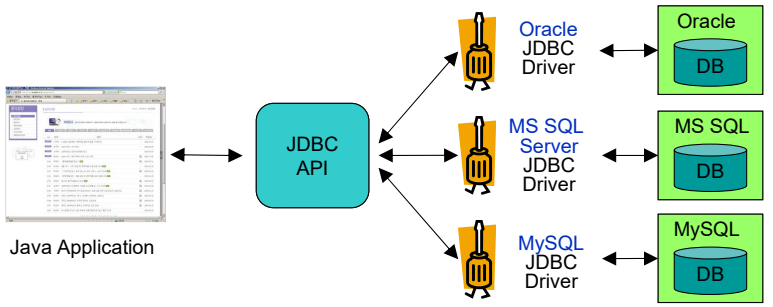
# 7. JDBC Programming

# JDBC

- JDBC(Java Database Connectivity)
  - Java 응용 프로그램에서 DBMS와 연동하기 위한 표준 API
    - JDBC API를 이용함으로써 DBMS의 종류에 상관없이 동일한 방법으로 데이터베이스 접속 및 질의 실행 가능
  - DBMS 접속 및 이용을 위한 interface와 class들을 제공
    - DBMS vendor에서 제공하는 JDBC Driver를 통해 구현됨






# JDBC Driver

- 종류
  - Type 1: JDBC-ODBC Bridge Driver
  - Type 2: Native-API Driver
  - Type 3: Network Protocol Driver
  - Type 4: Native-Protocol Driver(Thin Driver)
    - 100% Java로 구현된 JDBC Driver
    - DBMS에 종속적: 각 vendor에서 JDBC API를 구현한 JDBC Driver를 제공

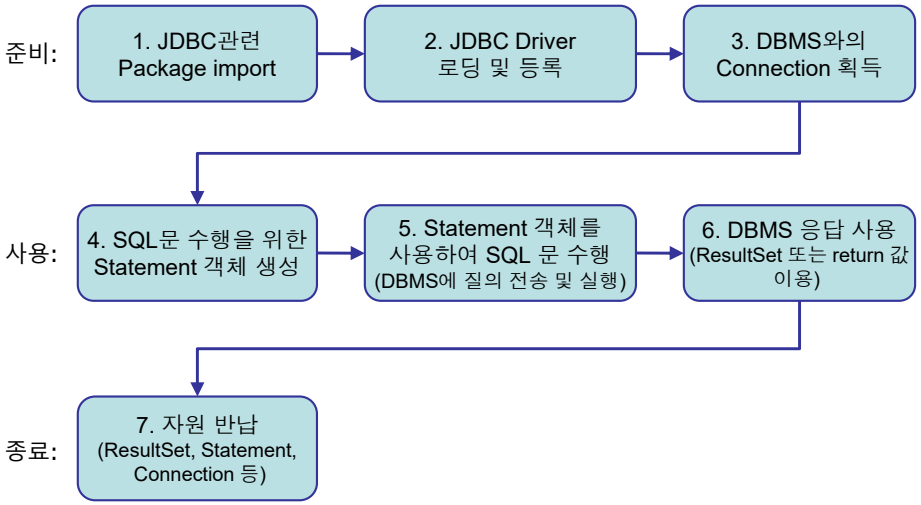
- Oracle JDBC Driver
  - Oracle homepage 또는 Maven repository에서 다운로드 가능
    - JDBC and UCP Downloads page (oracle.com)
      - ✓ ojdbc8.jar: supports JDBC 4.2 spec and for use with JDK 8~
      - ✓ ojdbc11.jar: supports JDBC 4.3 spec and for use with JDK 11~
    - Maven Repository: com.oracle.database.jdbc (mvnrepository.com)



# JDBC API

- 두 개의 package로 구성됨
  - `java.sql` package
    - data source(relational DB)에 저장된 데이터를 접근하고 처리하기 위한 API 제공
      - ✓ Class: `DriverManager`
      - ✓ Interface: `Connection`, `Statement`, `PreparedStatement`, `CallableStatement` , `ResultSet`, `DatabaseMetaData`, `ResultSetMetaData` 등
    - java.sql API Javadoc 참조 (https://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html)
  - `javax.sql` package
    - 서버에 존재하는 Data source를 접근하고 이용하기 위한 API 제공
      - ✓ Interface: `DataSource`, `XADataSource` 등
    - javax.sql API Javadoc 참조 (https://docs.oracle.com/javase/8/docs/api/javax/sql/package-summary.html)

