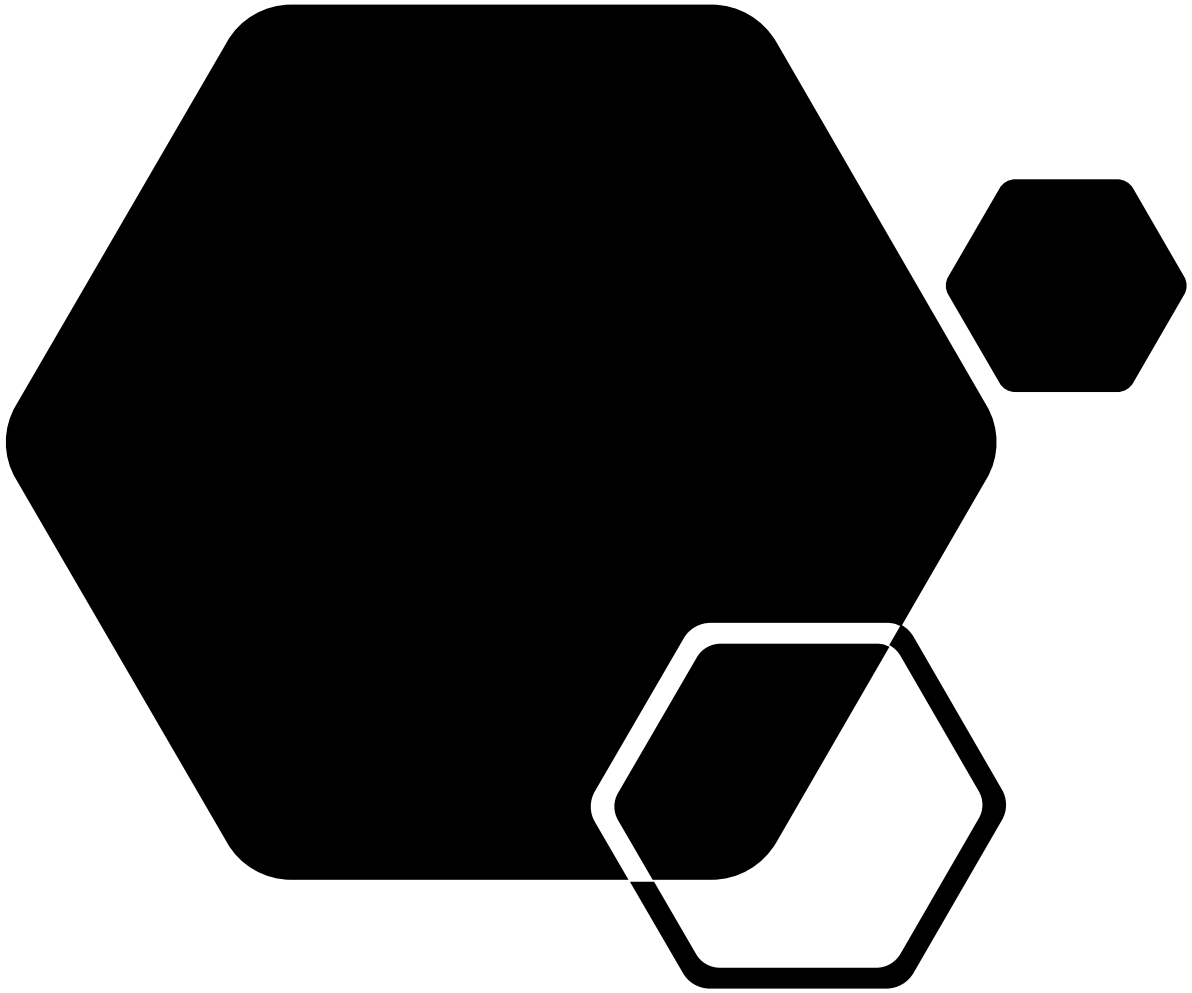
SQL 응용



SQL(Structured Query Language)

