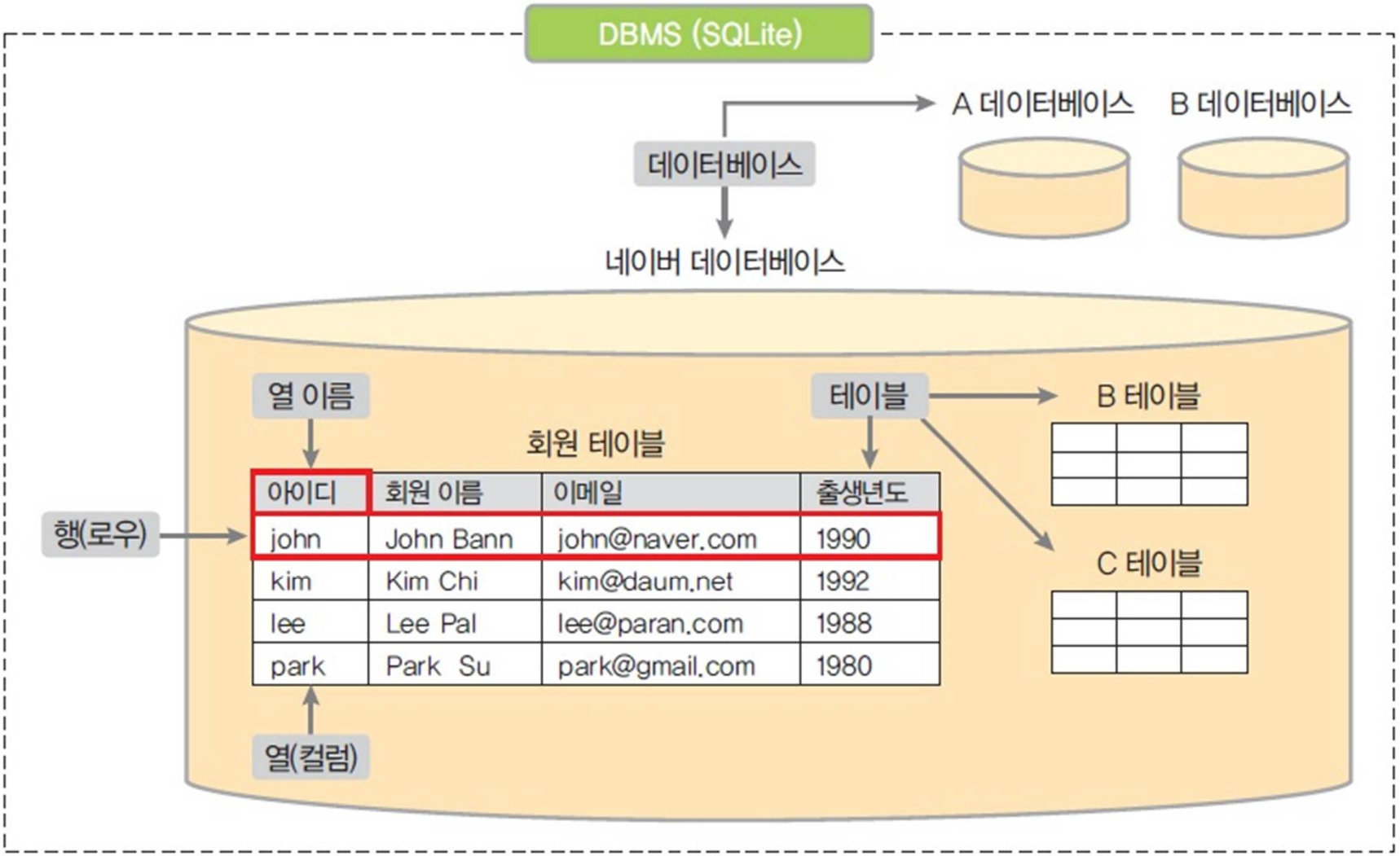
# 데이터베이스

- DBMS란?
- 데이터베이스란?
- 테이블이란?
- SQL = DML + DDL + DCL
- DML : CRUD2SG
  - C: INSERT
  - R: SELECT
  - U: UPDATE
  - D: DELETE
  - S: SELECT ~ WHERE
  - S: SELECT ~ ORDER BY
  - G: GROUP BY ~ HAVING

데이터베이스를 사용하는 방법을 학습합니다.

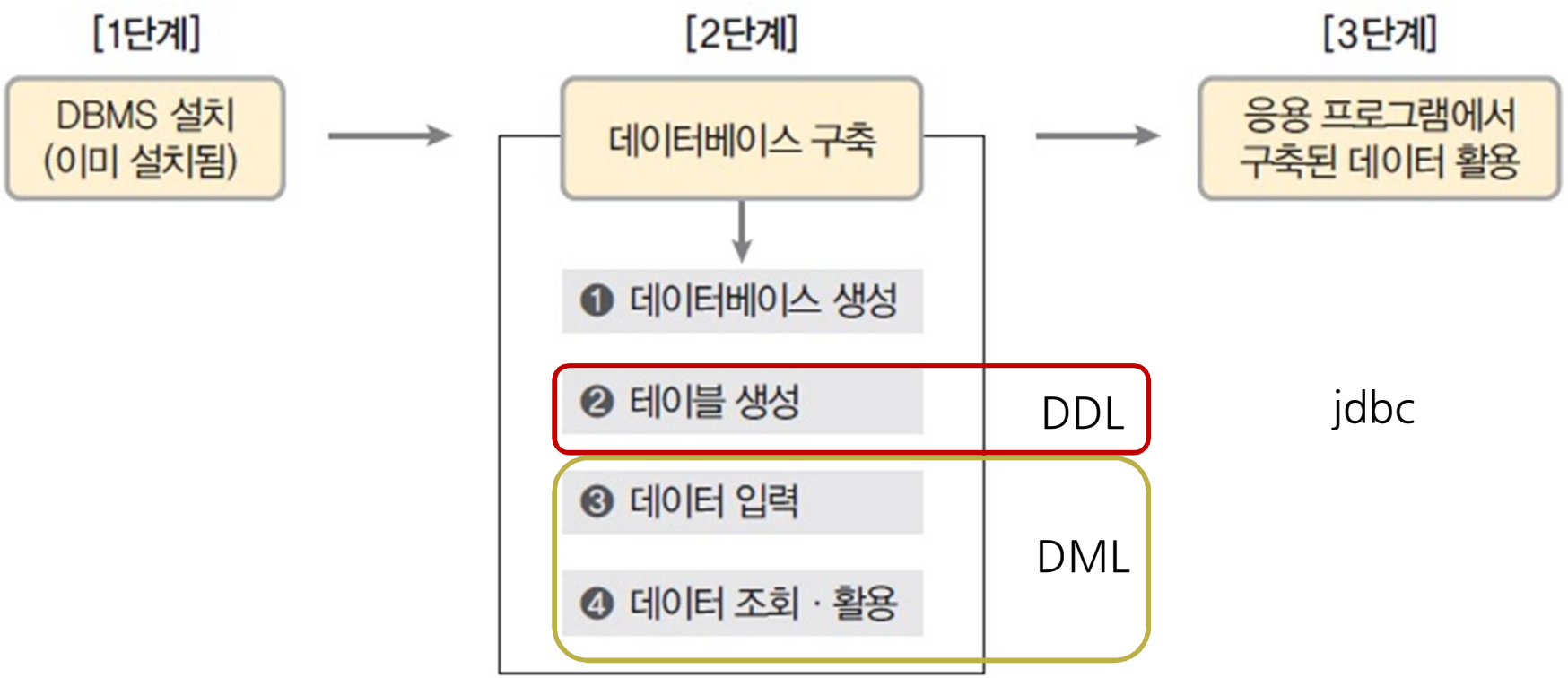
# 데이터베이스 기본 개념


DBMS (SQLite)
A 데이터베이스
B 데이터베이스
데이터베이스
네이버 데이터베이스
열 이름
테이블
B 테이블
회원 테이블
행(로우)
아이디
회원 이름
이메일
출생년도
john
John Bann
john@naver.com
1990
kim
Kim Chi
kim@daum.net
1992
lee
Lee Pal
lee@paran.com
1988
park
Park Su
park@gmail.com
1980
C 테이블
열(컬럼)

# 데이터베이스 기본 개념

- 데이터베이스 관련 용어
  - **데이터** : 하나하나의 단편적인 정보를 뜻함
  - **DBMS** : 데이터베이스를 관리하는 소프트웨어를 말함
    Oracle, MS-SQL, MySQL, SQLite 소프트웨어가 이에 해당
  - **데이터베이스**(DB) : 테이블이 저장되는 장소로 원통 모양으로 표현
    각 데이터베이스는 서로 다른 고유한 이름이 있어야 함
  - **테이블** : 데이터가 표 형태로 표현된 것
  - **열**(컬럼 또는 필드) : 각 테이블은 1개 이상의 열로 구성됨
  - 열 이름 : 각 열을 구분하는 이름, 열 이름은 각 테이블 안에서는 중복되지 않아야 함
  - 데이터 형식 : 열의 데이터 형식을 뜻함
    **숫자, 문자, 날짜**
    테이블을 생성할 때 열 이름과 함께 지정해줘야 함
  - **행**(로우) : 실제 데이터를 뜻함
  - SQL : 사용자와 DBMS와 소통하기 위해서 사용하는 언어
    SQL = **DML(CRUD2SG)** + DDL + DCL

# 데이터베이스 구축


[1단계]
DBMS 설치
(이미 설치됨)
[2단계]
데이터베이스 구축
❶ 데이터베이스 생성
❷ 테이블 생성
❸ 데이터 입력
❹ 데이터 조회 · 활용
DDL
DML
[3단계]
응용 프로그램에서
구축된 데이터 활용
jdbc

# Introduction

- DBMS
  - Database management system
  - Storing and organizing data
- Database
  - Collection of data
- Table
- Column
- Row, Record
- SQL
  - Relational database
  - Structured Query Language
- JDBC
  - Java Database Connectivity
  - JDBC driver

# SQL이란?

- SQL = Structured Query Language
- 데이터베이스에서 사용하기 위하여 설계된 언어
- DDL + DML + DCL
  - DDL : Data Definition Language : DB, 테이블 조작
  - DML : Data Manipulation Language : 테이블에서 데이터 조작
  - DCL : Data Control Language : DB 제어