# JDBC API 사용 절차






# JDBC Driver 준비

- JDBC Driver 준비
  - JDK에 설치
    - <JDK 설치폴더>/jre/lib/ext 폴더에 복사
  - Tomcat 서버에 설치
    - <Tomcat 설치폴더>/lib 폴더에 복사
  - Web application에 포함 보통 이 방법을 많이 쓴다
    - Java EE Web application의 경우 WEB-INF/lib 폴더에 포함
  - Java application은 JDBC driver 파일을 classpath에 포함시켜야 사용 가능
    - Eclipse 설정: project의 Properties → Java Build Path의 Libraries tab → Add JARs 또는 Add External JARs → JDBC Driver 파일 선택
  - Maven 설정 예 (pom.xml)

```
<dependency>
    <groupId>com.oracle.database.jdbc</groupId>
    <artifactId>ojdbc8</artifactId>
    <version>21.3.0.0</version>
</dependency>
```



# 1. JDBC Package Import

- Package import
  - Java 프로그램 내에서 `java.sql` package를 import함
    - 예: `import java.sql.*;`



# 2. JDBC Driver Loading 및 등록

- Class.forName(DRIVER_CLASS_NAME)

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

  - DRIVER_CLASS_NAME에 해당하는 클래스를 runtime에 동적으로 로딩
    - 클래스 이름은 프로그램에서 문자열로 지정 (변경 가능)
  - 메모리에 Driver 객체를 생성하고 DriverManager 객체에 등록함
  - JDBC Driver 클래스 예
    - Oracle: oracle.jdbc.driver.OracleDriver
    - MySQL: com.mysql.jdbc.Driver 또는 org.gjt.mm.mysql.Driver

# 3. DBMS와의 연결 획득

◆ 방법 1: DriverManager 이용

정적

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

- 데이터베이스 접속에 필요한 Connection 객체를 구함
- url
  - JDBC를 사용하여 접속할 DBMS Server와 데이터베이스의 주소 표현
  - 형식: jdbc:[DBMSServer주소:port번호]:[데이터베이스식별자]
  - 예:
    - ✓ jdbc:oracle:thin:@dblab.dongduk.ac.kr:1521:orcl (Oracle)
    - ✓ jdbc:mysql://dblab.dongduk.ac.kr:3306/testdb (MySQL)
    - ✓ jdbc:hsqldb:hsql://dblab.dongduk.ac.kr:9001/sample (HSQLDB)
- user, passwd
  - 접속할 데이터베이스 사용자 계정의 이름과 비밀번호



# 3. DBMS와의 연결 획득

◆ 방법 2: javax.sql.DataSource Interface 이용

- Data source를 나타내며, 그것에 대한 연결(connection)을 생성하는 factory 기능 제공
- DriverManager의 대안으로 connection 객체를 구하기 위한 더 좋은 방법
- 일반적으로 Application Server(ex. Tomcat)에서 클래스 구현 및 객체 생성
  - Database connection pooling(DBCP) 기능 지원
  - Java Naming and Directory Interface(JNDI) 서비스를 통해 DataSource 객체에 대한 참조를 제공
- Application에서는 JNDI 서비스를 검색(lookup)하여 DataSource 객체를 획득 및 사용

```
import javax.naming.*;   // Context, InitialContext
import javax.sql.*;      // DataSource
Context ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/OracleDS");
Connection conn = ds.getConnection();
```



# 4. Statement 객체 생성

◆ Connection#createStatement()

```
Statement stmt = conn.createStatement();
```

- 정적인 SQL 문을 실행하고 그 결과를 반환하기 위해 사용되는 객체



# 5. Statement 객체를 이용하여 SQL 문 실행

◆ 질의(SELECT문) 실행

```
ResultSet rs = stmt.executeQuery(query)
```

- *query*: 실행할 SQL 질의문
- ResultSet 타입의 객체를 반환

◆ DML(INSERT, UDPATE, DELETE) 실행

```
int recordCount = stmt.executeUpdate(dml)
```

- *dml*: INSERT, UPDATE, DELETE 문
- 삽입/변경/삭제된 행의 개수를 반환 잘못되면 0이 반환됨(삽입/변경삭제된 것이 없다는 뜻)

