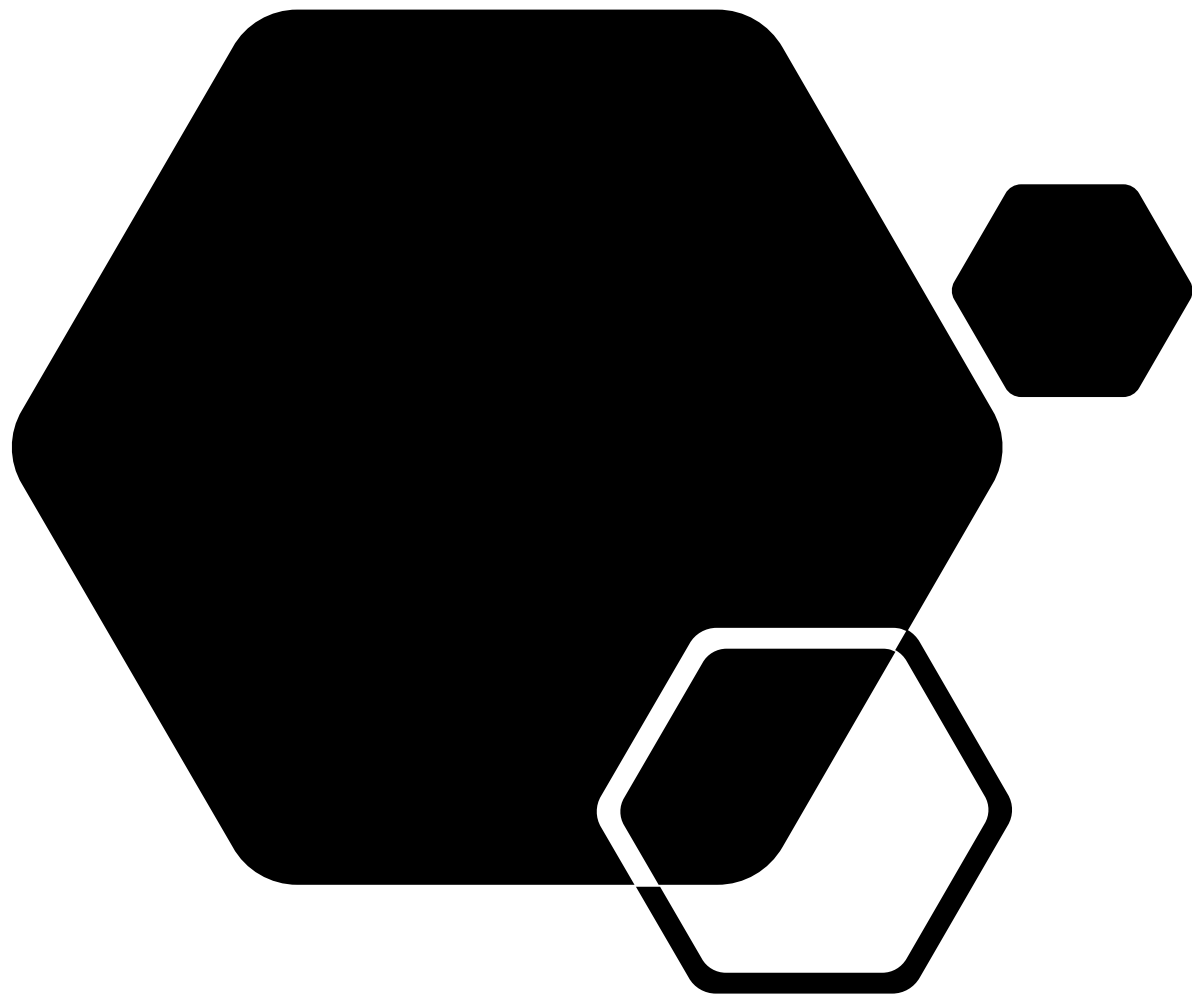


SQL 응용



# 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 가능하고 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<br>• 중첩 가능   | BEGIN<br>{SQL 문}<br>END  |
| IF-ELSE   | • 조건의 검사 결과에 따라 문장을 선택적으로 수행 | IF <조건> THEN {SQL 문}<br>[ELSE {SQL 문}]<br>END IF;                                  |
| LOOP      | • LEAVE 문을 만나기 전까지 LOOP을 반복  | [label:] LOOP<br>{SQL 문   LEAVE [label]}<br>END LOOP                               |
| WHILE     | • 조건이 참일 경우 WHILE 문의 블록을 실행  | WHILE <조건> DO<br>{SQL 문   BREAK   CONTINUE}<br>END WHILE                           |
| REPEAT    | • 조건이 참일 경우 REPEAT 문의 블록을 실행 | [label:] REPEAT<br>{SQL 문   BREAK   CONTINUE}<br>UNTIL <조건><br>END REPEAT [label:] |
| RETURN    | • 프로시저를 종료<br>• 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;**

매개변수 입력한 숫자까지 합을 출력

```
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
  - Function은 파라미터는 IN, OUT 등을 사용할 수 없으며 모두 입력 파라미터로 사용
  - Function은 반환할 값의 데이터 형식을 지정하고 본문에서 하나의 값을 반환
  - Procedure는 별도의 반환 구문이 없으며 필요에 따라 여러 개의 OUT 파라미터를 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 명시된 이벤트 발생시 감지하여 **자동 실행**되는 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  
