12장-데이터베이스 프로그래밍 기초

TOC

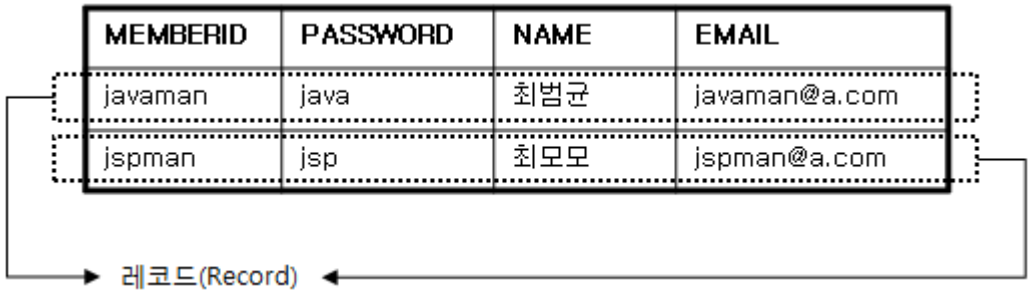
- 데이터베이스 기초
  - DB, 테이블, 주요키
- SQL 기초
- JDBC 프로그래밍
- 커넥션 풀

데이터베이스 & DBMS

- 데이터베이스(database)
  - 빠른 탐색과 검색을 위해 조직된 데이터의 집합체
- DBMS(Database Management System)
  - 데이터베이스를 관리하기 위한 시스템
  - 주요 기능
    - 데이터의 추가/조회/변경/삭제
    - 데이터의 무결성(integrity) 유지
    - 트랜잭션 관리
    - 데이터의 백업 및 복원
    - 데이터 보안

테이블 & 레코드

- 테이블 - 데이터가 저장되는 가상의 장소
- 테이블은 1개 이상의 칼럼으로 구성
  - 각 칼럼은 타입을 가지며, 제약(값의 길이, 가질 수 있는 값 등)을 갖는다.
  - 이런 테이블의 구성을 스키마(schema)라고 함
- 칼럼의 모음을 레코드(record)라고 표현
  - 하나의 테이블은 여러 개의 레코드로 구성

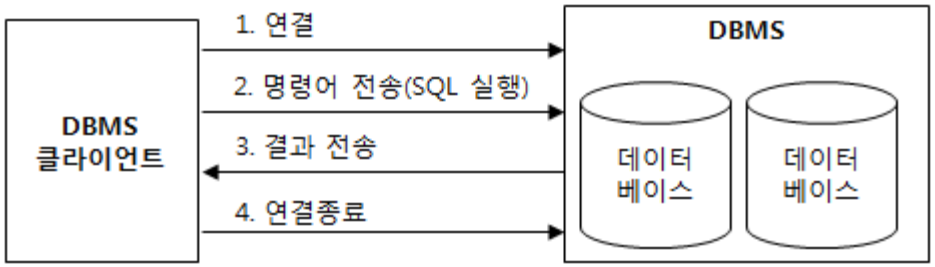


주요키와 인덱스

- 주요키(Primary Key)
  - 각각의 레코드를 구별하기 위해 사용되는 것
  - 각 레코드가 서로 다른 값을 갖는 칼럼
  - 주요키 값을 이용해서 빠른 검색 가능
- 인덱스
  - 지정한 칼럼에 맞춰 데이터의 정렬 순서를 미리 계산
  - 주요키도 인덱스의 종류
  - 인덱스로 사용되는 칼럼은 중복된 값을 가질 수도 있음

데이터베이스 프로그래밍

- 일반적 순서



- 필수 요소
  - DBMS
  - 데이터베이스
  - DBMS 클라이언트

SQL 기초 - 주요 타입

SQL 타입	설명
CHAR	확정 길이의 문자열을 저장. 표준의 경우 255 글자까지만 저장할 수 있다.
VARCHAR	가변 길이의 문자열을 저장. 표준의 경우 255 글자까지만 저장할 수 있다.
LONG VARCHAR	긴 가변 길이의 문자열을 저장
NUMERIC	숫자를 저장
DECIMAL	십진수를 저장
INTEGER	정수를 저장
TIMESTAMP	날짜 및 시간을 저장
TIME	시간을 저장
DATE	날짜를 저장
CLOB	대량의 문자열 데이터를 저장
BLOB	대량의 바이너리 데이터를 저장

