1. **MySQL Stored Procedure 작성하기**

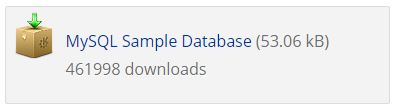
  우리는 구문(syntax)에 친숙해질 수 있도록 이름이 GetAllProducts()인 간단한 저장 프로시저를 개발할 것이다. GetAllProducts() 저장 프로시저는 products 테이블에 있는 모든 products을 선택하는 것이다.

mysql 클라이언트 툴을 실행시키고 다음 명령어를 입력해라:

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6 | DELIMITER //   CREATE PROCEDURE GetAllProducts()     BEGIN     SELECT \* FROM products;     END //  DELIMITER ; | [cs](http://colorscripter.com/info#e) |

접기

샘플 데이터는 <http://www.mysqltutorial.org/mysql-sample-database.aspx> 페이지에서 받을 수 있습니다!



이후 위 버튼을 클릭해서 받으시고 import 해주시면 가능합니다!

접기

명령어들을 세부적으로 확인해보자:

* 첫번째 명령어는 DELIMITER ( // ) 입니다. 이것은 저장 프로시저 구문과는 관련이 없다. DELIMITER문은 표준 구분 기호인 세미콜론( ; ) ??. 이 경우에 구분 기호는 세미콜론 ( ; ) 에서 더블 슬래시 ( // ) 로 변경되어진다. 왜 구분기호를 변경할 까? 왜냐하면 우리는 MySQL 도구가 매번 각 문장을 실행하는 것 보다 서버에 저장 프로시저를 통과 시키는 것을 원하기 때문이다.  END 키워드는 우리가 저장 프로시저의 끝을 표시할 때 구분기호 ( // ) 를 사용한다. 마지막 명령어 ( DELIMITER ; )는 구분기호를 다시 세미콜론 ( ; )으로 변경하기 위해 사용한다.
* 우리는 새로운 저장 프로시저를 작성할때 CREATE PROCEDURE 문을 사용한다. 우리는 CREATE PROCEDURE 문 뒤에 저장 프로시저의 이름을 명시한다. 위 예시의 경우에, 저장 프로시저의 이름은 GetAllProducts 이다. 우리는 저장 프로시저의 이름 뒤에 괄호를 놓는다.
* BEGIN 과 END 사이의 부분에 저장 프로시저의 바디 부분을 호출한다. 너는 비즈니스 로직을 처리하기 위해 바디 부분에 서술한 SQL 문을 작성할 수 있다. 위의 예제 저장 프로시저의 경우에, 우리는 products table 에서 data를 질의 하기 위해 간단한 SELECT 문법을 사용한다.

저장 프로시저 호출하기

저장 프로시저를 호출하기 위해, 아래와 같은 SQL 명령문을 사용해야 한다.

|  |  |  |
| --- | --- | --- |
| 1 | CALL STORED\_PROCEDURE\_NAME(); | [cs](http://colorscripter.com/info#e) |

저장 프로시저를 호출하기 위해 CALL 문을 사용할 수 있다. 예시에서 GetAllProducts() 저장 프로시저를 호출하기 위해 아래와 같은 코드를 사용할 수 있다.

|  |  |  |
| --- | --- | --- |
| 1 | CALL GetAllProducts(); | [cs](http://colorscripter.com/info#e) |

만약, 위의 명령문을 실행한다면, products 테이블의 모든 prodcuts 들을 볼 수 있을 것이다.

1. **변수 선언 및 사용**

변수는 데이터 객체에 의해서 이름이 지어진다. 데이터 객체의 값은 저장 프로시저가 실행되는 동안 값이 바뀔 수 있다. 우리는 전형적으로 실행결과를 저장(hold)하기 위해 저장 프로시저에서 변수를 사용한다. 이러한 변수들은 저장 프로시저의 로컬에 있다.

변수를 사용하기 전에는 변수를 선언해야 한다.

**변수 선언**

저장 프로시저 내부에서 변수를 선언하기 위해서는 아래와 같은 DECLARE 문을 사용해야 한다.

|  |  |  |
| --- | --- | --- |
| 1 | DECLARE variable\_name datatype(size) DEFAULT default\_value; | [cs](http://colorscripter.com/info#e) |

예시로 든 위의 명령문에 대해서 세부적으로 알아보자:

첫번째, DECLARE 키워드 뒤에 변수 이름을 지정 할 수 있다. 변수 이름은 MySQL table column names의 naming규칙을 따라야 한다.

두번째, 변수의 데이터 타입과 크기를 지정 할 수 있다. 변수는 INT, VARCHAR, DATETIME, 등 과 같은 MySQL 데이터 타입을 가질 수 있다.

세번째, 변수를 선언했을 때 초기값은 NULL 이다. DEFAULT 키워드를 사용해서 변수의 기본 값을 지정해 줄 수 있다.

예를들어, 아래와 같이 데이터 타입이 INT 이고, 기본 값이 0인 total\_sale의 이름을 가진 변수를 선언할 수 있다.

|  |  |  |
| --- | --- | --- |
| 1 | DECLARE total\_sale INT DEFAULT 0; | [cs](http://colorscripter.com/info#e) |

MySQL은 아래와 같이 하나의 DECLARE 문으로  같은 데이터 타입을 사용하는 두개 또는 그 이상의 변수들을 선언하는 것을 허용한다.

|  |  |  |
| --- | --- | --- |
| 1 | DECLARE x, y INT DEFAULT 0; | [cs](http://colorscripter.com/info#e) |

기본 값으로 0을 가지는 두개의 정수값 변수 x 와 y를 선언 했다.