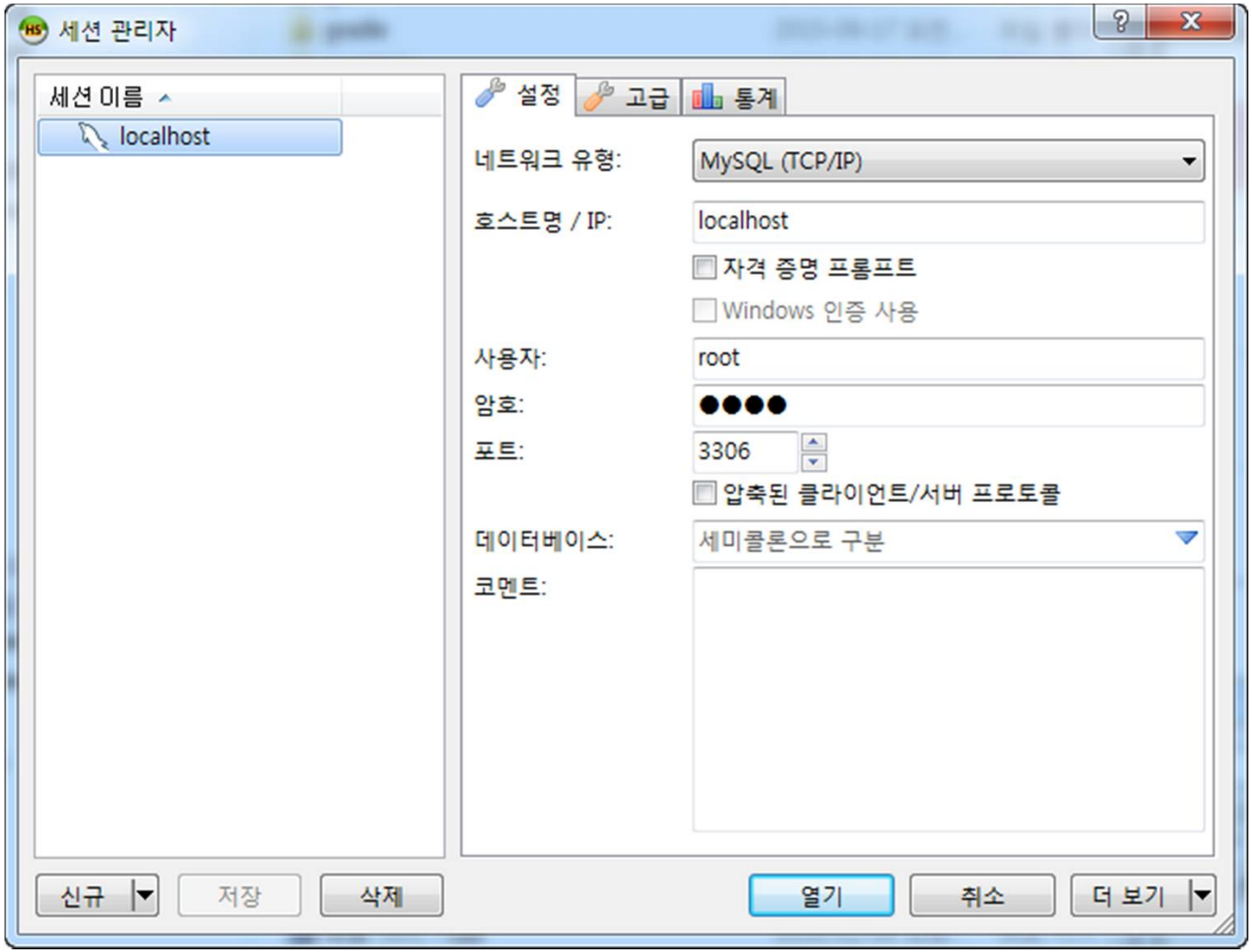
# DDL이란?

데이터베이스나 테이블을 생성하거나 수정하는 데 사용되는 언어.

- 생성(CREATE),
- 변경(ALTER),
- 제거(DROP)
- 권한 부여(GRANT)
- 권한 박탈(REVOKE)

| 구분 | 명령어 | 설명 |
|---|---|---|
| 데이터 정의 명령어 (Data Definition Language) | CREATE | 사용자가 제공하는 컬럼 이름을 가지고 테이블을 생성한다. 사용자는 컬럼의 데이터 타입도 지정하여야 한다. 데이터 타입은 데이터베이스에 따라 달라진다. CREATE TABLE은 보통 DML보다 적게 사용된다. 왜냐하면 이미 테이블이 만들어져 있는 경우가 많기 때문이다. |
| | ALTER | 테이블에서 컬럼을 추가하거나 삭제한다. |
| | DROP | 데이블의 모든 레코드를 제거하고 테이블의 정의 자체를 데이터베이스로부터 삭제하는 명령어이다. |
| | USE | 어떤 데이터베이스를 사용하는지를 지정 |

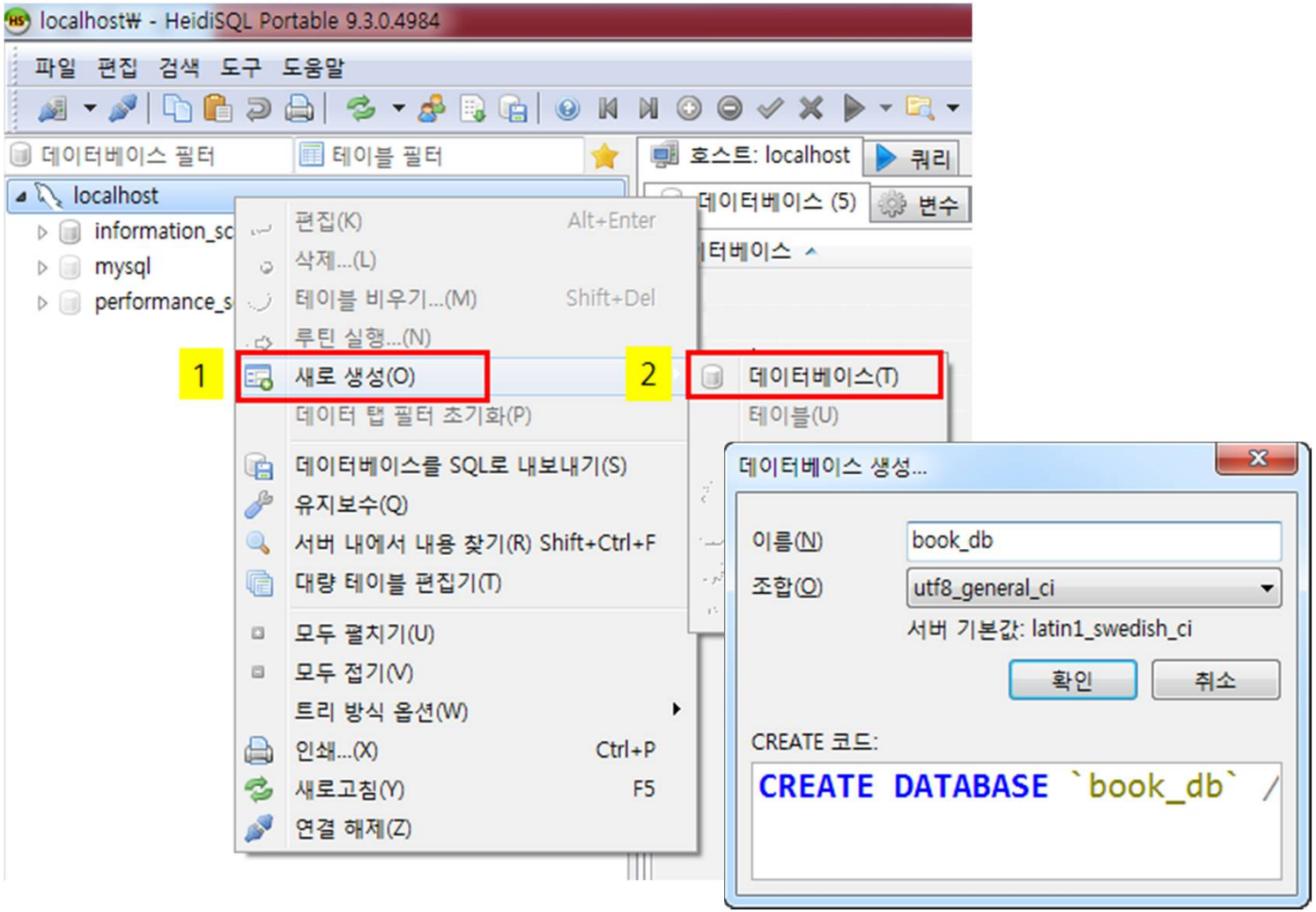
# 데이터베이스 접속하기


세션 관리자
세션 이름
localhost
설정
고급
통계
네트워크 유형:
MySQL (TCP/IP)
호스트명 / IP:
localhost
자격 증명 프롬프트
Windows 인증 사용
사용자:
root
암호:
●●●●
포트:
3306
압축된 클라이언트/서버 프로토콜
데이터베이스:
세미콜론으로 구분
코멘트:
신규
저장
삭제
열기
취소
더 보기

# 데이터베이스 생성하기

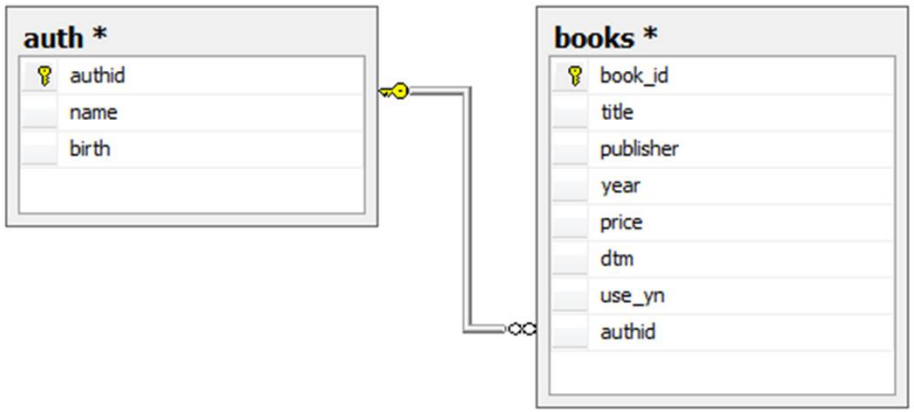
```
-- 데이터베이스 삭제
DROP DATABASE  IF  EXISTS book_db;


-- 데이터베이스 생성
CREATE  DATABASE  book_db  COLLATE  'utf8_general_ci';


-- 데이터베이스 사용
USE book_db;
```

