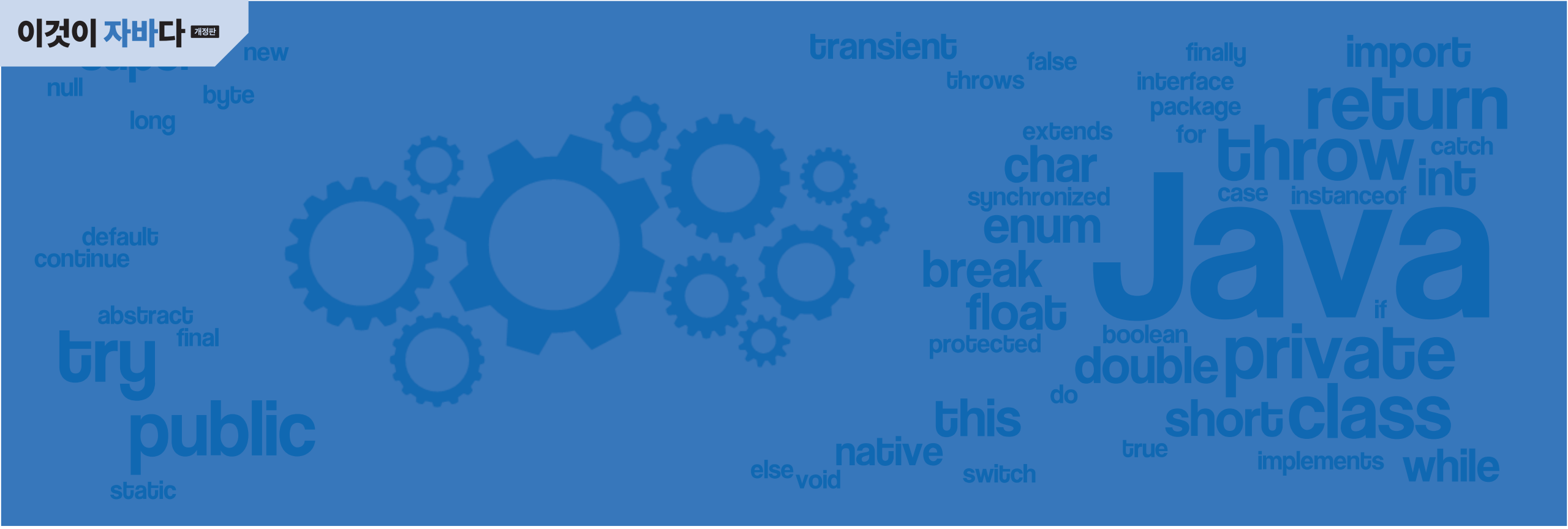
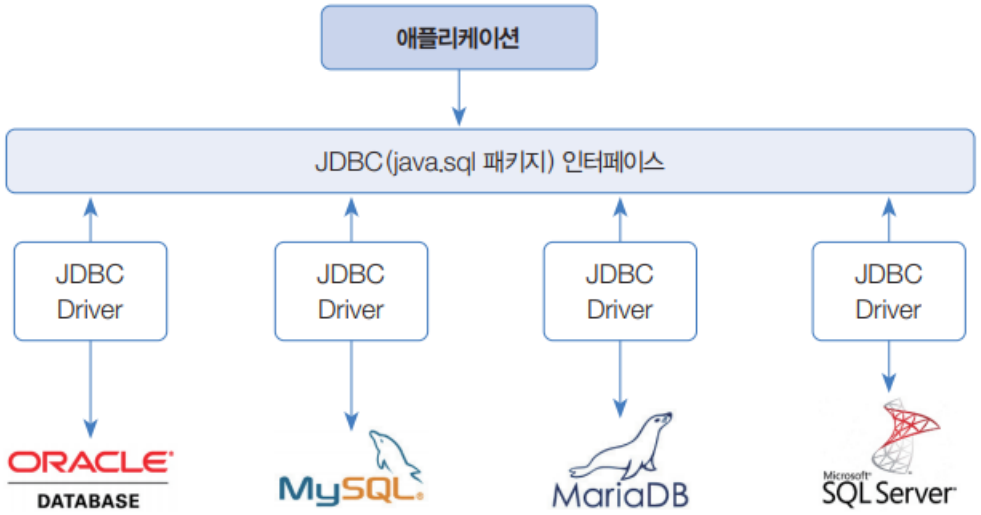


