5장 DB 프로그래밍 (DB Programming)

데이터베이스응용
한문석

목차

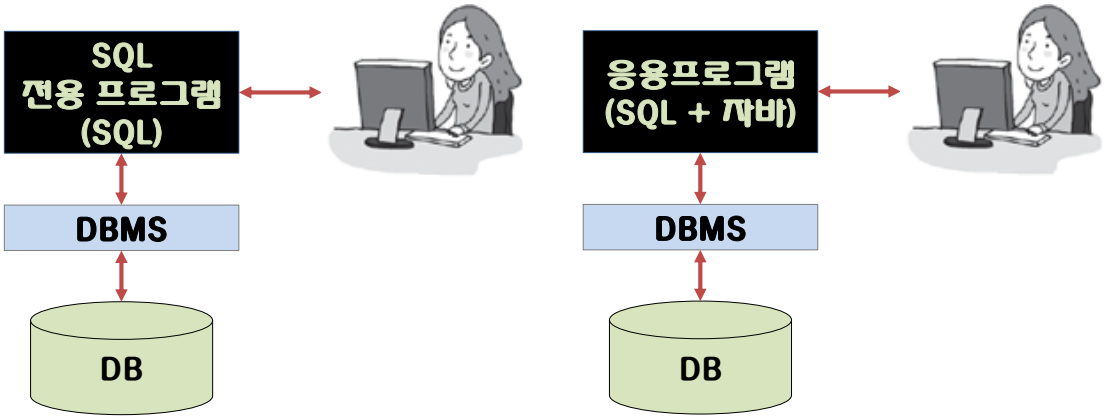
학습목표

- 데이터베이스 프로그래밍 개념 이해
- 저장프로그램 문법과 사용방법
- 자바 프로그램과 데이터베이스 연동 방법
- JSP 프로그램과 데이터베이스 연동 방법

01. 데이터베이스 프로그래밍의 개념

- '프로그래밍' 이란
  - 프로그램을 설계, 소스코드를 작성, 디버깅하는 과정
- 데이터베이스 프로그래밍
  - DB Programming 과정
    - DBMS에 데이터를 정의
    - 저장된 데이터를 읽어옴
    - 데이터를 변경하는 프로그램을 작성
  - 일반 프로그래밍과의 차이
    - 데이터베이스 언어인 SQL을 포함한다는 점이 다름

01. 데이터베이스 프로그래밍의 개념



(a) SQL Interface (b) 삽입 프로그래밍

01. 데이터베이스 프로그래밍 방법

- SQL 전용 언어를 사용하는 방법
  - SQL 자체 기능을 확장
  - 변수, 제어, 입출력 등의 기능을 추가한 새로운 언어를 사
  - MySQL: Stored Program
  - Oracle: PL/SQL, SQL Server: T-SQL 언어
- 일반 프로그래밍 언어에 SQL을 삽입하여 사용하는 방법
  - 자바, C, C++ 등 일반 프로그래밍 언어에 SQL 삽입하여 사용
  - Host 언어 응용에서 DB에 저장된 데이터를 관리, 검색
  - 삽입된 SQL문은 DBMS의 컴파일러가 처리함

01. 데이터베이스 프로그래밍 방법

- 웹 프로그래밍 언어에 SQL을 삽입하여 사용
  - 호스트 언어가 JSP, ASP, PHP 등 웹 스크립트 언어인 경우
  - 웹 프로그래밍 언어에서 데이터 관리 및 검색
  - 결과는 웹 브라우저에 출력, 웹 서버가 DB 연동 지원

- 4GL(4th Generation Language)
  - 데이터베이스 관리 기능과 비주얼 프로그래밍 기능을 갖춤
  - ‘GUI 기반 소프트웨어 개발 도구’ 를 사용
  - Delphi, Power Builder, Visual Basic 등

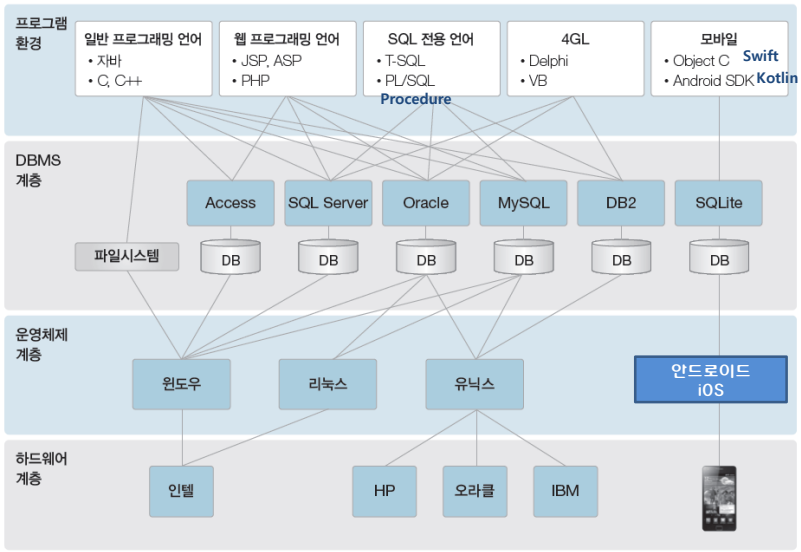
5장 DB 프로그래밍 (DB Programming)

