

짧고 굵게 배우는

JSP 웹 프로그래밍과 스프링 프레임워크

활화정 지음

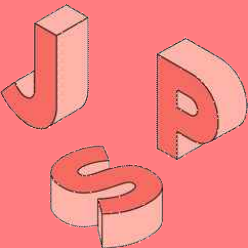
짧고 굵게 배우는
JSP 웹 프로그래밍과
스프링 프레임워크

[강의교안 이용 안내]

- 본 강의교안의 저작권은 황희정과 한빛아카데미㈜에 있습니다.
- 이 자료는 강의 보조 자료로 제공되는 것으로 무단으로 전제하거나 배포하는 것을 금합니다.

Chapter 09

데이터베이스와 JDBC



목차

1. 데이터베이스의 개요
2. 관계형 데이터베이스
3. H2 데이터베이스
4. SQL의 개요
5. [실습 9-1] SQL 실습 : 학생정보
6. 목록 생성
7. JDBC 기본 구조와 API의 이해
8. [실습 9-2] JDBC 종합 실습 : 학생정보b 조회와 등록

학습목표

- 데이터베이스의 개념을 이해하고 나에게 맞는 데이터베이스를 선택할 수 있다.
- 관계형 데이터베이스의 개념을 이해한다.
- H2 데이터베이스를 설치하고 실습한다.
- 테이블을 생성하고 데이터를 다룰 수 있다.
- JDBC의 개념을 이해하고 실습한다.

Section 01

데이터베이스의 개요

데이터베이스란?

- 데이터베이스(Database)
 - 사전적으로 여러 사람이 공유하여 사용할 목적으로 체계화하여 통합, 관리하는 데이터의 집합을 의미함
 - 데이터베이스는 방대한 데이터를 쉽게 검색하거나 찾을 수 있도록 체계적으로 분류하고 정리해둔 정보의 집합
 - 디지털화된 정보만 의미하는 것은 아님

데이터베이스란?

- DBMS(DataBase Management System)
 - 데이터베이스는 DBMS라고 불리는 소프트웨어 시스템을 사용함
 - DBMS는 효과적인 데이터 파일 관리와 운영을 위한 구조와 함께 인덱싱, 캐싱, 네트워크 서버, 사용자 및 권한 관리, 백업/복원, 클러스터링 등 다양한 기능을 제공함
 - DBMS의 종류: Oracle, MySQL, IBM DB2, MS SQL 등
 - 데이터베이스와 데이터베이스 관리 시스템인 DBMS는 다른 의미이지만 보통 데이터베이스라고 하면 DBMS를 포함한 개념을 의미함

데이터베이스란?

- 데이터베이스의 일반적인 특징
 - 데이터 중복을 최소화할 수 있음
 - 데이터를 쉽게 공유할 수 있음
 - 일관성, 무결성, 보안성이 유지됨
 - 최신 데이터를 유지할 수 있음
 - 데이터의 표준화가 가능함
 - 데이터의 논리적·물리적 독립성이 유지됨
 - 데이터 접근이 용이함
 - 데이터 저장 공간을 절약할 수 있음

데이터베이스의 종류

- 관계형 데이터베이스(Relational DataBase Management System, RDBMS)
 - 전통적이고 가장 보편적인 형태의 데이터베이스이며, 관계를 가지는 데이터 구조가 필요한 경우 가장 적합한 데이터베이스임
 - 테이블(Table): 칼럼과 로우 구조로 데이터 구조를 정의하고 관리함
 - SQL(Structured Query Language): 데이터를 관리하는 질의어임. 특히 테이블과 테이블의 관계 지정을 통해 연관성 있는 데이터를 체계적으로 구조화할 수 있음
 - 관계형 데이터베이스의 종류: Oracle, IBM DB2, MS SQL, MySQL 등

데이터베이스의 종류

- 관계형 데이터베이스(Relational DataBase Management System, RDBMS)

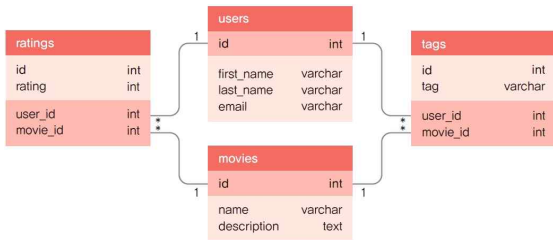


그림 9-1 관계형 데이터베이스의 영화 평점 테이블 구조

데이터베이스의 종류

■ 관계형 데이터베이스의 장점

- 다양한 용도로의 사용이 가능하며 높은 성능을 보여줌
- 데이터의 일관성이 보장됨
- 정규화를 통해 갱신 비용을 최소화할 수 있음
- 구조화되어 있고 동일한 구조를 가지는 데이터를 다룰 때 유리함