Chapter 20 데이터베이스 입출력

▶ Contents

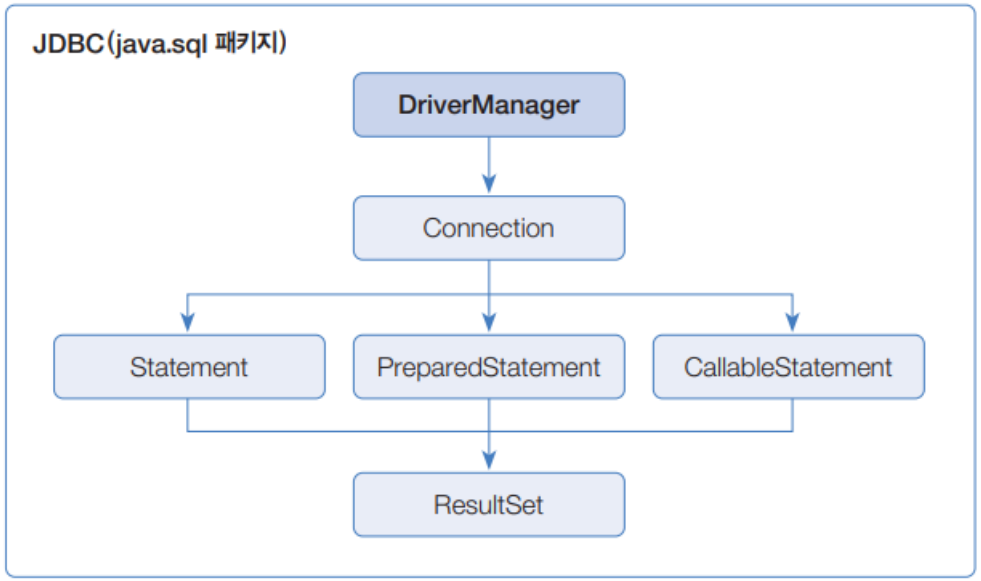
JDBC 라이브러리

- 자바는 데이터베이스(DB)와 연결해서 데이터 입출력 작업을 할 수 있도록 JDBC 라이브러리(java.sql 패키지)를 제공
- JDBC는 데이터베이스 관리시스템(DBMS)의 종류와 상관없이 동일하게 사용할 수 있는 클래스와 인터페이스로 구성



JDBC Driver

- JDBC 인터페이스를 구현한 것으로, DBMS마다 별도로 다운로드받아 사용
- DriverManager 클래스: JDBC Driver를 관리하며 DB와 연결해서 Connection 구현 객체를 생성
- Connection 인터페이스: Statement, PreparedStatement, CallableStatement 구현 객체를 생성하며, 트랜잭션 처리 및 DB 연결을 끊을 때 사용
- Statement 인터페이스: SQL의 DDL과 DML 실행 시 사용
- PreparedStatement: SQL의 DDL, DML 문 실행 시 사용. 매개변수화된 SQL 문을 써 편리성과 보안성 유리
- CallableStatement: DB에 저장된 프로시저와 함수를 호출
- ResultSet: DB에서 가져온 데이터를 읽음



Oracle 설치

- Enterprise Edition 설치 파일 다운로드:

  https://www.oracle.com/database/technologies/oracle-database-software-downloads.html#19c

- C:\Oracle\WINDOWS.X64_193000_db_home

Oracle Base: C:\Oracle
비밀번호: oracle
컨테이너 데이터베이스로 생성: 체크 해제 (매우 중요)

- 프로그램에서 사용할 DB 계정을 생성하기 위해 SQL Plus에서 다음 SQL 문을 실행

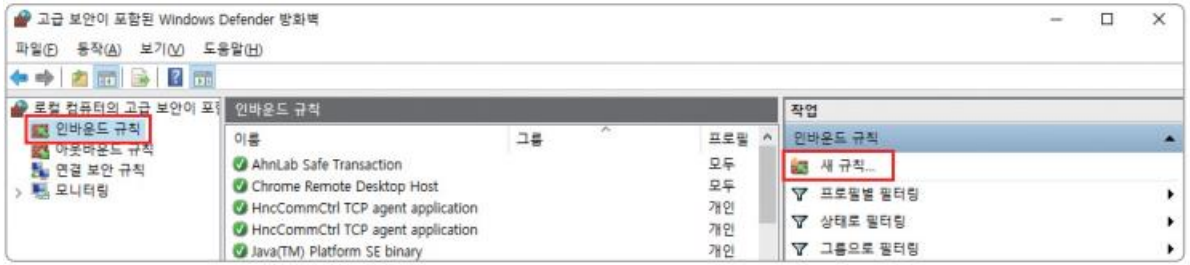
```
SQL> create user java identified by oracle;
SQL> grant connect to java;
SQL> grant resource to java;
SQL> grant unlimited tablespace to java;
```