SQL

- Structured Query Language
- 데이터 조회, 삭제 등의 데이터베이스 작업을 수행할 때 사용되는 언어
- SQL의 종류
  - DDL (Data Description Language) : 테이블 생성과 같이 데이터를 정의할 대 사용되는 SQL
  - DML (Data Manipluation Language) : 데이터 삽입, 조회, 삭제와 같이 데이터를 다루기 위해 사용되는 SQL

테이블 생성 쿼리

- 구문

```
create table TABLENAME (
    COL_NAME1       COL_TYPE1(LEN1),
    COL_NAME2       COL_TYPE2(LEN2),
    ...,
    COL_NAMEn       COL_TYPEn(LENn)
)
```

- 예

```
create table MEMBER (
   MEMBERID       VARCHAR(10) NOT NULL PRIMARY KEY,
   PASSWORD       VARCHAR(10) NOT NULL,
   NAME           VARCHAR(20) NOT NULL,
   EMAIL          VARCHAR(80)
)
```

데이터 삽입 쿼리

- 구문

```
insert into [테이블이름] ([칼럼1], [칼럼2], .., [칼럼n])
values ([값1], [값2], .., [값n])
```

  - 새로운 레코드를 삽입
  - 칼럼에 대해 값을 설정
  - 칼럼 목록을 지정하지 않은 경우 values 에 모든 칼럼에 대한 값을 지정

- 예

```
insert into MEMBER (MEMBERID, PASSWORD, NAME)
values ('madvirus', '1234', '최범균');
```

데이터 조회 쿼리

- 구문
  - select [칼럼1], [칼럼2], ..., [칼럼n] from [테이블이름]
- 예
  - select MEMBERID, NAME from MEMBER

- where 절
  - 조건에 맞는 레코드 검색
    - select * from MEMBER where NAME = '최범균'
  - and와 or로 다양한 조건 지정 가능
    - where NAME = '최범균' and EMAIL = 'madvirus@madvirus.net'
  - 주요 비교문
    - =, <>, >=, >, <=, <
    - is null, is not null, like

데이터 조회 쿼리 - 정렬, 집합

- order by를 이용한 조회 정렬 순서 지정
  - select .. from [테이블이름] where [조건절]
    order by [칼럼1] asc, [칼럼2] desc, ...

- 집합 관련 함수
  - select max(SALARY), min(SALARY), sum(SALARY) from ...
    - max() - 최대값
    - min() - 최소값
    - sum() - 합

데이터 수정/삭제 쿼리

- 수정 쿼리
  - update [테이블이름] set [칼럼1]=[값1], [칼럼2]=[값2], .. where [조건절]
  - where절을 사용하지 않을 경우 모든 레코드가 수정

- 삭제 쿼리
  - delete from [테이블이름] where [조건절]
  - where 절을 사용하지 않을 경우 모든 레코드가 삭제

조인

- 두 개 이상의 테이블로부터 관련 있는 데이터를 읽어올 때 사용
- 기본 구문

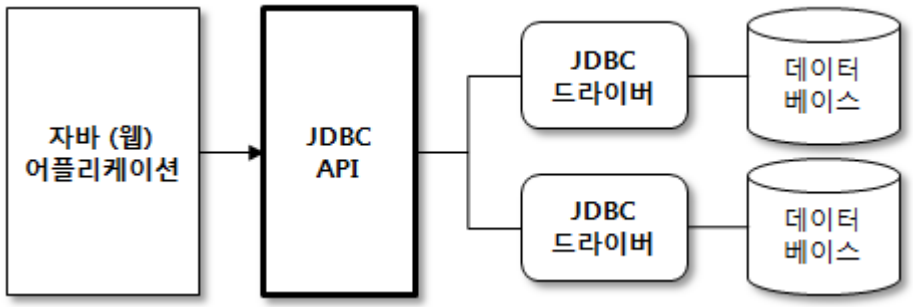
```
select A. 칼럼1, A. 칼럼2, B. 칼럼3, B. 칼럼4
from [테이블1] as A, [테이블2] as B
where A.[ 칼럼x] = B.[ 칼럼y]
```