**변수 대입**

변수를 선언 했으면, 이제 변수를 사용할 수 있다. 변수에 다른 값을 대입하고 싶다면, SET 문을 사용할 수 있다. 예를들어,

|  |  |  |
| --- | --- | --- |
| 1  2 | DECLARE total\_count INT DEFAULT 0;  SET total\_count = 10; | [cs](http://colorscripter.com/info#e) |

total\_count를 선언 한 후에 변수의 값은 10이다.

게다가, SET 문으로 Scalar 값(1개의 값)을 반환하는 질의의 결과 를 SELECT INTO 문을 사용해서 변수에 대입 할 수 있다.  아래의 예시를 보자:

|  |  |  |
| --- | --- | --- |
| 1  2  3  4 | DECLARE total\_products INT DEFAULT 0    SELECT COUNT(\*) INTO total\_products  FROM products | [cs](http://colorscripter.com/info#e) |

예시를 살펴보면,

첫번째로, 초기값을 0으로 가지는 total\_products 를 선언한다.

그리고, SELECT INTO 문을 사용하여 샘플 데이터베이스 안에 있는 products 테이블을 선택하고 products 의 레코드 수를 total\_products 변수에 대입하는 것을 볼 수 있다.

**변수 범위**

변수는 변수의 수명주기를 정의한 범위를 가지고 잇다. 만약 저장 프로시저 내부에 변수를 선언한다면 저장 프로시저의 END 문에 도달했을 때 변수는 사라진다.

**3. 매개변수**

**IN, OUT, INOUT**

IN - IN은 가장 기본 모드이다. 저장 프로시저에서 IN 매개변수를 정의할 경우, 호출 프로그램은 저장 프로시저에 매개변수를 전달한다. 게다가, IN 매개변수의 값은 보호되고 있다. 이것은 IN 매개변수 의 값은 저장 프로시저 내부에서 변경 될 수 있다는 것을 의미한다. 원본 값은 저장 프로시저가 끝난 후에도 유지 되어진다. 다른 말로, 저장 프로시저는 IN 매개변수의 복사본만 사용하게 된다.

OUT - OUT 매개변수의 값은 저장 프로시저 내부에서 변경되어 질 수 있고, 새로운 값은 호출 한 프로그램에 다시 전달할 수 있다. 프로그램을 시작 했을때, OUT 매개변수의 초기 값에 접근할 수 없는 것을 알고 있어야 한다.

INOUT - INOUT 매개변수는 IN 과 OUT 매개변수를 결합한 것이다. 이것은 호출 프로그램이 인수(argument)를 전달 하고, 저장 프로시저는 INOUT 매개변수를 수정 후에 호출 한 프로그램에 새로운 값을 다시 전달해주는 것을 의미한다.

저장 프로시저의 매개변수를 정의하는 문법은 아래와 같다:

|  |  |  |
| --- | --- | --- |
| 1 | MODE param\_name param\_type(param\_size) | [cs](http://colorscripter.com/info#e) |

MODE는 IN, OUT, INOUT 을 지정할 수 있다. 저장 프로시저의 매개변수의 목적에 의존되어진다.

param\_name은 매개변수의 이름이다. 매개변수의 이름은 MySQL의 column name의 naming rules을 따른다.

매개변수 이름 다음은 데이터 타입과 크기이다. 변수처럼 매개변수의 데이터 타입은 유효한 MySQL 데이터 타입을 가질 수 있다.

각각의 매개변수는 만약 저장 프로시저가 한개 이상의 매개변수를 가진다면 콤마( , )에 의해 구분되어진다.

더 쉽게 이해하기 위해서 몇가지 예제를 연습해보자. 연습은 이전에 받아 두었던 샘플 데이터 베이스의 테이블을 사용할 것이다.

**MySQL 저장 프로시저 매개변수 연습**

**IN 매개변수 연습**

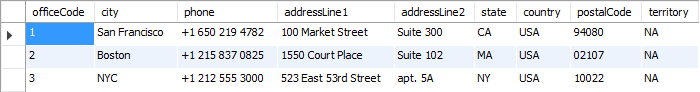
예제를 따라오면 특정한 도시에 위치한 사무실을 선택하는 GetOfficeByCountry 저장 프로시저에 있는 IN 매개변수를 사용하는 방법을 명확히 알 수 있다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8 | DELIMITER //  CREATE PROCEDURE GetOfficeByCountry(IN countryName VARCHAR(255))   BEGIN   SELECT \*   FROM offices   WHERE country = countryName;   END //  DELIMITER ;  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

countryName은 저장 프로시저의 IN 매개변수 이다. 저장 프로시저 내부에서, countryName 매개변수로 특정된 도시의 위치에 있는 모든 사무실을 선택한다.