■ 관계형 데이터베이스의 단점

- 데이터 구조의 변경(칼럼의 수정이나 확장)이 어려움
- 빠른 속도를 요구하는 단순한 처리에 대응하기 어려움
- 데이터 관계는 유용하지만 그로 인한 처리 속도 저하가 발생할 수 있음

데이터베이스의 종류

■ NoSQL 데이터베이스

- 말 그대로 SQL을 '사용하지 않는다(No)'는 의미로, 'SQL을 사용하는 전통적인 RDBMS가 아니다.'라는 의미로 사용되는 데이터베이스를 말함
- SQL과 다를 수 있지만 데이터 관리를 위해 별도의 쿼리 언어나 구조는 존재함
- NoSQL은 RDBMS의 가장 큰 특징인 테이블 형태의 데이터 구조를 사용하지 않기 때문에 형태가 고정되지 않은 비정형 데이터 처리에 유용함
- NoSQL의 종류: MongoDB, Redis, Casandra, Hbase, CouchDB 등

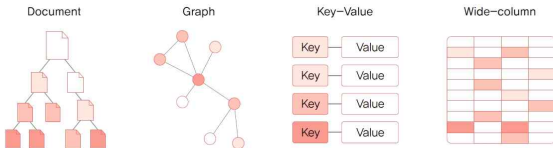


그림 9-2 NoSQL 데이터 구조

데이터베이스의 종류

■ NoSQL 데이터베이스의 장단점

- 대용량 데이터 처리에 유리함
- 분산 처리에 적합함
- 클라우드 컴퓨팅 환경에 적합함
- 빠른 읽기/쓰기 속도를 제공함
- 유연한 데이터 모델링(비정형)이 가능함
- 복잡한 데이터 관계를 표현할 때 중복 데이터가 발생할 수 있음
- 모든 데이터가 동일한 구조를 가지고 있지 않은 경우에 유리함
- 빠른 처리 속도를 위해 필요한 데이터를 다른 테이블 등에서 참조하지 않고 데이터 자체에 포함하는 구조에 적합함

나에게 맞는 데이터베이스 선택하기

■ 데이터베이스의 특징 비교

표 9-1 주요 데이터베이스의 특징 비교

데이터베이스	종류	특징	적용 규모
Oracle	RDBMS	가장 유명한 상용 데이터베이스다. 가격이 비싸고 운영을 위해 전문가가 필요하다.	대규모
MySQL	RDBMS	가장 유명한 공개 데이터베이스다. 오라클에 인수되었으며 무료 사용에 제약이 있다.	중소규모
MariaDB	RDBMS	MySQL을 대체하는 오픈소스 데이터베이스다. MySQL의 오라클 인수로 인한 제약 문제를 해결한 대체품이다.	중소~대규모
H2	RDBMS	경량의 자바 기반 데이터베이스다. 임베디드, 네트워크, 인메모리 등 다양한 운영이 가능하며 오픈소스다.	소규모
MongoDB	NoSQL	가장 유명한 NoSQL 데이터베이스다. 무료에서 상용, 클라우드 서버, 모바일 개발 등 다양한 옵션을 제공한다.	중소~대규모
Firebase	NoSQL	구글에서 제공하는 클라우드 데이터베이스다. 안드로이드, 프론트엔드 개발 및 구글 클라우드 연동에 최적화되어 있다.	중소~대규모

나에게 맞는 데이터베이스 선택하기

- 대형 상용 데이터베이스(Oracle, MS SQL, IBM DB2 등)의 문제점
 - 설치 과정이 복잡하거나 별도의 관리가 필요함
 - 설치 용량이 크고 메모리를 많이 차지함
 - 데이터베이스 운영 과정에서 발생하는 문제 해결에 불필요한 시간이 소요됨

나에게 맞는 데이터베이스 선택하기

■ 자바 및 자바 웹용 데이터베이스

- 일반적으로 자바 및 JSP, 서블릿 등의 자바 웹 프로그래밍을 배우는 과정에서 MySQL을 많이 추천함
- 장점
 - 커뮤니티 버전을 무료로 사용할 수 있음
 - 설치가 비교적 간단하고 관리 도구도 제공하며, 온라인에서 많은 도움을 받을 수 있음
- 단점
 - 설치 과정이 다소 복잡함
 - 네트워크 서버 방식으로만 운영되기 때문에 어느 정도 관리가 필요함

나에게 맞는 데이터베이스 선택하기

- H2 데이터베이스를 추천하는 이유
 - 별도의 설치나 관리가 필요 없음(JDBC 드라이버만으로 사용 가능)
 - 임베디드 모드, 네트워크, 인메모리 등 다양한 운영이 가능함
 - 앱 배포 시 포함이 가능함
 - 용량과 메모리 사용이 적음
 - 표준 SQL을 모두 지원하고 JPA와의 연계도 용이함