- 조인 사용에 따른 장단점
  - 다수의 테이블을 한번에 조회할 때 유용
  - 조인이 복잡해 질수록 조회 속도가 느려질 가능성 높음
    - 복잡한 인덱스 설계 등을 필요로 함

※ inner join, outer join 과 같은 내용은 DB 관련 서적 참고

JDBC

- Java Database Connectivity
- 자바에서 DB 프로그래밍을 하기 위해 사용되는 API
- JDBC API 사용 어플리케이션의 기본 구성



  - JDBC 드라이버 : 각 DBMS에 알맞은 클라이언트
    - 보통 jar 파일 형태로 제공

JDBC 프로그래밍 코딩 스타일

```
    // 1. JDBC 드라이버 로딩
    Class.forName("com.mysql.jdbc.Driver");
    Connection conn = null; Statement stmt = null; ResultSet rs = null;
    try {
        // 2. 데이터베이스 커넥션 생성
        conn = DriverManager.getConnection(""jdbc:mysql://localhost:3306/chap12", "user",
"pass");

        // 3. Statement 생성
        stmt = conn.createStatement();
        // 4. 쿼리 실행
        rs = stmt.executeQuery("select * from MEMBER order by MEMBERID");
        // 5. 쿼리 실행 결과 출력
        while(rs.next()) {
            String name = rs.getString(1);
        }
    } catch(SQLException ex) {
        ex.printStackTrace();
    } finally {
        // 6. 사용한 Statement 종료
        if (rs != null) try { rs.close(); } catch(SQLException ex) {}
        if (stmt != null) try { stmt.close(); } catch(SQLException ex) {}
        // 7. 커넥션 종료
        if (conn != null) try { conn.close(); } catch(SQLException ex) {}
    }
```

JDBC 드라이버

- DBMS와 통신을 담당하는 자바 클래스
- DBMS 별로 알맞은 JDBC 드라이버 필요
  - 보통 jar 파일로 제공
- JDBC 드라이버 로딩
  - DBMS와 통신하기 위해서는 먼저 로딩해 주어야 함
  - 로딩 코드
    - Class.forName("JDBC드라이버 클래스의 완전한 이름");
  - 주요 DBMS의 JDBC 드라이버
    - MySQL - com.mysql.jdbc.Driver
    - 오라클 - oracle.jdbc.driver.OracleDriver
    - MS SQL 서버 - com.microsoft.sqlserver.jdbc.SQLServerDriver

JDBC URL

- DBMS와의 연결을 위한 식별 값
- JDBC 드라이버에 따라 형식 다름
- 일반적인 구성
  - jdbc:[DBMS]:[데이터베이스식별자]
- 주요 DBMS의 JDBC URL 구성
  - MySQL : jdbc:mysql://HOST[:PORT]/DBNAME[?param=value&param1=value2&...]
  - Oracle: jdbc:oracle:thin:@HOST:PORT:SID
  - MS SQL : jdbc:sqlserver://HOST[:PORT];databaseName=DB

DB 연결 생성

- DriverManager를 이용해서 Connection 생성
  - DriverManager.getConnection(String jdbcURL)
  - DriverManager.getConnection(String jdbcURL, String user, String password)
- 일반적인 코드 구성

```
Connection conn = null;
try {
    String jdbcDriver = "jdbc:mysql://localhost:3306/chap11?" +
                        "useUnicode=true&characterEncoding=euc-kr";
    String dbUser = "jspexam";
    String dbPass = "jspex";

    conn = DriverManager.getConnection(jdbcDriver, dbUser, dbPass);
    ...
} catch(SQLException ex) {
    // 에러 발생
} finally {
    if (conn != null) try { conn.close(); } catch(SQLException ex) {}
}
```