데이터베이스응용

(10W-1) 2022-11-4(금)

한문석

01. 데이터베이스 프로그래밍의 유형



01. DBMS 종류 및 특징

특징	Access	SQL Server	Oracle	MySQL	DB2	SQLite
제조사	MS사	MS사	오라클사	오라클사	IBM사	리처드 힙 (오픈소스)
운영체제 기반	윈도우	윈도우	윈도우, 유닉스, 리눅스	윈도우, 유닉스, 리눅스	유닉스	모바일 OS :: 안드로이드 iOS
용도	개인용 DBMS	기업용 DBMS	대용량 DB 응용	소용량 DB 응용	대용량 DB 응용	모바일 DB

The basic development approaches:

- Use client-side programming to embed SQL statements in applications
  - a pre-compiler or Java translator before compilation
  - Alternatively, Java Database Connectivity (JDBC) or Oracle Call Interface (OCI)
- Use server-side programming to develop data logic that resides in the database
  - stored subprograms (procedures and functions)
    - written in Stored Procedure, PL/SQL or Java.
  - trigger, which is named program unit

목차

저장프로그램 학습내용

- 저장프로그램
- 트리거
- 사용자 정의 함수
- 저장프로그램 문법 요약

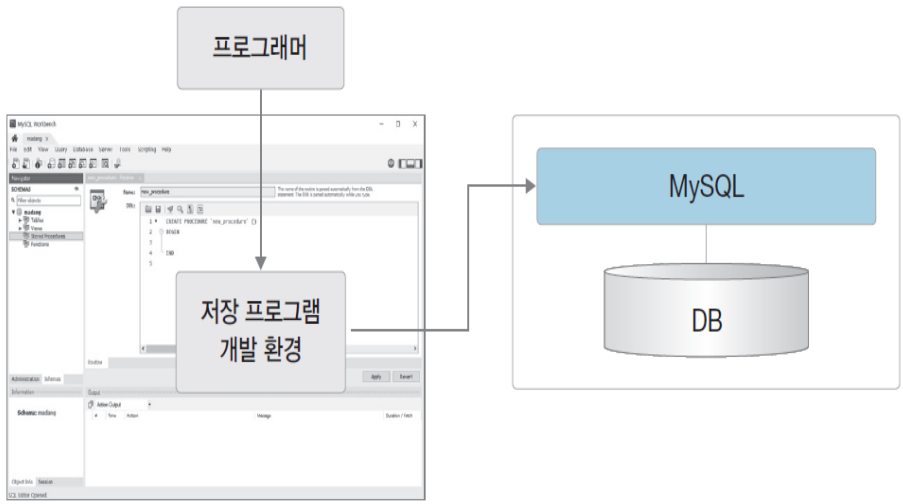
저장프로그램 개요

- 나중에 절차적으로 실행할 수 있음
- A server-side, stored procedural language
- A procedural extension to MySQL SQL
- 프로그램 논리를 프로시저로 구현
  - 객체 형태로 사용
- MySQL의 SQL 전용 언어
  - 데이터베이스 응용프로그램 작성에 사용
  - 함수와 비슷한 개념

저장프로그램 개요

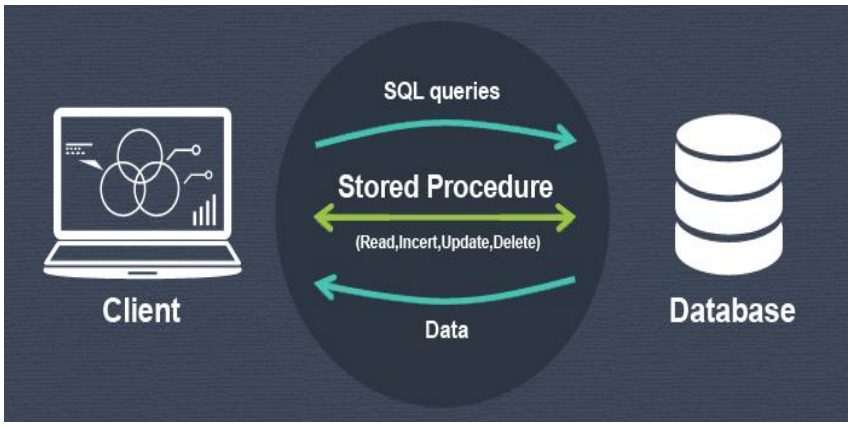
- SQL 문에 프로그래밍 기능 추가
  - 변수, 제어, 입출력 등
- SQL만으로 처리하기 어려운 문제를 해결
- 저장프로그램 작성 도구
  - MySQL Workbench에서 바로 작성
  - 컴파일하고 결과 실행

저장프로그램 개발환경



저장 프로그램

MYSQL Stored Procedures



저장 프로그램

- 작업 순서가 정해진 독립된 프로그램 수행 단위
- 종류
  - Stored Routine
    - Procedure
    - Function
  - Trigger
    - 데이터 변경문 실행 시 자동 실행하는 프로시저
    - 데이터 변경문: Insert, Delete, Update
  - Event Scheduler
    - manages the scheduling and execution of events