Section 02

관계형 데이터베이스

테이블

■ 테이블

- 관계형 데이터베이스에서 데이터 관리의 기본 구조
- 데이터가 가지는 공통적인 속성을 모아 정의한 칼럼(필드)으로 구성됨
- 칼럼에 저장되는 데이터는 숫자형, 문자형, 날짜형, 불형(Boolean) 등으로 구분됨
- 파일과 같은 바이너리 데이터를 저장하거나 매우 긴 텍스트를 저장할 수 있는 자료형도 있음

테이블

- 테이블 만들기의 예
 - 정리되지 않은 데이터

김길동, AA대학교, 1999-10-21, kim@aa.com
park@bb.com, 박사랑, BB대학교, 2000-01-21
나최고, 1998-07-11, na@cc.com, CC대학교
김길동, BB대학교, 1999-03-10, kim@bb.com
AA 대학교, 홍길동, 1999-12-10, hong@aa.com

- 데이터의 공통된 정보를 테이블로 표현(이름, 대학, 생년월일, 이메일)

이름	대학	생년월일	이메일
김길동	AA대학교	1999-10-21	kim@aa.com
박사랑	BB대학교	2000-01-21	park@bb.com
나최고	CC대학교	1998-07-11	na@cc.com
김길동	BB대학교	1999-03-10	kim@bb.com
홍길동	AA 대학교	1999-12-10	hong@aa.com

그림 9-3 학생정보 테이블

테이블

■ 칼럼(Column)


- 테이블을 구성하는 기본 속성으로 필드, 어트리뷰트라고도 불림
- [그림 9-3]에서의 칼럼: 이름, 대학, 생년월일, 이메일
- 각각의 칼럼에 저장되는 데이터는 동일한 타입이어야 하며 구체적인 자료형은 데이터베이스마다 차이가 있음

■ 로우(Row)

- 하나의 데이터셋을 의미하며 레코드 혹은 튜플이라고도 함
- [그림 9-3]에서의 로우: 한 사람의 데이터, 즉 김길동, AA대학교, 1999-10-21, kim@aa.com
 - [그림 9-3]의 로우의 총 개수: 5개

■ 자료형

- 칼럼에 들어갈 수 있는 데이터 유형
- 일반적으로 숫자형, 문자형, 날짜형, 불형(Boolean) 등으로 구분

여기서  **잠깐!** 데이터베이스에서 사용하는 자료형

데이터베이스에는 일반적인 자료형 이외에도 파일과 같은 바이너리 데이터를 저장하거나 매우 긴 텍스트를 저장하기 위한 자료형 등이 별도로 존재하는데, 데이터베이스마다 이러한 특수 타입을 지원하는 범위가 다르다. 또한 최근에는 JSON 데이터의 활용이 늘어나면서 JSON 데이터를 저장하기 위한 칼럼 구조 혹은 변환 함수 등을 제공하기도 한다.

또한 데이터베이스마다 사용하는 자료형의 이름에 차이가 있으니 주의해야 한다. 예를 들어 숫자의 경우 오라클은 number, MySQL은 int, double을 사용한다.

제약 조건

■ 제약 조건(Constraint)

- 칼럼에 부여되는 일종의 속성으로 저장될 데이터에 대한 요구사항
- 제약 조건은 데이터베이스 자체적으로 저장될 데이터에 대한 요구 조건을 설정함
- 따라서 제약 조건을 벗어나는 데이터는 원천적으로 차단됨
- 데이터베이스의 무결성(Integrity)을 지키기 위한 방법이 됨

제약 조건

■ 데이터베이스의 대표적인 제약 조건

표 9-2 데이터베이스 제약 조건

종류	설명	특징
NOT NULL	칼럼에 데이터를 비워둘 수 없다.	기본은 NULL 가능으로 설정되어 있다.
UNIQUE	기본키는 아니지만 중복값이 허용되지 않는다.	기본은 중복 가능으로 설정되어 있다.
Primary Key	데이터의 유일성을 보장하는 칼럼이다.	NOT NULL + UNIQUE, 테이블에 하나만 설정 가능하다.
Foreign Key	두 테이블 간의 참조 무결성을 보장한다.	외래키로 지정된 칼럼값은 참조 테이블에 존재하는 값만 사용 가능하다.
CHECK	저장할 값의 범위 등을 지정한다.	숫자 범위, 다른 칼럼값 비교 결과, 특정 문자 중 하나 등이다.
IDENTITY	유일한 값을 가지는 id 칼럼이다.	PK, Not Null, auto_increment가 자동 적용된다.

