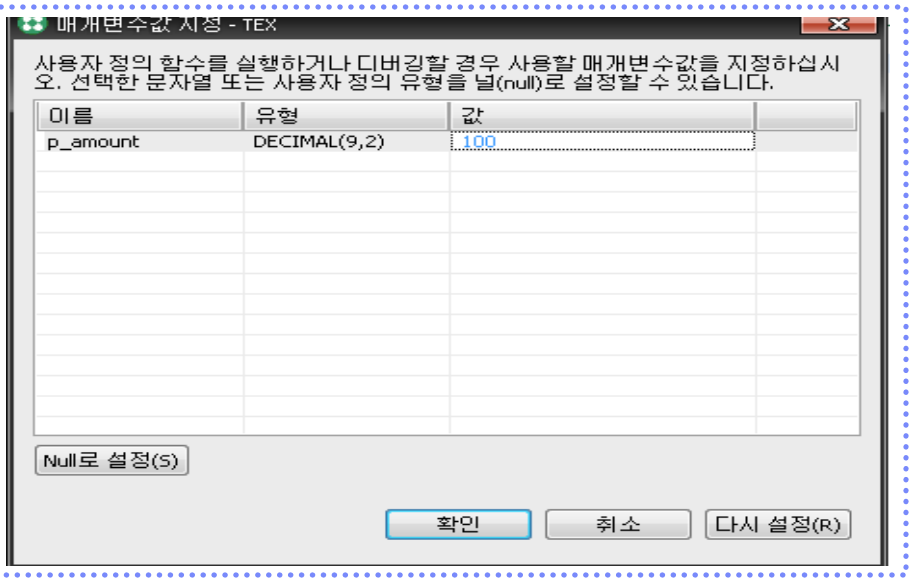


Figure 2403C IBM Data Studio Developer– 새로 작성-매개변수 지정

➲ 위와 같이. 입력변수 p_amount 에 대한 입력창이 팝업됩니다.
➲ 임의의 값을 입력하고 "확인" 버튼을 눌러 실행합니다.
➲ 함수가 성공적으로 수행되면, 다음과 같이 표시됩니다.

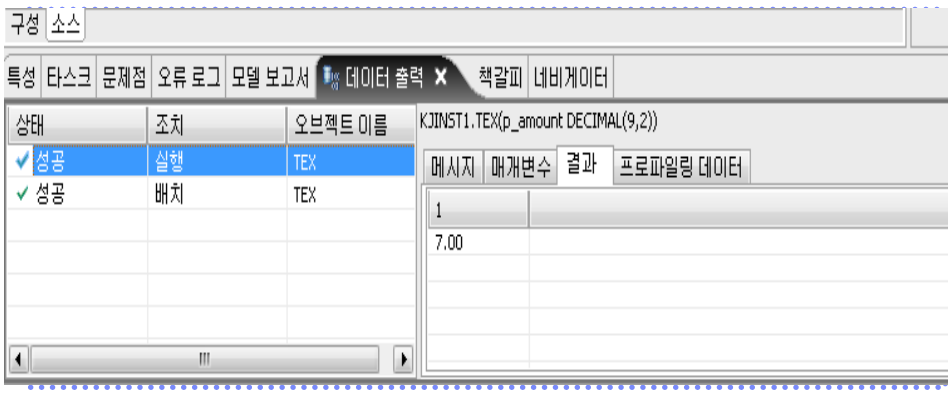


Figure 2403D IBM Data Studio Developer– UDF 새로 작성- 출력보기

TOPIC 03 03 UDF 생성

Point

IBM Data Studio Developer를 실행하여 User-Defined Function (UDF) 을 작성하는 방법입니다.

➲ 빠른 실행을 위해, 명령 창에서 다음과 같이 Query를 수행합니다.

```
SELECT   product_id, retail_price,
         KJINST1.TAX(retail_price) AS tax
   FROM KJINST1.product;
```

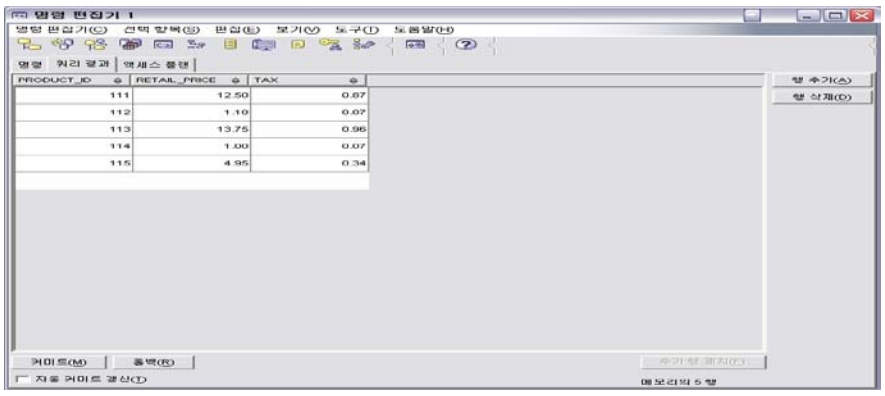


Figure 2403E IBM Data Studio Developer– UDF 새로 작성- 출력보기

Tip

TAX 함수의 Qualifier에 유의하세요.
함수 앞의 스키마를 제거하기 위해서는 "DB2 CURRENT PATH" 를 수정하세요. 이는 PATH와 같은 환경변수 역할을 합니다.

➲ DB2세션에서 함수에 대한 경로를 결정하는데 사용됩니다.
➲ 명령 창에서 다음 Query을 통해 "current path" 를 확인할 수 있습니다.

```
VALUES CURRENT PATH;
```

만일 "db2admin" 계정을 DB에 Connect하면, 다음과 같이 결과가 표시됩니다.

```
"SYSIBM","SYSFUN","SYSPROC","DB2ADMIN"
```

결과에서 보듯이, CURRENT PATH는 스키마 리스트 입니다. 뒤이어, 접속 사용자 스키마가 붙습니다.
➲ 함수이름 앞에 스키마가 정의되어 있지 않으면, CURRENT PATH에 따라 함수를 결정합니다. 스키마 순서에 따라, 실행할 함수를 검색하게 됩니다.

➲ CURRENT PATH를 변경하기 위해, "SET CURRENT PATH" 를 사용합니다.

```
SET CURRENT PATH = CURRENT PATH,"KJINST1"
VALUES CURRENT PATH

------------------------------------------------------
"SYSIBM","SYSFUN","SYSPROC","DB2ADMIN","KJINST1"

SET CURRENT PATH = SYSTEM PATH,"KJINST1"                    1
VALUES CURRENT PATH

------------------------------------------------------
"SYSIBM","SYSFUN","SYSPROC","KJINST1"                       2
```

TOPIC 03 03 UDF 생성

Point

IBM Data Studio Developer를 실행하여 User-Defined Function (UDF) 을 작성하는 방법입니다.

Tip

디폴트 SQL 경로(또는 사용자 스키마 앞에 SYSIBM이 있는 SQL 경로)를 사용하고 스키마에 새 SYSIBM 함수와 이름이 같은 기존 함수가 있는 경우, SYSIBM 함수가 대신 사용됩니다. 이 상황은 일반적으로 성능을 향상시키지만 예기치 않은 동작의 원인이 될 수 있습니다.

```
SET CURRENT PATH = USER,"KJINST1"
VALUES CURRENT PATH

------------------------------------------------------------
"DB2ADMIN","KJINST1"                                          3

SET CURRENT PATH =
"DB2ADMIN","SYSIBM","SYSFUN","SYSPROC","KJINST1"
VALUES CURRENT PATH

------------------------------------------------------------  4
"DB2ADMIN","SYSIBM","SYSFUN","SYSPROC","KJINST1"
```

➲ CURRENT PATH에 스키마 “KJINST1”를 추가합니다.
➲ CURRENT PATH를 SYSTEM PATH와 “KJINST1”로 변경합니다.
➲ 현재의 USER ID와 “KJINST1”를 포함하여 CURRENT PATH를 변경합니다.
➲ CURRENT PATH에 모든 SCHMA를 기술하여 적용합니다.

TOPIC 03 04 UDF 생성 - 예

Point

IBM Data Studio Developer 를 실행하여 UDF 를 작성하는 예제입니다.

1 복잡한 Query 실행하기

다음은 특정 기간 동안에 판매금액에 대한 이익률을 계산하는 UDF를 작성합니다.
해당 정보가 다음과 같이 세 개의 Table에 분산 되어 있습니다.