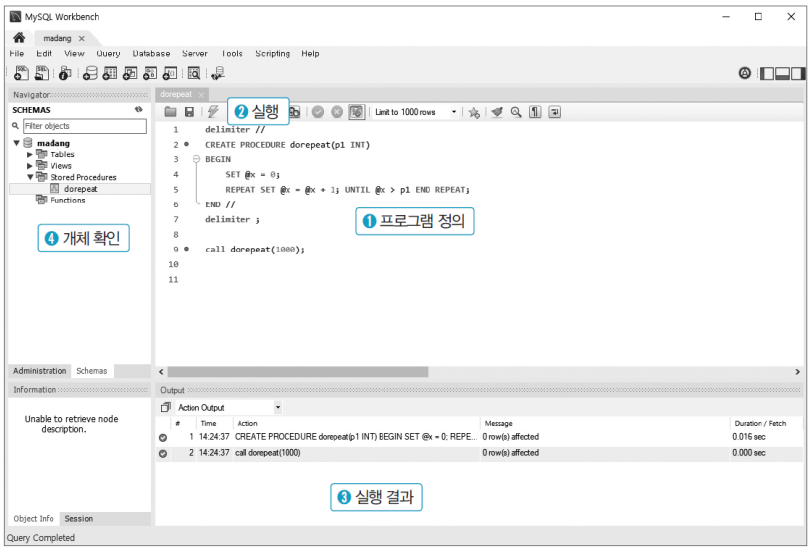
저장 프로그램

- **the syntax of creating the stored procedure**

```
CREATE PROCEDURE name_of_SP [(Parameter-1 datatype [,
Parameter-i datatype])]

BEGIN
        // Declaration part of stored procedure
        // Execution part of stored procedure
END;
```


저장 프로그램



저장 프로그램: 프로시저

- 프로시저 정의: CREATE PROCEDURE 문 사용
- 정의 방법
  - 프로시저 구성:
  - A compound statement made up of several statements separated by semicolon (;) characters.

```
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
        SET @x = 0;
        REPEAT SET @x = @x + 1;
        UNTIL @x > p1 END REPEAT;
END;
```


저장 프로그램: 프로시저

- 프로시저 정의: CREATE PROCEDURE 문 사용
- 정의 방법
  - 선언부와 실행부(BEGIN-END)
    - 선언부: 변수와 매개변수를 선언
    - 실행부: 프로그램 로직 구현

```
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
        SET @x = 0;
        REPEAT SET @x = @x + 1;
        UNTIL @x > p1 END REPEAT;
END;
```

선언부

실행부

저장 프로그램: 프로시저

- 프로시저 정의: CREATE PROCEDURE 문 사용
- 정의 방법
  - 매개변수(parameter)
    - 저장 프로시저가 호출될 때 그 프로시저에 전달되는 값

```
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
        SET @x = 0;
        REPEAT SET @x = @x + 1;
        UNTIL @x > p1 END REPEAT;
END;
```


저장 프로그램: 프로시저

- 정의 방법
  - 변수(variable)
    - 저장 프로시저나 트리거 내에서 사용되는 값
  - 소스코드 설명문
    - /*와 */ 사이에 기술
    - 설명문이 한 줄이면 이중 대시(--) 기호 다음에 기술해도 됨

```
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
        SET @x = 0;
        REPEAT SET @x = @x + 1;
        UNTIL @x > p1 END REPEAT;
END;
```

저장 프로그램: 프로시저 정의 및 실행 방법

- 기본 Mysql 문장구분기호(delimiter): 세미콜론;
- mysql client program을 사용한 정의
  - 세미콜론 문자를 포함하는 저장프로그램을 정의 시
    - 문제 발생 ⇒ 프로시저 끝과 문장구분기호 혼돈
- 임시로 문장구분기호 재정의 필요
  - Mysql이 서버로 전체 저장프로그램을 전달하기 위해
- To redefine the mysql delimiter
  - Use the delimiter command
    - 재정의: delimiter //
    - 실행 후 완원: delimiter ;

저장 프로그램: 프로시저 정의 및 실행 방법

- The following example: dorepeat() procedure

```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
    -> BEGIN
    ->   SET @x = 0;
    ->   REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL dorepeat(1000);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x;
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

5장 DB 프로그래밍 (DB Programming)

데이터베이스응용

(11W-2) 2022-11-14(월)

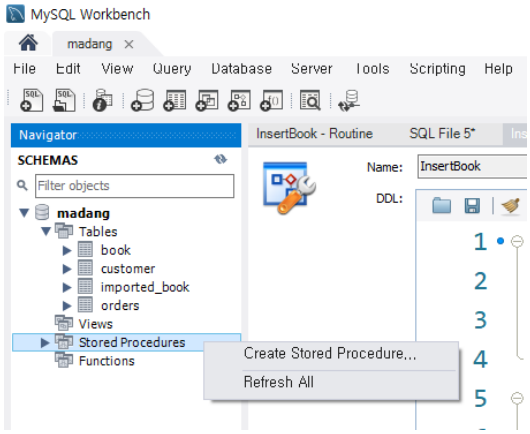
안문석

삽입 작업 프로시저

- 프로시저로 데이터를 삽입하는 작업
  - INSERT문 사용하여 입력 가능
- 프로시저 사용 삽입 장점
  - 복잡한 조건의 삽입 작업 수행 가능
    - 인자 값만 바꾸어 수행할 수도 있음
  - 저장해두었다가 필요할 때마다 호출하여 사용 가능

Procedure 작성(at Workbench)

- 프로시저 생성하기
  - 워크벤치를 실행
  - 왼쪽 Navigator에서
  - 다음 순서로 선택
    - SCHEMAS
    - Stored Procedure
    - Create Stored Procedure



삽입 작업 프로시저

예제 5-1 Book 테이블에 한 개의 튜플을 삽입하는 프로시저 (InsertBook)