■ 키(Key)

- 관계형 데이터베이스의 제약 조건 중 하나
- 데이터의 유일성 및 관계 설정을 위해 사용됨
- 적절한 키의 사용은 데이터베이스 설계에서 매우 중요한 요소임

■ 기본키(Primary Key, PK)

- 주키, 프라이머리키라고도 불림
- 테이블에 저장된 레코드(로우)를 서로 구분할 수 있도록 특정 칼럼에 설정하는 제약 조건
- 기본키가 존재하지 않으면 데이터가 중복되고 특정 데이터 검색에 오류가 발생하기 때문에 꼭 설정되어 있어야 함
- 보통 기본키는 시퀀스(Sequence)라 불리는 단순 증가 값을 사용함
 - 단순 증가 값은 1부터 시작해서 1씩 단계적으로 증가하는 숫자로, 단순히 데이터를 서로 구분하는 용도로만 사용함

■ 학생정보 테이블 변경

- [그림 9-3]에서 김길동 학생이 2명이기 때문에 이름은 중복이 가능함
- 따라서 별도의 중복되지 않는 값이 기본키로 필요함
- 하지만 학번, 주민등록번호 등의 고유번호는 개인정보에 해당하므로 외부에 노출되지 않는 것이 좋기 때문에 기본키로 부적합함

번호(PK)	이름	대학	생년월일	이메일
1	김길동	AA대학교	1999-10-21	kim@aa.com
2	박사랑	BB대학교	2000-01-21	park@bb.com
3	나최고	CC대학교	1998-07-11	na@cc.com
4	김길동	BB대학교	1999-03-10	kim@bb.com
5	홍길동	AA 대학교	1999-12-10	hong@aa.com

그림 9-4 기본키가 추가된 학생정보 테이블

■ 외래키(Foreign Key, FK)

- 테이블 간의 관계를 설정하기 위해 사용하며 참조 무결성을 제공하기 위한 용도
- [그림 9-3]에서 대학 칼럼의 자료형은 문자열로, 다음의 데이터는 모두 동일한 학교를 의미함

AA대학교, AA 대학교, 에이에이 대학교, A A 대학교, AA대학 ...

- 데이터의 의미는 동일하지만 검색에서 문제가 발생함
 - 같은 학교임에도 AA대학교 or AA 대학교 or 에이에이 대학교 등과 같이 조건 검색을 해야 하는 문제가 발생함
- 이러한 문제를 원천적으로 차단하기 위해서는 등록 가능한 대학정보 테이블을 별도로 두고 이를 참조할 수 있도록 외래키를 등록해 사용해야 함

■ 대학정보 테이블과 학생정보 테이블 변경

- 대학코드를 기본키(PK)로 설정함

대학코드(PK)	대학명	위치
10011	AA대학교	서울
10012	BB대학교	인천
10013	CC대학교	대전

그림 9-5 대학정보 테이블

- 학생정보 테이블([그림 9-6])은 다음과 같이 수정됨

- 대학 칼럼은 외래키(FK)로 설정, 대학명 대신 대학정보 테이블의 대학코드를 외래키로 등록

번호(PK)	이름	대학코드(FK)	생년월일	이메일
1	김길동	10011	1999-10-21	kim@aa.com
2	박사랑	10012	2000-01-21	park@bb.com
3	나최고	10013	1998-07-11	na@cc.com
4	김길동	10012	1999-03-10	kim@bb.com
5	홍길동	10011	1999-12-10	hong@aa.com

그림 9-6 외래키가 추가된 학생정보 테이블

■ 대학정보 테이블과 학생정보 테이블 변경

- 학생정보 테이블의 대학 칼럼에 대학정보 테이블에 없는 값을 등록하는 경우
 - 에러가 발생하고 데이터가 등록되지 않음
- 대학코드 테이블에서 특정 대학을 삭제하는 경우
 - 해당 데이터를 학생정보 테이블에서 참조하고 있기 때문에 단순한 삭제 쿼리로는 데이터가 바로 삭제되지 않음

시퀀스, 트랜잭션

■ 시퀀스(Sequence)

- 기본키 칼럼을 관리하기 위해 사용하는 데이터베이스 객체
- 보통 데이터를 추가할 때 순차적으로 증가하는 값을 자동으로 생성함
- 최근의 데이터베이스는 `auto increment`, `auto_increment`와 같은 속성을 칼럼에 추가하는 것으로 자동 증가 값을 사용할 수 있음

■ 트랜잭션(Transaction)

- 프로그램에서 어떤 이벤트가 발생했을 때 여러 테이블의 데이터를 차례로 변경해야 하는 경우가 많이 발생함
- 예) 동일 은행의 A계좌에서 B계좌로 이체한다고 했을 때
 - 계좌는 테이블로 볼 수 있음
 - A계좌에서 100만 원을 차감한 후 B계좌에 100만 원을 추가하면 계좌이체가 완료됨
 - 이때 A계좌에서 차감 후 B계좌에 추가하는 과정에서 에러가 발생하면 A계좌에서 차감했던 금액은 다시 원래대로 되돌려야 함
- 이때 트랜잭션이란 하나의 논리적 기능을 수행하기 위해 여러 작업을 묶어서 처리하는 것을 의미함
- 이러한 트랜잭션은 데이터베이스 혹은 미들웨어 레벨에서 처리할 수 있어야 함