원격 연결

- 원격 연결 요청을 수락하기 위해 Net Configuration Assistant를 실행
- [시작] 메뉴 - [Oracle] - [OraDB19Home1] - [Net Configuration Assistant]

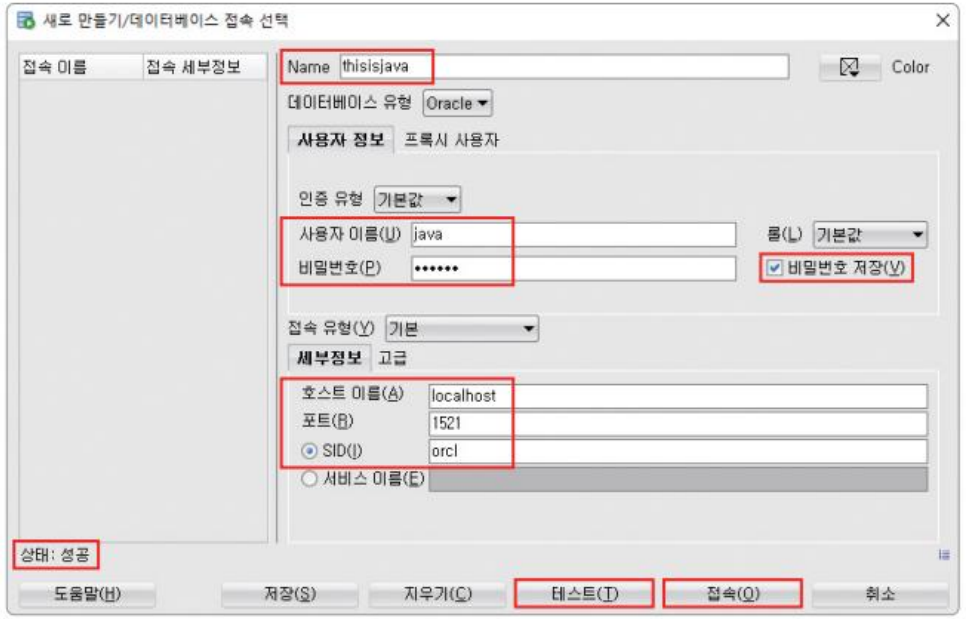
방화벽 해제

- Oracle을 설치한 운영체제의 방화벽 설정에서 1521 포트를 개방
- Windows Defender 방화벽 대화상자에서 [인바운드 규칙]을, 오른쪽 작업 창에서 [새 규칙]을 선택해 설정 변경