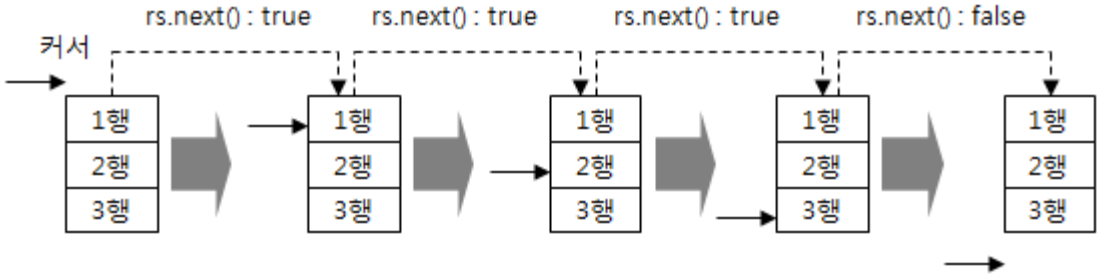
Statement를 이용한 쿼리 실행

- Connection.createStatement()로 Statement 생성
- Statement가 제공하는 메서드로 쿼리 실행
  - ResultSet executeQuery(String query) - SELECT 쿼리를 실행
  - int executeUpdate(String query) - INSERT, UPDATE, DELETE 쿼리를 실행

```
Statement stmt = null;
ResultSet rs = null;
try {
    stmt = conn.createStatement();
    int insertedCount = stmt.executeUpdate("insert .....");
    rs = stmt.executeQuery("select * from ....");
    ...
} catch(SQLException ex) {
    ...
} finally {
    if (rs != null) try { rs.close(); } catch(SQLException ex) {}
    if (stmt != null) try { stmt.close(); } catch(SQLException ex) {}
}
```

ResultSet에서 값 조회

- next() 메서드로 데이터 조회 여부 확인



- 데이터 조회 위한 주요 메서드
  - getString()
  - getInt(), getLong(), getFloat(), getDouble()
  - getTimestamp(), getDate(), getTime()

ResultSet에서 데이터 조회하는 코드

- 1개 행 처리

```
rs = stmt.executeQuery("select * from member");
if (rs.next()) {  // 다음 행(첫 번째 행)이 존재하면 rs.next()는 true를 리턴
    // rs.next()에 의해 다음 행(첫 번째 행)으로 이동
    String name = rs.getString("NAME");
} else {
    // 첫 번째 행이 존재하지 않는다. 즉, 결과가 없다.
}
```

- 1개 이상 행 처리

```
rs = stmt.executeQuery(...);
if (rs.next()) {
    do {
        String name = rs.getString("NAME");
        ...
    } while( rs.next() );
}
```

PreparedStatement를 이용한 처리

- SQL의 틀을 미리 정해 놓고, 나중에 값을 지정하는 방식
- PreparedStatement의 일반적 사용

```
pstmt = conn.prepareStatement(
   "insert into MEMBER (MEMBERID, NAME, EMAIL) values (?, ?, ?)");
pstmt.setString(1, "madvirus"); // 첫번째 물음표의 값 지정
pstmt.setString(2, "최범균");   // 두번째 물음표의 값 지정
pstmt.executeUpdate();
```

- 쿼리 실행 관련 메서드
  - ResultSet executeQuery() - SELECT 쿼리를 실행할 때 사용되며 ResultSet을 결과값으로 리턴한다.
  - int executeUpdate() - INSERT, UPDATE, DELETE 쿼리를 실행할 때 사용되며, 실행 결과 변경된 레코드의 개수를 리턴한다