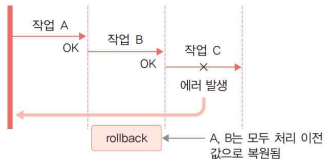
시퀀스, 트랜잭션

■ 데이터베이스에서 제공하는 트랜잭션 관리를 위한 명령

- commit: 모든 데이터의 변화를 실제 적용함
- rollback: 문제 발생 시 현재까지의 변화를 원래대로 되돌림
- A~C 순으로 진행되는 트랜잭션의 처리 과정



(a) commit 처리 과정



(b) rollback 처리 과정

그림 9-7 commit과 rollback 처리 과정

Section 03

H2 데이터베이스

H2 데이터베이스란?

■ H2 데이터베이스

- 보통 임베디드 데이터베이스로 알려져 있으며 MySQL과 같은 관계형 데이터베이스 관리 시스템
- 복잡한 프로그램의 설치가 필요 없고 데이터베이스 파일만 있으면 언제든지 데이터베이스를 실행할 수 있어 프로그램에 포함해서 배포하는 것도 가능함
- 운영 데이터베이스와 상관없이 관계형 데이터베이스를 사용하고 특정 데이터베이스 종속 기능 없이 구현하는 경우 H2와 같은 경량 데이터베이스를 이용해 개발하기도 함

■ H2 데이터베이스의 세 가지 모드

- 임베디드 모드
- 인메모리 모드
- 네트워크 서버 모드

H2 데이터베이스란?

■ 임베디드 모드

- 프로그램에서 JDBC URL을 이용해 접속하거나 console 웹을 통해 관리함
- 데이터 파일만 있으면 동작함
- 가장 간단하고 지정된 데이터 파일이 없다면 자동 생성하므로 언제든지 실행 가능함
- 톰캣이 실행된 상태에서는 별도의 도구를 통해 데이터베이스 관리가 불가능함
- 파일을 사용하기 때문에 동시 다중 접속이 불가능하므로 개발이나 테스트가 불편함

■ 인메모리 모드

- 프로그램에서 JDBC URL을 이용해 접속하거나 console 웹을 통해 관리함

데이터베이스를 메모리상에서만 운영하는 모드로 처리 속도가 빠름

- 프로그램 종료 시 데이터가 소멸되기 때문에 일반적인 애플리케이션에는 부적합함

H2 데이터베이스란?

■ 네트워크 서버 모드

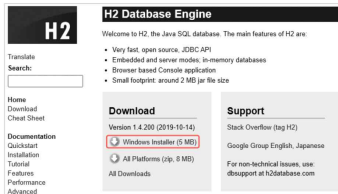
- H2 데이터베이스를 내려받아 설치한 후에 JDBC를 이용해 접속하거나 console 웹을 통해 관리함
- 일반적인 데이터베이스와 같이 네트워크 서버로 동작하는 방식임
- 동시 다중 접속이 가능함
- 별도의 프로그램 실행과 관리가 필요하기 때문에 데이터베이스 서버를 별도로 실행해야 함
- 이번 실습에서는 일반적인 데이터베이스 운영 형태인 네트워크 서버 모드를 사용함

H2 설치

■ H2 데이터베이스 설치하기

- H2 자체가 자바로 구현되어 있기 때문에 H2 데이터베이스 실행을 위해서는 자바가 설치되어 있어야 함

1) H2 데이터베이스 홈페이지(<https://www.h2database.com/>)에 접속하여 최신 버전 (1.4.2)의 H2 데이터베이스를 내려받아 설치하기

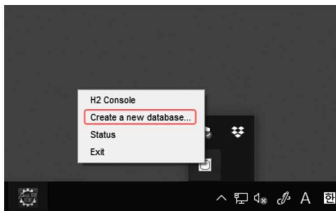


2) 내려받은 설치 프로그램을 실행하면 C:\Program Files (x86)\H2에 설치됨

H2 설치

■ H2 데이터베이스 설치하기

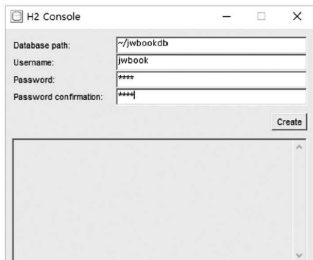
- 3) 설치가 완료되면 윈도우 시작 버튼을 누르고 <H2 console> 앱을 실행함. 앱이 실행되면 자동으로 웹 기반의 관리 콘솔 화면이 뜬다. 이때 윈도우 트레이의 H2 아이콘에서 마우스 오른쪽 버튼을 클릭하여 'Create a new database...'를 선택하기