1) PRODUCT_PURCHASE : 각 제품별 판매 가격 정보
2) PRODUCT : 제품별 비용
3) SALES : 판매에 대한 일자/시간 관리 정보

특정 기간 동안의 이익률을 계산 하려면, 매번 세 개의 Table을 Join하여야 합니다.
이를 단순화 하기 위하여, PRODUCT ID, 시작일자 및 종료일자를 입력 파라미터를 갖는 UDF를 작성합니다.

➲ Table Layout

```
CREATE TABLE KJINST1.PRODUCT (
      PRODUCT_ID        INT         NOT NULL,
      DESCRIPTION       VARCHAR(40)  NOT NULL,
      COST              DECIMAL(7,2) NOT NULL,
      RETAIL_PRICE      DECIMAL(7,2) NOT NULL,
      INVENTORY         INT         NOT NULL,
      MINIMUM_INVENTORY  INT         NOT NULL
                        WITH DEFAULT 0,
      PRIMARY KEY (PRODUCT_ID) );

CREATE TABLE KJINST1.CUSTOMER (
     CUSTOMER_ID        INT         NOT NULL
       GENERATED ALWAYS AS IDENTITY
       (START WITH 0 INCREMENT BY 1),
     CREDIT_CARD         CHAR(16),
     EXPIRY_DATE         CHAR(4),
     LASTNAME            VARCHAR(28),
     FIRSTNAME           VARCHAR(28),
     ADDRESS             VARCHAR(300),
     ZIP_CODE            CHAR(6),
     PHONE               CHAR(10),
     PRIMARY KEY (CUSTOMER_ID) );

CREATE TABLE KJINST1.SALES (
     SALES_TRANSACTION_ID  INT
       GENERATED ALWAYS AS IDENTITY
       (START WITH 0 INCREMENT BY 1),
     CUSTOMER_ID           INT
       REFERENCES KJINST1.CUSTOMER(CUSTOMER_ID),
     SUB_TOTAL             DECIMAL(7,2),
     TAX                   DECIMAL(7,2),
     TYPE                  INT NOT NULL,
     TRANSACTION_TIMESTAMP TIMESTAMP,
     PRIMARY KEY (SALES_TRANSACTION_ID) );
```

TOPIC 03 04 UDF 생성 - 예

Point

IBM Data Studio Developer 를 실행하여 UDF 를 작성하는 예제입니다.

```
CREATE TABLE KJINST1.PRODUCT_PURCHASES (
    SALES_TRANSACTION_ID  INT
     REFERENCES KJINST1.SALES(SALES_TRANSACTION_ID)
          ON DELETE CASCADE,
    PRODUCT_ID          INT
     REFERENCES KJINST1.PRODUCT(PRODUCT_ID),
    PRICE               DECIMAL(7,2) NOT NULL,
    QTY                 INT NOT NULL );

CREATE TABLE KJINST1.AUDIT_STOCKJINST1HK (
    STAFF               VARCHAR(50),
    CHECKTIME           TIMESTAMP );
```

➲ PROD_PROFIT UDF 생성하기

```
CREATE FUNCTION KJINST1.PROD_PROFIT
  ( p_pid INTEGER, p_sdate DATE, p_edate DATE )                    1
  RETURNS DECIMAL(9,2)                                             2
----------------------                                             3
-- SQL UDF (Scalar)
----------------------
F1: BEGIN ATOMIC
  DECLARE v_retail_price DECIMAL(9,2);                             4
  DECLARE v_cost DECIMAL(9,2);                                     5
  DECLARE v_err VARCHAR(70);

  SET (v_retail_price, v_cost) =                                   6
    ( SELECT SUM(retail_price)
           , SUM(cost)
       FROM KJINST1.product p
         , KJINST1.product_purchases pp
         , KJINST1.sales s
      WHERE p.product_id = pp.product_id
       AND pp.sales_transaction_id
          = s.sales_transaction_id
       AND p.product_id = p_pid
       AND DATE(s.transaction_timestamp)
          BETWEEN p_sdate AND p_edate );                           7
                    SET v_err = 'Error: product ID '               8
                              || CHAR(p_pid)
                            || ' was not found.';

                    IF ( v_retail_price IS NULL                    9
                     OR v_cost        IS NULL )
                          THEN
                     SIGNAL SQLSTATE '80000'
                     SET MESSAGE_TEXT = v_err;
                          END IF;

                                                                   10
                          RETURN
                ( v_retail_price - v_cost ) / v_cost * 100;
                          END                                      11
```

TOPIC 03 04 UDF 생성 - 예

Point

IBM Data Studio Developer 를 실행하여 UDF 를 작성하는 예제입니다.

1) UDF KJINST1.PROD_PROFIT를 생성합니다.
2) PROD_PROFIT에 세개의 INPUT Parameter를 지정합니다.
   - p_pid : Product ID
   - p_sdate : 시작일자 조건
   - p_edate : 종료일자 조건
3) DECIMAL(9,2)형태의 이익률을 반환합니다.
4) UDF "KJINST1.TAX"는 단문으로 이루어진 함수 입니다. PROD_PROFIT처럼 복문으로 구성된 UDF는 BEGIN ATOMIC(4) 와 END(11)사이에 함수 Logic을 기술 합니다.
5) 함수 내에서 사용될 변수를 기술합니다.
   - v_retail_price : 판매 금액
   - v_cost : 판매 비용
   - v_err : PRODUCT ID가 없을 경우의 Error Message
6) PRODUCT, PRODUCT_PURCHASES와 SALES를 Join하여 판매금액과 비용을 산정한다. "SET"문장을 통해 값을 변수에 Assign한다.
7) SALES Table의 transaction_timestamp가 TIMESTAMP형태로 저장되어 있습니다. 이를 p_sdate, p_edate와 비교하기 위하여 DATE()함수를 사용하여 DATE 값으로 변환한다.
8) PRODUCT ID가 없을 경우 Error Message를 생성합니다.
9) 만일, PRODUCT ID가 없을 경우, UDF는 Error를 발생한다. 이후, Function 수행을 종료하고, 호출한 Application에 Error를 반환한다. v_err를 VARCHAR(70)으로 정의하였습니다. "SIGNAL SQLSTATE"의 error text의 한계가 70자 입니다. 만일, Message가 한계를 초과하면, 경고 없이 절삭됩니다.
10) 판매금액과 비용을 통해 이익률을 계산하여 반환합니다.

IBM Data Studio Developer를 통해 빌드 후, 실행합니다.

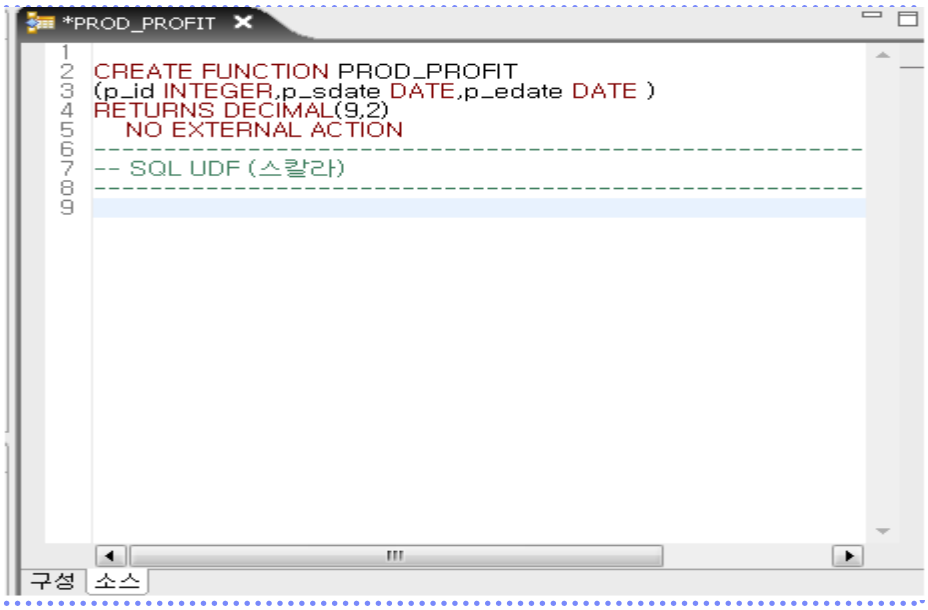


Figure 2404A••• IBM Data Studio Developer– UDF 실행하기 -1

TOPIC 03 04 UDF 생성 - 예

Point

IBM Data Studio Developer 를 실행하여 UDF 를 작성하는 예제입니다.

Tip

SP를 명령 창에서 실행하고자 할 때는 아래와 같은 방법으로 사용합니다.