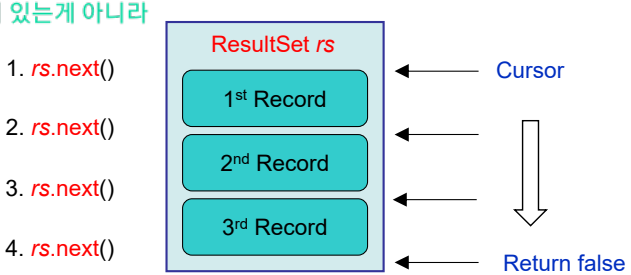
◆ 참고: boolean execute(String sql)

- 질의, DML, DDL 모두 실행 가능
  - 질의 실행 후 결과가 있으면 true 반환 → stmt.getResultSet()으로 ResultSet 구함
  - DML 실행 후 false 반환 → stmt.getUpdateCount()로 행의 개수 구함

# 6. DBMS 응답 사용

- ResultSet
  - Statement#executeQuery()의 실행 결과로 반환되는 행들의 집합을 저장
  - 내부적으로 커서(cursor)를 사용하여 결과 행들을 순차적으로 접근
- ResultSet#next()
  - ResultSet에 저장된 행들을 커서가 순서대로 가리키도록 함
    - 최초 호출 시 첫 번째 행을 가리키고, 이후 호출될 때 마다 커서를 다음 행으로 이동시킴
  - 가리킬 행이 있으면 true, 없으면 false를 반환

레코드들은 테이블에 있는게 아니라 검색 후 넘어온 결과






# 6. DBMS 응답 사용

- ResultSet에서 현재 커서가 가리키는 행의 컬럼 값 읽기
  - ResultSet#getXXX() methods 이용

| method 명 | return type | 기능 |
|---|---|---|
| getString(*columnName*) | String | 컬럼명에 해당하는 컬럼 값을 String으로 읽어 반환 |
| getCharacterStream(*columnName*) | java.io.Reader | 컬럼명에 해당하는 컬럼 값을 문자열 Stream으로 반환 (Reader 객체 반환) |
| getInt(*columnName*) | int | 컬럼명에 해당하는 컬럼 값을 int 타입으로 읽어 반환 |
| getLong(*columnName*) | long | 컬럼명에 해당하는 컬럼 값을 long 타입으로 읽어 반환 |
| getFloat(*columnName*) | float | 컬럼명에 해당하는 컬럼 값을 float 타입으로 읽어 반환 |
| getDouble(*columnName*) | double | 컬럼명에 해당하는 컬럼 값을 double 타입으로 읽어 반환 |
| getDate(*columnName*) | java.sql.Date | 컬럼명에 해당하는 컬럼 값을 Date 객체로 반환 |
| getTime(*columnName*) | java.sql.Time | 컬럼명에 해당하는 컬럼 값을 Time 객체로 반환 |
| getTimestamp(*columnName*) | java.sql.Timestamp | 컬럼명에 해당하는 컬럼 값을 Timestamp 객체로 반환 |

  - 참고: getXXX(int *columnIndex*)
    - 질의의 SELECT 절에 지정된 *columnIndex* 번째 컬럼 값을 반환 (Column index: 1~n)



# 7. 자원 반납

- 데이터베이스 작업이 끝나면 각 자원을 반드시 반환해야 함

```
Connection conn;
Statement stmt;
ResultSet rs;

…

if (rs != null) {                    // ResultSet 객체 존재
      try { rs.close(); }
      catch(SQLException ex) { … }
}
if (stmt != null) {                  // Statement 객체 존재
      try { stmt.close(); }
      catch(SQLException ex) { … }
}
if (conn != null) {                  // Connection 객체 존재
      try { conn.close(); }
      catch(SQLException ex) { … }
}
```

사용한 순서의 역순대로 close()하면 됨



# JDBC Program Example

- StmtEx.java

```
package jdbc.examples;
import java.sql.*;                                   // 1. JDBC 관련 package import

public class StmtEx {
      public static void main(String args[])
      {
            Connection conn = null;
            Statement stmt = null;
            ResultSet rs = null;
            String url = "jdbc:oracle:thin:@localhost:1521/xepdb1“;
            String user = "scott", passwd = "TIGER";
            try {
                  Class.forName("oracle.jdbc.driver.OracleDriver");  // 2. JDBC Driver 로딩 및 등록
            } catch (ClassNotFoundException ex) { ex.printStackTrace(); }
            try {
                  conn = DriverManager.getConnection(url, user, passwd);  // 3. DBMS와의 연결
                                                                          //    획득
                  String query =  "SELECT * FROM EMP WHERE DEPTNO = 10";
                  stmt = conn.createStatement();      // 4. SQL문을 위한 Statement 객체 생성
                  rs = stmt.executeQuery(query);      // 5. Statement 객체를 사용하여 SQL문 실행
```