H2 설치

■ H2 데이터베이스 설치하기

- 4) 데이터베이스 생성 화면에서 다음 내용을 참조해 원하는 이름으로 데이터베이스를 생성함. 작성을 완료한 후 <Create> 버튼을 눌러 DB를 생성하기
- 이때 Database path는 실제 데이터 파일이 생성되는 경로와 파일명이므로 신중히 작성해야 함
 - Database path: ~/jwbookdb
 - Username: jwbook
 - Password: 1234



H2 설치

■ H2 데이터베이스 설치하기

- 5) 정상적으로 데이터베이스가 생성되었으면 다시 콘솔 화면으로 돌아와 연결 시범을 수행하기
 - JDBC URL을 다음과 같이 설정하고 사용자명, 비밀번호를 넣은 다음 <연결 시범> 버튼을 눌렀을 때 '시범 성공'이 나오는지 확인하기



Section 04

SQL의 개요

SQL이란?

- SQL(Structured Query Language)
 - 관계형 데이터베이스에서 데이터를 관리하기 위한 쿼리 언어
 - 로 대부분의 프로그래밍 언어보다는 단순한 구조를 가지고 있음
 - SQL 자체는 표준 언어이지만 데이터베이스마다 세부적인 차이가 있을 수 있기 때문에 데이터베이스와 호환이 되지 않을 수도 있음
 - SQL은 단순히 데이터 관련 작업 이외에 데이터베이스 자체의 관리 기능 수행에도 사용됨

SQL이란?

- SQL에서 할 수 있는 일
 - 새로운 테이블 생성
 - 내장 프로시저Stored Procedure 생성
 - 뷰 생성
 - 테이블, 프로시저, 뷰 등의 접근 권한 부여
 - 데이터베이스에 대해 쿼리 실행
 - 데이터베이스로부터 데이터 조회
 - 데이터베이스에 기록 삽입, 갱신, 삭제
 - 새로운 데이터베이스 생성

SQL의 유형

■ DDL(Data Definition Language)

- 테이블의 생성, 수정, 삭제와 같은 관리 기능을 제공하는 SQL 문
- 스키마, 테이블, 시퀀스, 인덱스, 사용자, 권한 객체를 생성하고 관리하기 위한 명령

■ DML(Data Manipulation Language)

- 테이블의 데이터를 조작할 때 사용하는 SQL 문
- 데이터 조작의 기본 기능인 CRUD(Create, Read, Update, Delete)와 관계된 명령으로 이루어짐

SQL의 유형

■ DDL과 DML

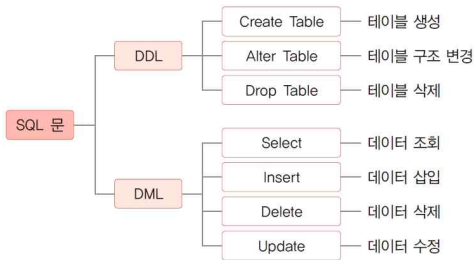


그림 9-8 기본 SQL의 유형

■ H2의 대표적인 자료형

표 9-3 H2 데이터베이스의 자료형

자료형	종류	크기
INT	정수형	32bit
DOUBLE	실수형	64bit
DATE	날짜형	날짜와 시간을 포함
CHAR	문자열	고정길이의 INT 범위
VARCHAR	문자열	가변길이의 INT 범위
CLOB	문자열	INT 범위를 넘어서는 대용량 문자열
BLOB	바이너리형	대용량 바이너리형

■ SQL의 기본 규칙

- 모든 SQL 구문과 식별자는 대소문자를 구분하지 않음
 - 데이터베이스에 따라 구분하는 경우도 있음
- 명령과 키워드는 대문자, 식별자는 정해진 이름 규칙에 따를 것을 권장함
- 식별자는 영문으로 작성하며 공백을 허용하지 않음
- 키워드는 식별자로 사용할 수 없음
- 서술적인 접두어 사용은 자제함
 - 예) tbl_member, idx_, pk_

DDL이란?

■ DDL의 종류

- 테이블의 생성, 수정, 삭제와 같은 관리 기능을 제공하는 SQL 문을 의미함
- DDL을 통해 데이터베이스 스키마, 테이블, 인덱스 등 데이터 저장 및 운영을 위한 객체의 생성과 관리가 가능함
- CREATE, ALTER, DROP, SHOW

DDL이란?

■ CREATE

- 테이블을 생성할 때 사용하는 명령어
- 각 칼럼의 자료형과 최대 크기를 명시해야 함
- 필요에 따라 칼럼에 제약 조건과 속성을 추가할 수 있음
- 마지막 칼럼 설정 뒤에 ';'를 넣지 않도록 주의

```
CREATE TABLE 테이블_이름 (  
    칼럼_이름 자료형(크기) 제약 조건 | 속성,  
    칼럼_이름 자료형(크기),  
    ...  
)
```

DDL이란?