PreparedStatement의 값 바인딩 관련 메서드

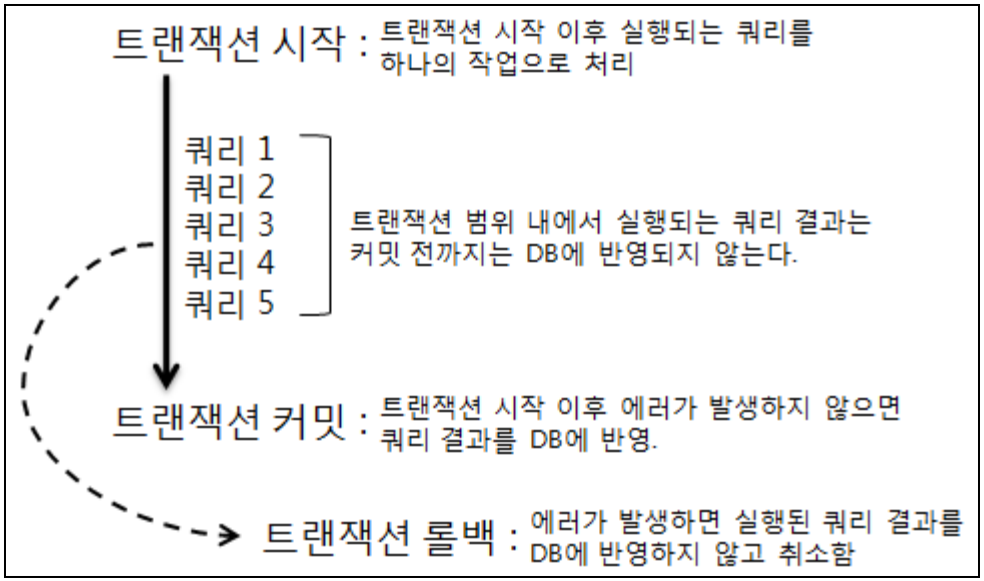
메서드	설명
setString(int index, String x)	지정한 인덱스의 파라미터 값을 x로 지정한다.
setInt(int index, int x)	지정한 인덱스의 파라미터 값을 int 값 x로 지정한다.
setLong(int index, long x)	지정한 인덱스의 파라미터 값을 long 값 x로 지정한다.
setDouble(int index, double x)	지정한 인덱스의 파라미터 값을 double 값 x로 지정한다.
setFloat(int index, float x)	지정한 인덱스의 파라미터 값을 float 값 x로 지정한다.
setTimestamp(int index, Timesetamp x)	지정한 인덱스의 값을 SQL TIMESTMAP 타입을 나타내는 java.sql.Timestamp 타입으로 지정한다.
setDate(int index, Date x)	지정한 인덱스의 값을 SQL DATE 타입을 나타내는 java.sql.Date 타입으로 지정한다.
setTime(int index, Time x)	지정한 인덱스의 값을 SQL TIME 타입을 나타내는 java.sql.Time 타입으로 지정한다.

PreparedStatement의 사용 이유

- 반복해서 실행되는 동일 쿼리의 속도를 향상
  - DBMS가 PreparedStatement와 관련된 쿼리 파싱 회수 감소
- 값 변환 처리
  - 작은 따옴표 등 값에 포함된 특수 문자의 처리
- 코드의 간결함
  - 문자열 연결에 따른 코드의 복잡함 감소