- 저장 프로시저(Stored Procedure)
  - 저장 프로시저는 일련의 쿼리를 마치 하나의 함수처럼 실행하기 위한 쿼리의 집합
    - 저장 프로시저는 데이터베이스에서 처리해야 하는 어떤 논리적 순서를 구성하고 그것을 하나의 명령어로 처리할 수 있게 한 것이며, 복잡한 처리 및 조회 등의 쿼리를 작성할 때 사용한다
    - 저장 프로시저를 만들기 위한 SQL문

```
CREATE PROCEDURE 저장프로시저_이름(IN 매개변수, OUT 매개변수)
BEGIN
        SQL문장들
END
```

SQL(Structured Query Language)

- 저장 프로시저(Stored Procedure)의 특징
  - SQL Server의 성능을 향상 시킬 수 있다.
    - 최적화, 컴파일 등의 과정을 거쳐서 그 결과가 캐시(메모리)에 저장
  - 유지 관리가 간편하다.
    - 응용프로그램에서 직접 SQL 문을 작성하지 않고 저장 프로시저 이름만 호출하게 설정
  - 모듈식 프로그래밍이 가능하다.
    - 언제든지 실행이 가능고 저장 프로시저로 저장해 놓은 쿼리의 수정, 삭제 등의 관리가 수월
  - 보안을 강화할 수 있다.
    - 사용자별로 테이블에 접근 권한을 주지 않고 저장 프로시저에만 접근 권한을 줌으로써, 보안을 더 강화
      - GRANT EXECUTE ON PROCEDURE `procedure_name` TO 'user_id'@'host' ;
      - FLUSH PRIVILEGES;

SQL(Structured Query Language)

- 제어문을 사용하는 프로시저
  - 저장 프로그램의 제어문은 어떤 조건에서 어떤 코드가 실행되어야 하는지를 제어하기 위한 문법으로, 절차적 언어의 구성요소를 포함함

구문	의미	문법
DELIMITER	• 구문 종료 기호 설정	DELIMITER {기호}
BEGIN-END	• 프로그램 문을 블록화시킴 • 중첩 가능	BEGIN {SQL 문} END
IF-ELSE	• 조건의 검사 결과에 따라 문장을 선택적으로 수행	IF <조건> THEN {SQL 문} [ELSE {SQL 문}] END IF;
LOOP	• LEAVE 문을 만나기 전까지 LOOP을 반복	[label:] LOOP {SQL 문 | LEAVE [label] } END LOOP

WHILE	• 조건이 참일 경우 WHILE 문의 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	• 프로시저를 종료 • 상태값을 반환 가능	RETURN [<식>]

SQL(Structured Query Language)

- IF문 구문

```
IF 조건식 THEN
        SQL문
[ELSEIF 조건식 THEN
        SQL문]
[ELSE
        SQL문]
END IF;
```

```
DELIMITER $$
CREATE PROCEDURE GetGradeByCredit(
        IN sID CHAR(13),                --학생번호
        OUT nGrade TINYINT) --학년
BEGIN
        DECLARE nTotalCredit SMALLINT; --총 이수학점
        SELECT SUM(이수학점) INTO nTotalCredit FROM 전공 WHERE 학생번호 = sID;
        IF nTotalCredit >= 120 THEN
                SET nGrade= 4;
        ELSEIF (nTotalCredit >=80 AND nTotalCredit < 120) THEN
                SET nGrade= 3;
        ELSEIF (nTotalCredit >= 40 AND nTotalCredit <80) THEN
                SET nGrade=3:
        ELSE
                SET nGrade= 1;
        END IF;
END$$
DELIMITER;
```

```
CALL GetGradeByCredit('202026-590930', @grade);
SELECT @grade
```

SQL(Structured Query Language)