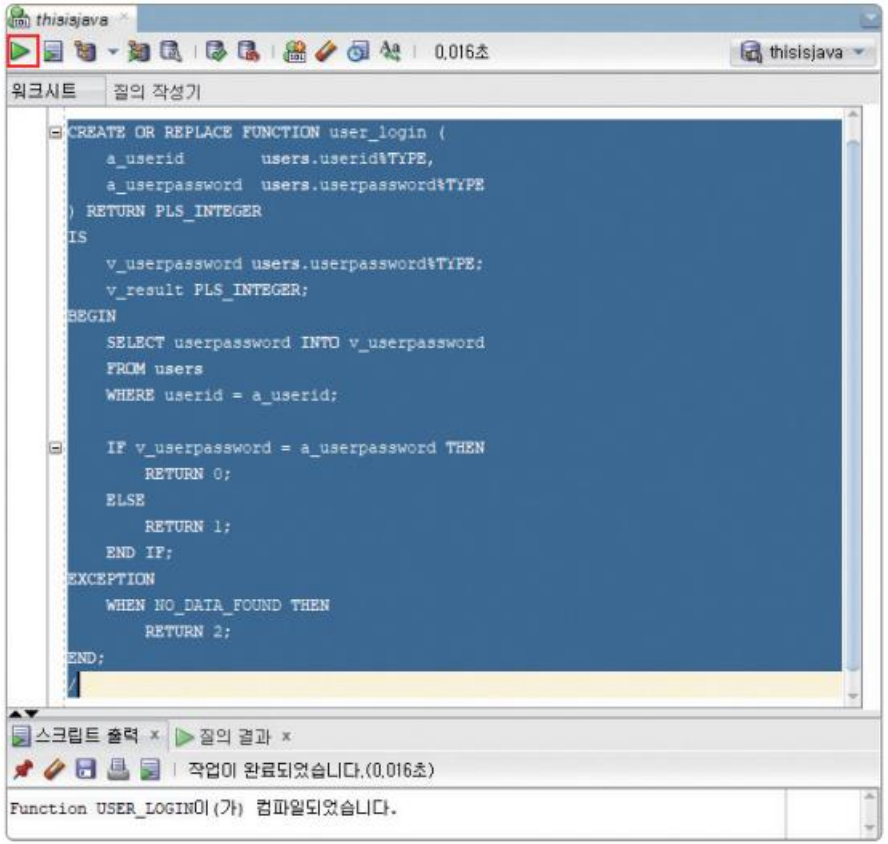
SQL Developer

- Oracle DB 모델링에서부터 DB 상태 확인, SQL 스크립트 및 PL/SQL 개발 등 용이한 무료 Client Tool
- 설치 파일 다운로드: https://www.oracle.com/tools/downloads/sqldev-downloads.html
- C:\Oracle\sqldeveloper
- 설치 후 [새로 만들기/데이터베이스 접속 선택] 대화상자에서 설정 후 테스트



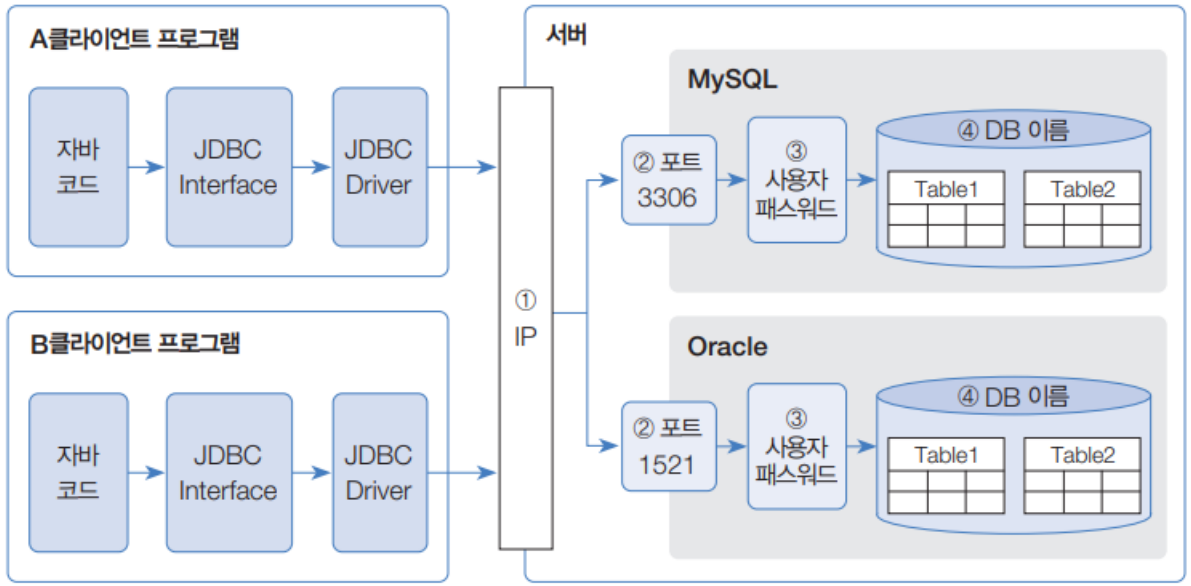
데이터베이스 구성

- 테이블, 시퀀스, 프로시저, 함수를 생성하여 데이터베이스를 구성



데이터베이스 연결

- 클라이언트 프로그램에서 DB와 연결하려면 해당 DBMS의 JDBC Driver가 필요
- ① DBMS가 설치된 컴퓨터의 IP 주소, ② DBMS가 허용하는 포트(Port) 번호, ③ 사용자(DB 계정) 및 비밀번호 ④ 사용하고자 하는 DB 이름 필요



JDBC Driver 설치

- 로컬 PC에 Oracle을 설치하면 JDBC Driver 파일 찾을 수 있음

  (C:\Oracle\WINDOWS.X64_193000_db_home\jdbc\lib\ojdbc8.jar)

- 원격 PC에 Oracle을 설치하면 JDBC Driver만 별도로 다운로드

  (https://mvnrepository.com/artifact/com.oracle.database.jdbc/ojdbc8)

DB 연결

- Class.forName() 메소드는 문자열로 주어진 JDBC Driver 클래스를 BuildPath에서 찾고, JDBC Driver를 메모리로 로딩

```
Class.forName("oracle.jdbc.OracleDriver");
```

```
Connection conn = DriverManager.getConnection("연결 문자열", "사용자", "비밀번호");
```