>db2
"call KJINST1.PROD_PROFIT (111, 2009-07-01, 2009-07-31)

➲ PROD_PROFIT를 실행

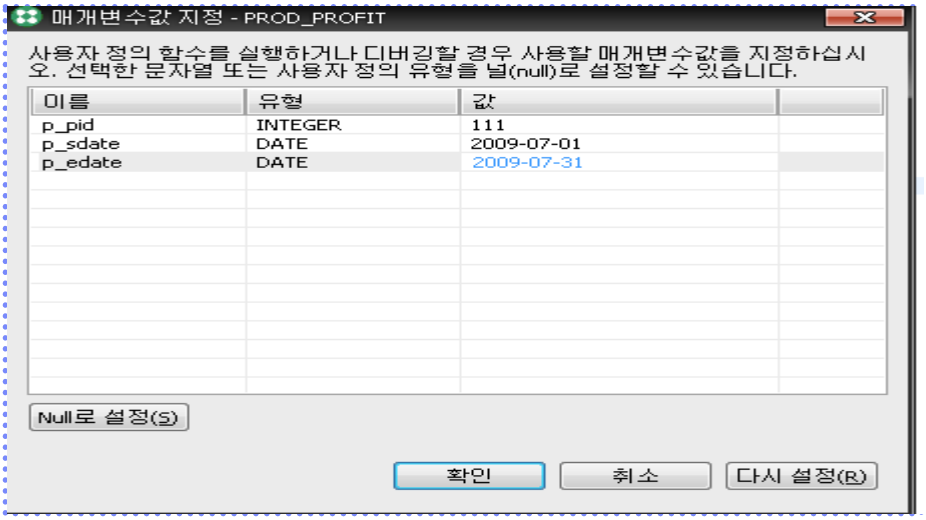


Figure 2404B ••• IBM Data Studio Developer– UDF 실행하기 -2

➲ 실행결과 화면 & 잘못된 Product ID를 입력했을 때의 화면

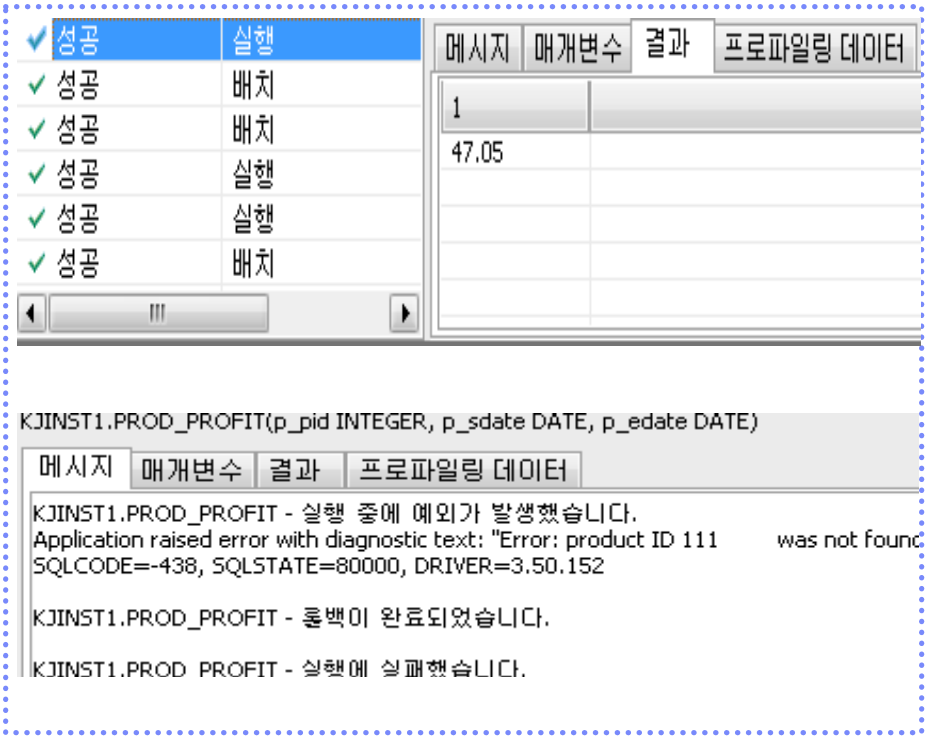


Figure 2404C ••• IBM Data Studio Developer– UDF 실행하기 -2

TOPIC 03 05 Table UDF 작성

Point

IBM Data Studio Developer 를 실행하여 Table UDF 를 작성하는 예제입니다.

1 Table UDF

Query의 FROM절에서 사용되며, Table형xo의 Row를 반환합니다. 다음은 입고가 필요한 모든 PRODUCT를 반환하는 Table 함수입니다. 누가, 언제 STOCK를 점검했는지 LOG을 위해 AUDIT_STOCKCHK Table을 사용합니다.

➲ STOCKCHK UDF 생성하기

```
CREATE FUNCTION KJINST1..STOCKCHK()                          1

   RETURNS TABLE ( PRODUCT_ID       INTEGER                  2
                 , DESCRIPTION      VARCHAR(40)
                 , INVENTORY        INTEGER
                 , MINIMUM_INVENTORY INTEGER )

   MODIFIES SQL DATA                                         3
------------------------------------------------
-- SQL UDF (TABLE)
------------------------------------------------
F1: BEGIN ATOMIC

   INSERT INTO KJINST1.AUDIT_STOCKCHK                        4
   VALUES (USER, CURRENT TIMESTAMP);

   RETURN
      SELECT PRODUCT_ID
           , DESCRIPTION
           , INVENTORY
           , MINIMUM_INVENTORY
        FROM KJINST1.PRODUCT
       WHERE INVENTORY < MINIMUM_INVENTORY;

END
```

1) UDF KJINST1.STOCCHK를 입력 Parameter없이 생성합니다.
2) RETURN절에 Table function이 반환할 Column과 Type을 정의합니다.
3) Table function은 기본적으로 실행 위주로 수행됩니다. “MODIFIES SQL DATA”를 기술하여, Function내에서 INSERT, UPDATE 및 DLETE를 할 수 있도록 합니다.
4) 누가 Call을 했는지를 관리하기 위해, AUDIT_STOCKCHK Table에 데이터를 입력합니다.
   - Special Register
   - USER : DB에 접속한 current user ID
   - CURRENT TIMESTAMP : 현재 시간

TOPIC 03 05 Table UDF 작성

Point

IBM Data Studio Developer 를 실행하여 Table UDF 를 작성하는 예제입니다.

```
1  CREATE FUNCTION KJINST1.STOCKCHK( )
2     RETURNS TABLE( product_id INTEGER
3                  , descriiption VARCHAR(40)
4                  , inventory INTEGER
5                  , minimum_inventory INTEGER)
6      MODIFIES SQL DATA
7     NO EXTERNAL ACTION
8  ------------------------------------------------
9  -- SQL UDF (스칼라)
10 ------------------------------------------------
11 F1: BEGIN ATOMIC
12    INSERT INTO KJINST1.AUDIT_STOCKCHK VALUES(USER, CU
13    RETURN
14        SELECT product_id, description,inventory,minimum_inve
15        FROM KJINTS1.PRODUCT
16        WHERE inventory < minimum_inventory;
17 END
18
```

Figure 2405A ••• IBM Data Studio Developer– Table UDF

➲ STOCKCHK UDF 수행

```
SELECT * FROM TABLE (KJINST1.STOCKCHK())  AS STOCKCHK;
```

STOCKCHK.sql

메시지 | 매개변수 | 결과 | 프로파일링 데이터

PRODUCT_ID	DESCRIIPT...	INVENTORY	MINIMUM_I...

Figure 2405B ••• IBM Data Studio Developer– Table UDF

TOPIC 03 06 Stored Procedure 작성

Point

IBM Data Studio Developer 를 실행하여 Stored Procedure를 생성하는 예입니다.

1 Stored Procedure 이해하기

SP는 DB내의 데이터를 Access하거나 수정하는 하나 이상의 SQL문장으로 구성된 DB Object입니다. SP는 DB2의 제어 하에 수행되고 관리 됩니다. SP는 SQL PL, C/C++, Java, COBOL, CLR 및 OLE를 사용하여 작성됩니다. SQL Procedure가 간단하기 때문에 주로 사용됩니다.

➲ SQL Stored Procedure의 장점

- ➢ Code의 재 사용을 통한 Business Logic 통합
- ➢ 보안 Level 강화
- ➢ 성능 개선

➲ SP는 DB에 저장되며, SQL 문장과 Business Logic을 encapsulate합니다. 적절한 권한을 가진 모든 Application Client에서 SP를 호출할 수 있습니다. 또한 Code의 재 사용률을 증대합니다. 또한, SP내의 Business Logic 변경이 관련된 모든 Application또는 Client와 독립적이므로 관리 비용을 절감할 수 있습니다.