[ ①BEFORE | ②AFTER ][ INSERT|UPDATE|DELETE ] ON 테이블이름 FOR EACH ROW  
  
BEGIN  
...  
SQL 명령문 ;  
...  
END
```

```
DROP TRIGGER 트리거_이름 ;
```

# SQL(Structured Query Language)

- 트리거 종류

| 트리거 이벤트 | 실행시점   | 기능                   |
|---------|--------|----------------------|
| INSERT  | BEFORE | 테이블에 데이터가 입력되기 전에 실행 |
|         | AFTER  | 테이블에 데이터가 입력된 후에 실행  |
| UPDATE  | BEFORE | 테이블의 데이터가 수정되기 전에 실행 |
|         | AFTER  | 테이블의 데이터가 수정된 후에 실행  |
| DELETE  | BEFORE | 테이블의 데이터가 삭제되기 전에 실행 |
|         | AFTER  | 테이블의 데이터가 삭제된 후에 실행  |

- BEGIN-END 블록 안에 있는 트리거의 SQL문에서는 OLD와 NEW라는 키워드가 사용

- 테이블에 어떠한 처리가 이루어지기 직전의 값과 직후의 값들을 저장하는 특별한 테이블이 **OLD** 테이블과 **NEW** 테이블

- OLD.열이름 : 처리 직전의 특정 열 값

- ```
insert into deleteDept values (old.dno, old.budget, curdate());
```

- NEW.열이름 : 처리 직후의 특정 열 값

- ```
if new.salary < 0 then set new.salary = 0;
```

# 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 재고수량이 변경 되도록 트리거를 작성한다.
  - 문제3) 입고 테이블에 DELETE 트리거를 작성 한다.
    - [입고] 테이블의 자료가 삭제되는 경우 [상품] 테이블의 재고수량이 변경 되도록 트리거를 작성한다.

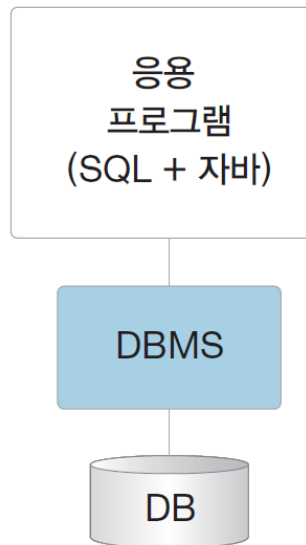
# SQL(Structured Query Language)