```
01 use madang;
02 delimiter //
03 CREATE PROCEDURE InsertBook(
04   IN myBookID INTEGER,
05   IN myBookName VARCHAR(40),
06   IN myPublisher VARCHAR(40),
07   IN myPrice INTEGER)
08 BEGIN
09   INSERT INTO Book(bookid, bookname, publisher, price)
10     VALUES(myBookID, myBookName, myPublisher, myPrice);
11 END;
12 //
13 delimiter ;
```

프로시저 정의

프로시저 실행

```
A /* 프로시저 InsertBook을 테스트하는 부분 */
B CALL InsertBook(13, '스포츠과학', '마당과학서적', 25000);
C SELECT * FROM Book;
```

삽입 작업 프로시저

예제 5-1 Book 테이블에 한 개의 투플을 삽입하는 프로시저 (InsertBook)

```
01 use madang;
02 delimiter //
03 CREATE PROCEDURE InsertBook(
04    IN myBookID INTEGER,
05    IN myBookName VARCHAR(40),
06    IN myPublisher VARCHAR(40),
07    IN myPrice INTEGER)
08 BEGIN
09    INSERT INTO Book(bookid, bookname, publisher, price)
10      VALUES(myBookID, myBookName, myPublisher, myPrice);
11 END;
12 //
13 delimiter ;
```

```
A  /* 프로시저 InsertBook을 테스트하는 부분 */
B  CALL InsertBook(13, '스포츠과학', '마당과학서적', 25000);
C  SELECT * FROM Book;
```

프로시저 정의

프로시저 매개변수 정의
IN: 입력인자
OUT: 출력인자

명령문

InsertBook 프로시저 실행 후: Book 테이블

bookid	bookname	publisher	price
1	축구의 역사	굿스포츠	7000
2	축구아는 여자	나무수	13000
3	축구의 이해	대한미디어	22000
4	골프 바이블	대한미디어	35000
5	피겨 교본	굿스포츠	8000
6	역도 단계별기술	굿스포츠	6000
7	야구의 추억	이상미디어	20000
8	야구를 부탁해	이상미디어	13000
9	올림픽 이야기	삼성당	7500
10	Olympic Champions	Pearson	13000
13	스포츠과학	마당과학...	25000

프로시저 삭제문:

```
DROP PROCEDURE InsertBook;
```

제어문 사용 프로시저

- 어떤 조건에서 어떤 코드가 실행되어야 하는지 제어하는 문법
- 절차적 언어의 구성요소를 포함함.

구문	의미	문법
DELIMITER	•구문 종료 기호 설정	DELIMITER {기호}
BEGIN-END	• 저장프로그램 문 블록화 • 중첩 가능	BEGIN { SQL 문 } END
IF-ELSE	• 조건의 검사 결과에 따라 문장을 선택적으로 수행	IF <조건> SQL 문 [ELSE SQL 문] END IF;

제어문 사용 프로시저

구문	의미	문법
LOOP	• LEAVE 문을 만나기 전까지 LOOP을 반복	[label:] LOOP {SQL 문 | LEAVE [label] } END LOOP

WHILE	• 조건이 참일 경우 WHILE문 블록을 실행	WHILE <조건> DO
		{ SQL 문 | BREAK | CONTINUE }
		END WHILE

REPEAT	• 조건이 참일 경우 REPEAT 문의 블록을 실행	[label:] REPEAT {SQL 문 | BREAK | CONTINUE} UNTIL <조건> END REPEAT [label:]
RETURN	• 프로시저 종료 • 상태 값을 정수로 반환 가능	RETURN [<식>]

제어문 사용 프로시저

use madang

delimiter //

- 동일한 도서가 있는지 점검한 후
- 삽입하는 프로시저(BookInsertOrUpdate.sql)
- 삽입도서와 동일도서 있음면 도서의 가격만 변경

```
CREATE PROCEDURE BookInsertOrUpdate(
    myBookID INTEGER,
    myBookName VARCHAR(40),
    myPublisher VARCHAR(40),
    myPrice INT)
```


제어문 사용 프로시저

```
BEGIN
  DECLARE mycount INTEGER;     -- 지역변수 선언

  --myBookName과 동일한 이름의 책 확인하여 mycount에 저장
  SELECT count(*) INTO mycount FROM Book
    WHERE bookname LIKE myBookName;

  IF mycount!=0 THEN
      SET SQL_SAFE_UPDATES=0; /* DELETE, UPDATE 연산에 필요한 설정 문, 1로 설정하면 변경문 실행오류 */

      UPDATE Book SET price = myPrice
          WHERE bookname LIKE myBookName;
  ELSE

     INSERT INTO Book(bookid, bookname, publisher, price)
          VALUES(myBookID, myBookName, myPublisher, myPrice);
  END IF;
END;