- CASE문 구문

CASE 변수
WHEN 비교변숫값1 THEN SQL문
[WHEN 비교변숫값2 THEN SQL문]
…
[ELSE SQL문]
END CASE;

```
DELIMITER $$
CREATE PROCEDURE GetOTPlaceByGrade(
        IN nGrade INT(4),                       - 학년
        OUT sOTplace VARCHAR(30))               - 장소
BEGIN
        CASE nGrade
                WHEN 1 THEN
                        SET sOTplace='춘천' ;
                WHEN 2 THEN
                        SET sOTplace="인천';
                WHEN 3 THEN
                        SET sOTplace='광주';
                ELSE
                        SET sOTplace='제주';
        END CASE;
END$$
DELIMITER;
```

CASE
WHEN 조건식1 THEN SQL문
[WHEN 조건식2 THEN SQL문]
…
[ELSE SQL문]
END CASE;

```
DELIMITER $$
CREATE PROCEDURE GetRoomSize(
        IN sClassCode CHAR(5),                  -과목코드
        OUT sClassSize VARCHAR(20))             - 강의실 규모
BEGIN
        DECLARE nClass Volumn INT;
        SELECT COUNT(*) INTO nClassVolumn
        FROM 수강
        WHERE 과목코드 = sClassCode;
        CASE
                WHEN nClassVolumn>4 THEN
                        SET ClassSize = '대강의실' :
                WHEN (nClassVolumn >= 2 AND
                        nClassVolumn <= 4) THEN
                        SET sClassSize = '중강의실';
                ELSE
                        SET sClassSize = '소강의실';
        END CASE;
END$$
DELIMITER;
```

SQL(Structured Query Language)

- WHILE문 구문

WHILE 조건식 DO
SQL문
END WHILE;

- REPEAT문 구문

REPEAT
SQL문
UNTIL 조건식
END REPEAT;

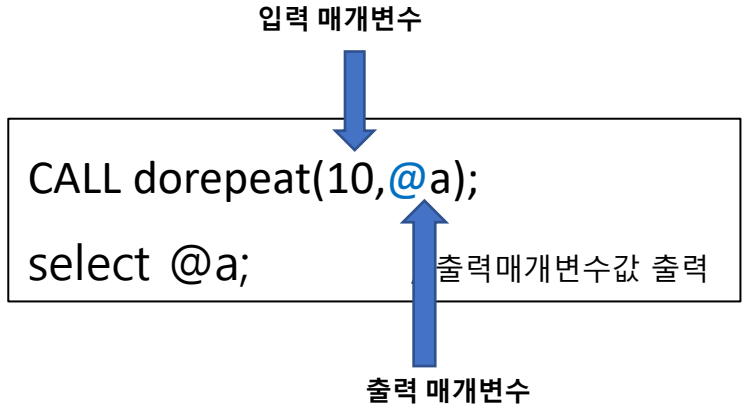
```
DELIMITER $$
 CREATE PROCEDURE test_mysql_while_loop()
 BEGIN
                DECLARE x INT;
                DECLARE str VARCHAR(255);
                SET x = 1;
                SET str =  '';
                WHILE x  <= 5 DO
                                SET  str = CONCAT(str,x,',');
                                SET  x = x + 1;
                END WHILE;
                SELECT str;
 END$$
DELIMITER ;
```

```
DELIMITER $$
 CREATE PROCEDURE test_mysql_repeat_loop()
 BEGIN
                DECLARE x INT;
                DECLARE str VARCHAR(255);
                SET x = 1;
                SET str =  '';
                REPEAT
                                SET  str = CONCAT(str,x,',');
                                SET  x = x + 1;
               UNTIL x > 5
                END REPEAT;
                SELECT str;
 END$$
DELIMITER ;
```

SQL(Structured Query Language)

- 저장 프로시저(Stored Procedure)
  - DB없는 저장 프로시저

```
/* 저장 프로시저 dorepeat가 있으면 삭제 */
drop procedure if exists dorepeat;
/* 마침기호 변경 */
delimiter //
CREATE PROCEDURE dorepeat(IN p1 INT, OUT y INT)
BEGIN
        declare x int;
        SET x=1;
        REPEAT
                SET x=x+1;
                SET y=x;
        UNTIL x>=p1
        END REPEAT;
END //
/* 마침기호 취소 */
delimiter;
```