- 프로시저, 트리거, 사용자 정의 함수의 공통점과 차이점

| 구분     | 프로시저                          | 트리거  | 사용자 정의 함수                       |
|--------|-------------------------------|--|---------------------------------|
| 공통점    | 저장 프로시저                       |  |                                 |
| 정의 방법  | CREATE<br>PROCEDURE 문         | CREATE TRIGGER 문                                   | CREATE FUNCTION 문               |
| 호출 방법  | CALL 문으로 직접 호출                | INSERT, DELETE, UPDATE 문이<br>실행될 때 자동으로 실행됨        | SELECT 문에 포함                    |
| 기능의 차이 | SQL 문으로 할 수 없는 복<br>잡한 로직을 수행 | 기본값 제공, 데이터 제약 준수,<br>SQL 뷰의 수정, 참조무결성 작업<br>등을 수행 | 속성 값을 가공하여 반환,<br>SQL 문에서 직접 사용 |

# SQL(Structured Query Language)

- 데이터베이스 연동 자바 프로그래밍
- 데이터베이스 프로그래밍이란?
  - DBMS에 데이터를 정의하고 저장된 데이터를 읽어와 데이터를 변경하는 프로그램을 작성하는 과정
  - 일반 프로그래밍과는 데이터베이스 언어인 SQL을 포함한다는 점이 다름



- 일반 프로그래밍 언어에 SQL을 삽입하여 사용하는 방법
  - 자바, C, C++ 등 일반 프로그래밍 언어에 SQL 삽입하여 사용하는 방법.
  - 일반 프로그래밍 언어로 작성된 응용 프로그램에서 데이터베이스에 저장된 데이터를 관리, 검색.
  - 삽입된 SQL문은 DBMS의 컴파일러가 처리.

# SQL(Structured Query Language)

- 데이터베이스 연동 자바 프로그래밍
- 데이터베이스 연동 자바 프로그래밍 실습 환경

| 항목                    | 프로그램  |
|-----------------------|---|
| 데이터베이스 프로그램           | MySQL DBMS  |
| 자바 컴파일러               | JDK 버전 12   |
| 데이터베이스와 자바를 연결하는 드라이버 | MySQL JDBC 드라이버 Connector/J<br>(파일명:mysql-connector-java-8.x.jar) |

# SQL(Structured Query Language)

- 데이터베이스 연동 자바 프로그래밍
- 데이터베이스 접속 자바 클래스(java.sql)

| 클래스 구분    | 클래스 혹은 인터페이스  | 주요 메소드 이름   | 메소드 설명  |
|-----------|---------------|---|---|
| java.lang | Class         | Class. forName(〈클래스이름〉)                           | 〈클래스이름〉의 JDBC 드라이버를 로딩                            |
| java.sql  | DriverManager | Connection getConnection<br>(url, user, password) | 데이터베이스 Connection 객체를 생성                          |
|           | Connection    | Statement createStatement()                       | SQL 문을 실행하는 Statement 객체를 생성                      |
|           |               | void close()                                      | Connection 객체 연결을 종료                              |
|           | Statement     | ResultSet executeQuery<br>(String sql)            | SQL 문을 실행해서 ResultSet 객체를 생성                      |
|           |               | ResultSet executeUpdate<br>(String sql)           | INSERT/DELETE/UPDATE 문을 실행<br>해서 ResultSet 객체를 생성 |
|           | ResultSet     | boolean first()                                   | 결과 테이블에서 커서가 처음 튜플을<br>가리킴                        |
|           |               | boolean next()                                    | 결과 테이블에서 커서가 다음 튜플을<br>가리킴                        |
|           |               | int getInt(〈int〉)                                 | 〈int〉가 가리키는 열 값을 정수로 반환                           |
|           |               | String getString(〈int〉)                           | 〈int〉가 가리키는 열 값을 문자열로 반환                          |



# SQL(Structured Query Language)

- 데이터베이스 연동 자바 프로그래밍
  - 프로그램 실습 순서
    - [1단계] DBMS 설치 및 환경설정
    - [2단계] 데이터베이스 준비
    - [3단계(A)] 자바 실행 – (이클립스)를 이용하는 방법
      - JDBC 드라이버 설치
        - <https://www.mysql.com/products/connector/>
      - 자바프로그램 작성(Sqlconnection.java)
      - 컴파일 및 실행