추측컨데, 우리는 USA에 있는 모든 사무실을 얻기 위해서, 아래처럼 저장 프로시저에 값(USA)을 전달 해야한다.

|  |  |  |
| --- | --- | --- |
| 1 | CALL GetOfficeByCountry('USA'); | [cs](http://colorscripter.com/info#e) |



프랑스에 있는 모든 사무실을 얻기 위해서는 우리는 아래처럼 GetOfficeByCountry 저장 프로시저에 프랑스 문자만 전달하면 된다:

|  |  |  |
| --- | --- | --- |
| 1 | CALL GetOfficeByCountry('France') | [cs](http://colorscripter.com/info#e) |



**OUT 매개변수 예제**

아래는 주문 상태의 수를 반환하는 저장 프로시저이다. 이 프로시저는 두개의 매개변수를 가지고 있다:

orderStatus : IN 매개변수는 주문 상태의 이름이다.

total : OUT 매개변수는 지정된 주문 상태의 갯수를 저장하는 변수이다.

아래는 CountOrderByStatus 저장 프로시저의 소스 코드 이다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | DELIMITER $$  CREATE PROCEDURE CountOrderByStatus(   IN orderStatus VARCHAR(25),   OUT total INT)  BEGIN   SELECT count(orderNumber)   INTO total   FROM orders   WHERE status = orderStatus;  END$$  DELIMITER ; | [cs](http://colorscripter.com/info#e) |

완료된 주문수를 얻기 위해서 우리는 CountOrderByStatus 저장 프로시저를 호출하고, Shipped인 주문 상태를 전달한다. 그리고 또한 반환 값을 얻기 위해서 인수 ( @total ) 을 전달한다.

|  |  |  |
| --- | --- | --- |
| 1  2 | CALL CountOrderByStatus('Shipped', @total);  SELECT @total; | [cs](http://colorscripter.com/info#e) |



처리 중인 주문의 수를 얻기 위해서 우리는 아래처럼 CountOrderByStatus 저장 프로시저를 호출한다.

|  |  |  |
| --- | --- | --- |
| 1  2 | CALL CountOrderByStatus('in process',@total);  SELECT @total AS total\_in\_process; | [cs](http://colorscripter.com/info#e) |



**INOUT 매개변수 예제**

아래 예제는 저장 프로시저에서 INOUT 매개변수를 어떻게 사용하는지 설명하고 있다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6 | DELIMITER $$  CREATE PROCEDURE set\_counter(INOUT count INT(4),IN inc INT(4))  BEGIN   SET count = count + inc;  END$$  DELIMITER ;  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

동작 방법

set\_counter 저장 프로시저는 하나의 INOUT 매개변수( count ) 와 하나의 IN 매개변수 ( inc ) 를 받는다.

저장 프로시저 내부에서, inc 매개변수의 값으로 받은 counter ( count ) 의 값을 증가시킨다.

set\_counter 저장 프로시저를 어떻게 호출 하는지 볼 수 있다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5 | SET @counter = 1;  CALL set\_counter(@counter,1); -- 2  CALL set\_counter(@counter,1); -- 3  CALL set\_counter(@counter,5); -- 8  SELECT @counter; -- 8 | [cs](http://colorscripter.com/info#e) |

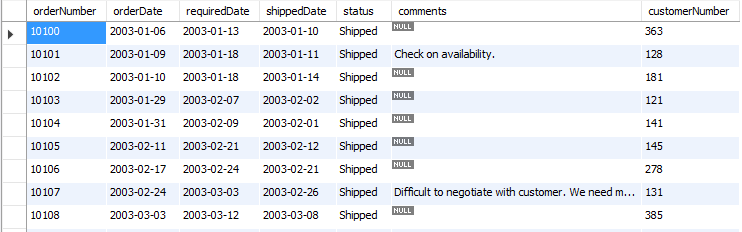


이번 강좌에서는 저장 프로시저에서 어떻게 매개변수를 정의하는지 보여 주었고, 매개변수의 모드(IN, OUT, INOUT)들에 대해서 소개하였다.

**4. 다중값 반환**

**다중 값을 반환하는 저장프로시저 예제**

샘플 데이터베이스의 orders 테이블을 살펴보자.



아래의 저장 프로시저는 customerNumber를 받고 shipped, cancled, resolved, disputed의 총 갯수를 반환하고 있다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | DELIMITER $$    CREATE PROCEDURE get\_order\_by\_cust(   IN cust\_no INT,   OUT shipped INT,   OUT canceled INT,   OUT resolved INT,   OUT disputed INT)  BEGIN   -- shipped  SELECT count(\*) INTO shipped    FROM orders    WHERE customerNumber = cust\_no AND status = 'Shipped';     -- canceled    SELECT count(\*) INTO canceled    FROM orders    WHERE customerNumber = cust\_no AND status = 'Canceled';     -- resolved    SELECT count(\*) INTO resolved    FROM orders    WHERE customerNumber = cust\_no AND status = 'Resolved';     -- disputed   SELECT count(\*) INTO disputed    FROM orders    WHERE customerNumber = cust\_no AND status = 'Disputed';    END $$  DELIMITER ;  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

IN 매개변수에 더하여, 저장프로시저는 4개의 OUT 매개변수 (shipped, canceled, resolved, disputed) 를 추가로 받는다(takes). 저장 프로시저 내부에서, 주문들의 상태로 부터 주문의 총 갯수를 얻는 COUNT함수와 SELECT 문 을 사용하고, 각각의 매개변수에 할당한다.

get\_order\_by\_cust 저장 프로시저를 사용하기 위해서는 customer number와 결과 값을 얻기 위한 4개의 사용자 정의 변수를 전달 해야 한다.