➲ 사용자는 SP를 통해 Access하는 Table 또는 View에 대해 권한이 필요치 않습니다. 다만 SP를 호출할 수 있는 권한만 있으면 됩니다. 이를 통해, 사용자의 예상치 않은 접근에 대한 통제를 할 수 있습니다.

SP는 DBMS내에 SQL과 Business Logic을 가지고 수행합니다. 즉, Application과 DB 사이의 N/W Traffic을 줄일 수 있습니다. 또한 성능을 위해 SQL이 컴파일 됩니다.

2 Stored Procedure 개발하기

Tip

Table에 Column을 추가 하기 위해, “ALTER TABLE” 문장을 사용하거나 제어센터 (Control Center)를 사용할 수 있습니다.

다음은 SP를 통해 할 수 있는 몇 가지 예를 소개합니다.
이를 위해 DB내의 table을 변경합니다. 예제에서는, Internet On-Line 주문을위한 Web site를 SP를 소개합니다.
On-Line 주문을 위해, SALES 와 PRODUCT_PURCHASES table을 POS (Point-of-Sale) 시스템과 같이 변경합니다. 주문 상태를 관리하기 위해, Column을 추가 합니다.

SALES Table의 ORDER_STATUS column :
Column 에 Check Constraints (N, C, P & I)를 추가합니다.
- N : New Order
- C : 주문 완료
- P : 부분적 주문 완료
- I : 고객정보 부족

PRODCUT_PURCHASES Table의 STATUS column :
Column에 Check Constraints (N, C & O)를 추가합니다.
- N : New Order
- C : 주문 완료
- O : 재고 부족

TOPIC 03 06 Stored Procedure 작성

Point

IBM Data Studio Developer 를 실행하여 Stored Procedure를 생성하는 예입니다.

➲ Stored procedure (SP) 생성

다음은 새로운 계약에 관해 정보를 조회하는 SP를 IBM Data Studio Developer를 통해 생성하는 과정을 보여줍니다.

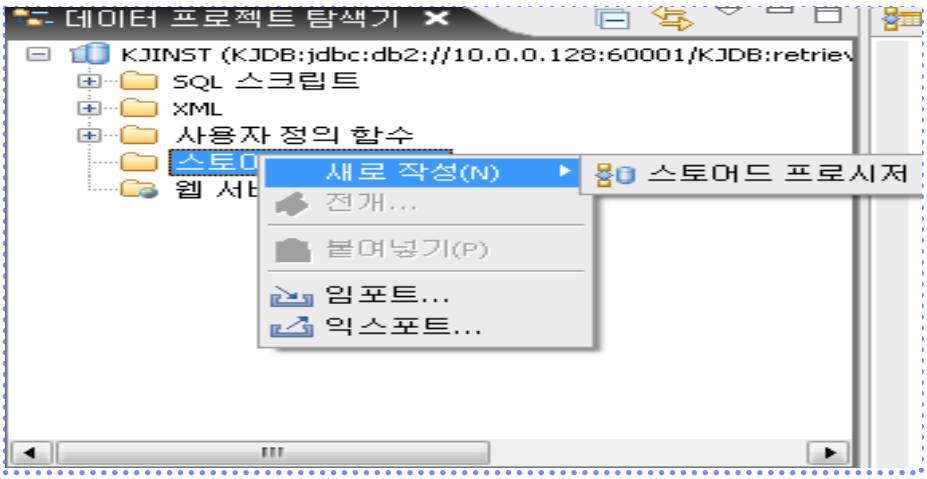


Figure 2406A IBM Data Studio Developer 에서 SP구현하기 -1

Procedure PROCEDD_NEWORDER를 Parameter없이 정의 합니다.
Procedure 본문은 "BEGIN" 과 "END" 블록 사이에 기술합니다. 다음의 Query 결과를 처리하기 위해 FOR loop를 정의합니다. FOR Loop내에, 새로운 주문에 대한 처리를 위해 Business Logic을 기술합니다.

```
SELECT SALES_TRANSACTION_ID, CUSTOMER_ID
  FROM SALES
 WHERE ORDER_STATUS = 'N'
```

➲ Stored Procedure PROCESS_NEW 생성

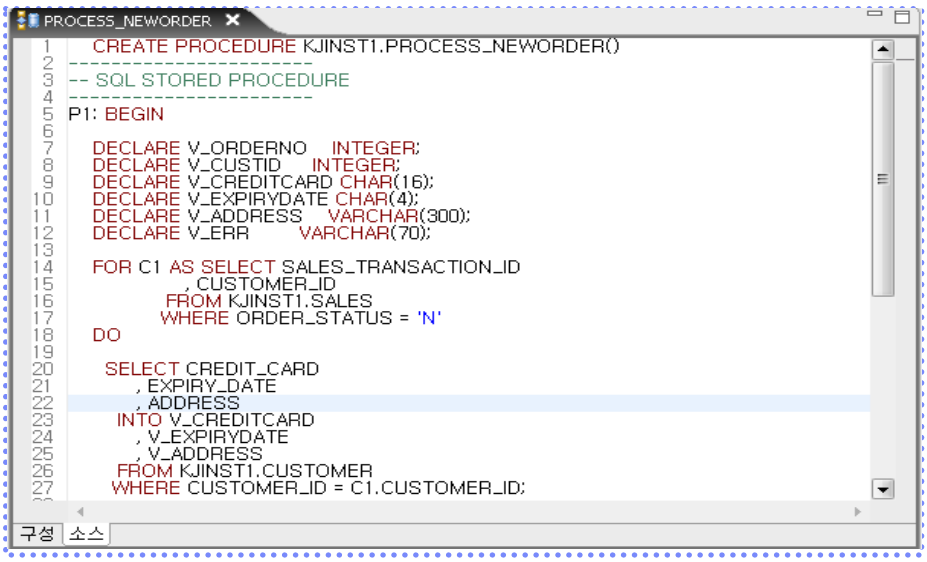


Figure 2406B IBM Data Studio Developer 에서 SP구현하기 -2

IBM Data Studio Developer 에서 PROCESS_NEW를 Build합니다.

TOPIC 03 06 Stored Procedure 작성

Point

IBM Data Studio Developer 를 실행하여 Stored Procedure를 생성하는 예입니다.

이제까지, 단순 SP를 생성하는 과정을 살펴 보았습니다.
예제에서는 단순한 SP를 작성하였습니다. 다음은 stored procedure의 강력한 기능을 통해 새로운 SP를 전개합니다.
PROCESS_NEWORDER에서 Web application에 입력한 새로운 주문을 처리합니다.
이제까지는 새로운 주문을 검색하는 Logic을 작성하였습니다. FOR loop내의 각 주문에 다음과 같은 처리를 합니다.

- 제품을 주문한 고객 정보를 점검합니다.
- 만일 고객의 주소나 신용카드 정도가 불충분하면 처리하지 않습니다.
- 검증된 주문은 처리를 계속합니다.

➲ 수정된 KJINST1.PROCESS_NEWORDER

```
CREATE PROCEDURE KJINST1.PROCESS_NEWORDER()
-----------------------
-- SQL STORED PROCEDURE
-----------------------
P1: BEGIN

    DECLARE V_ORDERNO    INTEGER;
    DECLARE V_CUSTID     INTEGER;
    DECLARE V_CREDITCARD CHAR(16);
    DECLARE V_EXPIRYDATE CHAR(4);
    DECLARE V_ADDRESS    VARCHAR(300);
    DECLARE V_ERR        VARCHAR(70);

    FOR C1 AS SELECT SALES_TRANSACTION_ID                          (1)
                   , CUSTOMER_ID
                FROM KJINST1.SALES
               WHERE ORDER_STATUS = 'N'
    DO

      SELECT CREDIT_CARD                                           (2)
           , EXPIRY_DATE
           , ADDRESS
        INTO V_CREDITCARD
           , V_EXPIRYDATE
           , V_ADDRESS
        FROM KJINST1.CUSTOMER
       WHERE CUSTOMER_ID = C1.CUSTOMER_ID;

      IF    (V_CREDITCARD IS NULL AND V_EXPIRYDATE IS NULL)        (3)
         OR (V_ADDRESS IS NULL)
      THEN

          SET V_ERR = 'THE CUSTOMER INFORMATION IS'
                    || ' NOT COMPLETE TO PROCESS THE ORDER NO '
                    || CHAR(V_ORDERNO);

          SIGNAL SQLSTATE'80000' SET MESSAGE_TEXT = V_ERR;         (4)

      END IF;

      --CALL ANOTHER SP TO PROCESS THE VALID ORDERS                (5)
      SET V_ORDERNO = C1.SALES_TRANSACTION_ID;
      CALL KJINST1.FILLORDER(V_ORDERNO, V_CUSTID);

    END FOR;

END P1
```

TOPIC 03 06 Stored Procedure 작성

Point

IBM Data Studio Developer 를 실행하여 Stored Procedure를 생성하는 예입니다.