# SQL(Structured Query Language)

- 데이터베이스 연동 자바 프로그래밍

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class Test {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        try {
            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/login_DB","root","1004");
            System.out.println("success");
            Statement stmt = conn.createStatement();
        }
        catch (SQLException ex) {
            System.out.println("SQLException" + ex);
            System.err.println("Got an exception!");
            System.err.println(e.getMessage());
        }
    }
}
```

# SQL(Structured Query Language)

- Using JDBC **CallableStatements** to Execute Stored Procedures

```
CREATE PROCEDURE `insert학과`(  
  IN 학과번호 CHAR(2),  
  IN 학과명 CHAR(20),  
  IN 전화번호 CHAR(20))  
  
BEGIN  
    INSERT INTO 학과 VALUES(학과번호, 학과명, 전화번호);  
END
```

```
import java.sql.CallableStatement;  
  
...  
  
CallableStatement cs = con.prepareCall("{ call insert학과(?, ?, ?) }");  
cs.setString(1, no);  
cs.setString(2, name);  
cs.setString(3, tel);  
cs.execute();
```

# JDBC 프로그래밍 실습

- 학생수강성적 릴레이션으로 데이터베이스 시스템을 구축하시오

| 학생번호 | 학생이름 | 주소       | 학과     | 학과사무실  | 강좌이름   | 강의실     | 성적  |
|------|------|----------|--------|--------|--------|---------|-----|
| 501  | 박지성  | 영국 맨체스타  | 컴퓨터공학과 | 공학관101 | 데이터베이스 | 공학관 110 | 3.5 |
| 401  | 김연아  | 대한민국 서울  | 체육학과   | 체육관101 | 데이터베이스 | 공학관 110 | 4.0 |
| 402  | 장미란  | 대한민국 강원도 | 체육학과   | 체육관101 | 스포츠경영학 | 체육관 103 | 3.5 |
| 502  | 추신수  | 미국 텍사스   | 컴퓨터공학과 | 공학관101 | 자료구조   | 공학관 111 | 4.0 |
| 501  | 박지성  | 영국 맨체스타  | 컴퓨터공학과 | 공학관101 | 자료구조   | 공학관 111 | 3.5 |
| 403  | 손흥민  | 영국 토트넘   | 로봇학과   | 과학관101 | 데이터베이스 | 공학관 110 | 4.0 |
| 403  | 손흥민  | 영국 토트넘   | 로봇학과   | 과학관101 | 자료구조   | 공학관 111 | 4.0 |
| 403  | 손흥민  | 영국 토트넘   | 로봇학과   | 과학관101 | 스포츠경영학 | 체육관 103 | 4.0 |

- 테이블을 생성하고, Insert, Select, Update, Delete가 가능하도록 프로그래밍 하세요

# SQL(Structured Query Language)

- AutoCommit
  - 현재 AutoCommit 값 확인
    - `SELECT @@AUTOCOMMIT;`
  - AutoCommit 설정
    - `SET AUTOCOMMIT = 1;`
  - AutoCommit 해제
    - `SET AUTOCOMMIT = 0;`
- MySQL Workbench 상단 메뉴
  - Edit - Preferences
  - SQL Editor > SQL Execution - General > Leave autocommit mode enabled by default 를  
체크 해제.

# SQL(Structured Query Language)

- commit/rollback 실습
  - 학과 테이블 select
  - '컴퓨터정보학과'를 '컴퓨터공학과'로 변경
  - 학과 테이블 select
  - rollback
  - '컴퓨터정보학과'를 '컴퓨터공학과'로 변경
  - commit
  - rollback

# SQL(Structured Query Language)

- savepoint / truncate 실습
  - savepoint는 변경된 지점의 위치를 저장
    - savepoint 지점;
  - 지점까지 복구
    - rollback to 지점
  - truncate table
    - truncate table 테이블명;
    - rollback으로 복구 안됨

```
Create Table new학과 ( select * from 학과 );  
  