SQL(Structured Query Language)

- 저장 프로시저(Stored Procedure)
  - DB있는 저장 프로시저

```
drop procedure if exists 새수강신청;
delimiter //
CREATE PROCEDURE 새수강신청(IN 학번 CHAR(7), OUT 수강신청_번호 INT)
BEGIN
        SELECT MAX(수강신청번호) INTO 수강신청_번호 FROM 수강신청;
        SET 수강신청_번호=수강신청_번호+1;
        INSERT INTO 수강신청(수강신청번호,학번,날짜,연도, 학기)
        VALUES(수강신청_번호, 학번, CURDATE(), '2020', '2');
END//
delimiter;
```

```
CALL 새수강신청('1804004', @수강신청_번호);
SELECT @수강신청_번호;
```

SQL(Structured Query Language)

- 저장 프로시저(Stored Procedure) 실습(1)
  - <학과> 테이블에 새로운 레코드를 삽입하고 삽입한 레코드를 보여주는 '새학과' stored procedure를 만드시오. 아래 실행 결과는 이 프로시저를 이용하여 출력한 예이다.

```
CALL 새학과('08', '컴퓨터보안학과', '022-200-7000');
```

실행결과

학과번호	학과명	전화번호
08	컴퓨터보안학과	022-200-7000

SQL(Structured Query Language)

- 저장 프로시저(Stored Procedure) 실습(2)
  - "수강신청" 데이터베이스의 총 학생 수, 교수 수, 과목 수를 계산하는 "통계" stored procedure를 만드시오. 아래 실행 결과는 이 프로시저를 이용하여 출력한 예이다.

```
CALL 통계(@a, @b, @c);
SELECT @a AS 학생수, @b AS 교수수, @c AS 과목수:
```

실행결과

학생수	교수수	과목수
10	7	10

SQL(Structured Query Language)

- 저장 함수(Stored Function)
  - 사용자가 직접 함수를 만들어서 사용할 필요가 있는 경우

```
DELIMITER $$
CREATE FUNCTION Stored_Function_Name ( Parameter )
            RETURN Return_Type
BEGIN
            Coding ...
    RETURN Return_Value;
END $$
DELIMITER ;

SELECT Stored_Function_Name();
```

```
DELIMITER $$
CREATE FUNTION userFunc(value1 INT, value2 INT)
            RETURN INT
BEGIN
            RETURN value1 + value2;
END $$
DELIMITER ;

-- userFunc 함수 실행
SELECT userFunc(100, 200);
```

SQL(Structured Query Language)

- Stored Procedure와 Stored Function의 차이점
  - Fuction은 파라메터는 IN, OUT 등을 사용할 수 없으며 모두 입력 파라메터로 사용
  - Function은 반환할 값의 데이터 형식을 지정하고 본문에서 하나의 값을 반환
  - Procedure는 별도의 반환 구문이 없으며 필요에 따라 여러 개의 OUT 파라메터를 사용하여 반환 가능
  - Procedure는 CALL로 호출하고 Function은 SELECT 문장 안에서 호출
  - Function 문장안에서는 DML(Insert/Update/Delete) 문을 사용할 수 없다.
  - Procedure는 코딩을 서버(DB)에서 하고 Function은 Client에서 한다.
  - 처리 속도는 Procedure가 빠르다.
  - Procedure는 여러 SQL문이나 숫자 계산 등의 다양한 용도로 사용되지만, Function은 어떤 계산을 통해서 하나의 값을 반환하는데 주로 사용된다.