INSERT 문

- users 테이블에 새로운 사용자 정보를 저장하는 INSERT 문 실행
- INSERT 문을 String 타입 변수 sql에 문자열로 대입

```
INSERT INTO users (userid, username, userpassword, userage, useremail)
VALUES (?, ?, ?, ?, ?)
```

```
String sql = new StringBuilder()
  .append("INSERT INTO users (userid, username, userpassword, userage, useremail) ")
  .append("VALUES (?, ?, ?, ?, ?)")
  .toString();

또는

String sql = "" +
  "INSERT INTO users (userid, username, userpassword, userage, useremail) " +
  "VALUES (?, ?, ?, ?, ?)";
```

20.6 데이터 저장

- 매개변수화된 SQL 문을 실행하기 위해 Connection의 prepareStatement() 메소드로부터 PreparedStatement를 얻음

```
PreparedStatement pstmt = conn.prepareStatement(sql);
```

- 값을 지정한 후 executeUpdate() 메소드를 호출하면 SQL 문이 실행되면서 users 테이블에 1개의 행이 저장

```
pstmt.setString(1, "winter");
pstmt.setString(2, "한겨울");
pstmt.setString(3, "12345");
pstmt.setInt(4, 25);
pstmt.setString(5, "winter@mycompany.com");
```

```
int rows = pstmt.executeUpdate();
```

- close() 메소드를 호출하면 PreparedStatement가 사용했던 메모리 해제

```
pstmt.close();
```

UPDATE 문

- JDBC를 이용해서 UPDATE 문을 실행

```
UPDATE boards SET
  btitle=?,
  bcontent=?,
  bfilename=?,
  bfiledata=?
WHERE bno=?
```

```
String sql = new StringBuilder()
    .append("UPDATE boards SET ")
    .append("btitle=?, ")
    .append("bcontent=?, ")
    .append("bfilename=?, ")
    .append("bfiledata=? ")
    .append("WHERE bno=?")
    .toString();
```

- prepareStatement() 메소드로부터 PreparedStatement를 얻고, ?에 해당하는 값을 지정
- executeUpdate() 메소드를 호출. 수정된 행의 수가 리턴

```
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setString(1, "눈사람");
pstmt.setString(2, "눈으로 만든 사람");
pstmt.setString(3, "snowman.jpg");
pstmt.setBlob(4, new FileInputStream("src/ch20/oracle/sec07/snowman.jpg"));
pstmt.setInt(5, 3);
```

```
int rows = pstmt.executeUpdate();
```

DELETE 문

- JDBC를 이용해서 DELETE 문 실행. 매개변수화된 DELETE 문을 String 타입 변수 sql에 대입

```
DELETE FROM boards WHERE bwriter=?
```

```
String sql = "DELETE FROM boards WHERE bwriter=?";
```

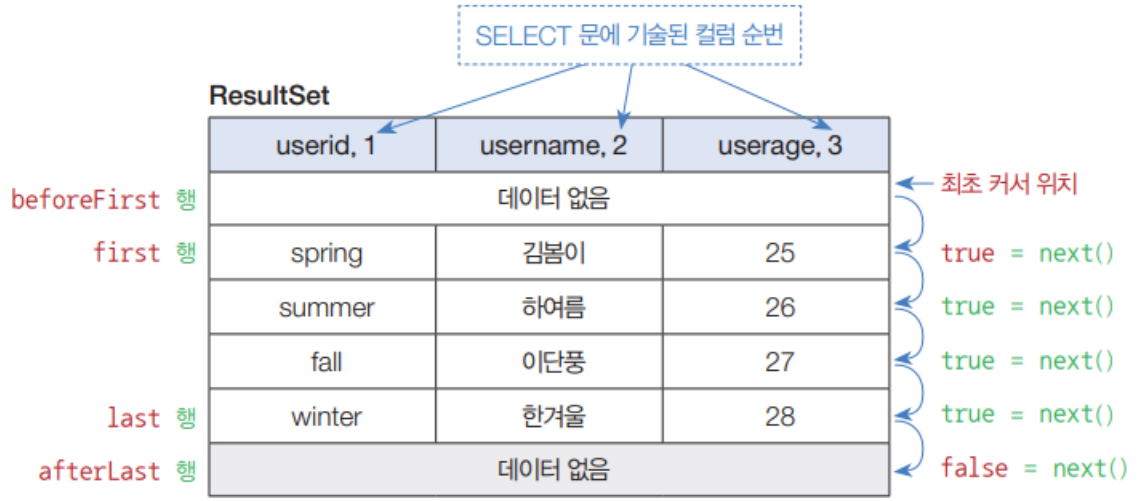
- prepareStatement() 메소드로부터 PreparedStatement를 얻고 ?에 값을 지정한 후, executeUpdate로 SQL 문을 실행

```
String sql = "DELETE FROM boards WHERE bwriter=?";
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setString(1, "winter");
int rows = pstmt.executeUpdate();
```