# JDBC Program Example

- JdbcTest.java (계속)

```
                System.out.println("EmpNo      Name");
                while (rs.next()) { // rs.next() == true                    // 6. DBMS 응답 사용
                        int no = rs.getInt("EMPNO");
                        String name = rs.getString("ENAME");
                        System.out.println(no + "    " + name);
                }
        } catch (SQLException ex) { ex.printStackTrace(); }
        finally {                                                           // 7. 자원 반납
                if (rs != null) {
                        try { rs.close(); } catch (SQLException ex) { ex.printStackTrace(); }
반납할 때도      }
오류 발생 가능  if (stmt != null) {
                        try { stmt.close(); } catch (SQLException ex) { ex.printStackTrace(); }
                }
                if (conn != null) {
                        try { conn.close(); } catch (SQLException ex) { ex.printStackTrace(); }
                }
        }
    }
}
```

# JDBC에서의 SQL 문 사용

- Statement 객체
  - 일반적인 SQL 문 사용 시
  - SQL 문을 실행할 때마다 컴파일 및 실행 과정을 반복
    - SQL 문 compile → compile된 SQL 문 실행 → 실행결과 반환
  - 반복적인 작업 수행 시 DBMS의 부하 증가
- PreparedStatement 객체
  - 객체 생성 시 주어진 SQL 문을 미리 컴파일
    - 반복적인 컴파일을 피함으로써 DBMS 부하 감소 효과
  - SQL 문 내에 매개변수(parameter) 사용 가능
  - 매개변수를 제외하고 구조가 동일한 SQL 문을 반복 실행 시 효과적
- CallableStatement 객체
  - Stored Procedure/Function 호출 시 이용



# Statement 사용 시의 문제점

- 동일한 형태의 SQL 문이 반복적으로 컴파일 됨 SQL: 작은 따옴표
  - SQL 문을 실행할 때마다 DBMS는 SQL 문을 컴파일하고 실행을 준비함

```
INSERT INTO EMP (EMPNO, ENAME, JOB, DEPTNO)
VALUES (7950, 'Tom', 'Analyst', 20);
. . .
INSERT INTO EMP (EMPNO, ENAME, JOB, DEPTNO)
VALUES (7960, 'Jain', 'Researcher', 10);
```

- 코드 작성 및 가독성 문제 자바: 큰 따옴표
  - 문자열 값은 SQL 문 내에서 반드시 따옴표(')로 둘러싸야 함

```
String empName ='Jain', empJob ='Researcher';
String query  = "INSERT INTO EMP (EMPNO, ENAME, JOB, DEPTNO) VALUES "
              + "(" + empNo + " , '" + empName  +  "' , '" + empJob + "' , '" + deptNo + ")";
```



# PreparedStatement

- PreparedStatement를 이용한 DML문 실행
  - SQL 문을 미리 컴파일 한 후 parameter 값은 질의를 실행할 때 설정

```
String dml = "INSERT INTO EMP (EMPNO, ENAME, JOB, DEPTNO)
              VALUES (?, ?, ?, ?)";                          // DML 문 작성

Connection conn = DriverManager.getConnection();             // Connection 생성

// PreparedStatement 객체 생성
PreparedStatement pstmt = conn.prepareStatement(dml);

// parameter 값 설정: ? 자리에 들어갈 값을 지정
pstmt.setInt(1, empNo);                                      1: 첫번째 물음표
pstmt.setString(2, empName);
pstmt.setString(3, empJob);
pstmt.setInt(4, deptNo);

// DML 문 실행
int count = pstmt.executeUpdate();                파라미터 필요없음
                                                  // 주의: method 인자 없음!!!
```

# PreparedStatement

◆ PreparedStatement를 이용한 질의문 실행

```
String query = "SELECT * FROM EMP
                WHERE JOB = ? AND DEPTNO = ?";        // 질의문 작성

Connection conn = DriverManager.getConnection();      // Connection 생성

// PreparedStatement 생성
PreparedStatement pstmt = conn.prepareStatement(query);

// parameter 값 지정
pstmt.setString(1, "CLERK");
pstmt.setInt(2, 30);

// 질의문 실행
ResultSet rs = pstmt.executeQuery();          // 주의: method 인자 없음!!!
```