저장 프로시저 실행 후에 변수 값을 출력하기 위해 SELECT 문을 사용한다.

|  |  |  |
| --- | --- | --- |
| 1  2 | CALL get\_order\_by\_cust(141,@shipped,@canceled,@resolved,@disputed);  SELECT @shipped,@canceled,@resolved,@disputed; | [cs](http://colorscripter.com/info#e) |



**5. MySQL IF 문 구조**

MySQL IF 문은 식의 특정한 조건이나 값에 맞는 SQL 문을 실행할 수 있다. MySQL에서 식을 형성하려면, 문자, 변수 연산자 심지어 함수도 결합할 수 있다. 표현식은 세개의 값(TRUE, FALSE, NULL) 중 하나를 반환한다.

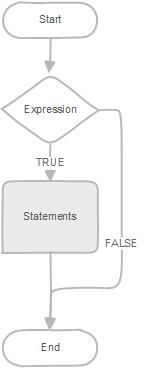
MySQL IF 문 구조

IF 문의 구조는 아래의 설명을 따른다.

|  |  |  |
| --- | --- | --- |
| 1  2  3 | IF expression THEN     statements;  END IF; | [cs](http://colorscripter.com/info#e) |

만약 조건식(expression)이 TRUE 일 경우에는 statements 식을 실행 할 것이고, 그렇지 않으면 다음 상태인 END IF로 넘어 갈 것이다.

IF 문을 순서도(FlowChart) 그림은 아래와 같다.

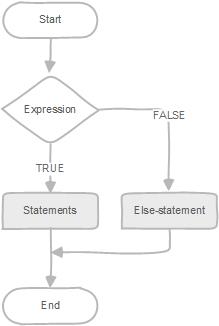


**MySQL IF ELSE 문**

이번 케이스에서는 조건식(expression)의 값이 FALSE 일때의 식(statement)을 실행하고 싶다면, 아래처럼 IF ELSE 문을 사용할 수 있다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5 | IF expression THEN     statements;  ELSE     else-statements;  END IF; | [cs](http://colorscripter.com/info#e) |

아래는 IF ELSE 문의 순서도 그림이다.



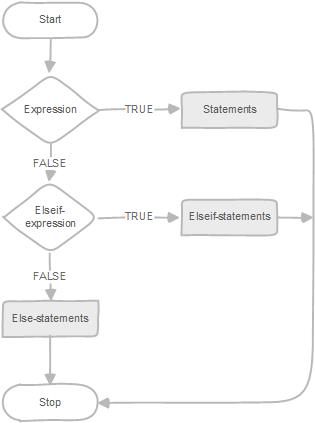
**MySQL IF ELSEIF ELSE 문법**

만약 다양한 조건 식을 실행하고 싶다면, 아래 처럼 IF ELSEIF ELSE 문을 사용할 수 있다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8 | IF expression THEN     statements;  ELSEIF elseif-expression THEN     elseif-statements;  ...  ELSE     else-statements;  END IF; | [cs](http://colorscripter.com/info#e) |

만약 조건식(expression)의 값이 TRUE 이면, statement를 실행할 것이다. 만약 조건식(expression)의 값이 FALSE 이면, MySQL은 다른 조건식(elseif-expression)을 체크하고 다른 조건식의 값이 TRUE이면 elseif-statement를 실행 할 것이다.

IF문은 다양한 조건식을 확인하기 위해 많은 ELSEIF 가지(branch)를 가질 수 있다. 만약 맞는 조건식이 없다면, ELSE에 있는 else-statement 가 실행 될 것이다.



**MySQL IF 문 예제**

아래의 예제는 어떻게 IF ELSEIF ELSE 문을 사용하는지 보여주고 있다. GetCustomerLevel()은 두개의 매개변수(customer number,  customer level)를 받는 저장 프로시저이다.

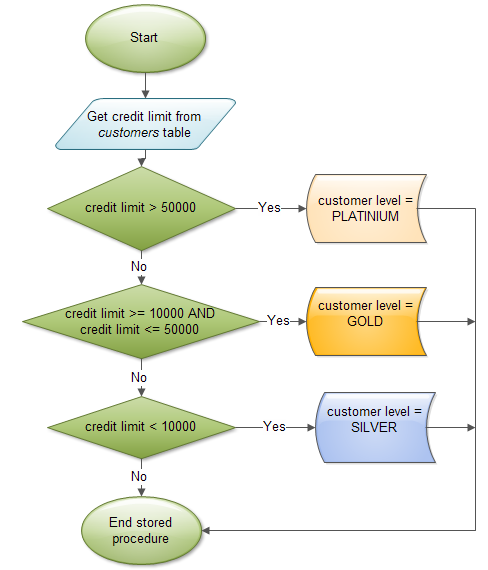
첫번째로, customer 테이블로 부터 신용 한도를 받아온다.

그 후, 신용 한도에 맞는 customer level(PLATINUM, GOLD, SILVER)를 결정한다.