ResultSet 구조

- SELECT 문에 기술된 컬럼으로 구성된 행(row)의 집합

```
SELECT userid, username, userage FROM users
```



- 커서cursor가 있는 행의 데이터만 읽을 수 있음
- first 행을 읽으려면 next() 메소드로 커서 이동

```
boolean result = rs.next();
```

1개의 데이터 행만 가져올 경우

```
ResultSet rs = pstmt.executeQuery();
if(rs.next()) {
  //첫 번째 데이터 행 처리
} else {
  //afterLast 행으로 이동했을 경우
}
```

n개의 데이터 행을 가져올 경우

```
ResultSet rs = pstmt.executeQuery();
while(rs.next()) {
  //last 행까지 이동하면서 데이터 행 처리
}
//afterLast 행으로 이동했을 경우
```

데이터 행 읽기

- 커서가 있는 데이터 행에서 각 컬럼의 값은 Getter 메소드로 읽음
- SELECT 문에 연산식이나 함수 호출이 포함되어 있다면 컬럼 이름 대신에 컬럼 순번으로 읽어야 함

컬럼 이름으로 읽기

```
String userId =
  rs.getString("userid");
String userName =
  rs.getString("username");
int userAge = rs.getInt("userage");
```

컬럼 순번으로 읽기

```
String userId = rs.getString(1);
String userName = rs.getString(2);
int userAge = rs.getInt(3);
```

```
SELECT userid, userage - 1
FROM users
```

```
String userId =
  rs.getString("userid");
int userAge = rs.getInt(2);
```

사용자 정보 읽기

- 사용자 정보를 가져오는 SELECT 문.
  prepareStatement() 메소드로부터
  PreparedStatement를 얻고, ?에 값을 지정

```
String sql = "" +
  "SELECT userid, username, userpassword, userage, useremail " +
  "FROM users " +
  "WHERE userid=?";
```

```
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setString(1, "winter");
```

- executeQuery() 메소드로 SELECT 문을
  실행해서 ResultSet을 얻음.
- if 문을 이용해서 next() 메소드가 true를
  리턴할 경우에는 데이터 행을 User 객체에
  저장하고 출력

```
ResultSet rs = pstmt.executeQuery();
if(rs.next()) {                          //1개의 데이터 행을 가져왔을 경우
  User user = new User();
  user.setUserId(rs.getString("userid"));
  user.setUserName(rs.getString("username"));
  user.setUserPassword(rs.getString("userpassword"));
  user.setUserAge(rs.getInt(4));         //컬럼 순번을 이용해서 컬럼 지정
  user.setUserEmail(rs.getString(5));    //컬럼 순번을 이용해서 컬럼 지정
  System.out.println(user);
} else {                                 //데이터 행을 가져오지 않았을 경우
  System.out.println("사용자 아이디가 존재하지 않음");
}
```

게시물 정보 읽기

- boards 테이블에서 bwriter가 winter인 게시물의 정보를 가져오기

- bwriter가 winter인 게시물 정보를 가져오는 SELECT 문. prepareStatement() 메소드로부터 PreparedStatement를 얻고, ?에 값을 지정

```
String sql = "" +
  "SELECT bno, btitle, bcontent, bwriter, bdate, bfilename, bfiledata " +
  "FROM boards " +
  "WHERE bwriter=?";
```

```
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setString(1, "winter");
```

20.9 데이터 읽기

- executeQuery() 메소드로 SELECT 문을 실행해서 ResultSet을 얻음
- while 문을 이용해서 next() 메소드가 false를 리턴할 때까지 반복해서 데이터 행을 Board 객체에 저장하고 출력한다

```
ResultSet rs = pstmt.executeQuery();
while(rs.next()) {
  //데이터 행을 읽고 Board 객체에 저장
  Board board = new Board();
  board.setBno(rs.getInt("bno"));
  board.setBtitle(rs.getString("btitle"));
  board.setBcontent(rs.getString("bcontent"));
  board.setBwriter(rs.getString("bwriter"));
  board.setBdate(rs.getDate("bdate"));
  board.setBfilename(rs.getString("bfilename"));
  board.setBfiledata(rs.getBlob("bfiledata"));

  //콘솔에 출력
  System.out.println(board);
}
```