# 데이터베이스 생성하기

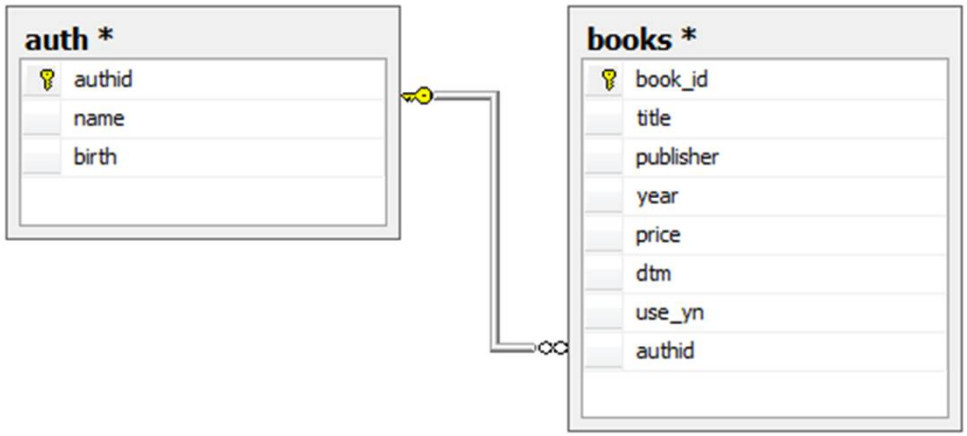

localhost₩ - HeidiSQL Portable 9.3.0.4984
파일 편집 검색 도구 도움말
데이터베이스 필터
테이블 필터
호스트: localhost
쿼리
데이터베이스 (5)
변수
localhost
information_sc
mysql
performance_s
편집(K) Alt+Enter
삭제...(L)
테이블 비우기...(M) Shift+Del
루틴 실행...(N)
1
새로 생성(O)
데이터 탭 필터 초기화(P)
데이터베이스를 SQL로 내보내기(S)
유지보수(Q)
서버 내에서 내용 찾기(R) Shift+Ctrl+F
대량 테이블 편집기(T)
모두 펼치기(U)
모두 접기(V)
트리 방식 옵션(W)
인쇄...(X) Ctrl+P
새로고침(Y) F5
연결 해제(Z)
2
데이터베이스(T)
테이블(U)
데이터베이스 생성...
이름(N) book_db
조합(O) utf8_general_ci
서버 기본값: latin1_swedish_ci
확인
취소
CREATE 코드:
CREATE DATABASE `book_db` /

# 테이블 생성하기

```
-- 테이블 생성
DROP  TABLE  IF  EXISTS book;
CREATE  TABLE  book (
      bookid        INT  NOT NULL auto_increment
    , bookname          VARCHAR(50)
    , publisher      VARCHAR(30)
    , year           VARCHAR(10)
    , price          INT
    , dtm            DATE
    , use_yn         BIT
    , authid         INT  NOT NULL

    , PRIMARY KEY(bookid)
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB  ;
```

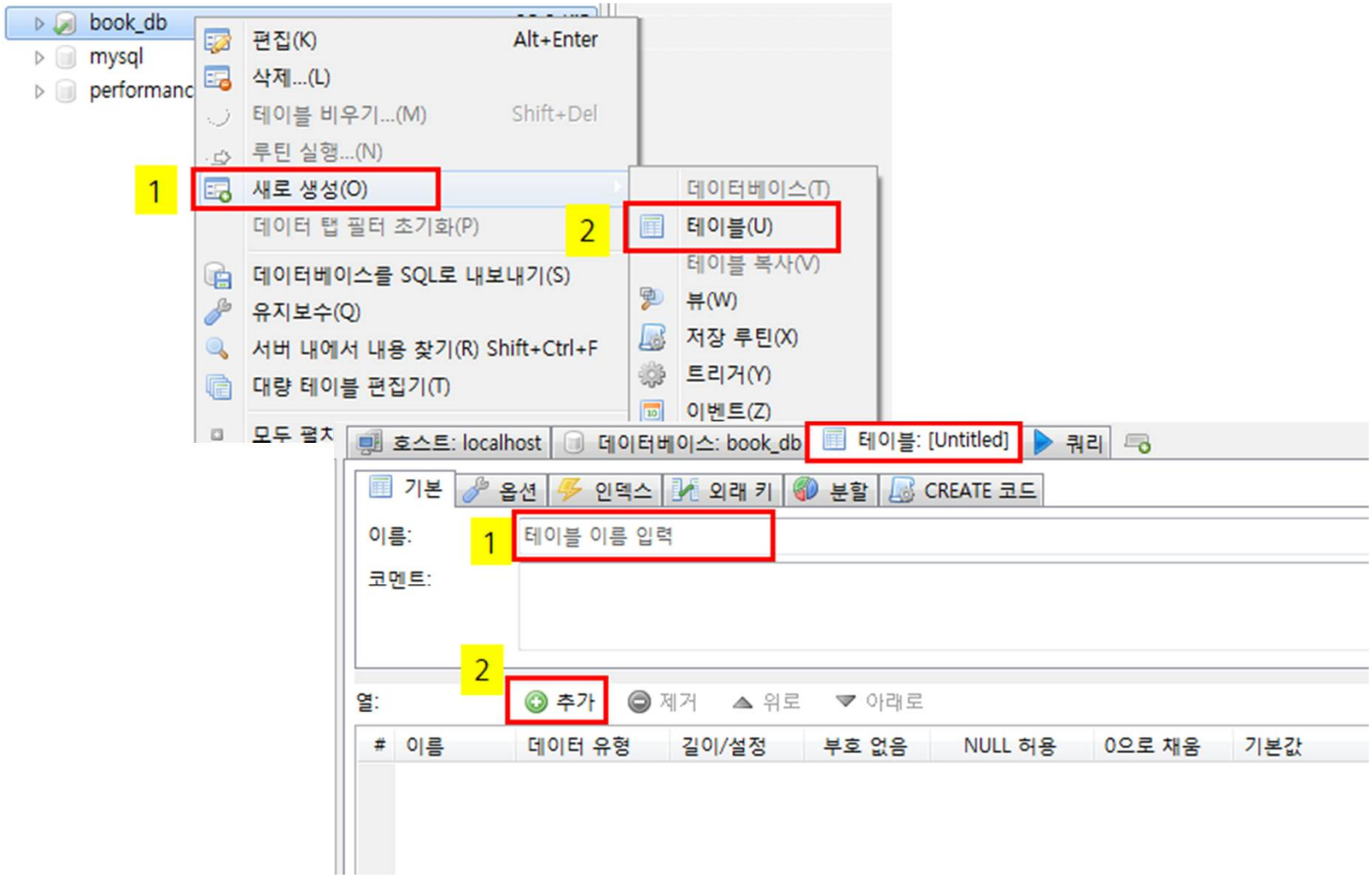


# 테이블 생성하기

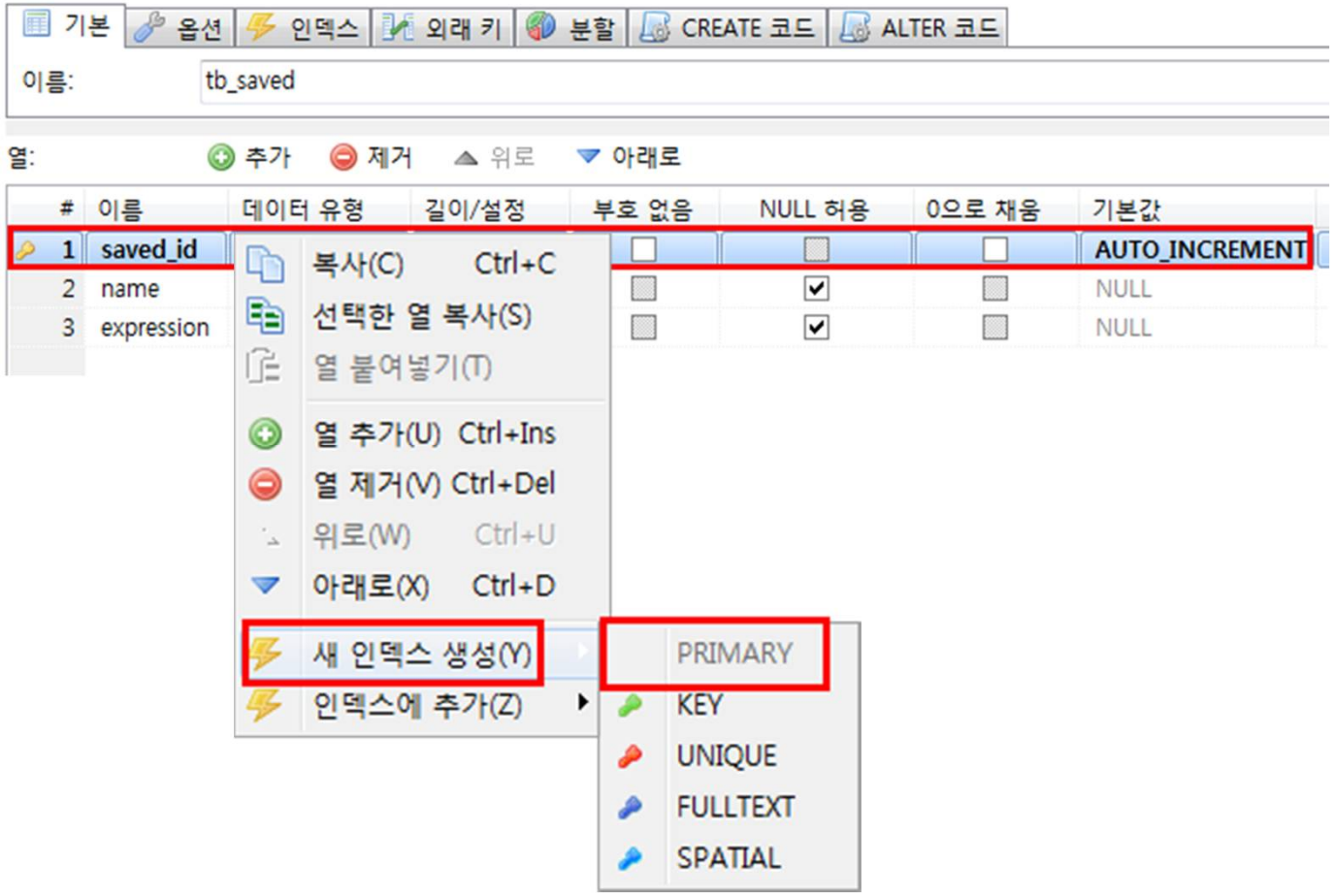



```
-- 테이블 생성
DROP TABLE IF EXISTS auth;

CREATE TABLE auth (
     authid     INT  NOT  NULL  auto_increment
   , name         VARCHAR(50)
   , birth      VARCHAR(10)

   , PRIMARY KEY(authid)
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB  ;
```