SELECT @@AUTOCOMMIT;  
  
savepoint t1;  
delete from new학과 where 학과번호=01;  
  
savepoint t2;  
delete from new학과 where 학과번호=02;  
  
savepoint t3;  
delete from new학과 where 학과번호=03;  
  
rollback to t2;
```

## SQL(Structured Query Language)

- 데이터베이스에서 트랜잭션을 정의하는 이유
  - 데이터베이스에서 데이터를 다룰 때 장애가 일어날 때 데이터를 복구하는 작업의 단위가 됨
  - 데이터베이스에서 여러 작업이 동시에 같은 데이터를 다룰 때가 이 작업을 서로 분리하는 단위가 됨



# SQL(Structured Query Language)

- 락(LOCK)
  - 데이터의 일관성과 무결성을 보장하기 위해 트랜잭션과 함께 병행 사용되며, MySQL의 **InnoDB 엔진**에서는 행과 테이블 단위의 공유 락(READ LOCK)과 배타 락(WRITE LOCK)을 트랜잭션 고립성 수준에 따라 자동으로 수행한다.
  - 또한 사용자에게 의해서도 명시적으로 락이 사용될 수 있다.
  - MySQL 서버의 InnoDB는 명시적으로 락을 사용하지 않아도 설정된 고립성 수준에 부합하여 자동으로 락을 사용한다
  - 고립성이 강하게 보장될수록 락 대상이 늘어남에 따라 **동시성(concurrency)**은 떨어진다. 또한 강한 고립성을 보장하는 수준은 자칫 **교착상태(dead-lock)**를 불러올 수 있으므로 이를 감시하는 작업이 필요하다
  - 작업의 효율을 위해 기본 락 단위는 '레코드'이다. 그러나 사용자는 InnoDB나 다른 데이터베이스 엔진에 '**테이블**' 단위의 락을 명시적으로 걸 수도 있다

# SQL(Structured Query Language)

- 락(LOCK)의 종류
  - **READ LOCK:** READ 잠금을 획득한 트랜잭션은 테이블에 대한 읽기만 가능해진다. 복수의 트랜잭션에 의해 동시에 취득될 수 있다.
  - **WRITE LOCK:** WRITE 잠금을 획득한 트랜잭션은 테이블에 대한 읽기/쓰기가 가능해진다. 오직 하나의 트랜잭션만 해당 테이블에 대한 WRITE 잠금을 획득할 수 있다. 다른 트랜잭션들은 이 잠금이 해제될 때까지 테이블에 접근 불가능하다.

# SQL(Structured Query Language)

- 락(LOCK)
  - 데이터베이스를 백업하거나 테이블의 스키마 구조를 변경하거나 기타 중요한 작업을 진행할 때 다른 사람이 해당 테이블에 작업을 하지 못하도록 막기 위해 locking을 한다.
  - 트랜잭션이 지원되지 않는 테이블 타입인 **MyISAM**에는 트랜잭션과 비슷한 LOCK를 사용
  - 굳이 InnoDB 타입으로 트랜잭션을 사용하는 것보다 MyISAM 테이블 타입을 사용하되 LOCK TABLES 로 트랜잭션 기능을 구현하는 것이 더 효율적일 수 있다
    - show engines; -MySQL의 스토리지 엔진확인
    - InnoDB : 트랜잭션 처리가 필요하고 대용량의 데이터를 다루기에 효율적
    - MyISAM : 트랜잭션 처리가 필요없고 운영에 Read only 기능이 많은 서비스에 효율적

# SQL(Structured Query Language)

- 락(LOCK) 실습
  - 락 실습을 위해 서로 다른 두 명의 사용자가 MySQL 워크벤치를 사용하여 로컬 데이터베이스에 연결되어 있음을 가정한다
  - student1은 '계좌' 테이블에 WRITE 모드로 잠금을 획득하게 한 후, student2는 '계좌' 테이블에 READ 모드로 잠금을 획득하게 하시오
    1. student1 사용자: LOCK TABLE WRITE;
    2. student2 사용자: LOCK TABLE READ; 대기상태(running)
    3. student1 사용자: UNLOCK TABLES;
    4. student2에 의한 READ 잠금 획득

# SQL(Structured Query Language)

- 락(LOCK)
- 형식