- Blob 객체에 저장된 바이너리 데이터를 얻기 위해서는 입력 스트림 또는 배열을 얻어냄
- Blob 객체에서 InputStream을 얻고, 읽은 바이트를 파일로 저장

```
Blob blob = board.getBfiledata();
InputStream is =
  blob.getBinaryStream();
```

```
Blob blob = board.getBfiledata();
byte[] bytes = blob.getBytes(0,
  blob.length());
```

```
InputStream is = blob.getBinaryStream();
OutputStream os = new FileOutputStream("C:/Temp/" + board.getBfilename());
is.transferTo(os);
os.flush();
os.close();
is.close();
```

프로시저와 함수

- Oracle DB에 저장되는 PL/SQL 프로그램. 클라이언트 프로그램에서 매개값과 함께 프로시저 또는 함수를 호출하면 DB 내부에서 SQL 문을 실행하고, 실행 결과를 클라이언트 프로그램으로 돌려줌
- JDBC에서 프로시저와 함수를 호출 시 CallableStatement를 사용. 프로시저와 함수의 매개변수화된 호출문을 작성하고 Connection의 prepareCall() 메소드로부터 CallableStatement 객체를 얻음

```
//프로시저를 호출할 경우
String sql = "{ call 프로시저명(?, ?, …) }";
CallableStatement cstmt = conn.prepareCall(sql);
```

```
//함수를 호출할 경우
String sql = "{ ? = call 함수명(?, ?, …) }"
CallableStatement cstmt = conn.prepareCall(sql);
```

- 프로시저도 리턴값과 유사한 OUT 타입의 매개변수를 가질 수 있기 때문에 괄호 안의 ?중 일부는 OUT값(리턴값)일 수 있음

20.10 프로시저와 함수 호출

- prepareCall() 메소드로 CallableStatement을 얻으면 리턴값에 해당하는 ?는 registerOutParameter() 메소드로 지정하고, 그 이외의 ?는 호출 시 필요한 매개값으로 Setter 메소드를 사용해서 값을 지정

```
String sql = "{ call 프로시저명(?, ?, ?) }";
CallableStatement cstmt = conn.prepareCall(sql);
cstmt.setString(1, "값");                  //프로시저의 첫 번째 매개값
cstmt.setString(2, "값");                  //프로시저의 두 번째 매개값
cstmt.registerOutParameter(3, 리턴타입);    //세 번째 ?는 OUT값(리턴값)임을 지정
```

```
String sql = "{? = call 함수명(?, ?)}";
CallableStatement cstmt = conn.prepareCall(sql);
cstmt.registerOutParameter(1, 리턴타입);    //첫 번째 ?는 리턴값임을 지정
cstmt.setString(2, "값");                  //함수의 첫 번째 매개값
cstmt.setString(3, "값");                  //함수의 두 번째 매개값
```

- execute() 메소드로 프로시저 또는 함수 호출. Getter 메소드로 리턴값 얻음

```
cstmt.execute();
```

프로시저

```
int result = cstmt.getInt(3);
```

함수

```
int result = cstmt.getInt(1);
```

프로시저 호출

- IN 매개변수는 호출 시 필요한 매개값으로 사용되며, OUT 매개변수는 리턴값으로 사용

```
CREATE OR REPLACE PROCEDURE user_create (
    a_userid        IN    users.userid%TYPE,
    a_username      IN    users.username%TYPE,
    a_userpassword  IN    users.userpassword%TYPE,
    a_userage       IN    users.userage%TYPE,
    a_useremail     IN    users.useremail%TYPE,
    a_rows          OUT   PLS_INTEGER
)
...
```

- 매개변수화된 호출문을 작성하고 CallableStatement를 얻음
- ?의 값을 지정하고 리턴 타입을 지정
- 프로시저를 실행하고 리턴값 얻음

```
String sql = "{call user_create(?, ?, ?, ?, ?, ?)}";
CallableStatement cstmt = conn.prepareCall(sql);
```

```
cstmt.setString(1, "summer");
cstmt.setString(2, "한여름");
cstmt.setString(3, "12345");
cstmt.setInt(4, 26);
cstmt.setString(5, "summer@mycompany.com");
cstmt.registerOutParameter(6, Types.INTEGER);
```

```
cstmt.execute();
int rows = cstmt.getInt(6);   //6번째 ? 값 얻기
```