1) Query 결과를 처리하기 위한 FOR Loop 정의
2) 고객 정보에 해당하는 카드 정보 및 주소 확인 하는 Process
3) 고객정보의 정합성 점검
4) 필요한 정보가 부적하면 Error 처리 : SQLSTATE 80000
5) 주문 번호 및 고객정보를 입력변수로 SP FILLORDER를 CALL힙니다.

➲ 수정된 KJINST1.PROCESS_NEWORDER

```
CREATE PROCEDURE KJINST1.FILLORDER
            ( IN V_SALESTXNID INTEGER
            , IN V_CUSTID INTEGER )

-----------------------
-- SQL STORED PROCEDURE
-----------------------

P1: BEGIN

   -- DECLARE VARIABLES
   DECLARE V_PRODID         INTEGER;
   DECLARE V_QTY            INTEGER;
   DECLARE V_INVENTORY      INTEGER;
   DECLARE V_PRICE          DECIMAL(5,2);
   DECLARE V_TOTALCOUNT     INTEGER DEFAULT 0;
   DECLARE V_COUNT          INTEGER DEFAULT 0;
   DECLARE V_LASTPRODUCT    INTEGER DEFAULT 0;
   DECLARE V_ERR            VARCHAR(70);

   -- DECLARE CURSORS
   DECLARE C1 CURSOR WITH HOLD FOR
      SELECT PP.PRODUCT_ID
          , RETAIL_PRICE, QTY
        FROM KJINST1.PRODUCT_PURCHASES PP
          , KJINST1.PRODUCT P
       WHERE PP.PRODUCT_ID      =P.PRODUCT_ID
         AND SALES_TRANSACTION_ID=V_SALESTXNID;

   -- DECLARE EXCEPTION HANDLER
   DECLARE CONTINUE HANDLER FOR NOT FOUND
      SET V_LASTPRODUCT = 1;
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
      RESIGNAL;
```

Stored Procedure 작성

Point

IBM Data Studio Developer 를 실행하여 Stored Procedure를 생성하는 예입니다.

```
IF V_SALESTXNID IS NULL THEN

   --CREATION A NEW SALES TXN
   INSERT INTO KJINST1.SALES
            ( SALES_TRANSACTION_ID
            , CUSTOMER_ID
            , SUB_TOTAL
            , TYPE
            , TRANSACTION_TIMESTAMP)
        VALUES ( DEFAULT
            , V_CUSTID
            , 0
            , 5
            , CURRENT TIMESTAMP);

   VALUES IDENTITY_VAL_LOCAL()
    INTO V_SALESTXNID;

END IF;

OPEN C1;
FETCH C1 INTO V_PRODID, V_PRICE, V_QTY;

WHILE (V_LASTPRODUCT = 0) DO

   BEGIN

      DECLARE C_NO_STOCK CONDITION
        FOR SQLSTATE '80000';

          DECLARE EXIT HANDLER FOR C_NO_STOCK
                                  BEGIN
                UPDATE KJINST1.PRODUCT_PURCHASES
                              SET STATUS = 'O'
                    WHERE SALES_TRANSACTION_ID
                              = V_SALESTXNID
                            AND PRODUCT_ID
                              = V_PRODID;
                                END;
SET V_TOTALCOUNT = V_TOTALCOUNT + 1;
       SELECT INVENTORY
         INTO V_INVENTORY
         FROM PRODUCT
        WHERE PRODUCT_ID = V_PRODID;
```

TOPIC 03 06 Stored Procedure 작성

Point

IBM Data Studio Developer 를 실행하여 Stored Procedure를 생성하는 예입니다.

```
        --CHECK IF INVENTORY SATISFY THE QTY
        IF V_INVENTORY >= V_QTY THEN
          BEGIN ATOMIC
            INSERT INTO KJINST1.PRODUCT_PURCHASES
                  (SALES_TRANSACTION_ID
                  , PRODUCT_ID
                  , PRICE
                  , QTY
                  , STATUS)
              VALUES (V_SALESTXNID
                  , V_PRODID
                  , V_PRICE
                  , V_QTY
                  , 'C');

              UPDATE KJINST1.PRODUCT
                SET INVENTORY=INVENTORY-V_QTY
                WHERE PRODUCT_ID = V_PRODID;

              SET V_COUNT = V_COUNT + 1;
            END;
      ELSE
          --NOT ENOUGH STOCK TO FILL ORDER
          SET V_ERR = 'THERE IS NOT ENOUGH '
                || 'STOCK FOR PRODUCT ID  '
                || CHAR(V_PRODID)
                ||' TO FILL THE ORDER '
                || CHAR(V_SALESTXNID);
          SIGNAL SQLSTATE '80000'
            SET MESSAGE_TEXT = V_ERR;
        END IF;

      END;

      FETCH C1 INTO V_PRODID, V_PRICE, V_QTY;

    END WHILE;

    IF V_COUNT = V_TOTALCOUNT THEN
      UPDATE KJINST1.SALES
        SET ORDER_STATUS = 'C'
       WHERE SALES_TRANSACTION_ID
          = V_SALESTXNID;
    ELSE
       UPDATE KJINST1.SALES
         SET ORDER_STATUS = 'P'
        WHERE SALES_TRANSACTION_ID
           = V_SALESTXNID;
    END IF;

    UPDATE KJINST1.SALES
      SET SUB_TOTAL
        = (SELECT SUM(PRICE)
           FROM PRODUCT_PURCHASES
           WHERE SALES_TRANSACTION_ID
             = V_SALESTXNID)
     WHERE SALES_TRANSACTION_ID = V_SALESTXNID;

  END P1
```

TOPIC 03 07 Trigger 작성

Point

Trigger 에 대해 알아봅니다.

1 Trigger 이해하기

➲ Trigger는 Table에 INSERT/UPDATE/DELETE 처리 전□ 후에 필요한 일련의 작업을 자동으로 수행하기 위한 Table과 관련 있는 Database Object입니다.
➲ Trigger를 발생 시키는 문장을 "Triggering SQL" 문장 이라 합니다. Trigger 를 Triggering SQL 문장 전 또는 후에 실행 하게 할 것인지를 선택할 수 있습니다.

➲ 세 가지 Trigger Type

Trigger Type	설명
BEFORE	Table의 Data가 Triggering SQL문장에 의해 변경되기 전에 수행됩니다.
AFTER	Triggering SQL문장이 성공적으로 완료되면 Trigger가 수행됩니다. Trigger에 따라, AFTER Trigger는 다른 Trigger를 수행할 수 있습니다. DB2는 최대 16 Level까지 다른 Trigger를 연쇄적으로 수행할 수 있습니다.
INSTRADE OF	Trigger가 View를 대상으로 정의됩니다. SQL 문장이 복잡한 View에 대해 INSERT/UPDATE/DELETE를 할 때 유용합니다. View에 대해 허용되지 않는 INSERT/UPDATE/DELETE에 대해 사용됩니다. View의 Column은 해당 Table에 Column과 자동으로 Mapping되지 않으므로 INSERT/UPDATE/DELETE할 수 없다. Business Logic과 해당 Table과 View의 column에 대한 Mapping정보를 알고 있다면, SQL의 제약 사항을 우회적으로 INSTEAD Trigger 본문에 해당 Business Logic에 정의할 수 있습니다. INSTEAD OF Trigger에 SQL문장을 정의하여 Application Interface를 단순화 할 수 있습니다.

Trigger는 Application 전반에 걸쳐 수행되어야 될 Business Rule을 항상 수행하게 될 때 유용하게 사용됩니다. 특정 Table의 Data가 다른 Table의 Data와 관련 있는 Business Rule이 있을 수 있다. 만일, Business Rule이 변경되면, DB에 있는 Trigger 정의만 변경하면 되며, 모든 Application은 추가적인 변경 없이 새로운 Business Rule을 따르게 됩니다.

TOPIC 03 07 Trigger 작성

Point

Trigger 에 대해 알아봅니다.

2 CLP/제어센터를 통한 Trigger 생성

Trigger를 생성하기 위한 많은 Tool들이 있습니다. “CREATE TRIGGER” 명령도 그 중 하나 입니다.