//
delimiter ;
```

제어문 사용 프로시저

```
-- BookInsertOrUpdate 프로시저를 실행하여 테스트하는 부분
CALL BookInsertOrUpdate(15, '스포츠 즐거움', '마당과학서적', 25000);
SELECT * FROM Book; -- 15번 투플 삽입 결과 확인
-- BookInsertOrUpdate 프로시저를 실행하여 테스트하는 부분
CALL BookInsertOrUpdate(15, '스포츠 즐거움', '마당과학서적', 20000);
SELECT * FROM Book; -- 15번 투플 가격 변경 확인
```

```
A  -- BookInsertOrUpdate 프로시저를 실행하여 테스트하는 부분
B  CALL BookInsertOrUpdate(15, '스포츠 즐거움', '마당과학서적', 25000);
C  SELECT * FROM Book; /* 15번 투플 삽입 결과 확인 */
D  -- BookInsertOrUpdate 프로시저를 실행하여 테스트하는 부분
E  CALL BookInsertOrUpdate(15, '스포츠 즐거움', '마당과학서적', 20000);
F  SELECT * FROM Book; /* 15번 투플 가격 변경 확인 */
```


BookInsertOrUpdate 프로시저 실행한 후 Book 테이블

```
A  /* BookInsertOrUpdate 프로시저를 실행하여 테스트하는 부분 */
B  CALL BookInsertOrUpdate(15, '스포츠 즐거움', '마당과학서적', 25000);
C  SELECT * FROM Book; /* 15번 투플 삽입 결과 확인 */
```

```
D  /* BookInsertOrUpdate 프로시저를 실행하여 테스트하는 부분 */
E  CALL BookInsertOrUpdate(15, '스포츠 즐거움', '마당과학서적', 20000);
F  SELECT * FROM Book; /* 15번 투플 가격 변경 확인 */
```

BOOKID	BOOKNAME	PUBLISHER	PRICE
1	축구의 역사	굿스포츠	7000
2	축구아는 여자	나무수	13000
3	축구의 이해	대한미디어	22000
4	골프 바이블	대한미디어	35000
5	피겨 교본	굿스포츠	8000
6	역도 단계별기술	굿스포츠	6000
7	야구의 추억	이상미디어	20000
8	야구를 부탁해	이상미디어	13000
9	올림픽 이야기	삼성당	7500
10	Olympic Champions	Pearson	13000
13	스포츠과학	마당과학서적	25000
15	스포츠 즐거움	마당과학서적	25000

➡

BOOKID	BOOKNAME	PUBLISHER	PRICE
1	축구의 역사	굿스포츠	7000
2	축구아는 여자	나무수	13000
3	축구의 이해	대한미디어	22000
4	골프 바이블	대한미디어	35000
5	피겨 교본	굿스포츠	8000
6	역도 단계별기술	굿스포츠	6000
7	야구의 추억	이상미디어	20000
8	야구를 부탁해	이상미디어	13000
9	올림픽 이야기	삼성당	7500
10	Olympic Champions	Pearson	13000
13	스포츠과학	마당과학서적	25000
15	스포츠 즐거움	마당과학서적	20000

결과 반환 프로시저

Book 테이블에 저장된 도서의 평균가격을 반환하는 프로시저 (AveragePrice.sql)

```
delimiter //
CREATE PROCEDURE AveragePrice(
                        OUT AverageVal INTEGER)
BEGIN
        SELECT AVG(price) INTO AverageVal
        FROM Book WHERE price IS NOT NULL;
END;
//
delimiter ;
```

OUT: 출력 매개변수
INTO문: 변수에 값을 저장

결과 반환 프로시저

Book 테이블에 저장된 도서의 평균가격을 반환하는 프로시저 (AveragePrice)

```
A   /* 프로시저 AveragePrice를 테스트하는 부분 */
B   CALL AveragePrice(@myValue);
C   SELECT @myValue;
H   END;
```

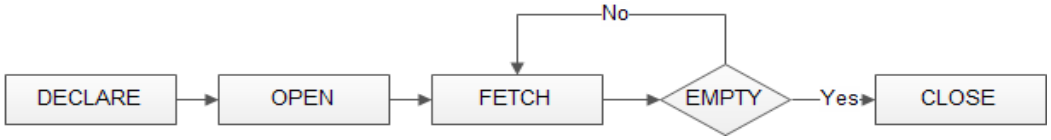
@myValue
15792

커서 사용 프로시저: Cursor

- 일련의 데이터에 순차적으로 액세스할 때
- 검색 및 "현재 위치"를 포함하는 데이터 요소
- 실행 결과 테이블을 한 번에 한 행씩 처리
  - 테이블의 행을 순서대로 가리키는 데 사용
- a pointer to a specific row within a query result
- The pointer can be moved from one row to the next.

커서 사용 프로시저: Cursor 사용법

1. 이용자는 검색 및 정렬 순서를 지정하여 커서를 정의한다.
2. 매개 변수를 사용하는 경우 값을 결정하고 데이터 액세스를 시작한다.
3. 커서 위치의 데이터를 검색한다. 검색 결과의 끝이라면 7에.
4. 필요에 따라 커서 위치의 데이터를 변경, 삭제한다.
5. 커서를 "다음 데이터"로 진행한다.
6. 3 반복.
7. 반복 종료 후 커서를 풀어 놓는다.

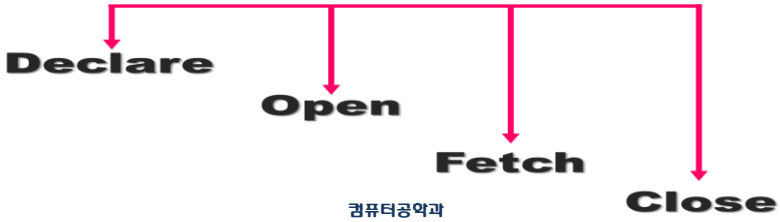


커서 사용 프로시저: Cursor 사용법

- To use cursors in SQL procedures
  1. Declare a cursor that defines a result set.
  2. Open the cursor to establish the result set.
  3. Fetch the data into local variables as needed from the cursor, one row at a time.
  4. Close the cursor when done.

커서 사용 프로시저

키워드	역할
CURSOR 〈cursor 이름〉 IS 〈커서 정의〉 DECLARE 〈cursor 이름〉 CURSOR FOR	커서를 생성
OPEN 〈cursor 이름〉	커서의 사용을 시작
FETCH 〈cursor 이름〉 INTO 〈변수〉	행 데이터를 가져옴
CLOSE 〈cursor 이름〉	커서의 사용을 끝냄



커서 사용 프로시저

Orders 테이블의 판매 도서에 대한 이익을 계산하는 프로시저(Interest.sql)