# 테이블 생성하기


book_db
mysql
performanc
편집(K) Alt+Enter
삭제...(L)
테이블 비우기...(M) Shift+Del
루틴 실행...(N)
1
새로 생성(O)
데이터 탭 필터 초기화(P)
데이터베이스를 SQL로 내보내기(S)
유지보수(Q)
서버 내에서 내용 찾기(R) Shift+Ctrl+F
대량 테이블 편집기(T)
데이터베이스(T)
2
테이블(U)
테이블 복사(V)
뷰(W)
저장 루틴(X)
트리거(Y)
이벤트(Z)
호스트: localhost
데이터베이스: book_db
테이블: [Untitled]
쿼리
기본
옵션
인덱스
외래 키
분할
CREATE 코드
이름:
1
테이블 이름 입력
코멘트:
2
열:
추가
제거
위로
아래로
# 이름 데이터 유형 길이/설정 부호 없음 NULL 허용 0으로 채움 기본값

# 테이블 Primary Key 생성하기



# DML이란?

데이터를 실질적으로 처리하는 데 사용되는 언어입니다. (SUDI)

- SELECT(검색)
- UPDATE(수정)
- INSERT(삽입)
- DELETE(삭제)

- COMMIT(트랜잭션)
- ROLLBACK(트랜잭션)

| 구분 | 명령어 | 설명 |
|---|---|---|
| 데이터 조작 명령어 (Data Manipulation Language) | SELECT | 데이터베이스로부터 데이터를 쿼리하고 출력한다. SELECT 명령어들은 결과 집합에 포함시킬 컬럼을 지정한다. SQL 명령어 중에서 가장 자주 사용된다. |
| | INSERT | 새로운 레코드를 테이블에 추가한다. INSERT는 새롭게 생성된 테이블을 채우거나 새로운 레코들를 이미 존재하는 테이블에 추가할 때 사용된다. |
| | DELETE | 지정된 레코드를 테이블로부터 삭제한다. |
| | UPDATE | 테이블에서 레코드에 존재하는 값을 변경한다. |

# 레코드 검색하기

```
SELECT * FROM book;

SELECT * FROM book  WHERE bookname like '%SQL%';

SELECT * FROM book  WHERE bookname = 'JAVA';

SELECT * FROM book  WHERE 30700 <= price and price <50000;

SELECT * FROM book  WHERE price BETWEEN 30700 AND 50000;

SELECT * FROM book  WHERE  price <= 30700 OR 58000 < price ;

SELECT bookname,  publisher,  price,  authid  FROM book;
```

# 레코드 정렬하기

```
SELECT * FROM book ORDER BY price ASC;

SELECT * FROM book ORDER BY price DESC;

SELECT * FROM book ORDER BY publisher ASC, price DESC;
```

# 레코드 추가하기

INSERT 테이블명( 컬럼명1, 컬럼명2, 컬럼명3, .. , 컬럼명N )
VALUES( 값1 , 값2, 값3, .. , 값N ) ;

```
INSERT INTO book (bookname   , publisher, year  , price, authid )
          VALUES('System', 'Wiley'  , '2003', 30700, 1      ) ;

INSERT INTO auth  (authid, name  , birth        )
          VALUES(1     , 'bob' , '1970.05.01' ) ;
```

# 레코드 수정하기

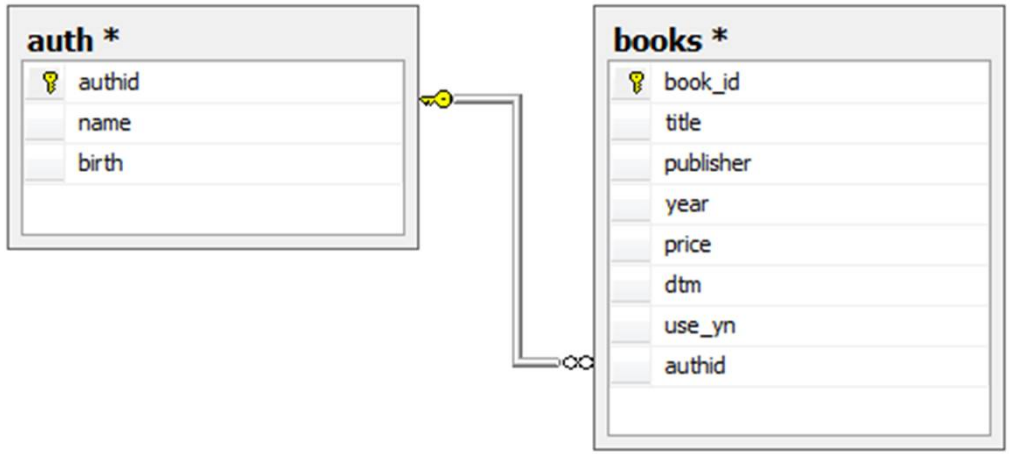
```
/* UPDATE 테이블명
        SET 컬럼명1=값1
          , ..
          , 컬럼명N=값N
   WHERE 컬럼명=값 ; */


UPDATE  book
      SET  year = '2016'
 WHERE  bookname like '%SQL%' ;


UPDATE  book
      SET  year = '2010'
          ,  price = 20000
 WHERE  bookname = 'JAVA' ;
```

# 레코드 삭제하기

```
/* DELETE FROM 테이블명
        WHERE 컬럼명=값 ;  */

DELETE FROM book
      WHERE bookname = 'JAVA' ;


DELETE FROM book
      WHERE publisher ='Wiley' AND year < '2015'  ;
```

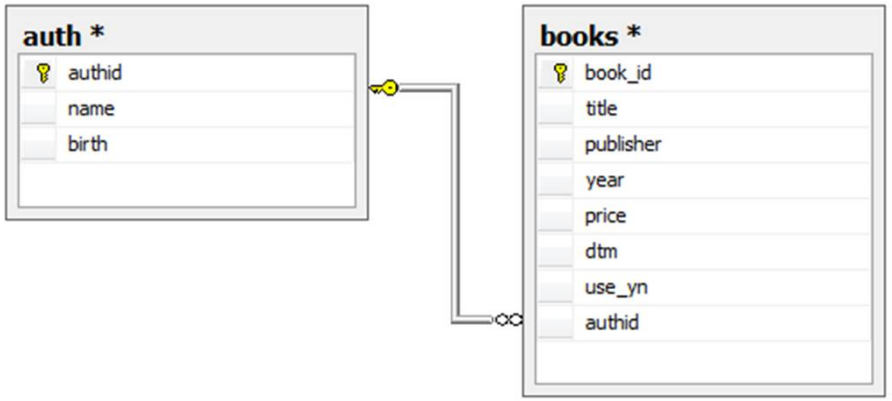
# 레코드 조인

- *조인이란?*
  - *테이블 연결*




- *조인의 종류*
  - ***inner join***
  - ***left join = left outer join***
  - *right join = right outer join*
  - *cross join*

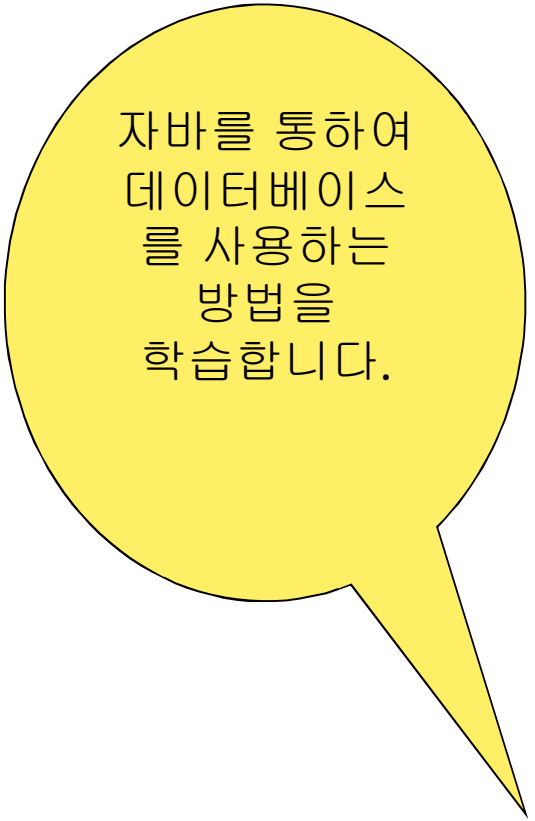
# 조인하기




```
-- inner  join
select book.*, auth.*
  from book
       inner join auth on book.authid = auth.authid ;


-- left join
select book.*, auth.*
  from book
       left join auth on book.authid = auth.authid ;
```