# PreparedStatement의 set methods

◆ 개요

- 질의에 포함된 parameter(?)들에 대해 값을 지정
- parameter들은 순서에 따라 **1부터 시작**하는 인덱스를 가짐 (1씩 증가)
- 질의문 상에서 값에 해당하는 부분만 지정 가능
  - "SELECT * FROM ?" → 오류 (테이블 이름은 매개변수로 지정 불가)
- parameter의 개수와 set method 호출의 개수가 일치해야 함
  - 불일치할 경우 SQLException 발생

◆ 형식

PreparedStatement#**setXXX**(*INDEX_NO*, *VALUE*);

- XXX : parameter의 데이터 타입
- *INDEX_NO* : parameter의 위치(1부터 시작)
- *VALUE* : parameter의 값



# PreparedStatement의 set methods

◆ 주요 methods

| Method명 | 기능 | 변환되는 SQL 타입 |
|---|---|---|
| setString(int *index*, String *val*) | index 위치에 String 값을 지정 | VARCHAR |
| setCharacterStream(int *index*, Reader *reader*, int *length*) | index 위치에 length 길이의 문자 Stream을 지정 (Reader 객체 이용) | LONG VARCHAR |
| setInt(int *index*, int *val*) | index 위치에 int 값을 지정 | INTEGER |
| setLong(int *index*, long *val*) | index 위치에 long 값을 지정 | BIGINT |
| setFloat(int *index*, float *val*) | index 위치에 float 값을 지정 | REAL |
| setDouble(int *index*, double *val*) | index 위치에 double 값을 지정 | |
| setDate(int *index*, Date *val*) | index 위치에 java.sql.Date 값을 지정 | DATE |
| setTime(int *index*, Time *val*) | index 위치에 java.sql.Time 값을 지정 | TIME |
| setTimestamp(int *index*, Timestamp *val*) | index 위치에 java.sql.Timestamp 값을 지정 | TIMESTAMP |
| setObject(int *index*, Object *val*) | index 위치에 Object 객체를 지정 | 자동 변환 |

- Column index: 1~n



# PreparedStatement 사용 예

```
public class PStmtEx {
    public static void main(String args[]) {
        Connection conn = null;
        PreparedStatement pStmt = null;
        ResultSet rs = null;
        ...
        try {
            conn = DriverManager.getConnection(url, user, passwd); // DBMS와의 연결 획득
            String query = "SELECT EMPNO, JOB FROM EMP WHERE ENAME=? ";
                                                    // parameter가 포함된 질의문 작성
            pStmt = conn.prepareStatement(query);   // PreparedStatement 객체 생성
            pStmt.setString(1, "SMITH");            // parameter 값 설정
            rs = pStmt.executeQuery();              // 질의 실행
            System.out.println("No    JOB");
            while (rs.next()) {                     // DBMS 응답 처리
                int no = rs.getInt("EMPNO");
                String job = rs.getString("JOB");
                System.out.println(no + " " + job);
            }
        } catch (SQLException ex) { ex.printStackTrace(); }
          finally { … }                             // 자원 반납
    } }
```

# Stored Procedure 호출

- 개요
  - DBMS의 Stored Procedure 및 Function을 JDBC API를 통해 호출 가능
  - **CallableStatement** interface사용

- Stored Procedure/Function의 장점
  - 직접적인 query 작성 및 실행이 아닌 Stored Procedure를 통한 간접 호출
    - 복잡도 감소
    - 코드 재사용
    - 성능 및 보안성 향상



# CallableStatement

- CallableStatement 생성 및 실행
  - Connection#prepareCall() method 사용

```
// Call Stored Procedure 문 작성
String storedProc = "{ call PROCEDURE_NAME }";              // parameter 없음
// 또는 String storedProc = "{ call PROCEDURE_NAME (?, ?, …, ?) }";
// Call Function 문 작성
String functionCall = "{ ? = call FUNCTION_NAME (?, ?, …, ?) }";

// Connection 생성
Connection conn = DriverManager.getConnection();

// CallableStatement 객체 생성
CallableStatement cstmt = conn.prepareCall(storedProc or functionCall );

// Stored Procedure/Function 실행
cstmt.execute();
```