➲ CLP에서 수행

```
CREATE TRIGGER KJINST1.UPD_PRODINV_TRIG
   AFTER INSERT ON KJINST1.PRODUCT_PURCHASES
   REFERENCING NEW AS NEWROW
   FOR EACH ROW  MODE DB2SQL
BEGIN ATOMIC
   IF ( NEWROW.QTY > 0 ) THEN
      UPDATE KJINST1.PRODUCT
        SET INVENTORY  = INVENTORY - NEWROW.QTY
       WHERE PRODUCT_ID = NEWROW.PRODUCT_ID;
   ELSEIF ( NEWROW.QTY < 0 ) THEN
      UPDATE KJINST1.PRODUCT
        SET INVENTORY  = INVENTORY + NEWROW.QTY
       WHERE PRODUCT_ID = NEWROW.PRODUCT_ID;
   END IF;
END@

db2 –td@ -vf <fie_name>
```

PRODUCT_PURCHASES Table에 대해 INSERT가 수행되면,
UPD_PRODINV_TRIG Trigger가 수행됩니다. 수량이 0보다 크면(판매) 재고가 감소됩니다. 수량이 0보다 작으면(반품) 재고가 증가됩니다.
제어센터를 통해 Trigger를 생성하는 방법을 소개합니다. 제어센터에서 Database를 선택한 후, PRODUCT_PURCHASES Table에서 마우스 오른쪽을 클릭합니다.

➲ 제어센터를 통한 Trigger 생성

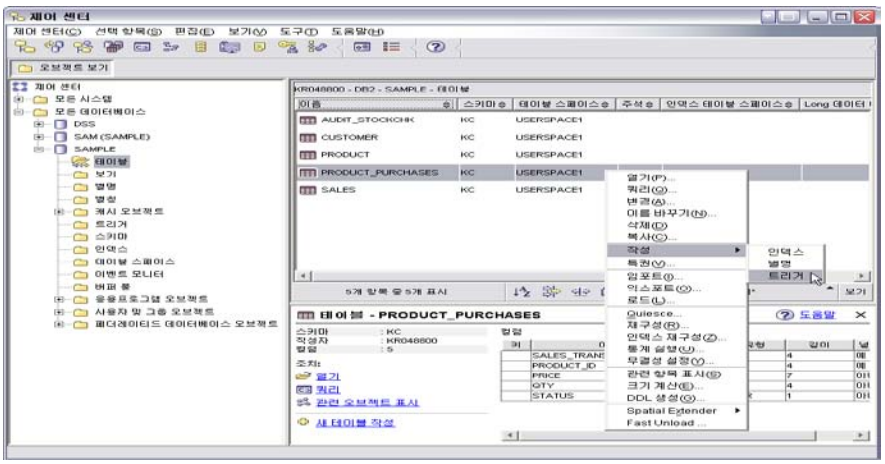


Figure 2407A 제어센터에서 Trigger 작성하기 -1

TOPIC 03 07 Trigger 작성

Point

Trigger 에 대해 알아봅니다.

"트리거 작성" 다이얼로그에서 Trigger 정의를 기술합니다.
화면이 "트리거" 와 "트리거 조치" 탭으로 나뉘어 집니다. "주석" 을 제외한 모든 항목을 정의합니다.

- 트리거 스키마 : KC
- 트리거 이름 : UPT_PRODINV_TRIG
- 테이블 또는 뷰 스키마 : KC
- 테이블 또는 뷰 이름 : PRODUCT_PURCHASE
- 트리거 조치 시간 : AFTER
- 트리거가 실행되도록 하는 조작 : 삽입

➲ 트리거 작성 : 트리거 탭

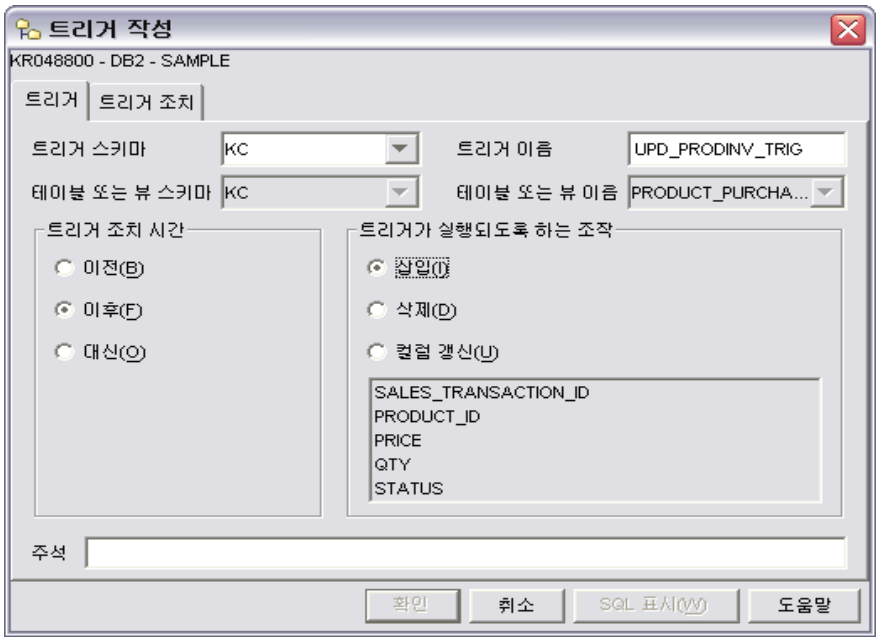


Figure 2407B ••• 제어센터에서 트리거 작성하기 -2

INSERT/UPDATE/DELETE된 이전 행 과 새 행에 대한 참조 명을 정의 할 수 있습니다. 이 전이(Transition) 변수는 Trigger 본문에서 사용됩니다. "OLD Transition 변수" 는 UPDATE/DELETE가 수행되면 생성됩니다.
한편, "NEW Transition 변수" 는 UPDATE /INSERT가 수행되면 생성됩니다.

Triggering SQL	OLD Transition	NEW Transition
DELETE	OLD	
INSERT		NEW
UPDATE	OLD	NEW

TOPIC 03 07 Trigger 작성

Point

Trigger 에 대해 알아봅니다.

➲ 트리거 작성 : 트리거 조치 탭

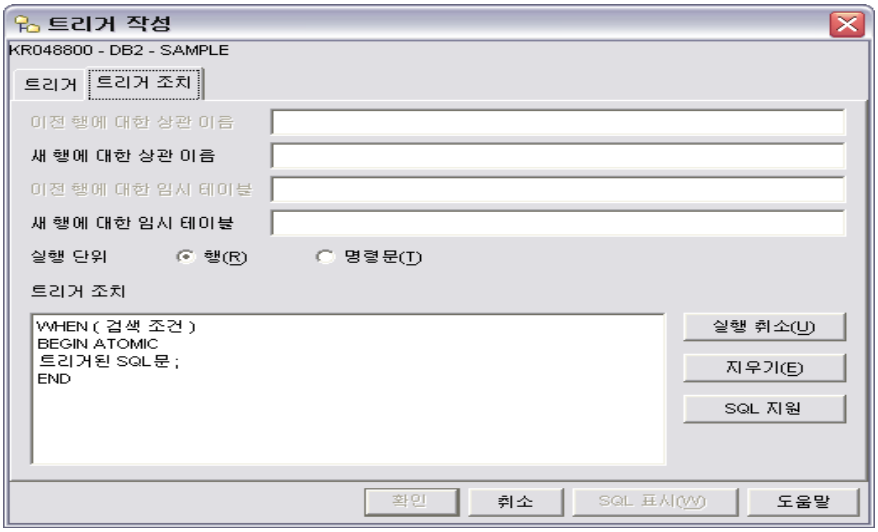


Figure 2407C 제어센터에서 트리거 작성하기 -3

INSERT Trigger를 작성하므로,
새 행에 관련된 참조만 활성화 됩니다. "새 행에 대한 상관 이름"에 "NEWROW"를 입력합니다. 다음은 Triggering SQL문장에 의해 영향을 받는 각 Row 단위로 Trigger를 실행 할 것인지, 또는 Row 수에 관계 없이 각 문장 별로 실행 할 것인지를 선택합니다.
UPT_PRODINV_TRIG Trigger에서는 PRODUCT_PURCHASE에 입력되는 각 Row 단위로 Trigger를 수행하게 합니다. 그러므로 실행단위로서 "행'을 선택합니다.
마지막으로, Trigger가 실행될 때의 SQL 문장을 "트리거 조치"에 기술합니다. 트리거 조치에는 기본적인 템플릿이 제공됩니다. "WHEN"절에는 Trigger 수행을 위한 조건을 정의합니다. 예를 들어, Trigger의 Base Table에 조건을 만족하는 경우에만 Trigger를 수행하도록 정의할 수 있습니다.