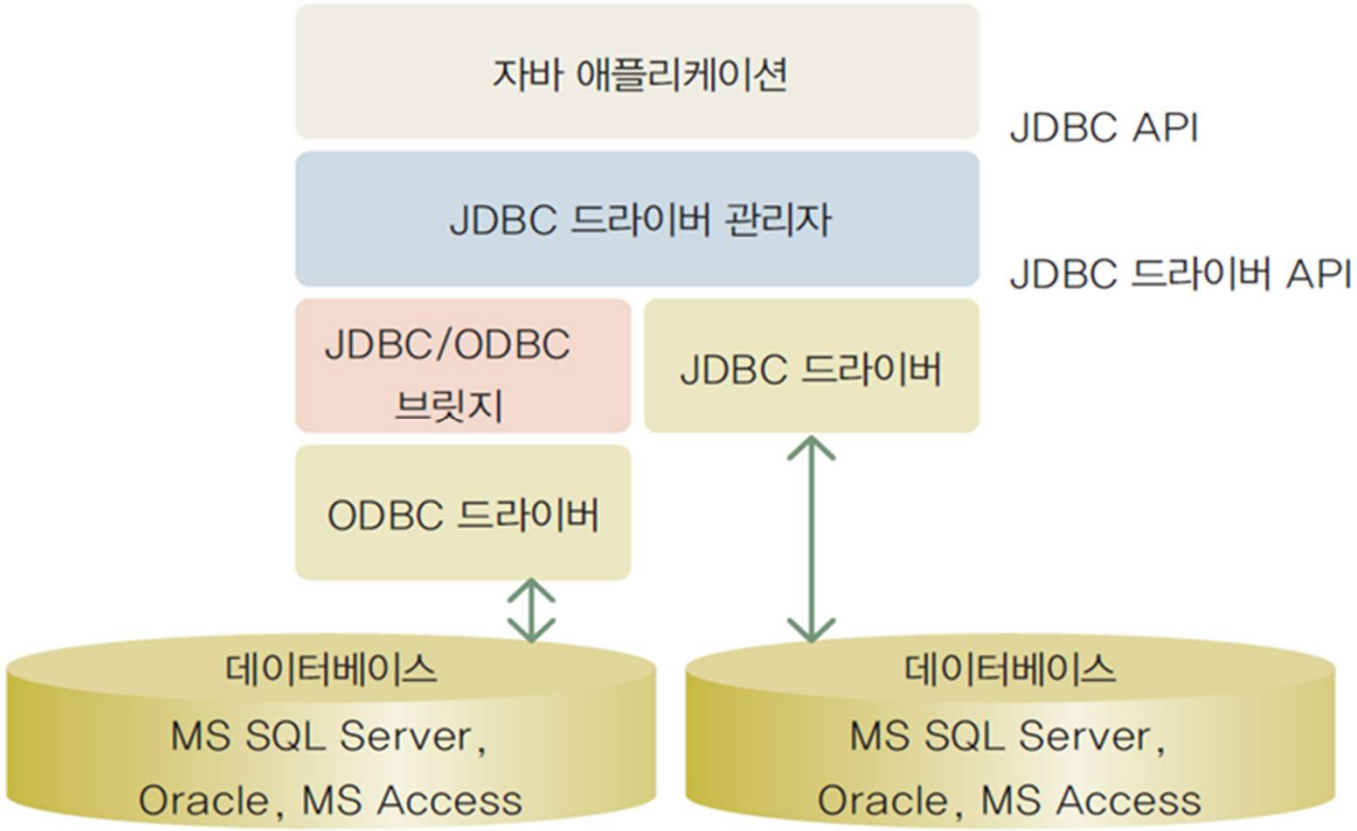
# JDBC 프로그래밍

- 자바와 데이터베이스
- JDBC란
  - 자바에서 DB 프로그래밍을 위해 사용되는 라이브러리
- JDBC 프로그래밍 절차
  1. JDBC 드라이버 로딩
  2. 데이터베이스 커넥션  연결
  3. PreparedStatement 객체 생성
  4. 쿼리 실행쿼리 실행 결과 사용
  5. PreparedStatement 종료
  6. 데이터베이스 커넥션 종료



# 자바와 데이터베이스

- JDBC(Java Database Connectivity)는 자바 API의 하나로서 데이터베이스에 연결하여서 데이터베이스 안의 데이터에 대하여 검색하고 데이터를 변경할 수 있게 한다.



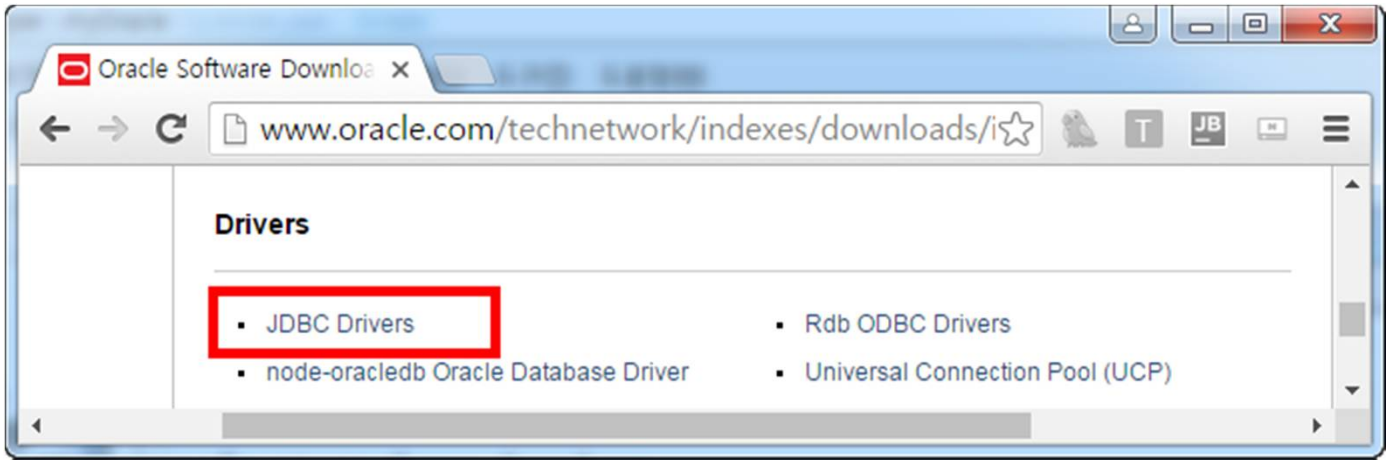
# JDBC 드라이버 다운로드

- Mysql / SQLite JDBC 드라이버 다운로드

1. http://mvnrepository.com/에서 드라이버를 다운로드 받는다.
   단, jar 파일이면 압축을 풀지 마시오.

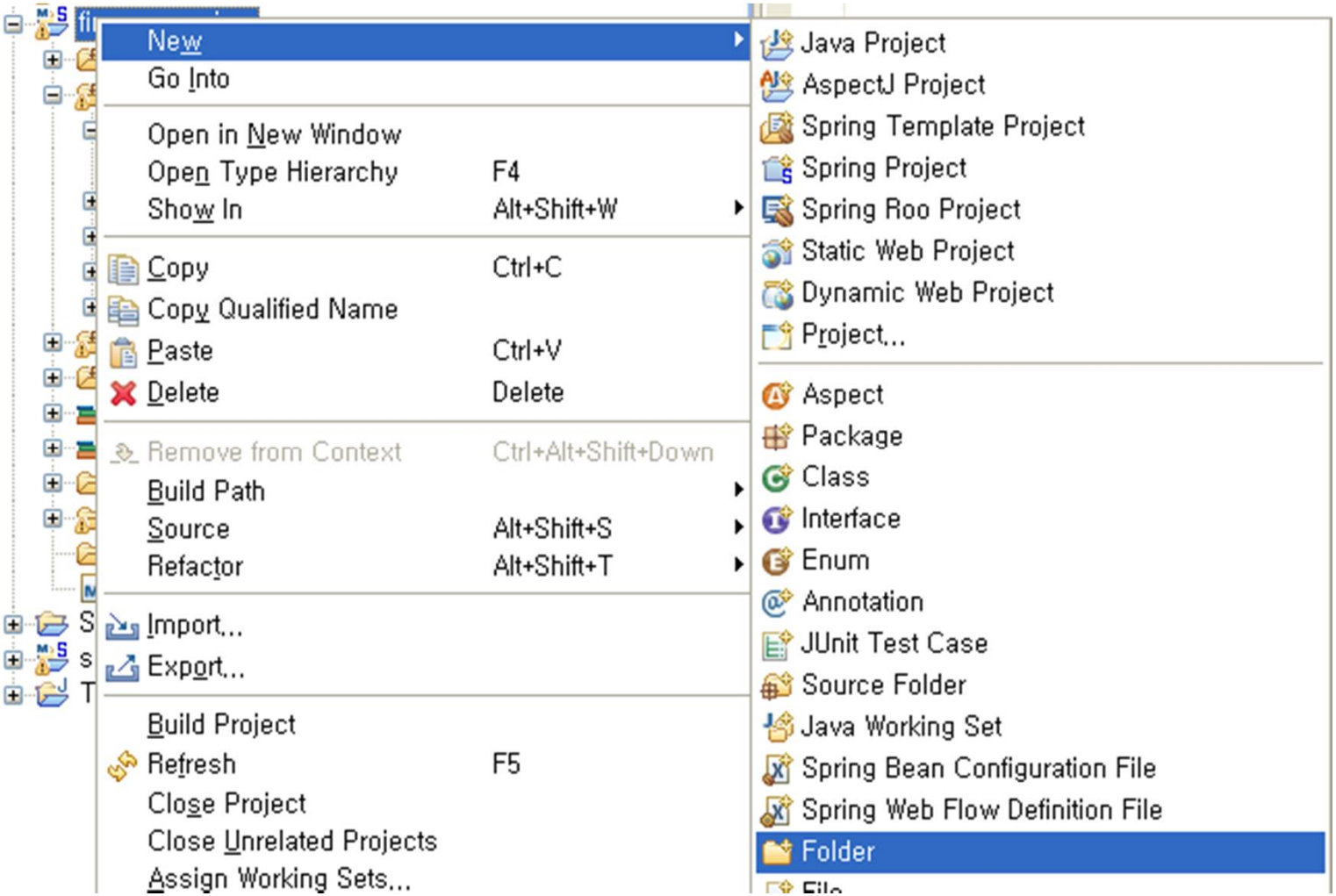
- Oracle JDBC 드라이버 다운로드

1. https://www.oracle.com/downloads/_에서 드라이버를 다운로드 받는다.
   단, jar 파일이면 압축을 풀지 마시오.