# IN mode parameter 처리

- void setXXX(index, value) 사용
  - Stored procedure로 전달될 IN mode parameter 값을 설정

```
// Stored Procedure가 testProc(?, ?) 일 경우
// 첫 번째 parameter는 정수, 두 번째 parameter는 문자열 타입으로 가정
String storedProc = "{ call testProc(?, ?) }";

// Connection 생성
Connection conn = DriverManager.getConnection();

// CallableStatement 생성
CallableStatement cstmt = conn.prepareCall(storedProc);

// IN mode parameter 설정
cstmt.setInt(1, 1000);
cstmt.setString(2, "보너스");

// CallableStatement 실행
cstmt.execute();
```



작성해둔 프로그램으로 데이터베이스에 저장해놓고 plSQL로 사용

# OUT mode parameter 처리

- void registerOutParameter(int *index*, int *JDBCType*) 사용
  - Stored procedure 실행 후 값을 반환받을 OUT mode parameter 설정
    - *index*: parameter의 위치, *JDBCType*: parameter의 데이터 타입

```
// 세 번째 parameter가 OUT mode parameter인 경우
String storedProc = "{ call testProc(?,?,?) }";
CallableStatement cstmt = conn.prepareCall(storedProc);

// IN mode parameter 설정
cstmt.setString(1, stuNo);
cstmt.setString(2, stuName);

// OUT mode parameter 설정
cstmt.registerOutParameter(3, java.sql.Types.VARCHAR);

// CallableStatement 실행
cstmt.execute();

// OUT mode parameter를 통해 결과 조회
String result = cstmt.getString(3);
```

# Function call의 parameter 처리

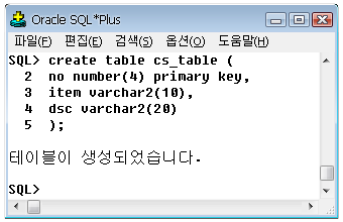
- Stored Procedure와 동일
  - 첫 번째 parameter가 함수의 결과 값을 반환하는 OUT mode parameter

```
// Function call을 위한 호출문 정의
String functionCall = "{ ? = call getStudentName(?) }";
CallableStatement cstmt = conn.prepareCall(functionCall);

// 첫번째 parameter를 OUT mode parameter로 설정
cstmt.registerOutParameter(1, java.sql.Types.VARCHAR);

// IN mode parameter 설정
cstmt.setString(2, stuNo);

// CallableStatement 실행
cstmt.execute();

// 결과 조회
String name = cstmt.getString(1);
```

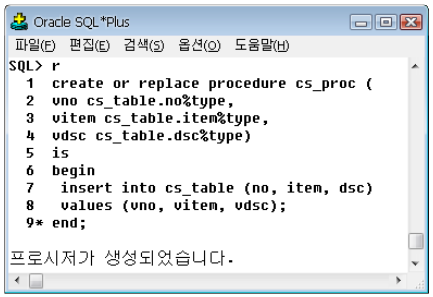


# CallableStatement 사용 예

- 예제 Table 생성
  - cs_table




- Stored Procedure 생성
  - cs_table에 insert 실행



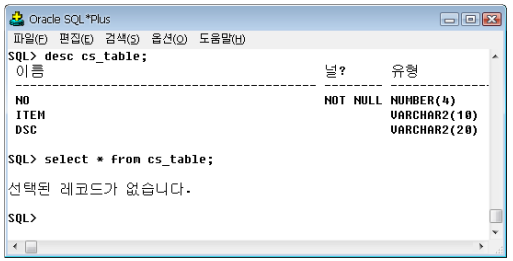



# CallableStatement 사용 예

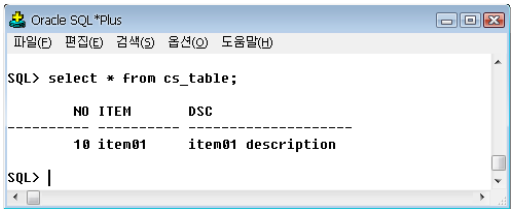
```
public class CStmtEx {
    public static void main(String args[]) {
        Connection conn = null;
        CallableStatement cStmt = null;
        ResultSet rs = null;
        …
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");     // JDBC Driver 로딩 및 등록
        } catch (ClassNotFoundException ex) { ex.printStackTrace(); }
        try {
            conn = DriverManager.getConnection(url, user, passwd); // DBMS와의 연결 획득
            String storedProc = "{call cs_proc(?, ?, ?)}";
            cStmt = conn.prepareCall(storedProc);          // CallableStatement 생성
            cStmt.setInt(1, 10);                           // parameter 값 설정
            cStmt.setString(2, "item01");
            cStmt.setString(3, "item01 is the best one.");
            cStmt.execute();                               // Stored Procedure 호출 실행
            System.out.println ("Insertion completes.");
        } catch (SQLException ex) { ex.printStackTrace(); }
        finally {   …   }                                  // 자원 반납
    }
}
```