함수 호출

- user_login()은 2개의 매개변수와 PLS_INTEGER 리턴 타입으로 구성

```
CREATE OR REPLACE FUNCTION user_login (
    a_userid        users.userid%TYPE,
    a_userpassword  users.userpassword%TYPE
) RETURN PLS_INTEGER
...
```

- 함수를 호출하기 위해 매개변수화된 호출문을 작성하고 CallableStatement를 얻음

```
String sql = "{? = call user_login(?, ?)}";
CallableStatement cstmt = conn.prepareCall(sql);
```

- ?의 값을 지정하고 리턴 타입을 지정

```
cstmt.registerOutParameter(1, Types.INTEGER);
cstmt.setString(2, "winter");
cstmt.setString(3, "12345");
```

- user_login() 함수는 userid와 userpassword가 일치하면 0을, userpassword가 틀리면 1을, userid가 존재하지 않으면 2를 리턴

```
cstmt.execute();
int result = cstmt.getInt(1);   //첫 번째 ? 값 얻기, 0|1|2 중 하나
```

트랜잭션

- 기능 처리의 최소 단위. 하나의 기능은 여러 소작업들로 구성
- 트랜잭션은 소작업들이 모두 성공하거나 실패해야 함

계좌 이체(트랜잭션)

```
//출금 작업
UPDATE accounts SET balance=balance-이체금액 WHERE ano=출금계좌번호

//입금 작업
UPDATE accounts SET balance=balance+이체금액 WHERE ano=입금계좌번호
```

- 커밋은 내부 작업을 모두 성공 처리하고, 롤백은 실행 전으로 돌아간다는 의미에서 모두 실패 처리
- JDBC에서 트랜잭션을 제어 시 Connection의 setAutoCommit() 메소드로 자동 커밋 기능을 꺼야 함

```
conn.setAutoCommit(false);
```

메인 메뉴

- main() 메소드는 BoardExample 객체를 생성하고 list() 메소드를 호출. list() 메소드는 게시물 목록을 출력하고 mainMenu() 메소드를 호출

메인 메뉴 선택 기능

- 키보드 입력을 받기 위해 Scanner 필드를 추가. mainMenu() 메소드에서 키보드 입력을 받기 위해 nextLine() 메소드를 호출. 메뉴 선택 번호에 따라 해당 메소드를 호출

Board 클래스 작성

- boards 테이블의 한 개의 행(게시물)을 저장할 Board 클래스를 작성.
- 컬럼 개수와 타입에 맞게 필드를 선언하고, 롬복 @Data 어노테이션을 이용해서 Getter, Setter, toString() 메소드를 자동 생성

게시물 목록 기능

- DB 연결이 필요하므로 Connection 필드를 추가하고, 생성자에서 DB 연결. boards 테이블에서 게시물 정보들을 가져와서 게시물 목록으로 출력하도록 list() 메소드를 수정

게시물 생성 기능

- 메인 메뉴에서 '1.Create'를 선택했을 때 호출되는 create() 메소드 수정

게시물 읽기 기능

- 메인 메뉴에서 '2.Read'를 선택했을 때 호출되는 read() 메소드 수정

게시물 수정 기능

- read() 메소드에서 보조 메뉴 '1.Update|2.Delete|3.List'를 추가하고, 보조 메뉴에서 '1.Update'를 선택하면 update() 메소드가, '2.Delete'를 선택하면 delete() 메소드가 호출
- update() 메소드는 매개값으로 받은 Board 객체를 수정해서 boards 테이블의 게시물 정보를 수정

게시물 삭제 기능

- 게시물 수정 기능을 구현할 때 보조 메뉴에서 '2.Delete'를 선택했을 때 delete() 메소드가 호출. delete() 메소드를 수정해 매개값으로 받은 Board 객체에서 bno를 얻어 boards 테이블에서 해당 게시물을 삭제

게시물 전체 삭제 기능

- 메인 메뉴에서 '3.Clear'를 선택했을 때 호출되는 clear() 메소드 수정

종료 기능

- 메인 메뉴에서 '4.Exit'를 선택했을 때 호출되는 exit() 메소드 수정


transient
false
finally
import
new
byte
null
long
throws
interface
return
package
extends
for
throw
catch
char
int
synchronized
case
instanceof
enum
Java
default
continue
if
abstract
final
boolean
protected
try
double
private
do
this
short
class
public
native
true
implements
while
else
void
switch
static

Thank you!