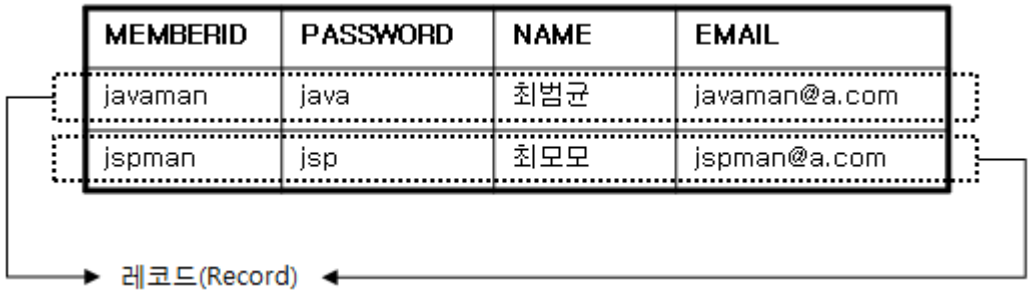
10장 데이터베이스 프로그래밍

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/row)라고 표현
  - 하나의 테이블은 여러 개의 레코드로 구성

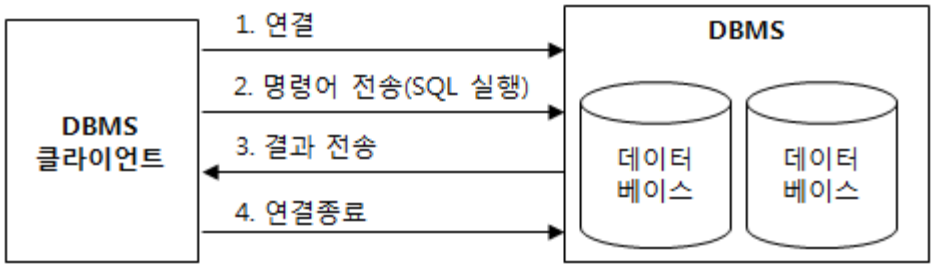


주요키와 인덱스

- 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

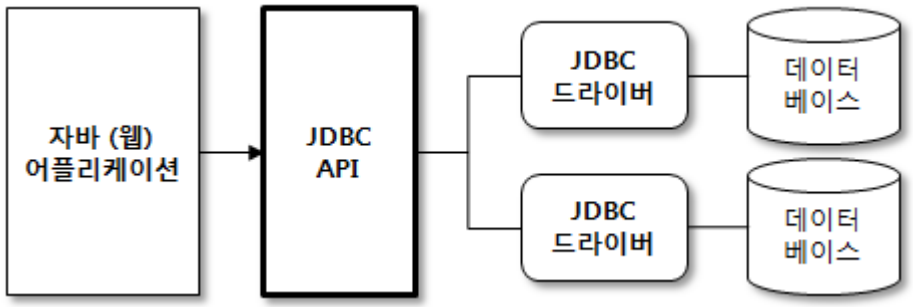
- 일반적 순서



- 필수 요소
  - DBMS
  - 데이터베이스
  - DBMS 클라이언트

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("oracle.jdbc.driver.OracleDriver");
Connection conn = null; PreparedStatement pstmt = null; ResultSet rs = null;
try {
    // 2. 데이터베이스 커넥션 생성
    conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE",
                                       "dev", "123456");
    // 3. Statement 생성
    pstmt = conn.prepareStatement("SELECT * FROM CUSTOMER");
    // 4. 쿼리 실행
    rs = pstmt.executeQuery();
    // 5. 쿼리 실행 결과 출력
    while(rs.next()) {
        System.out.print(rs.getInt(1)    + "₩t");
        System.out.print(rs.getString(2) + "₩t");
        System.out.print(rs.getString(3) + "₩t");
        System.out.print(rs.getString(4) + "₩t");
        System.out.println(rs.getBigDecimal(5));
    }
} catch(SQLException e) { e.printStackTrace();
} finally {
    // 6. 사용한 Statement 종료
    if (rs != null) try { rs.close(); } catch(SQLException e) {}
    if (pstmt != null) try { pstmt.close(); } catch(SQLException e) {}
    // 7. 커넥션 종료
    if (conn != null) try { conn.close(); } catch(SQLException e) {}
}
```

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 url      = "jdbc:oracle:thin:@localhost:1521:XE";
    String dbUser = “dev";
    String dbPass = “123456";

    conn = DriverManager.getConnection(url, dbUser, dbPass);
    ...
} catch(SQLException e) {
    // 에러 발생시 처리할 코드
} finally {
    if (conn != null) try { conn.close(); } catch(SQLException e) {}
}
```

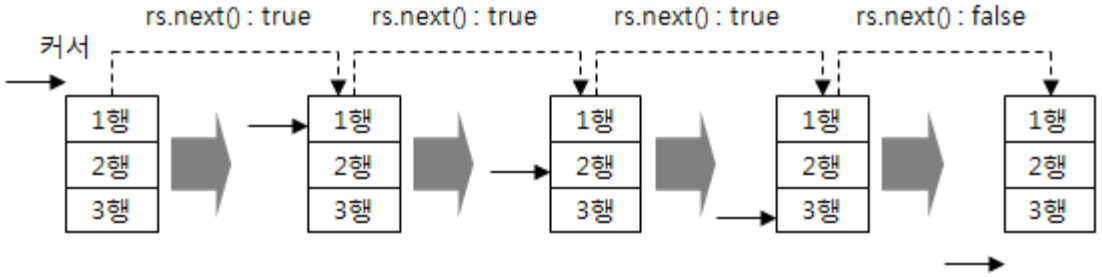
Statement를 이용한 쿼리 실행

- 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, “hong");      // 첫번째 물음표의 값 지정
pstmt.setString(2, “홍길동");    // 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