# CallableStatement 사용 예

- 실행 전




- 실행 후



# Metadata 사용

- DatabaseMetaData interface: 데이터베이스 전반에 관한 메타정보 제공
  - Connection 객체에서 getMetaData() 실행으로 객체를 구함
  - Methods
    - getDatabaseProductName(), getDatabaseProductVersion(), getUserName(), getSchemas(), getTables(), getColumns(), getPrimaryKeys(), getExportedKeys(), getImportedKeys() 등
- ResultSetMetaData interface: ResultSet 객체에 포함된 컬럼들에 대한 메타정보 제공
  - 질의 실행 후 ResultSet 객체에서 getMetaData() 실행으로 객체를 구함
  - Methods
    - getColumnCount(), getColumnName(int index), getColumnTypeName(int index), getColumnDisplaySize(int index) 등
      - ✓ Column index: 1~n



# Metadata 사용

- DatabaseMetaData 사용 예

```
Connection conn = DriverManager.getConnection(url, user, passwd);
DatabaseMetaData dbMetaData = conn.getMetaData();
String productName = dbMetaData.getDatabaseProductName();
System.out.println("Database: " + productName);
String productVersion =dbMetaData.getDatabaseProductVersion();
System.out.println("Version: " + productVersion);
String username = dbMetaData.getUserName();
System.out.println("UserName: " + username);
System.out.println("List of tables:");
ResultSet  tables = dbMetaData.getTables(null, username, "%", new String[] {"TABLE"});
while (tables.next()) {
      String tableName = tables.getString("TABLE_NAME");
      System.out.println(tableName);
}
```

  - ResultSet **getTables**(String catalog, String schemaPattern, String tableNamePattern, String[] **types**) throws SQLException
    - **types**: "TABLE", "VIEW", "SYSTEM TABLE", "GLOBAL TEMPORARY", "LOCAL TEMPORARY", "ALIAS", "SYNONYM"



# Metadata 사용

  - 실행 결과

```
Database: Oracle
Version: Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production
UserName: SCOTT
List of tables:
DEPT
EMP
SALGRADE
…
```



# Metadata 사용

- ResultSetMetaData 사용 예

```
conn = DriverManager.getConnection(url, user, passwd);
String query =  "SELECT * FROM DEPT";
stmt = conn.createStatement();
rs = stmt.executeQuery(query);
ResultSetMetaData rsMetaData = rs.getMetaData();
System.out.println("Field \tsize\tDataType");
int columnCount = rsMetaData.getColumnCount();
for (int i = 1; i <= columnCount; i++) {
      System.out.print(rsMetaData.getColumnName(i) + " \t");
      System.out.print(rsMetaData.getColumnDisplaySize(i) + "\t");
      System.out.println(rsMetaData.getColumnTypeName(i));
}
for(int i=1; i<=columnCount; i++)
      System.out.print(rsMetaData.getColumnName(i) + " \t");
System.out.println();
while (rs.next()) {
      for(int i=1; i<=columnCount; i++)
            System.out.print(rs.getObject(rsMetaData.getColumnName(i)) + " \t");
      System.out.println();
}
```

# Metadata 사용

- 실행 결과

```
Field          size  DataType
DEPTNO         3     NUMBER
DNAME          14    VARCHAR2
LOC            13    VARCHAR2

DEPTNO         DNAME          LOC
10             ACCOUNTING     NEW YORK
20             RESEARCH       DALLAS
30             SALES          CHICAGO
40             OPERATIONS     BOSTON
```



# References


- Java JDBC API (oracle.com)
  - https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/
- java.sql API Javadoc
  - https://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html
- Introduction to JDBC
  - https://www.baeldung.com/java-jdbc
- JDBC Tutorial - W3schools
  - https://www.w3schools.blog/jdbc-tutorial
- JDBC Developer's Guide and Reference
  - https://docs.oracle.com/en/database/oracle/oracle-database/21/jjdbc/toc.htm