SQL(Structured Query Language)

- 뷰(View)
  - 기존의 테이블들을 기반으로 만들어진 가상 테이블
  - 뷰는 데이터를 저장하지 않으며 쿼리 만을 저장하고 있어 뷰를 실행시킬 때마다 쿼리를 실행해서 동적으로 데이터를 가져온다
  - DB 사용자는 기존 테이블과 같은 방법으로 뷰를 사용
  - 복잡한 SQL문을 간단하게 표현할 수 있다

```
CREATE VIEW 학생정보 AS
SELECT 학생.학번, 학생.이름, 학과 학과번호, 학과 학과명
FROM 학생, 학과
WHERE 학생.학과 = 학과 학과번호:
```

```
SELECT
FROM 학생정보;
```

```
SELECT
FROM 학생정보
 WHERE 학과번호='01;
```

SQL(Structured Query Language)

- 뷰(View) 실습(1)
  - 교수의 사번, 이름, 소속 학과번호, 소속 학과명을 필드로 갖는 <교수정보>view를 만드시오. 아래 실행결과는 이 뷰를 이용하여 컴퓨터정보학과 소속 교수의 정보를 출력한 예이다.

```
SELECT
FROM 교수정보
WHERE 학과번호 = '01';
```

실행결과

사번	이름	학과번호	학과명
1000001	김교수	01	컴퓨터정보학과
1000002	이교수	01	컴퓨터정보학과
1000003	박교수	01	컴퓨터정보학과
1000004	최교수	01	컴퓨터정보학과

SQL(Structured Query Language)

- 뷰(View) 실습(2)
  - 교수의 사번, 이름, 소속 학과명, 담당 과목명, 과목 학점을 필드로 갖는 <담당교과> view를 만드시오. 아래 실행결과는 이 뷰를 이용하여 김교수의 담당 교과 정보를 출력한 예이다.

```
SELECT
FROM 담당교과
WHERE 사번 = 1000001';
```

실행결과

사번	이름	학과명	과목명	학점
1000001	김교수	컴퓨터정보학과	컴퓨터네트워크	2
1000001	김교수	컴퓨터정보학과	정보보호개론	4
1000001	김교수	컴퓨터정보학과	인터넷프로그래밍	3

SQL(Structured Query Language)

- 뷰(View) 실습(3)
  - 교수의 사번, 이름, 소속 학과명, 담당과목 수, 담당 총 학점 수를 필드로 갖는 <교수별담당과목> view를 만드시오. 아래 실행결과는 이 뷰를 이용하여 김교수의 담당 교과 정보를 출력한 예이다.

```
SELECT
FROM 교수별담당과목
WHERE 사번 = '1000001';
```

실행결과

사번	이름	학과명	과목수	학점수
1000001	김교수	컴퓨터정보학과	3	9

SQL(Structured Query Language)

- 트리거(Trigger)
  - 데이터 변경 등 명세된 이벤트 발생시 감지하여 자동 실행되는 사용자 정의 프로시저
  - INSERT, UPDATE, DELETE 명령문의 실행 직전·후 자동으로 호출되어 실행
  - 보통 무결성 제약 조건을 유지하거나 업무 규칙 등을 적용하기 위해 사용
  - TRIGGER 명령문의 형식

```
CREATE TRIGGER 트리거_이름
[ ❶BEFORE | ❷AFTER ][ INSERT|UPDATE|DELETE ] ON 테이블이름 FOR EACH ROW

BEGIN
...
 SQL 명령문 ;
...
END
```

```
DROP TRIGGER 트리거_이름 ;
```

SQL(Structured Query Language)

- 트리거 실습
- 문제1) 입고테이블에 INSERT 트리거를 작성한다.
  - [입고] 테이블에 자료가 추가 되는 경우 [상품] 테이블의 재고수량이 되도록 트리거를 작성한다.
- 문제2) 입고 테이블에 UPDATE 트리거를 작성 한다.
  - [입고] 테이블의 자료가 변경 되는 경우 [상품] 테이블의 [재고수량]이 변경 되도록 트리거를 작성한다.
- 문제3) 입고 테이블에 DELETE 트리거를 작성 한다.
  - [입고] 테이블의 자료가 삭제되는 경우 [상품] 테이블의 [재고수량]이 변경 되도록 트리거를 작성한다.