매개변수 p\_customerlevel은 customer의 level을 저장하고 호출 프로그램에서 사용되어진다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | DELIMITER $$    CREATE PROCEDURE GetCustomerLevel(      IN  p\_customerNumber int(11),      OUT p\_customerLevel  varchar(10))  BEGIN      DECLARE creditlim double;        SELECT creditlimit INTO creditlim      FROM customers      WHERE customerNumber = p\_customerNumber;        IF creditlim > 50000 THEN    SET p\_customerLevel = 'PLATINUM';      ELSEIF (creditlim <= 50000 AND creditlim >= 10000) THEN          SET p\_customerLevel = 'GOLD';      ELSEIF creditlim < 10000 THEN          SET p\_customerLevel = 'SILVER';      END IF;    END$$    DELIMITER ;  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

아래의 순서도는 customer level을 결정하는 로직을 설명하고 있다.



**6. 간단한 CASE 문**

간단한 CASE문의 구조에 대해서 살펴보자:

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6 | CASE  case\_expression     WHEN when\_expression\_1 THEN commands     WHEN when\_expression\_2 THEN commands     ...     ELSE commands  END CASE; | [cs](http://colorscripter.com/info#e) |

고유값의 집합에 대해 표현식의 값을 체크하는 간단한 CASE문을 사용할 수 있다.

case\_expression은 올바른 표현식이 될 수 있다. 우리는 각 WHEN절(when\_expression\_1, when\_expression\_2, etc)의 값과 case\_expression의 값을 비교한다. 만약 case\_expression 의 값과 when\_expression\_n 의 값이 동일 할 때, 해당하는 WHEN의 commands를 실행한다.

WHEN의 when\_expression 과 case\_expression의 값이 일치하는 조건이 없을 경우에는, ELSE 절에 있는 commands를 실행하게 된다. ELSE 절은 옵션으로 제공한다. 만약 ELSE 절을 작성하지 않고 조건에 맞는 값을 찾지 못할 경우에, MySQL은 에러를 출력한다.

아래 예시는 간단한 CASE 문의 사용방법을 보여주고 있다.

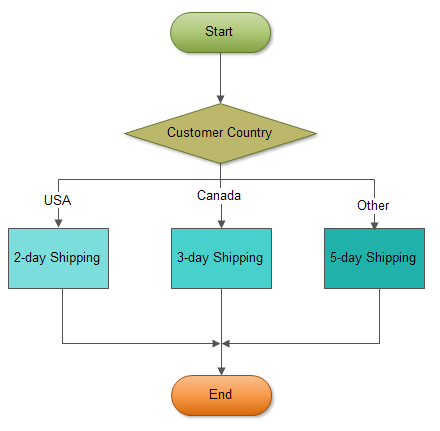
|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | DELIMITER $$    CREATE PROCEDURE GetCustomerShipping(   IN  p\_customerNumber int(11),   OUT p\_shiping        varchar(50))  BEGIN      DECLARE customerCountry varchar(50);        SELECT country INTO customerCountry    FROM customers    WHERE customerNumber = p\_customerNumber;        CASE customerCountry    WHEN  'USA' THEN       SET p\_shiping = '2-day Shipping';   WHEN 'Canada' THEN      SET p\_shiping = '3-day Shipping';   ELSE      SET p\_shiping = '5-day Shipping';   END CASE;  END$$    DELIMITER ; | [cs](http://colorscripter.com/info#e) |

저장 프로시저의 동작 방법을 알아보자.

GetCustomerShipping 저장 프로시저는 IN 매개변수로 customer number를 받아서, customer의 나라에 맞는 배송기간을 반환한다.

저장 프로시저 내부에서, 첫번째로 입력받은 customer number에 맞는 customer의 country를 얻는다. 그 후, 간단한 CASE문을 사용하여 customer 의 country에 맞는 배송 기간을 결정한다. 만약 USA에 살고 있는 customer 이라면 배송 기간은 2일이 걸린다. Canada에 customer가 있다면 배송 기간은 3일이 걸린다. 이외의 countries에 있는 customers 은 5일의 기간이 소요된다.

아래의 순서도는 배송기간을 결정하는 로직을 설명한다.



아래의 코드는 위의 저장 프로시저의 테스트 스크립트 이다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | SET @customerNo = 112;    SELECT country INTO @country  FROM customers  WHERE customernumber = @customerNo;    CALL GetCustomerShipping(@customerNo,@shipping);    SELECT @customerNo AS Customer,         @country    AS Country,         @shipping   AS Shipping; | [cs](http://colorscripter.com/info#e) |



**검색된(Searched) CASE 문**

간단한 CASE 문은 명확한 값을 가지는 집합에 대한 표현식의 값을 매칭하는것만 허용한다. 범위 같은 더 복잡한 매칭을 수행하려면, 검색된(Searched) CASE 문을 사용해야 한다. Searched CASE 문은 IF문과 비슷하다. 하지만 IF문에 비해 더 가독성이 좋다.

아래는 Searched CASE 문의 구조를 설명하고 있다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6 | CASE      WHEN condition\_1 THEN commands      WHEN condition\_2 THEN commands      ...      ELSE commands  END CASE; | [cs](http://colorscripter.com/info#e) |

MySQL은 WHEN 절에 있는 조건의 값이 TRUE인 것을 찾아 해당하는 절의 commands 를 실행합니다.

만약 모든 조건이 TRUE가 아닐 경우 ELSE절에 있는 commands 를 실행하게 된다. 만약 특별히 ELSE 절을 작성하지 않고, 일치하는 조건이 없을 경우에는, MySQL은 에러 메세지를 출력하게 된다.