```
delimiter //
CREATE PROCEDURE Interest()  --프로시저 선언
BEGIN
    DECLARE myInterest INTEGER DEFAULT 0.0;
    DECLARE Price INTEGER;
    DECLARE endOfRow BOOLEAN DEFAULT FALSE;
    DECLARE InterestCursor CURSOR FOR
        SELECT saleprice FROM Orders;
    DECLARE CONTINUE handler
        FOR NOT FOUND SET endOfRow=TRUE;
    OPEN InterestCursor;  --커서 열기
```

지역변수 선언

InterestCursor: 커서변수

행 끝 확인(endOfRow) 준비. 처음에는 행의 끝이 아니므로 FALSE로 초기화

반복 조건을 준비하는 예약어

커서 사용 프로시저

Local Variable DECLARE Statement

DECLARE var_name [, var_name] ... type [DEFAULT value]

- To declare local variables within stored programs
- To provide a default value for a variable, include a DEFAULT clause.
- If the DEFAULT clause is missing, the initial value is NULL.

커서 사용 프로시저

Cursor DECLARE Statement

```
DECLARE cursor_name CURSOR FOR select_statement
```

- To declare a cursor
- To fetch the rows later, use a FETCH statement.
- The number of columns retrieved by the SELECT statement must match the number of output variables specified in the FETCH statement.
- The SELECT statement cannot have an INTO clause.

커서 사용 프로시저

DECLARE ... HANDLER Statement

```
DECLARE handler_action HANDLER
        FOR condition_value [, condition_value] ...
        statement

handler_action: {
        CONTINUE | EXIT | UNDO }

condition_value: {
        mysql_error_code
        | SQLSTATE [VALUE] sqlstate_value
        | condition_name
        | SQLWARNING
        | NOT FOUND
        | SQLEXCEPTION
}
```

커서 사용 프로시저

DECLARE ... HANDLER Statement

- a handler that deals with one or more conditions
- The handler_action value
  - CONTINUE: Execution of the current program continues.
  - EXIT: Execution terminates
  - UNDO: Not supported.

커서 사용 프로시저

DECLARE ... HANDLER Statement

- The **condition_value**
  - mysql_error_code:
    - MySQL error code, such as 1051 to specify "unknown table"
  - SQLSTATE [VALUE] sqlstate_value:
    - such as '42S01' to specify "unknown table" :
  - condition_name:
    - A condition name previously specified with DECLARE ... CONDITION.
  - SQLWARNING:
    - Shorthand for the class of SQLSTATE values that begin with '01'.
  - NOT FOUND:
    - Shorthand for the class of SQLSTATE values that begin with '02'.
    - the context of cursors
      - when a cursor reaches the end of a data set.

커서 사용 프로시저

Orders 테이블의 판매 도서에 대한 이익을 계산하는 프로시저(Interest.sql)

```
cursor_loop: LOOP  --Loop 반복문
    FETCH InterestCursor INTO Price; -- 다음 투플 가져옴
    IF endOfRow THEN LEAVE cursor_loop; --조건 만족하면 exit
    END IF;

    IF Price >= 30000 THEN
        SET myInterest = myInterest + Price * 0.1;
    ELSE
        SET myInterest = myInterest + Price * 0.05;
    END IF;

    END LOOP cursor_loop;
```

조건에 따라 이익금 저장

커서 사용 프로시저

Orders 테이블의 판매 도서에 대한 이익을 계산하는 프로시저(Interest.sql)

```
[begin_label:] LOOP
        statement_list
END LOOP [end_label]
```

- LOOP 문
  - To implement a simple loop construct
  - To enable repeated execution of the statement list
  - The statements within the loop are repeated until the loop is terminated.
  - To terminate: a LEAVE statement
    - Within a stored function, RETURN can also be used

커서 사용 프로시저

Orders 테이블의 판매 도서에 대한 이익을 계산하는 프로시저(Interest.sql)