■ ALTER

- 테이블 구조를 수정할 때 사용하는 명령어
- 테이블에 데이터가 들어가 있는 상태에서는 구조 변경에 여러 제약이 따름
- 따라서 테이블을 수정하기 전에 신중히 테이블을 생성하는 것이 좋음

```
ALTER TABLE 테이블_이름 [ADD | ALTER | DROP] 컬럼명 자료형 제약 조건
```

- 테이블에 데이터가 들어가 있을 때 수정 제약 사항
 - 컬럼의 자료형은 변경할 수 없음
 - 컬럼의 크기를 줄일 수는 없고 늘리는 것만 가능함
 - NOT NULL 속성을 갖는 필드는 추가할 수 있으나, NULL 속성이 있는 필드는 추가할 수 없음

DDL이란?

■ DROP

- 테이블 자체를 삭제하는 명령어
- 데이터와 함께 테이블과 연관되어 정의된 인덱스, 룰, 트리거, 제약 조건도 함께 삭제되므로 주의해야 함

```
DROP TABLE 테이블_이름 [RESTRICT | CASCADE]
```

- RESTRICT: 기본값으로 삭제 테이블이 다른 곳에서 참조되고 있다면 삭제를 중지함
- CASCADE: 삭제 테이블과 의존관계가 있는 모든 개체를 함께 삭제함

```
DROP TABLE student  
DROP TABLE student CASCADE
```

DDL이란?

■ SHOW

- 테이블 정보를 조회하기 위해서는 데이터베이스마다 제공되는 별도의 명령 혹은 스키마 구조에 접근하는 쿼리를 사용해야 함
- H2에서는 다음과 같이 SHOW 명령어를 이용하여 테이블 정보를 확인할 수 있음

```
SHOW COLUMNS FROM student
```

DML이란?

- DML의 종류
 - 테이블의 데이터를 조작할 때 사용하는 SQL 문
 - 데이터의 입력, 수정, 삭제, 검색 등에 사용됨
 - 주로 프로그램 코드 안에서 사용하는 쿼리문
 - INSERT, SELECT, DELETE, UPDATE

DML이란?

■ INSERT

- 테이블에 데이터를 추가하기 위한 명령어
- 전체 칼럼값을 모두 추가하는 경우
 - VALUES에 오는 값의 순서는 테이블을 생성할 때 지정한 칼럼 순서와 반드시 일치해야 함

// 전체 칼럼값을 모두 저장하는 경우

```
INSERT INTO 테이블_이름 VALUES(칼럼에 넣을 데이터)
```

```
INSERT INTO student VALUES('김길동','AA대학교','1999-10-21','kim@aa.com')
```

- 부분 칼럼 데이터만 저장하는 경우
 - NOT NULL 칼럼은 반드시 포함되어야 하며 auto_increment 속성이 적용될 칼럼은 비움

// 특정 칼럼값만 저장하는 경우

```
INSERT INTO 테이블_이름(칼럼_이름1, 칼럼_이름2...) VALUES(칼럼에 넣을 데이터)
```

```
INSERT INTO student(username, email) VALUES('김길동','kim@aa.com')
```


DML이란?

■ SELECT

- 테이블에서 데이터를 조회하기 위한 명령어
- 전체 데이터 혹은 조건에 맞는 데이터만 조회가 가능함
- 데이터베이스에서 제일 중요한 쿼리임
- 효율적인 조회를 위한 작업
 - 여러 테이블의 데이터를 조합해서 조회하기
 - 외래키 칼럼의 코드 데이터를 참조 테이블의 이름 칼럼으로 대체하기
 - 날짜 형식 변경하기
 - 데이터 정렬 또는 집계하기

DML이란?

■ SELECT를 통한 단일 테이블 조회

```
SELECT [ALL | DISTINCT] 칼럼명(혹은 *)..  
FROM 테이블_이름  
[ WHERE 조건절 ]  
[ GROUP BY 칼럼명 ]  
[ HAVING 검색조건 ]  
[ ORDER BY 칼럼명 [ ASC | DESC ]
```

전체 조회

- 특정 칼럼을 지정하거나 전체 칼럼(*)을 조회할 수 있음
- DISTINCT: 중복된 값은 제거하고 가지고 옴
- WHERE: 검색 조건을 지정할 때 사용함
- GROUP BY: 특정 칼럼을 그룹화할 때 사용함
- HAVING: 특정 칼럼을 그룹화한 결과에 조건을 설정할 때 사용함
- ORDER BY: 특정 칼럼을 기준으로 오름차순(ASC)/내림차순(DESC) 정렬할 때 사용함

여기서 잠깐! WHERE 조건절의 사용

조건절에서는 일반적인 비교연산, 논리연산과 LIKE 연산 등이 가능하다. LIKE 검색의 경우 %를 이용해 특정 단어를 포함하거나 특정 단어로 시작 또는 끝나는 데이터를 검색할 수 있으나 컬럼의 데이터를 모두 비교하기 때문에 별도의 인덱스 설정이 필요하다. 하지만 조건절을 남용할 경우 성능 저하의 원인이 될 수 있으니 사용에 주의한다.