MySQL은 THEN절이나 ELSE절에 빈 명령줄을 가지는 것을 허용하지 않는다. 만약 MySQL이 에러를 출력하지 않게 하고, ELSE 절에 있는 로직을 처리하고 싶지 않는 것을 원한다면, ELSE 절에 빈 BEGIN END 문맥을 작성하면 된다.

아래 예제 소비자들의 신용한도에 맞는 소비자 레벨 (SILVER, GOLD, PLATINUM)을 찾을 때 Searched CASE 문을 이용하는것을 설명하고 있다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | DELIMITER $$    CREATE PROCEDURE GetCustomerLevel(   IN  p\_customerNumber int(11),   OUT p\_customerLevel  varchar(10))  BEGIN      DECLARE creditlim double;        SELECT creditlimit INTO creditlim    FROM customers    WHERE customerNumber = p\_customerNumber;        CASE    WHEN creditlim > 50000 THEN       SET p\_customerLevel = 'PLATINUM';   WHEN (creditlim <= 50000 AND creditlim >= 10000) THEN       SET p\_customerLevel = 'GOLD';    WHEN creditlim < 10000 THEN       SET p\_customerLevel = 'SILVER';   END CASE;    END$$    DELIMITER ;  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

신용한도가 50000 이상이면 소비자는 PLATINUM 회원이다.

10000 이상 50000 미만의 소비자는 GOLD 회원이다.

10000 미만 소비자는 SILVER 회원이다.

아래의 테스트 스크립트를 실행해서 저장 프로시저를 테스트 할 수 있다.

|  |  |  |
| --- | --- | --- |
| 1  2 | CALL GetCustomerLevel(112, @level);  SELECT @level AS 'Customer Level'; | [cs](http://colorscripter.com/info#e) |



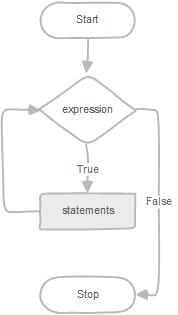
**7. WHILE 반복**

WHILE문의 구조는 아래와 같다:

|  |  |  |
| --- | --- | --- |
| 1  2  3 | WHILE expression DO     statements  END WHILE | [cs](http://colorscripter.com/info#e) |

  WHILE문은 각 반복의 처음에 표현식(expression)을 체크한다. 만약 표현식의 값이 TURE이면 MySQL은 WHILE 문과 END WHILE 문 사이에 있는 식(statement)을 표현식의 값이 FALSE가 될때까지 실행할 것이다. WHILE문은 사전 검사라고 불리운다. 왜냐하면 WHILE문은 식(statement)를 실행하기 전에 표현식(expression)을 검사하기 때문이다.

아래의 흐름도(Flowchart)는 WHILE문 의 동작원리를 설명하고 있다.



아래는 저장 프로시저에서 WHILE문을 사용하는 예시를 보여준다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | DELIMITER $$   DROP PROCEDURE IF EXISTS test\_mysql\_while\_loop$$   CREATE PROCEDURE test\_mysql\_while\_loop()   BEGIN   DECLARE x INT;   DECLARE str VARCHAR(255);     SET x = 1;   SET str =  '';     WHILE x  <= 5 DO   SET  str = CONCAT(str,x,',');   SET  x = x + 1;   END WHILE;     SELECT str;   END$$  DELIMITER ;  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

위의 test\_mysql\_while\_loop 저장 프로시저에서는:

첫 번째로, x변수의 값이 5보다 클 때까지 str 문자열을 반복적으로 추가한다. 그 후, SELECT문을 사용해서 최종 문자열을 표시한다.

  만약 x변수를 초기화 하지 않는다면, x변수의 기본값은 NULL이 된다는 것을 알고 있어야 한다. 그러므로, WHILE문의 조건은 항상 TRUE가 된다. 그리고 무한 반복에 빠지게 된다.

test\_mysql\_while\_loop 저장 프로시저를 테스트 해봅시다.

|  |  |  |
| --- | --- | --- |
| 1 | CALL test\_mysql\_while\_loop(); | [cs](http://colorscripter.com/info#e) |



**REPEAT 문**

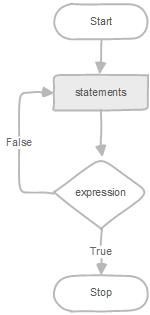
REPEAT 문법의 구조는 아래와 같다:

|  |  |  |
| --- | --- | --- |
| 1  2  3  4 | REPEAT   statements;  UNTIL expression  END REPEAT | [cs](http://colorscripter.com/info#e) |

첫 번째로, MySQL은 식(statement)를 실행한고, 그 후에 표현식(expression)을 확인한다. 만약 표현식의 값이 FALSE일 경우에 MySQL은 표현식의 결과가 TRUE가 될때 까지 반복적으로 식(statement)를 실행한다.

왜냐하면 REPEAT문은 식(statement)을 실행한 후에 표현식(expression)을 확인하기 때문이다. 그래서, REPEAT문은 사후 테스트(post-test) 반복문으로 알려져 있다.