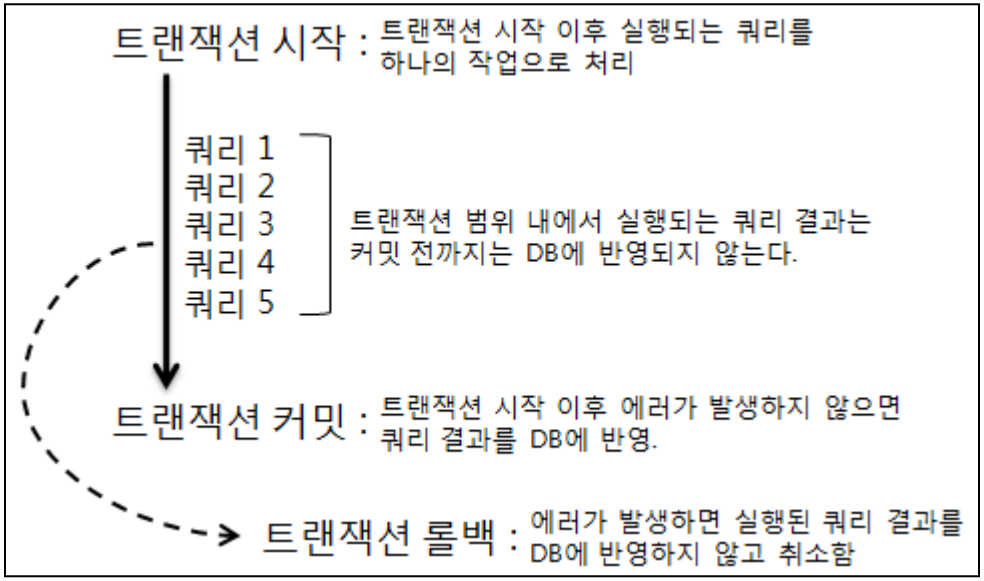
메서드	설명
setString(int index, String x)	지정한 인덱스의 파라미터 값을 x로 지정한다.
setInt(int index, int x)	지정한 인덱스의 파라미터 값을 int 값 x로 지정한다.
setLong(int index, long x)	지정한 인덱스의 파라미터 값을 long 값 x로 지정한다.
setDouble(int index, double x)	지정한 인덱스의 파라미터 값을 double 값 x로 지정한다.
setFloat(int index, float x)	지정한 인덱스의 파라미터 값을 float 값 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



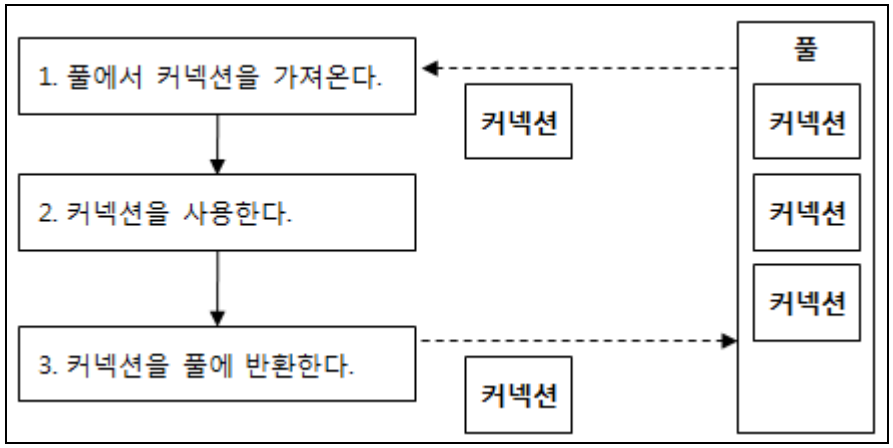
JDBC API에서의 트랜잭션 처리

- 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 필요한 연결 시간이 소비되지 않는다.
  - 커넥션을 재사용하기 때문에 생성되는 커넥션 수가 많지 않다.

커넥션 풀 사용 절차

- context.xml 수정

```
<Resource name="jdbc/jsp" auth="Container" type="javax.sql.DataSource"
          maxActive="50" maxIdle="5" maxWait="-1"
          driverClassName="oracle.jdbc.driver.OracleDriver"
          url="jdbc:oracle:thin:@localhost:1521:XE"
          username="dev" password="123456" />
```

- web.xml 수정

```
<resource-ref>
    <description>DB Connection</description>
    <res-ref-name>jdbc/jsp</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

- 등록된 네이밍 사용

```
Context initCtx = new InitialContext();
Context envCtx = (Context) initCtx.lookup("java:/comp/env");
DataSource ds = (DataSource)envCtx.lookup("jdbc/jsp");
Connection conn = ds.getConnection();
```