```
CLOSE InterestCursor; --커서 사용 완료
  SELECT CONCAT(' 전체 이익 금액 = ', myInterest);
END;
//
delimiter ;
```

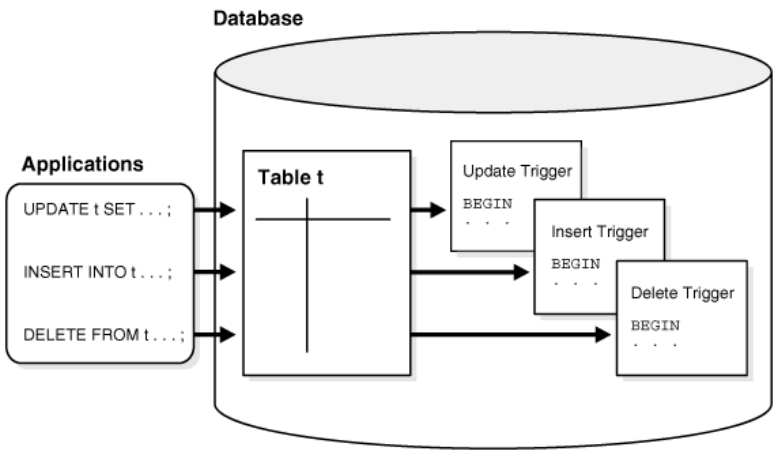
```
/* Interest 프로시저를 실행하여 판매된 도서에 대한 이익금을 계산 */
CALL Interest();
```

CONCAT(' 전체 이익 금액 = ', myInterest)
전체 이익 금액 = 5900

트리거

- stored programs
- a named database object that is associated with a table
- automatically executed or fired when some events occur
- 데이터의 변경(INSERT, DELETE, UPDATE)문이 실행될 때 자동 실행되는 프로시저

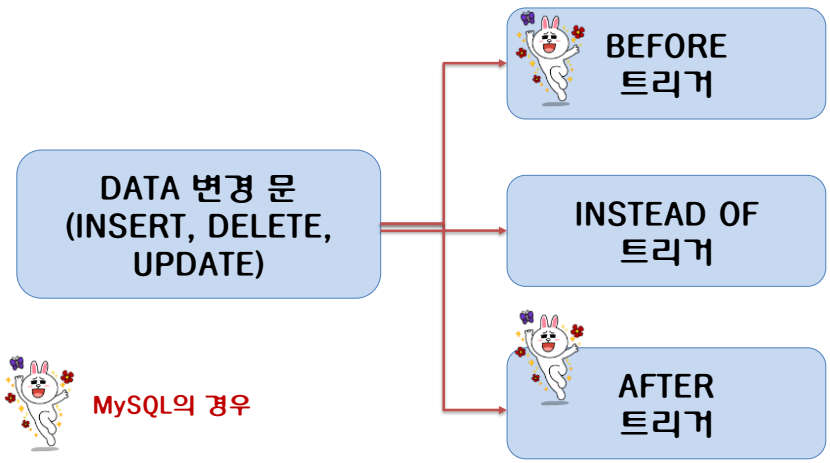
트리거



트리거 사용할 때

- 수행작업
  - 데이터 기본값 제공, 데이터 제약 준수
  - SQL 뷰의 수정, 참조무결성
- Automatically generate derived column values
- Prevent invalid transactions
- Provide auditing and event logging
- Record information about table access

데이터 변경과 트리거의 수행



The syntax for creating a trigger

```
CREATE
    [DEFINER = user]
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    [trigger_order]
    trigger_body
trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }
trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

The DEFINER clause:
specify the MySQL account to be used when checking access privileges at trigger activation time.

tbl_name: permanent table

the trigger action time

the kind of operation

With FOLLOWS, the new trigger activates after the existing trigger.

With PRECEDES, the new trigger activates before the existing trigger.

트리거 예

새로운 도서를 삽입한 후
자동으로 Book_log 테이블에 삽입한 내용을
기록하는 트리거

```
SET global log_bin_trust_function_creators=ON;
/* 실습을 위해 root 계정에서 실행
```

It controls whether stored function creators can be **trusted** not to create stored functions.

트리거 예

새로운 도서를 삽입한 후
자동으로 Book_log 테이블에 삽입한 내용을
기록하는 트리거

```
/* madang 계정에서 실습 위한 Book_log 테이블 생성*/
CREATE TABLE Book_log(
  bookid_l INTEGER,
  bookname_l VARCHAR(40),
  publisher_l VARCHAR(40),
  price_l INTEGER);
```


트리거 예

```
delimiter //
-- 트리거 선언
CREATE TRIGGER AfterInsertBook
-- Book 테이블에 INSERT문이 실행되면 트리거 자동 실행
    AFTER INSERT ON Book FOR EACH ROW

BEGIN
   DECLARE average INTEGER;
   --Book_log 테이블에 삽입, new=새로 삽입될 투플 값 지정자
   INSERT INTO Book_log
   VALUES(new.bookid, new.bookname, new.publisher, new.price);
END;
//
delimiter ;
```

트리거 예

```
A /* 삽입한 내용을 기록하는 트리거 확인 */
B    INSERT INTO Book VALUES(14, '스포츠 과학 1', '이상미디어', 25000);
C    SELECT * FROM Book WHERE bookid='14';
D    SELECT * FROM Book_log WHERE bookid_l='14'; /* 결과 확인 */
```