아래의 흐름도는 REPEAT 반복문을 설명하고 있다:



우리는 test\_mysql\_while\_loop 저장 프로시저를 재 작성할 것이다. 위의 WHILE 반복문 대신에 REPEAT 반복문을 사용한다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | DELIMITER $$   DROP PROCEDURE IF EXISTS test\_mysql\_repeat\_loop$$   CREATE PROCEDURE test\_mysql\_repeat\_loop()   BEGIN   DECLARE x INT;   DECLARE str VARCHAR(255);     SET x = 1;   SET str =  '';    REPEAT  SET str = CONCAT(str,x,',');  SET x = x + 1;  UNTIL x > 5  END REPEAT;     SELECT str;   END$$  DELIMITER ;  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

UNTIL 표현식(expression)에서는 세미콜론( ; )을 사용하지 않는것을 알고 있어야 한다.

|  |  |  |
| --- | --- | --- |
| 1 | CALL mysql\_test\_repeat\_loop(); | [cs](http://colorscripter.com/info#e) |

**LOOP, LEAVE, ITERATE 문**

LOOP를 제어할 수 있게 하는 2개의 문법이 있다:

- LEAVE 문은 조건을 만족할 때 까지 기다리지 않고 즉시 반복문을 빠져 나갈 수 있게 허용한다. LEAVE문은 PHP, C/C+, JAVA 와 같은 다른 언어에서 break 처럼 작동한다.

- ITERATE 문은 코드를 전부 실행 시키지 않고 새로운 반복을 시작하도록 한다. ITERATE 문은 PHP, C/C++, JAVA 에서 continue문과 비슷하게 작동한다.

MySQL은 또한 반복 상태의 코드 구간에서 코드를 조금 더 유용성 있게 하는 loop label을 제공한다.

아래는 LOOP 반복문을 사용한 예제이다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | CREATE PROCEDURE test\_mysql\_loop()   BEGIN    DECLARE x  INT;    DECLARE str  VARCHAR(255);      SET x = 1;    SET str =  '';      loop\_label: LOOP    IF x > 10 THEN    LEAVE  loop\_label;    END IF;      SET x = x + 1;    IF (x mod 2) THEN     ITERATE loop\_label;    ELSE     SET str = CONCAT(str,x,',');    END  IF;    END LOOP;      SELECT str;     END; | [cs](http://colorscripter.com/info#e) |

- 저장 프로시저는 짝수(2, 4, 6 등)의 문자열만 나열한다.

- 우리는 LOOP문을 실행하기 전에 loop\_label의 이름을 가진 loop label을 배치한다.

- 만약 x의 값이 10보다 크다면  LEAVE문에 의해서 종료 되어진다.

- 만약 x의 값이 홀수 라면, ITERATE문 아래의 모든 것을 무시하고 새로운 반복을 시작한다.

- 만약 x의 값이 짝수 라면, ELSE문의 구간에 짝수만 존재하는 문자열을 만들 것이다.

**7. MySQL Cursor**

저장 프로세저 내부에 있는 결과 집합들을 처리하는데 cursor를 사용할 수 있다. cursor는 query에 의해 반환된 rows의 set을 반복하고 그에 따라 각 row를 처리할 수 있다.

MySQL cursor는 Read Only, Non-Scrollable, Asensitive 이다.

- ReadOnly : cursor를 통해서 테이블에 있는 데이터를 업데이트 할 수 없다.

- Non-scrollable : SELECT문에 의해 결정된 순서로 rows를 가져 올 수 있다. 반대 순서로 rows를 가져 올 수 없다. 추가로, 결과 set에 대해 특정한 row를 건너 뛰거나 rows 생략할 수 없다.

- Asensitive : 두 종류의 cursor가 있다. asensitive cursor와 insensitive cursor가 있다. asensitive cursor는 실제 데이터를 가르킨다. 그에 비해 insensitive cursor는 데이터의 임시 복사본을 사용한다. asensitive cursor는 insensitive cursor보다 빠르게 수행한다. 왜냐하면 데이터의 임시 복사본을 만들지 않기 때문이다. 하지만 다른 connections으로 부터 데이터를 만드는 것 같은 변화가 생긴다면 asensitive cursor가 사용하고 있는 데이터에 영향을 받는다. 그러므로, 만약 asensitive cursor에서 사용하고 있는 데이터를 업데이트 하지 않는다면 안전하다. MySQL cursor는 asensitive 이다.

stored procedures, stored functions, triggers에서 MySQL cursor를 사용할 수 있다.

**MySQL cursor와 함께 작업하기**

첫번째로, DECLARE문을 사용해서 cursor를 선언할 수 있다.

|  |  |  |
| --- | --- | --- |
| 1 | DECLARE cursor\_name CURSOR FOR SELECT\_statement; | [cs](http://colorscripter.com/info#e) |

cursor는 어떤 변수를 선언한 후에 선언 해야 한다. 만약 변수를 선언하기 전에 cursor를 선언한다면 MySQL은 에러를 뿌릴 것이다. CURSOR는 항상 SELECT문과 함께 조합되어야 한다.

다음에는, OPEN문을 사용하여 cursor를 열 수 있다. OPEN문은 cursor의 결과 값으로 초기화 된다. 그러므로 결과 set으로 부터 rows를 가져오기 전에 OPEN문을 호출 해야 한다.

|  |  |  |
| --- | --- | --- |
| 1 | OPEN cursor\_name; | [cs](http://colorscripter.com/info#e) |

그리고, cursor에 의해 다음 행이 가르키는 것을 가져오고 그리고 결과 집합의 다음 행에 cursor를 옮기는데 FETCH 문을 사용한다.

|  |  |  |
| --- | --- | --- |
| 1 | FETCH cursor\_name INTO variables list; | [cs](http://colorscripter.com/info#e) |

그 후에, 다음 row을 가져오기 전에 이용 가능한 행이 있는지 확인 할 수 있다.