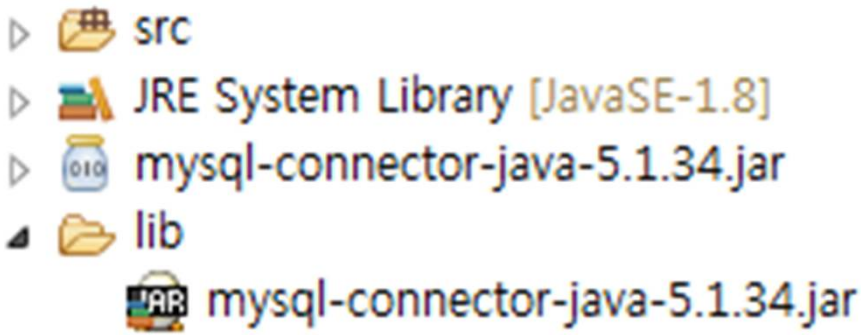
# JDBC를 Build Path에 추가하기

- **프로젝트에 lib 폴더 생성**



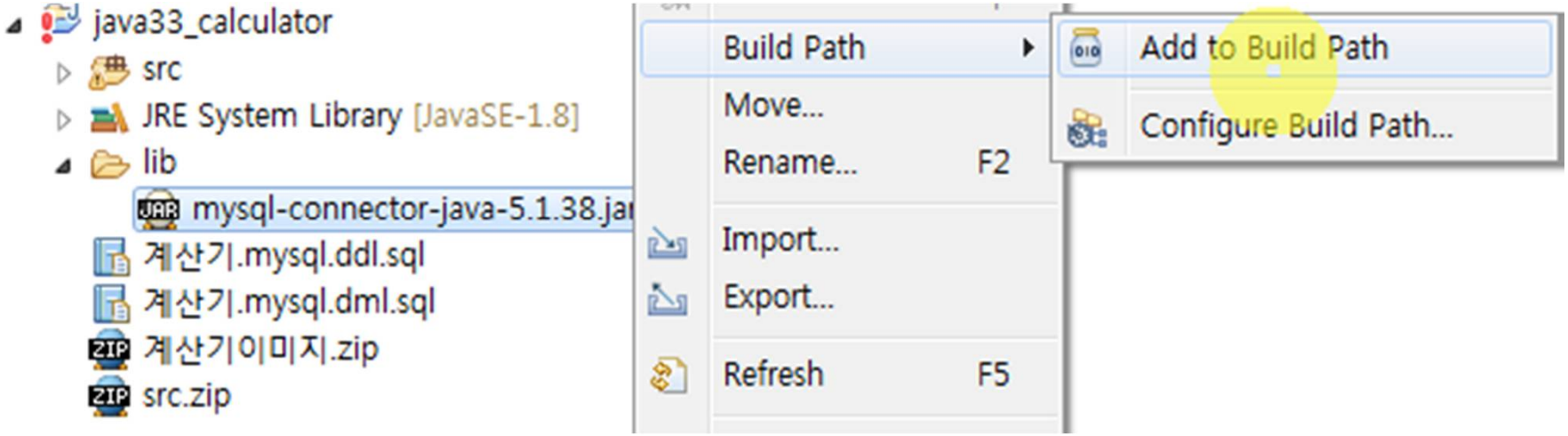
# JDBC를 Build Path에 추가하기

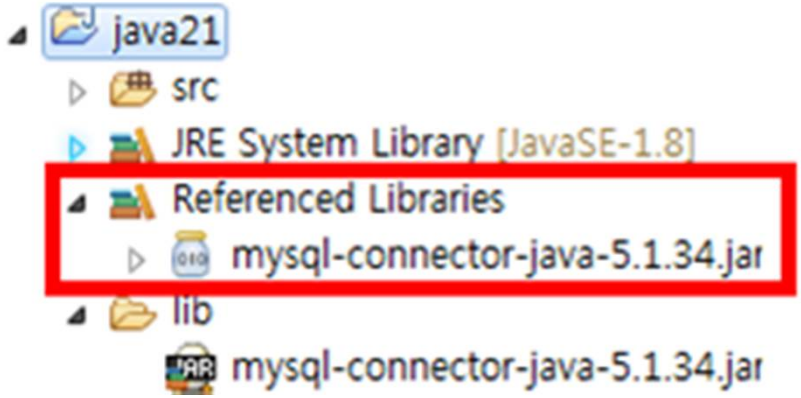
- **lib 폴더에 다운로드 받은 jar 파일을 복사**



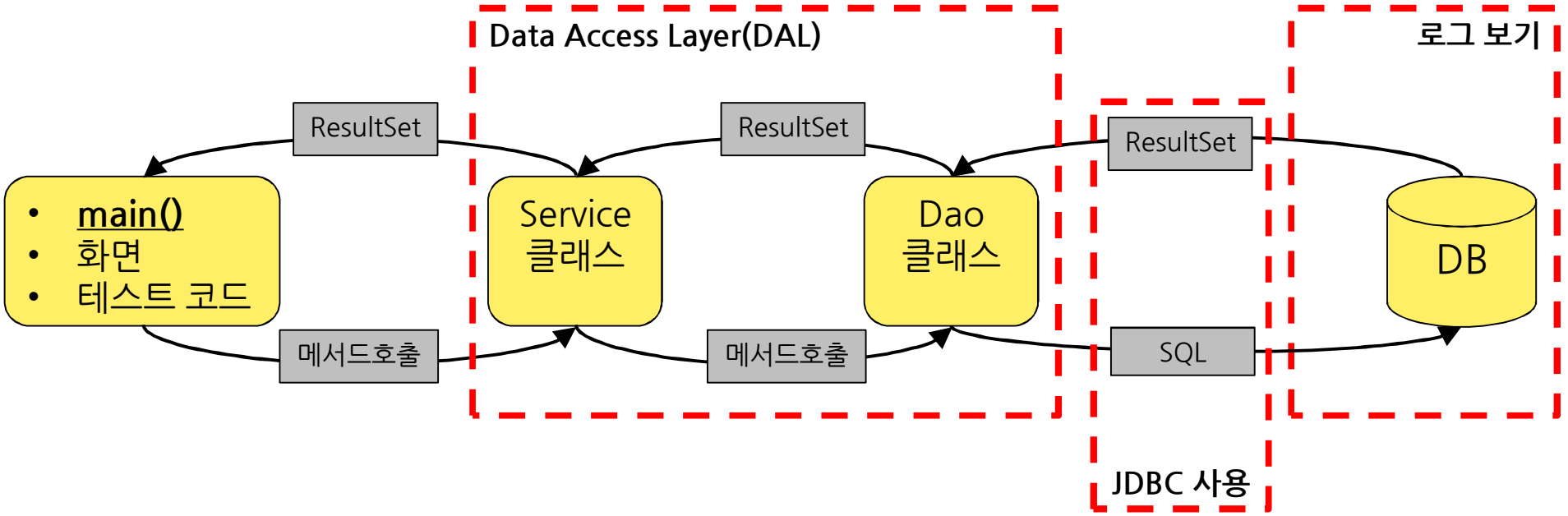
# JDBC를 Build Path에 추가하기

- **jar 파일 Java Build  Path에 추가**



# JDBC를 이용한 데이터 처리 과정



# JDBC 프로그래밍 방식

- Statement 방식
  - *SQL 인젝션 공격에 취약*
  - *느리다*
- PreparedStatement 방식
  - *SQL 인젝션 공격을 무력화*
  - *빠르다*