```
SET autocommit=0;  
START TRANSACTION;  
LOCK TABLES 테이블이름 WRITE |READ;  
SQL문...  
COMMIT;  
UNLOCK TABLES;
```

- READ LOCK을 사용하면 다른 사용자가 해당 테이블을 읽기만 가능하고 쓰기를 할 때에는 LOCK이 걸린다
  - SELECT문은 실행할 수 있지만 INSERT, UPDATE, DELETE를 실행할 수 없다.
- WRITE LOCK을 사용하면 다른 사용자가 해당테이블에 대해 읽기와 쓰기를 할 때 모두 LOCK이 걸린다
  - 해당테이블에 대해 읽기와 쓰기를 할 때 모두 LOCK이 걸린다
- mysql>lock tables student read, professor write;
- mysql>unlock tables;

# SQL(Structured Query Language)

- 락(LOCK)
  - 트랜잭션의 읽기(read)/쓰기(write) 시나리오

|        | 트랜잭션1 | 트랜잭션2 | 발생 문제                      | 동시접근           |
|--------|-------|-------|----------------------------|----------------|
| [상황 1] | 읽기    | 읽기    | 읽음(읽기만 하면 아무 문제가 없음)       | 허용             |
| [상황 2] | 읽기    | 쓰기    | 오손 읽기, 반복불가능 읽기, 유령 데이터 읽기 | 허용 혹은 불가 선택    |
| [상황 3] | 쓰기    | 쓰기    | 갱신손실(절대 허용하면 안 됨)          | 허용불가(LOCK을 이용) |

- 갱신손실(lost update): 두 개의 트랜잭션이 한 개의 데이터를 동시에 갱신(update)할 때 발생하며, 데이터베이스에서 절대 발생하면 안 되는 현상

| 트랜잭션 T1                       | 트랜잭션 T2                       | 버퍼의 데이터 값 |
|-------------------------------|-------------------------------|-----------|
| A=read_item(X); ①<br>A=A-100; |                               | X=1000    |
|                               | B=read_item(X); ②<br>B=B+100; | X=1000    |
| write_item(A → X); ③          |                               | X=900     |
|                               | write_item(B → X); ④          | X=1100    |

| 트랜잭션 T1                                  | 트랜잭션 T2  | 버퍼의 데이터 값 |
|--|--|-----------|
| LOCK(X)<br>A=read_item(X); ①<br>A=A-100; |  | X=1000    |
|  | LOCK(X)<br>(wait... 대기)  | X=1000    |
| write_item(A→X); ②<br>UNLOCK(X);         |  | X=900     |
|  | B=read_item(X); ③<br>B=B+100;<br>write_item(B→X); ④<br>UNLOCK(X) | X=1000    |

데이터를 수정 중이라는 사실을 알리는 방법의 잠금 장치

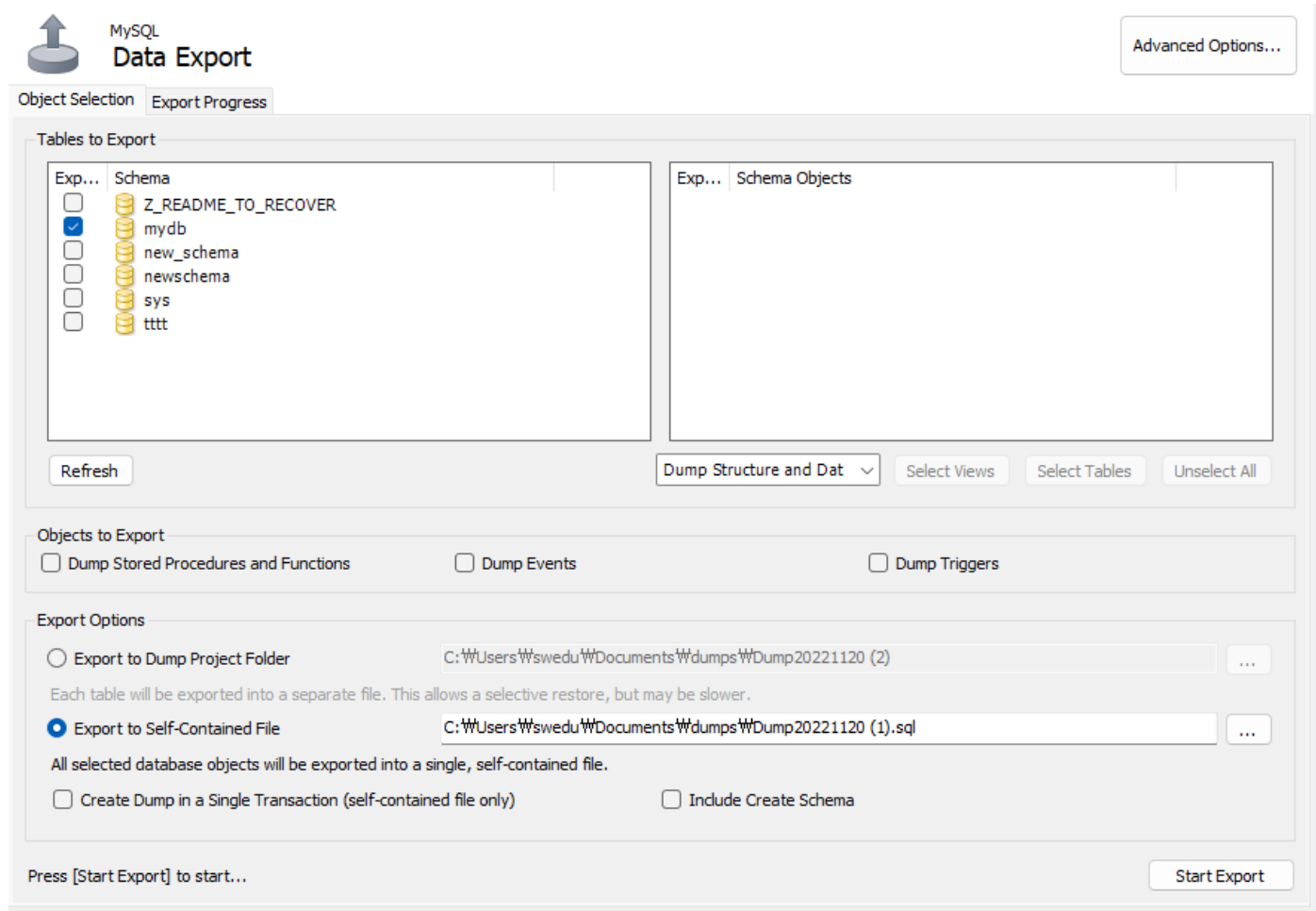
# SQL(Structured Query Language)

- MySQL에서 두 트랜잭션을 동시에 실행시키는 방법
  - 트랜잭션 실패를 진행하기 위해서는 MySQL 접속 시 서로 다른 두 세션(세션은 데이터베이스 접속단위)에서 진행해야 한다.