마지막으로, 아래처럼 관련 메모리를 해제하고 cursor를 회수하는데 CLOSE문을 사용할 수 있다.

|  |  |  |
| --- | --- | --- |
| 1 | CLOSE cursor\_name; | [cs](http://colorscripter.com/info#e) |

그 후에 cursor가 더 이상 사용되지 않는다면, cursor를 닫아야 한다.

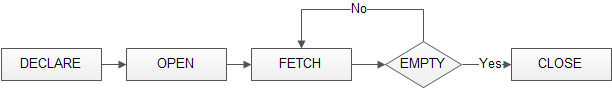
MySQL cursor로 일을 했을 때에는, cursor가 row를 찾지 못했을 때의 상황을 처리하기 위해 NOT FOUND 핸들러를 선언해야 한다. 왜냐하면 FETCH문을 호출 할 때 마다, cursor는 결과 set에 있는 다음 row를 읽기를 시도하기 때문이다. cursor가 결과 set의 끝에 도달 했을 때에는 data를 가져 올 수 없어 질 것이고 상태는 업데이트 되어진다. 핸들러는 이러한 상태를 처리하기 위해 사용되어 진다.

NOT FOUND 핸들러를 선언하기 위해서는 아래 구조로 사용할 수 있다.

|  |  |  |
| --- | --- | --- |
| 1 | DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1; | [cs](http://colorscripter.com/info#e) |

여기서 finished는 cursor가 결과 set의 끝데 도달 했다는 것을 나타내는 변수이다. 저장 프로시저 내부에 variable 와 cursor를 나타낸 후에 handler를 선언해야 한다는 것을 알고 있어야 한다.

아래 다이어그램은 MySQL cursor의 동작 원리를 나타내고 있다.



**MySQL Cursor 예제**

우리는 MySQL 샘플 데이터베이스에 있는 employees 테이블의 모든 고용인의 이메일 리스트를 구축하는 저장 프로시저를 개발 할 것이다.

첫번째로, 몇 가지 변수, employees의 이메일을 반복적으로 가져오는 커서, 그리고 NOT FOUND 핸들러를 선언한다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | DECLARE finished INTEGER DEFAULT 0;  DECLARE email varchar(255) DEFAULT "";    -- declare cursor for employee email  DEClARE email\_cursor CURSOR FOR   SELECT email FROM employees;    -- declare NOT FOUND handler  DECLARE CONTINUE HANDLER  FOR NOT FOUND SET finished = 1; | [cs](http://colorscripter.com/info#e) |

다음은 OPEN문을 이용해서 email\_cursor를 열 것이다.

|  |  |  |
| --- | --- | --- |
| 1 | OPEN email\_cursor; | [cs](http://colorscripter.com/info#e) |

그 후, email\_list를 반복해서 모든 이메일을 연결하고 각 이메일은 세미콜론(;)에 의해서 구분 되어 진다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8 | get\_email: LOOP   FETCH email\_cursor INTO v\_email;   IF v\_finished = 1 THEN   LEAVE get\_email;   END IF;   -- build email list   SET email\_list = CONCAT(v\_email,";",email\_list);  END LOOP get\_email; | [cs](http://colorscripter.com/info#e) |

그리고, 반복문 내부에서 반복을 종료하기 위해 리스트에 있는 이메일이 어떤 이메일인지 확인하기 위해 v\_finished변수를 사용한다.

마지막으로 CLOSE문을 사용하여 cursor를 닫게 된다.

|  |  |  |
| --- | --- | --- |
| 1 | CLOSE email\_cursor; | [cs](http://colorscripter.com/info#e) |

build\_email\_list stored procedure는 아래와 같다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | DELIMITER $$    CREATE PROCEDURE build\_email\_list (INOUT email\_list varchar(4000))  BEGIN     DECLARE v\_finished INTEGER DEFAULT 0;          DECLARE v\_email varchar(100) DEFAULT "";     -- declare cursor for employee email   DEClARE email\_cursor CURSOR FOR   SELECT email FROM employees;     -- declare NOT FOUND handler   DECLARE CONTINUE HANDLER          FOR NOT FOUND SET v\_finished = 1;     OPEN email\_cursor;     get\_email: LOOP     FETCH email\_cursor INTO v\_email;     IF v\_finished = 1 THEN   LEAVE get\_email;   END IF;     -- build email list   SET email\_list = CONCAT(v\_email,";",email\_list);     END LOOP get\_email;     CLOSE email\_cursor;    END$$    DELIMITER ;  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

아래 스크립트를 사용하여 build\_email\_list stored procedure를 테스트 할 수 있다.

|  |  |  |
| --- | --- | --- |
| 1  2  3 | SET @email\_list = "";  CALL build\_email\_list(@email\_list);  SELECT @email\_list; | [cs](http://colorscripter.com/info#e) |

이번 강좌에서는, MySQL cursor로 결과 set을 반복하고 그에 따라 각 row를 처리하는지에 대해서 배워 보았다.

**[사용법]**

https://202psj.tistory.com/1491