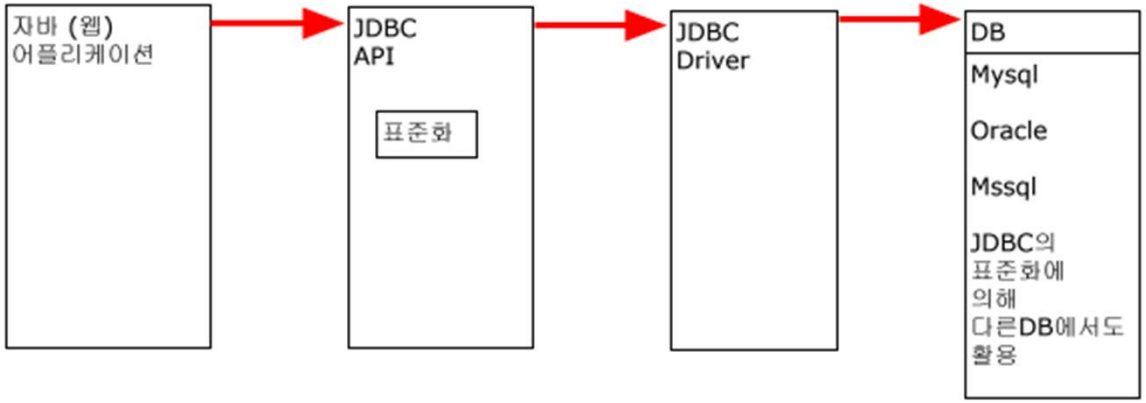
# JDBC 프로그래밍 방식

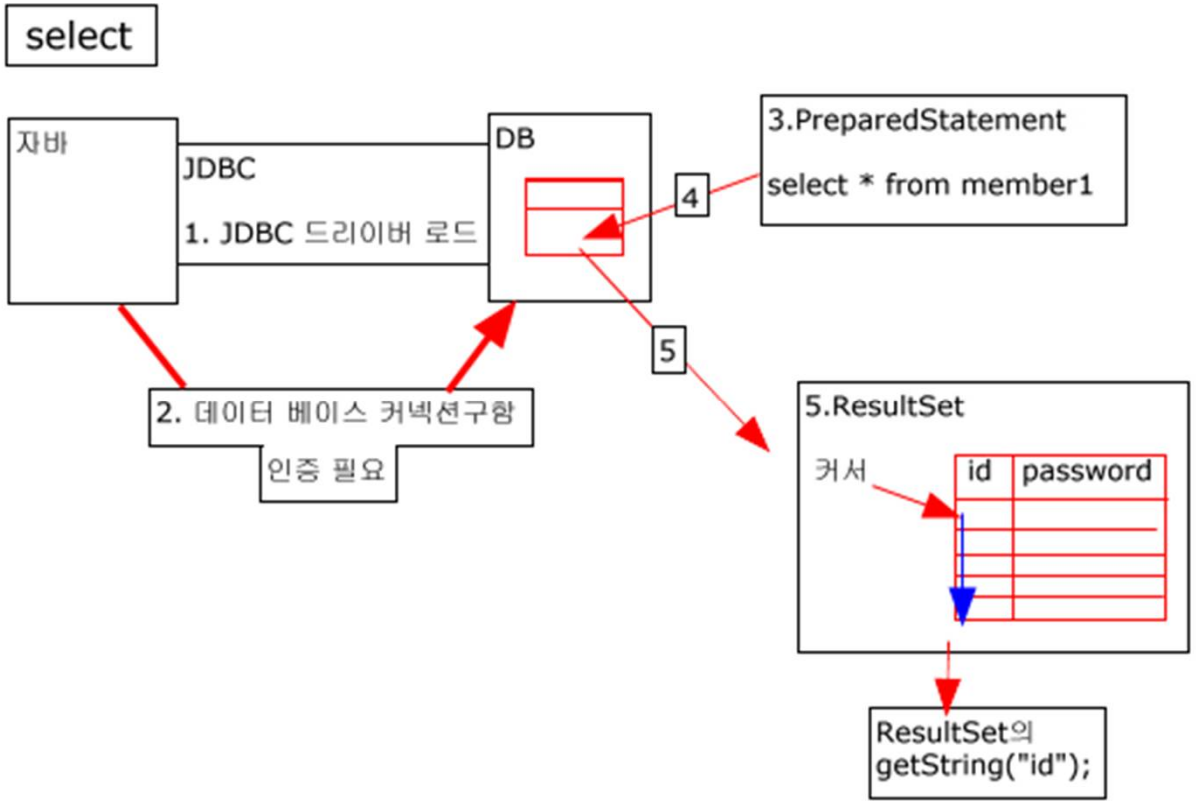
<table>
<tr><th>Application</th><th colspan="2">DB I/O</th><th>DBMS</th></tr>
<tr><td>GUI화면</td><td>Service<br>Layer</td><td>DAO(Data Access Object)<br>Layer</td><td>TABLE</td></tr>
<tr><td></td><td></td><td>
<table>
<tr><td>SELECT</td><td>INSERT<br>DELETE<br>UPDATE</td></tr>
<tr><td>executeQuery()</td><td>executeUpdate()</td></tr>
<tr><td>ResultSet</td><td>int</td></tr>
</table>
</td><td></td></tr>
</table>

# JDBC프로그램 실행 순서(1/3)

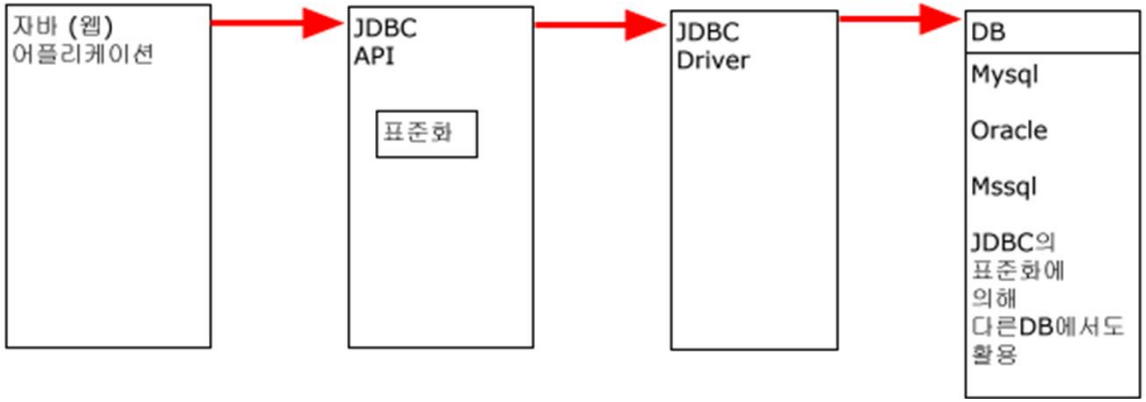
1. JDBC 드라이버 로딩
2. 데이터베이스 커넥션  연결
3. 쿼리 실행을 위한 PreparedStatement 객체 생성
4. 쿼리 실행
   - executeQuery()
   - executeUpdate()
5. 쿼리 실행 결과 사용
6. PreparedStatement 종료
7. 데이터베이스 커넥션 종료

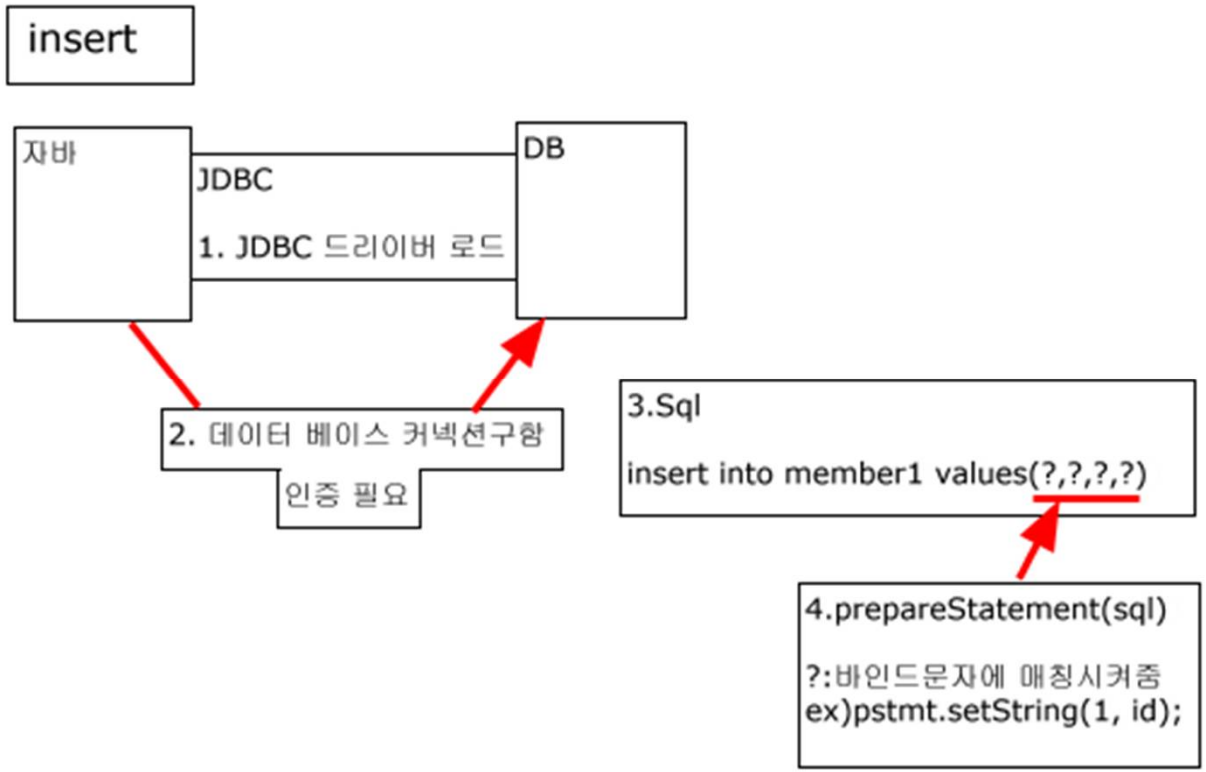
# JDBC프로그램 실행 순서(2/3)



# JDBC프로그램 실행 순서(3/3)



# 데이터베이스 프로그래밍의 순서

❶ JDBC 드라이버를 적재
❷ 데이터베이스 연결

```
Class.forName("                ?                ");

Connection conn =
DriverManager.getConnection(url, user, pw);
```



❸ SQL 문장 작성 및 전송

```
String query = "SELECT * FROM book ";
PreparedStatement stmt =
conn.prepareStatement(query);
ResultSet rs = stmt.executeQuery();
```


❹ 결과 집합 사용 후 연결 해제

```
while( rs.next ) {
    int bookid      = rs.getInt( "bookid" );
    String bookname = rs.getString( "bookname" );
}
```

# 타입 매핑

| sqlite( mysql, oracle) | jdbc 함수 | java |
|---|---|---|
| BIT(1) | getBoolean()<br>setBoolean() | boolean, Boolean |
| BIT(1), TINYIN<br>T, SMALLINT, INT, BIGINT | getInt()<br>setInt() | byte, short, int, long |
| CHAR , VARCHAR<br>TEXT | getString()<br>setString() | String |
| FLOAT, DOUBLE | getDouble()<br>setDouble() | float, double |
| NUMERIC, DECIMAL | | java.math.BigDecimal<br>java.math.BigInteger |
| DATE, DATETIME<br>TIME, TIMESTAMP | getDate()<br>setDate() | Java.util.Date<br>Java.sql.Date<br>Java.sql.Time<br>Java.sql.Timestamp |
| BINARY, VARBINARY, BLOB | | byte[] |

# DBMS별 타입 매핑

| MySQL | Oracle | SQLite 타입 |
|---|---|---|
| INT | NUMBER( 자리수, 0 ) | INTEGER |
| INTEGER | | |
| TINYINT | | |
| SMALLINT | | |
| MEDIUMINT | | |
| BIGINT | | |
| NUMERIC | | |
| BOOLEAN | | |
| CHAR | VARCHAR2 | TEXT |
| VARCHAR | | |
| NCHA | | |
| NARACTER | | |
| TEXT | | |
| DATE | DATE | TEXT |
| DATETIME | | |
| BLOB | BLOB | NONE |
| CLOB | CLOB | |
| REAL | NUMBER( 자리수, 소수점 ) | REAL |
| DOUBLE | | |
| FLOAT | | |
| DECIMAL(10,5) | | |

# DBConnect 만들기(1/4)

```
public class DBConnect {
    public static Connection connectionOracle(){
        String url      = "jdbc:oracle:thin:@//localhost:1521/pdborcl";
        String user     = "tester1";
        String password = "1234";

        Connection conn = null;
        try {
            Class.forName("oracle.jdbc.OracleDriver"); // driver 적재
            conn = DriverManager.getConnection(url, user, password); // DB 연결
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return conn;
    }
}
```