➲ WHEN 절

Ex) "가격 x 수량"이 100보다 큰 경우만 수행
WHEN (newrow.price * newrow.qty > 100)

WHEN절 다음에는 "BEGIN ATOMIC" ~ "END"에 해당하는 SQL문장이 정의됩니다.
SQL 문장 중, 하나라도 실패하면 Trigger 조치가 실패 하도록 반드시 "ATOMIC"으로 기술하여야 합니다.
Trigger 문장을 기술할 때는 몇 가지 규칙이 있습니다. 그 중 하나가, BEFORE Trigger에서는 INSERT/UPDATE/DELETE를 기술할 수 없습니다.
만일, Data 수정을 원한다면, AFTER Trigger를 정의하세요. 자세한 내용은 "DB2 SQL Reference Guide"를 참조 하세요!.
계속해서, Trigger 조치를 작성합니다. PRODUCT_PURCHASES Table에 대해 INSERT가 수행되면, UPD_PRODINV_TRIG Trigger가 수행됩니다. 수량이 0보다 크면(판매) 재고가 감소됩니다. 수량이 0보다 작으면(반품) 재고가 증가됩니다.
각 Trigger문장 뒤에는 ";"으로 끝을 맺습니다.

Tip

UPT_PRODINV_TRIG Trigger에서는 PRODUCT_PURCHASE Table에 입력되는 모든 Operation 에 대해 Trigger를 수행하므로, WHEN절을 정의하지 않습니다.

TOPIC 03 07 Trigger 작성

Point

Trigger 에 대해 알아봅니다.

Figure 2407D ••• 제어센터에서 트리거 작성하기 -4

Tool에 의해 생성된 "CREATE TRIGGER" 에 대한 문장을 보려면 "SQL 표시" 를 Click합니다.

"트리거 작성" 창에서 바로 생성하기 위하여 "확인" 을 Click합니다.
제어센터에서 트리거 Object를 선택하여 생성된 Trigger를 확인합니다.

Tip

해당 창에서 SQL문장을 복사 하거나, 다른 파일로 저장 할 수 있습니다.

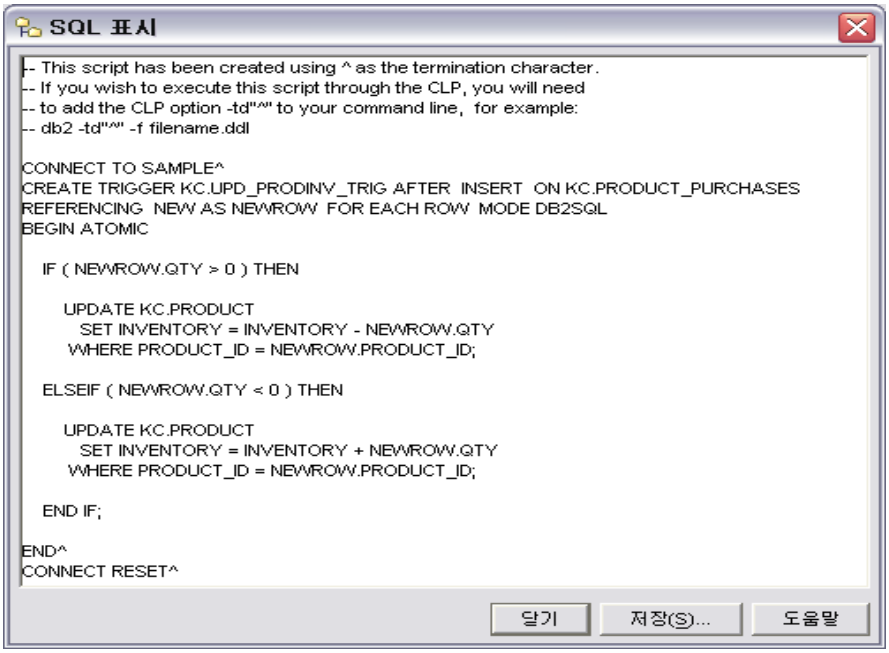


Figure 2407E ••• 제어센터에서 트리거 작성하기 -5

TOPIC 03 07 Trigger 작성

Point

Trigger 에 대해 알아봅니다.

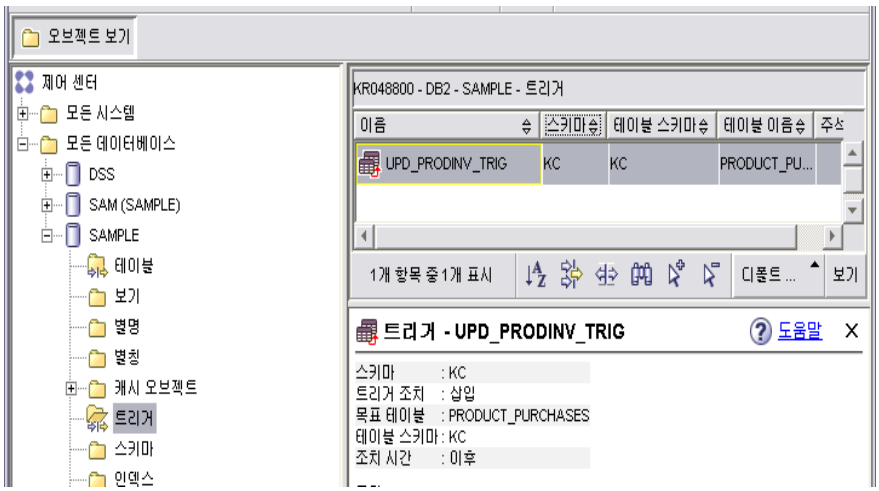


Figure 2407F 제어센터에서 트리거 작성하기 -6

3 Create Trigger 문

```
>>-CREATE TRIGGER--trigger-name--+-NO CASCADE BEFORE-+---------->
                                 +-AFTER-------------+
                                 '-INSTEAD OF--------'
>--+-INSERT-------------------------+--ON--+-table-name-+------->
   +-DELETE-------------------------+      '-view-name--'
   '-UPDATE--+--------------------+-'
             |    .-,-----------. |
             |    V             | |
             '-OF---column-name-+-'
>--+-------------------------------------------------------------+-->
   |              .-------------------------------------------. |
   |              V                      .-AS-.                 | |
   '-REFERENCING -----------------+-OLD--+----+--correlation-name-+-+-'
                                  |      .-AS-.                  |
                                  +-NEW--+----+--correlation-name-+
                                  |            .-AS-.            |
                                  +-OLD_TABLE--+----+--identifier-+
                                  |            .-AS-.            |
                                  '-NEW_TABLE--+----+--identifier-'
>--+-FOR EACH ROW--------------+--MODE DB2SQL------------------->
   '-FOR EACH STATEMENT--------'
>--| triggered-action |------------------------------------><

triggered-action:
|--+-----------------------------------+---------------------->
   '--------WHEN--(--search-condition--)-'
>--+---------+----SQL-procedure-statement----------------------|
   '--label--'
```

TOPIC 03 07 Trigger 작성

Point

Trigger 에 대해 알아봅니다.

4 Trigger 사용 예제

➲ BEFORE INSERT TRIGGER

```
CREATE TRIGGER KJINST1.default_class_end
   NO CASCADE
   BEFORE INSERT ON KJINST1.cl_sched
   REFERENCING NEW AS n
   FOR EACH ROW
   MODE DB2SQL
WHEN (n.ending IS NULL)
SET n.ending = n.starting + 1 HOUR
```

➲ AFTER UPDATE TRIGGER

```
CREATE TRIGGER KJINST1.audit_emp_sal
   AFTER UPDATE OF salary ON KJINST1.employee
   REFERENCING OLD AS o
               NEW AS n
   FOR EACH ROW
   MODE DB2SQL
INSERT INTO KJINST1.audit
VALUES (CURRENT TIMESTAMP
     ,  ' Employee '
        | | o.empno
        | | ' salary changed from '
        | | CHAR(o.salary)
        | | ' to '
        | | CHAR(n.salary)
        | | ' by '
        | | USER)
```

➲ SQL PL을 이용한 BEFORE INSERT TRIGGER

```
CREATE TRIGGER KJINST1.validate_sched
   NO CASCADE BEFORE INSERT ON KJINST1.cl_sched
   REFERENCING NEW AS n
   FOR EACH ROW
   MODE DB2SQL
vs: BEGIN ATOMIC

  -- supply default value for ending time if null
  IF (n.ending IS NULL) THEN
     SET n.ending = n.starting + 1 HOUR;
  END IF;

  -- ensure that class does not end beyond 9PM
  IF (n.ending > '21:00') THEN

     SIGNAL SQLSTATE '80000'
       SET MESSAGE_TEXT='class ending time is'
                  | | ' beyond 9pm';

  ELSEIF (n.DAY=1 or n.DAY=7) THEN

     SIGNAL SQLSTATE '80001'
       SET MESSAGE_TEXT='class cannot be'
                  | | 'scheduled on a weekend';

  END IF;
END vs
```

TOPIC 03 07 Trigger 작성

Point

Trigger 에 대해 알아봅니다.