트랜잭션

- 데이터의 무결성을 위해 하나의 작업을 위한 쿼리는 트랜잭션으로 처리될 필요



- 트랜잭션 구현 방법: 오토 커밋 해제, JTA 이용 방식

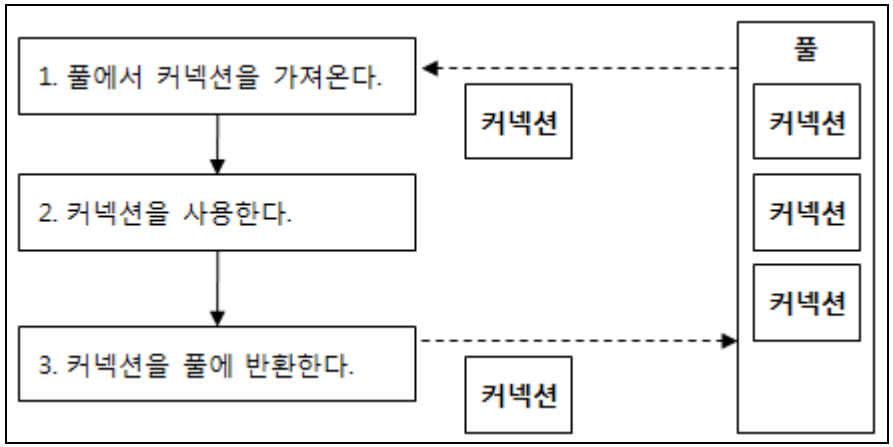
JDBC API에서의 트랜잭션 처리

- Connection.setAutoCommit(false)

```
try {
    conn = DriverManager.getConnection(...);
    // 트랜잭션 시작
    conn.setAutoCommit(false);
    ... // 쿼리 실행
    ... // 쿼리 실행
    // 트랜잭션 커밋
    conn.commit();
} catch(SQLException ex) {
    if (conn != null) {
        // 트랜잭션 롤백
        conn.rollback();
    }
} finally {
    if (conn != null) {
        try {
            conn.close();
        } catch(SQLException ex) {}
    }
}
```

커넥션 풀

- 데이터베이스와 연결된 커넥션을 미리 만들어서 풀(pool) 속에 저장해 두고 있다가 필요할 때에 커넥션을 풀에서 가져다 쓰고 다시 풀에 반환하는 기법



- 특징
  - 커넥션을 생성하는 데 드는 연결 시간이 소비되지 않는다.
  - 커넥션을 재사용하기 때문에 생성되는 커넥션 수가 많지 않다.

DBCP API를 이용한 커넥션 풀 사용

- 필요 라이브러리
  - Commons-DBCP API 관련 Jar 파일: commons-dbcp-1.2.2.jar
  - Commons-Pool API의 Jar 파일: commons-pool-1.4.jar

- DBCP가 제공하는 JDBC 드라이버 로딩
  - Class.forName("org.apache.commons.dbcp.PoolingDriver");
  - 실제 DBMS에 연결할 때 사용될 JDBC 드라이버도 로딩해야 함

- DBCP가 제공하는 커넥션 풀로부터 커넥션 가져오기
  - JDBC URL: jdbc:apache:common:dbcp:설정파일명
  - 예, 클래스패스:/pool.jocl 파일에 설정된 풀 사용
    - DriverManager.getConnection("jdbc:apache:commons:dbcp:/pool");

커넥션 풀 설정 파일 예

```
<object class="org.apache.commons.dbcp.PoolableConnectionFactory"
   xmlns="http://apache.org/xml/xmlns/jakarta/commons/jocl">
   <object class="org.apache.commons.dbcp.DriverManagerConnectionFactory">
      <string value="jdbc:mysql://localhost:3306/chap11" />
      <string value="jspexam" />
      <string value="jspex" />
   </object>
   <object class="org.apache.commons.pool.impl.GenericObjectPool">
      <object class="org.apache.commons.pool.PoolableObjectFactory"
            null="true" />
      <int value="10" />   <!-- maxActive -->
      <byte value="1" />   <!-- whenExhaustedAction -->
      <long value="10000" /> <!-- maxWait -->
      <int value="10" /> <!-- maxIdle -->
      <int value="3" /> <!-- minIdle -->
      <boolean value="true" /> <!-- testOnBorrow -->
      <boolean value="true" /> <!-- testOnReturn -->
      <long value="600000" /> <!-- timeBetweenEvctionRunsMillis -->
      <int value="5" /> <!-- numTestsPerEvictionRun -->
      <long value="3600000" /> <!-- minEvictableIdleTimeMillis -->
      <boolean value="true" /> <!-- testWhileIdle -->
   </object>
   <object class="org.apache.commons.pool.impl.GenericKeyedObjectPoolFactory"
      null="true" />
   <string null="true" />
   <boolean value="false" />
   <boolean value="true" />
</object>
```