  - MySQL을 두 번 연결하기 위해서는 MySQL Workbench에서 접속을 2개 따로 만들어야 한다( 동일한 내용으로 2개의 접속을 만들고 접속)

| 트랜잭션 T1   | 트랜잭션 T2  |
|---|--|
| START TRANSACTION;<br>SELECT *<br>FROM Book<br>WHERE bookid=1;<br><br><b>UPDATE</b> Book<br>SET price=7100<br>WHERE bookid=1;<br><br>SELECT *<br>FROM Book<br>WHERE bookid=1;<br><br><b>COMMIT;</b> | START TRANSACTION;<br>SELECT *<br>FROM Book<br>WHERE bookid=1;<br><br><b>UPDATE</b> Book<br>SET price=price+100<br>WHERE bookid=1;<br><br>SELECT *<br>FROM Book<br>WHERE bookid=1;<br><br><b>COMMIT;</b> |

# SQL(Structured Query Language)

## ■ MySQL 백업과 복원 실습



The image shows the MySQL Data Export dialog box. The 'Object Selection' tab is active. Under 'Tables to Export', the 'mydb' schema is selected. The 'Export Options' section shows 'Export to Self-Contained File' selected. The 'Start Export' button is at the bottom right.

MySQL Data Export

Object Selection | Export Progress

Tables to Export

| Exp...                              | Schema              |
|-------------------------------------|---------------------|
| <input type="checkbox"/>            | Z_README_TO_RECOVER |
| <input checked="" type="checkbox"/> | mydb                |
| <input type="checkbox"/>            | new_schema          |
| <input type="checkbox"/>            | newschema           |
| <input type="checkbox"/>            | sys                 |
| <input type="checkbox"/>            | tttt                |

Refresh

Dump Structure and Dat | Select Views | Select Tables | Unselect All

Objects to Export

☐ Dump Stored Procedures and Functions ☐ Dump Events ☐ Dump Triggers

Export Options

☐ Export to Dump Project Folder C:\Users\swedu\Documents\dumps\Dump20221120 (2) ...

Each table will be exported into a separate file. This allows a selective restore, but may be slower.

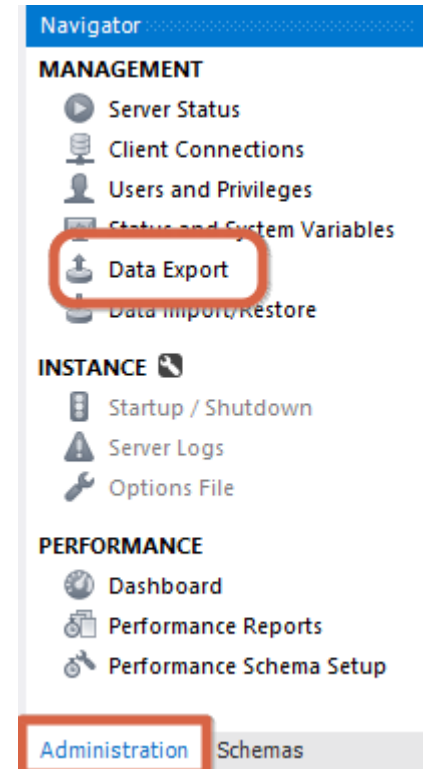
☒ Export to Self-Contained File C:\Users\swedu\Documents\dumps\Dump20221120 (1).sql ...

All selected database objects will be exported into a single, self-contained file.