# DBConnect 만들기(2/4)

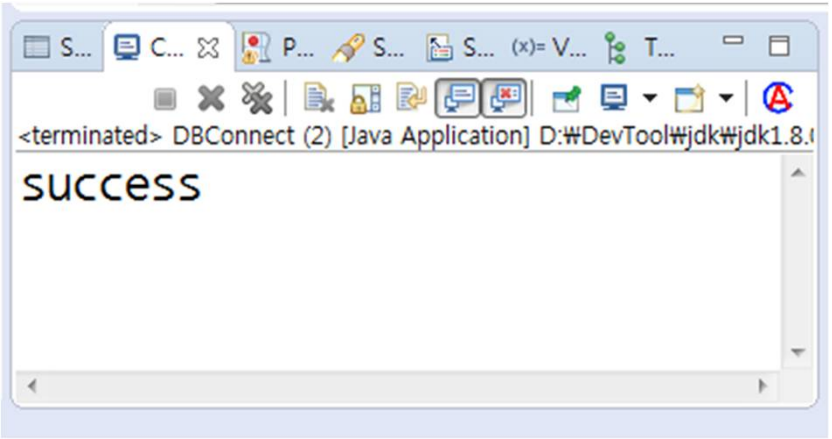
```
public class DBConnect {
    public static Connection connectionOracle(){
        String url      = "jdbc:mysql://localhost:3306/book_db";
        String user     = "root";
        String password = "root";
        Connection conn = null;
        try {
            Class.forName("com.mysql.jdbc.Driver"); // driver 적재
            conn = DriverManager.getConnection(url, user, password); // DB 연결
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return conn;
    }
}
```

# DBConnect 만들기(3/4)

```
public class DBConnect {
    public static Connection connectionSQLite(){
        String url      = "jdbc:sqlite::resource:resources/data/test.db";
        Connection conn = null;

        try {
            Class.forName("org.sqlite.JDBC"); // driver 적재
            conn = DriverManager.getConnection( url ); // DB 연결
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return conn;
    }
}
```

# DBConnect 만들기(4/4)

```
public class DBConnect {

    public static Connection makeConnection(){
        return connectionOracle(); // Oracle과 연결할 때
    }

    public static void main(String[] args) {

        // 디비 연결
        Connection conn = makeConnection();

        if( conn != null) {
            System.out.println("success");
        }
        else {
            System.out.println("fail");
        }
    }
}
```



# DaoBook의 getSQLSelectAll(...) 만들기

```
Connection  conn = DBConnect.makeConnection();

String query  = "SELECT * FROM book ORDER BY bookid ASC "; // SQL 문장 생성
PreparedStatement stmt = conn.prepareStatement(query); // 문장 객체 생성
ResultSet rs           = stmt.executeQuery(); // SQL 문장 실행

for (  ; rs.next();  ) {
    int id           = rs.getInt    ("bookname"   );
    String bookname  = rs.getString ("bookname"     );
    String publisher = rs.getString ("publisher" );
    String year      = rs.getString ("year"      );
    int price        = rs.getInt    ("price"     );
    Date dtm         = rs.getDate   ("dtm"       );
    boolean use_yn   = rs.getBoolean("use_yn"    );

    String out = String.format(" %d \t %15s \t %10s \t %4s \t %7d \t %10s \t %b",
id, bookname, publisher, year, price, dtm, use_yn);
    System.out.println(out);
}
conn.close();
```

SQL문장을 실행하고
결과 **row** 를 반환한다.

# DaoBook의 getSQLSelectEqual(...) 만들기

```
Connection  conn = DBConnect.makeConnection();

String query = "SELECT * FROM book  WHERE bookname LIKE  ?   "; // SQL 문장 생성
PreparedStatement stmt = conn.prepareStatement(query); // 문장 객체 생성
stmt.setString ( 1, "%"+ bookname +"%"  );  // ? 에 값을 설정하는 부분

ResultSet rs = stmt.executeQuery(); // SQL 문장 실행

for (  ; rs.next();  ) {
    int id           = rs.getInt    ("bookid"   );
    String bookname  = rs.getString ("bookname"     );
    Date dtm         = rs.getDate   ("dtm"       );
    boolean use_yn   = rs.getBoolean("use_yn"    );

    String out = String.format(" %d \t %s \t %d \t %b", id, bookname, dtm,
use_yn);
    System.out.println(out);
}

conn.close();
```

SQL문장을 실행하고
결과 row 를 반환한다.

# DaoBook의 getSQLSelectEqual(..) 만들기

```
Connection  conn = DBConnect.makeConnection();

String query = "SELECT * FROM book  WHERE bookname = ?   "; // SQL 문장 생성
PreparedStatement stmt = conn.prepareStatement(query); // 문장 객체 생성
stmt.setString ( 1,    bookname    );  // ? 에 값을 설정하는 부분

ResultSet rs = stmt.executeQuery(); // SQL 문장 실행

for (  ; rs.next();  ) {
    int id           = rs.getInt    ("bookid"   );
    String bookname  = rs.getString ("bookname" );
    Date dtm         = rs.getDate   ("dtm"      );
    boolean use_yn   = rs.getBoolean("use_yn"   );

    String out = String.format(" %d \t %s \t %s \t %s \t %d \t %s \t %b", id,
bookname, publisher, year, price, dtm, use_yn);
    System.out.println(out)
}

conn.close();
```

SQL문장을 실행하고
결과 row 를 반환한다.

# DaoBook의 setSQLInsert(...) 만들기

```
Connection  conn = DBConnect.makeConnection();

// SQL 문장 생성
String query  =" INSERT INTO                                                  \n";
query += "   book( bookname, publisher, year, price, dtm, use_yn, authid) \n";
query += " VALUES(    ?    , ?         , ?   , ?    ,  ? ,  ?    ,  ?    ) \n";

PreparedStatement stmt = conn.prepareStatement(query); // 문장 객체 생성
stmt.setString ( 1, bookname );
stmt.setString ( 2, publisher);
stmt.setString ( 3, year     );
stmt.setInt    ( 4, price    );
stmt.setString ( 5, dtm      );
stmt.setBoolean( 6, use_yn   );
stmt.setInt    ( 7, authid   );

// SQL 문장 실행
int rs = stmt.executeUpdate();

conn.close();
```

SQL문장을 실행하고
결과 **row** 수를 반환한다.

# DaoBook의 setSQLUpdate(...) 만들기

```
Connection  conn = DBConnect.makeConnection();

// SQL 문장 생성
String query = " UPDATE book  SET year = ? , price = ? WHERE bookname LIKE ? ";

PreparedStatement stmt = conn.prepareStatement(query); // 문장 객체 생성
stmt.setString ( 1, year    );
stmt.setInt    ( 2, price   );
stmt.setString ( 3,  "%" + bookname + "%"   );

// SQL 문장 실행
int rs = stmt.executeUpdate();

conn.close();
```

SQL문장을 실행하고
결과 row 수를 반환한다.

# DaoBook의 setSQLDelete(...) 만들기

```
Connection  conn = DBConnect.makeConnection();

// SQL 문장 생성
String query = " DELETE FROM book WHERE bookname = ?  " ;

// 문장 객체 생성
PreparedStatement stmt = conn.prepareStatement(query);
stmt.setString ( 1, bookname  );

// SQL 문장 실행
int rs = stmt.executeUpdate();

conn.close();
```

SQL문장을 실행하고
결과 row 수를 반환한다.

# ServiceBook의 getSQLSelectAll(...) 만들기

```
public ResultSet getSQLSelectAll() throws SQLException {

    ResultSet rs = null;
    Connection conn = DBConnect.makeConnection(); // 디비 연결
    DaoBook dao = new DaoBook();

    try {
        conn.setAutoCommit(false); //transaction block start

        rs = dao.getSQLSelectAll(conn);

        conn.commit(); //transaction block end
    } catch (SQLException e) {
        e.printStackTrace();
        conn.rollback(); //transaction block end
    }

    return rs;
}
```

# ServiceBook의 getSQLSelectEqual(...) 만들기

```
public ResultSet getSQLSelectEqual(String expr) throws SQLException {

    ResultSet rs = null;
    Connection conn = DBConnect.makeConnection(); // 디비 연결
    DaoBook dao = new DaoBook();

    try {
        conn.setAutoCommit(false); //transaction block start

        rs = dao.getSQLSelectEqual(conn, expr);

        conn.commit(); //transaction block end
    } catch (SQLException e) {
        conn.rollback(); //transaction block end
    }

    return rs;
}
```

# ServiceBook의 setSQLInsert(...) 만들기

```
public int setSQLInsert( String bookname, String publisher, String year, int price
        , String dtm, boolean use_yn, int authid ) throws SQLException {

    int rs = 0;
    Connection conn = DBConnect.makeConnection(); // 디비 연결
    DaoBook dao = new DaoBook();

    try {
        conn.setAutoCommit(false); //transaction block start

        rs = dao.setSQLInsert(conn, bookname, publisher, year, price, dtm, use_yn,
authid);

        conn.commit(); //transaction block end
    } catch (SQLException e) {
        conn.rollback(); //transaction block end
    }

    return rs;
}
```

# ServiceBook의 setSQLUpdate(...) 만들기

```
public int setSQLUpdate(String bookname, String year, int price ) throws
SQLException {

    int rs = 0;
    Connection conn = DBConnect.makeConnection(); // 디비 연결
    DaoBook dao = new DaoBook();

    try {
        conn.setAutoCommit(false); //transaction block start

        rs = dao.setSQLUpdate( conn, bookname, year, price );

        conn.commit(); //transaction block end
    } catch (SQLException e) {
        conn.rollback(); //transaction block end
    }

    return rs;
}
```

# ServiceBook의 setSQLDelete(...) 만들기

```
public int setSQLDelete(String bookname) throws SQLException {

    int rs = 0;
    Connection conn = DBConnect.makeConnection(); // 디비 연결
    DaoBook dao = new DaoBook();

    try {
        conn.setAutoCommit(false); //transaction block start

        rs = dao.setSQLDelete(conn, bookname);

        conn.commit(); //transaction block end
    } catch (SQLException e) {
        conn.rollback(); //transaction block end
    }

    return rs;
}
```