Book 테이블에 투플을 삽입하여 트리거가 실행된 결과

Action	Message
INSERT INTO Book VALUES(14, '스포츠 과학 1', '이상미디어', 25...	1 row(s) affected

Book 테이블 Insert (B 행)

bookid	bookname	publisher	price
14	스포츠 과학 1	이상미디어	25000

Book 테이블 (C 행)

bookid_l	bookname_l	publisher_l	price_l
14	스포츠 과학 1	이상미디어	25000

Book_log 테이블 (D 행)

사용자 정의 함수

사용자 정의 함수

- 입력된 값을 가공하여 결과 값을 반환
- 내장함수 vs. 사용자정의함수
- 프로시저: CALL 명령에 의해 실행
  - 독립프로그램
- 사용자정의함수
  - Select문 또는 프로시저에서 호출
  - 결과값을 제공
- 스칼라함수가 일반적
  - 단일값 반환

사용자 정의 함수

- CREATE FUNCTION Statement: for User-Defined Functions
- An external stored function
- RETURN문: 반드시 반환 값의 데이터 타입을 선언
- 파라미터: 함수의 매개변수는 항상 IN 파라미터

사용자 정의 함수

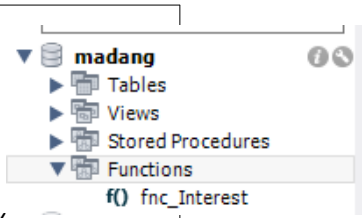
- 함수정의
  - --사용자정의함수(user-defined function (UDF))를 생성
  - -- 함수 실행블록에는 적어도 한 개의 RETURN 문이 있어야 한다.
  - -- Block은 함수가 수행 할 내용을 정의한 몸체부분이다.

```
CREATE FUNCTION 함수_이름[(매개변수, …)]
    RETURNS type      -- type은 반환되는 값의 datatype
BEGIN
    [변수 선언 부분]
    [SQL 실행문]          블록
    RETURN 변수;
END;
```


사용자 정의 함수

판매된 도서에 대한 이익을 계산하는 함수(fnc_Interest.sql) 실행

```
01 delimiter //
02 CREATE FUNCTION fnc_Interest(
03       Price INTEGER) RETURNS INT
04 BEGIN
05 DECLARE myInterest INTEGER;
06 -- 가격이 30,000원 이상이면 10%, 30,000원 미만이면 5%
07     IF Price >= 30000 THEN SET myInterest = Price * 0.1;
08                       ELSE SET myInterest := Price * 0.05;
09     END IF;
10 RETURN myInterest;
11 END; //
12 delimiter ;
```



log_bin_trust_function_creators

- It controls whether stored function creators can be trusted not to create stored functions
  - SET GLOBAL log_bin_trust_function_creators = 1;

```
ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO
SQL, or READS SQL DATA in its declaration and binary logging is enabled
(you *might* want to use the less safe log_bin_trust_function_creators
variable)
```


사용자 정의 함수

```
A /* Orders 테이블에서 각 주문에 대한 이익을 출력 */
B    SELECT custid, orderid, saleprice, fnc_Interest(saleprice) interest
C    FROM Orders;
```

Result Grid | Filter Rows: | Export:

custid	orderid	saleprice	interest
1	1	6000	300
1	2	21000	1050
2	3	8000	400
3	4	6000	300
4	5	20000	1000
1	6	12000	600
4	7	13000	650
3	8	12000	600
2	9	7000	350
3	10	13000	650

프로시저, 트리거, 사용자 정의 함수 공통점과 차이점

	프로시저	트리거	사용자 정의 함수
공통점	저장 프로시저		
정의 방법	CREATE PROCEDURE	CREATE TRIGGER	CREATE FUNCTION
호출 방법	CALL문 직접 호출	INSERT, DELETE, UPDATE 문이 실행될 때, 자동으로 실행됨	SELECT, Procedure 문에서 호출
기능 차이	SQL문으로 할 수 없는 복잡한 로직 수행	기본 값 제공, 데이터 제약 준수, SQL 뷰의 수정, 참조무결성 작업 등을 수행	속성 값 가공 반환, SQL문에 직접 사용

저장프로그램 문법 요약

구분	명령어
Data Definition Language (데이터 정의어)	CREATE TABLE CREATE PROCEDURE CREATE FUNCTION CREATE TRIGGER ALTER, DROP
Data Manipulation Language (데이터 조작어)	SELECT INSERT DELETE UPDATE
Data Types(데이터 타입)	INTEGER, VARCHAR(n), DATE
Variables(변수)	DECLARE 문으로 선언 치환(SET, = 사용)
Operator(연산자)	산술연산자 (+, −, *, /) 비교연산자 (=, <, >, >=, <=, <>) 문자열연산자 (||) 논리연산자 (NOT, AND, OR)

저장프로그램 문법 요약

구분	명령어
Language Element(주석)	--, /* */
Built-in Function (내장함수)	숫자 함수 (ABS, CEIL, FLOOR, POWER 등) 집계 함수 (AVG, COUNT, MAX, MIN, SUM) 날짜 함수 (SYSDATE, DATE, DAYNAME 등) 문자 함수 (CHR, LEFT, LOWER, SUBSTR 등)
Control of Flow (제어문)	BEGIN-END IF-THEN-ELSE WHILE , LOOP
Data Control Language (데이터 제어어)	GRANT REVOKE