☐ Create Dump in a Single Transaction (self-contained file only) ☐ Include Create Schema

Press [Start Export] to start...

Start Export



The image shows the MySQL Navigator sidebar. The 'Data Export' option is highlighted with a red box. The 'Administration' tab is also highlighted with a red box.

Navigator

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export**
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

PERFORMANCE


- Dashboard
- Performance Reports
- Performance Schema Setup

**Administration** | Schemas



# SQL(Structured Query Language)

- MySQL 백업과 복원 실습

 MySQL  
Data Import

Import from Disk Import Progress

Import Options

☐ Import from Dump Project Folder C:\Users\Wswedu\Documents\dumps ...

Select the Dump Project Folder to import. You can do a selective restore.

Load Folder Contents

☒ Import from Self-Contained File C:\Users\Wswedu\Documents\dumps\Dump20221120.sql ...

Select the SQL/dump file to import. Please note that the whole file will be imported.

Default Schema to be Imported To

Default Target Schema: mydb New...

The default schema to import the dump into.  
NOTE: this is only used if the dump file doesn't contain its schema,  
otherwise it is ignored.

Select Database Objects to Import (only available for Project Folders)







Imp... Schema

Imp... Schema Objects




Dump Structure and Dat Select Views Select Tables Unselect All

Press [Start Import] to start... Start Import




## MANAGEMENT

-  Server Status
-  Client Connections
-  Users and Privileges
-  Status and System Variables
-  Data Export
-  Data Import/Restore

## INSTANCE

-  Startup / Shutdown
-  Server Logs
-  Options File

## PERFORMANCE

-  Dashboard
-  Performance Reports
-  Performance Schema Setup

# SQL(Structured Query Language)

- **명령어**로 MySQL 백업과 복원 실습
  - mysqldump -u[ID이름] -p [DB이름] > [파일이름]
  - mysql -u root -p < [파일이름] -- 모든DB를 복구
  - mysql -u root -p DB명 < [파일이름] -- 특정 DB만 복구
  - mysql > source [파일이름] --dump받은 파일이 있는 위치로 이동한 후 mysql 에 접속