➲ INSTEAD OF TRIGGER

```
CREATE VIEW KJINST1.org_by_division
 (division, number_of_dept)
AS
 SELECT division, count(*)
 FROM KJINST1.org
 GROUP BY division


CREATE TRIGGER KJINST1.upd_org
   INSTEAD OF UPDATE
   ON KJINST1.org_by_division
   REFERENCING OLD AS o
               NEW AS n
   FOR EACH ROW
   MODE DB2SQL
BEGIN ATOMIC

 IF (o.number_of_dept != n.number_of_dept) THEN

   SIGNAL SQLSTATE '80001'
   SET MESSAGE_TEXT
   =‘The number of department is not updatable.';

 END IF;

 UPDATE KJINST1.org
   SET division = n.division
 WHERE division = o.division;

END
```

View에 대한 Update후의 처리 결과를 확인할 수 있습니다.

```
select * from KJINST1.org_by_division;

DIVISION   NUMBER_OF_DEPT
---------- --------------
Corporate           1
Eastern             5
Midwest             2
Western             2

UPDATE KJINST1.org_by_division
  SET division='Eastern_1'
 WHERE division='Eastern‘;

select * from KJINST1.org_by_division;

DIVISION   NUMBER_OF_DEPT
---------- --------------
Corporate           1
Eastern_1           5
Midwest             2
Western             2
```

TOPIC 03 07 Trigger 작성

Point

Trigger 에 대해 알아봅니다.

➲ SP를 Call하는 Trigger

```
CREATE TRIGGER KJINST1.tr_autoproc_order
    AFTER INSERT ON KJINST1.sales
    FOR EACH ROW MODE DB2SQL
    BEGIN ATOMIC

      CALL KJINST1.process_neworder();

    END
```

5 View Trigger 와 Table Trigger 비교

다음 예를 통해 View Trigger와 Table Trigger의 차이점을 살펴 봅니다.
이를 위해 Table 과 View, Table Insert Trigger와 View Insert Trigger를 사용합니다.

➲ Table, View 및 Trigger 생성

```
CREATE TABLE KJINST1.T_AIRPORT (
  AIRPORT_CODE CHAR( 3) NOT NULL,
  AIRPORT_NAME CHAR(50)
);

CREATE VIEW KJINST1.V_AIRPORT
AS
  SELECT *
    FROM KJINST1.T_AIRPORT
;

CREATE TRIGGER KJINST1.INSERT_T_AIRPORT
    AFTER INSERT
    ON KJINST1.T_AIRPORT
    FOR EACH ROW
    MODE DB2SQL
BEGIN ATOMIC
END;

CREATE TRIGGER KJINST1.INSERT_V_AIRPORT
    INSTEAD OF INSERT
    ON KJINST1.V_AIRPORT
    FOR EACH ROW
    MODE DB2SQL
BEGIN ATOMIC
END;
```

TOPIC 03 07 Trigger 작성

Point

Trigger 에 대해 알아봅니다.

➲ Table, View 및 Trigger 생성

```
INSERT INTO KJINST1.t_airport
      VALUES ('KOR', 'SEOUL_T');
INSERT INTO KJINST1.v_airport
      VALUES ('KOR', 'SEOUL_V');

select * from KJINST1.t_airport;

AIRPORT_CODE AIRPORT_NAME
------------ -------------------------
KOR          SEOUL_T

select * from KJINST1.v_airport;

AIRPORT_CODE AIRPORT_NAME
------------ -------------------------
KOR          SEOUL_T
```

➲ INSTEAD OF Trigger 수정

```
CREATE TRIGGER KJINST1.insert_v_airport
   INSTEAD OF INSERT
   ON KJINST1.v_airport
   REFERENCING NEW AS n
   FOR EACH ROW
   MODE DB2SQL
BEGIN ATOMIC
   INSERT INTO KJINST1.t_airport
   VALUES (n.airport_code, n.airport_name);
END;
```

TOPIC 03 08 모듈 작성

Point

DB2 모듈은 저장 프로시저, 함수, 변수, 커서드의 오브젝트를 하나의 묶음으로 관리합니다. 오라클의 패키지(Package)와 유사한 기능을 수행합니다.

1 모듈(Module)

➲ DB2 9.7에서 새롭게 제공하는 모듈을 통해서 일련의 DBMS의 오브젝트를 그룹으로 관리합니다.
➲ 오브젝트 리스트: 저장프로시저, 사용자 함수, 배열,레코드, 사용자 정의 커서등 서브 모듈로 지정할 수 있습니다.

2 모듈 작성 실습 : 모듈에 프로시저 포함하기

Tip

모듈에 등록된 프로시저나 함수는 독립적으로도 실행이 가능합니다. 하지만 모듈을 사용하면 모듈간의 의존성 관리가 쉽습니다.

1. 간단 테스트 테이블 하나 작성

```
create table  TEST_TEST (name varchar(20) , entrydate
timestamp)
```

4. 모듈 선언

```
CREATE MODULE mod_test1
```

5. 선언한 모듈에 프로시져 추가 (고정변수)

```
ALTER MODULE mod_test1
  PUBLISH PROCEDURE proc1_test2  (name  varchar(20))
        BEGIN
         call  proc1_test  ('kasung');
        END
```

6. 모듈 실행

```
call mod_test1. proc1_test3 ('HONG')
```

7. 데이타 이상유무 확인

```
select * from test_test

NAME         ENTRYDATE
------------  --------------------
kasung       2009. 4. 3 오전 3:37:36
```

8. 스키마.모듈명.프로시져명으로 도 실행 예시

```
call  db2inst1.mod_test1. proc1_test3 ('KOREA')
```

TOPIC 03 08 모듈 작성

Point

DB2 모듈은 저장 프로시저, 함수, 변수, 커서등의 DB2 오브젝트를 하나의 묶음으로 관리합니다. 오라클의 패키지(Package)와 유사한 기능을 수행합니다.

3 모듈 작성 실습: 모듈에 함수 포함하기

```
1.        기존의 모듈에 appending 하기

          ALTER MODULE mod_test1
            PUBLISH function  fn_addsum123  (p1 int)
                              returns int
                    BEGIN
                          return ( select fn_addsum(p1) from
          sysibm.sysdummy1 );
                    END

2.모듈의 함수 실행

select  mod_test1.fn_addsum123 (10)  from sysibm.sysdummy1

        1
          ----
          1000
```

4 모듈 작성 실습: 모듈에 사용자 정의 타입 포함하기

```
1. CREATE MODULE INVENTORY

2. ALTER MODULE INVENTORY ADD
  TYPE ITEMLIST AS INTEGER ARRAY[VARCHAR(100)]

3. ALTER MODULE INVENTORY ADD VARIABLE ITEMS ITEMLIST
```

TOPIC 03 08 모듈 작성

Point

사용자 정의 모듈과 서브 모듈 정보를 조회해 봅니다.

➲ 모듈 정보 조회 (1) : DB2 9.7에 추가된 관리자 뷰 이용

```
SQL> SELECT MODULENAME, DIALECT , MODULETYPE ,
REMARKS   FROM SYSCAT.MODULES

 MODULENAME     DIALECT     MODULETYPE     REMARKS
 -------------  ----------  -------------  --------------------
EMP_ADMIN       PL/SQL      P              PL/SQL Package
Body
 MOD_TEST1      DB2 SQL PL  M              (null)

SQL>SELECT OBJECTMODULENAME , OBJECTNAME  ,
OBJECTTYPE  FROM SYSCAT.MODULEOBJECTS

 OBJECTMODULENAME     OBJECTNAME      OBJECTTYPE
 -------------------  --------------  -------------
 EMP_ADMIN            GET_DEPT_NAME   FUNCTION
 EMP_ADMIN            UPDATE_EMP_SAL  FUNCTION
...
```

➲ 모듈 정보 조회 (2) : DB2 9.7 이전 버전에서 지원하는 관리자 뷰 예시

```
SQL> SELECT  ROUTINEMODULENAME ,   ROUTINENAME   ,
ROUTINETYPE FROM SYSCAT.ROUTINES

ROUTINEMODULENAME     ROUTINENAME              ROUTI
NETYPE
 --------------------  -----------------------  -------------
 DBMS_OUTPUT           ENABLE                     P
 DBMS_OUTPUT           GET_LINES                  P
 DBMS_ALERT            INIT                     P
```

Memo