데이터 조회 함수

■ 데이터 조회 함수란?

- 데이터 조회의 편의를 위해 제공되는 데이터베이스마다의 전용 함수
- 이번 예제는 데이터 조회 함수 테스트를 위해 가상의 테이블인 DUAL과 SELECT 문을 사용함

■ 숫자 관련 함수

- 숫자를 조작하기 위한 함수
- ABS(절댓값), CEILING(올림), ROUND(반올림), FLOOR(버림), SQRT(제곱근) 등

```
SELECT ABS(-20), CEILING(20,25), ROUND(20,25), FLOOR(20,25), SQRT(4) from dual
```

20	21	20	20	2.0
----	----	----	----	-----

임의의 가상 테이블

데이터 조회 함수

■ 문자 관련 함수

- 문자, 문자열을 조작하기 위한 함수
- ASCII(아스키코드값), LENGTH(길이), CONCAT(문자열 결합), TRIM(양쪽 공백 제거)
- LOWER(소문자 변환), UPPER(대문자 변환), SUBSTRING(부분 선택) 등

```
SELECT ASCII('A'), LENGTH('HELLO'), CONCAT('Hello','World'), TRIM('Hello World'),  
       LOWER('ABC'), UPPER('abc'), SUBSTRING('HelloWorld' FROM 2 FOR 5) from dual
```

65	5	'HelloWorld'	'Hello World'	'abc'	'ABC'	'elloW'
----	---	--------------	---------------	-------	-------	---------

데이터 조회 함수

■ 날짜/기간 함수

- NOW(현재 날짜 시간), CURRENT_TIMESTAMP(현재 날짜 시간), DAYNAME(요일)
- PARSEDATETIME(문자열 포맷을 날짜 시간 정보로 변환) 등

```
SELECT NOW(), CURRENT_TIMESTAMP(),  
       DAYNAME(now()), PARSEDATETIME('10-01-2020','MM-dd-yy','GMT') from dual
```

NOW()	CURRENT_TIMESTAMP	DAYNAME(NOW())	TIMESTAMP '2020-10-01 00:00:00'
2021-04-01 12:07:47.297208	2021-04-01 12:07:47.297208+09	Thursday	2020-10-01 00:00:00

■ 집계 함수

- COUNT(레코드 수), SUM(칼럼값 더하기), AVG(칼럼값 평균)
- MAX(칼럼 최댓값), MIN(칼럼 최솟값) 등

```
SELECT COUNT(id), SUM(id), AVG(id), MAX(id), MIN(id) FROM student
```

COUNT(ID)	SUM(ID)	AVG(ID)	MAX(ID)	MIN(ID)
5	15	3	5	1

■ 조인(Join)

- 관계형 데이터베이스에서 2개 이상의 테이블이나 데이터베이스를 조합해 데이터를 검색하는 것
- 조회하고자 하는 칼럼이 서로 다른 테이블에 있을 경우에 주로 사용함
- 여러 개의 테이블을 마치 하나의 테이블인 것처럼 사용할 수 있는 방법임
- 기본키(PK)와 외래키(FK)로 연결된 두 테이블의 데이터를 조합하기 위해 사용할 수 있음
- 조인은 여러 유형이 있으며 조인을 통해 여러 번 쿼리를 보내거나 결과를 프로그램에서 조합할 필요 없이 한 번의 쿼리로 원하는 데이터 구조를 받아볼 수 있음
- 조인 형태: Inner Join, Outer Join, Cross Join, Self Join 등

Section 05

[실습 9-1] SQL 실습 : 학생정보 목록 생성

- 이번 실습의 개요

- SQL을 이용해 데이터베이스를 사용하려면 SQL 클라이언트 프로그램이 필요함
- 이번 실습에서는 H2 데이터베이스를 실행하면 함께 실행되는 데이터베이스 관리 프로그램인 H2 콘솔 프로그램을 사용함
- H2 콘솔은 웹 기반의 도구로 간단한 관리 기능과 함께 SQL 편집 기능도 제공함
- 웹 기반이기 때문에 별도의 프로그램 설치 없이 사용할 수 있는 것이 장점임

- 1) 윈도우에서 'H2 Console'을 검색하면 나오는 프로그램을 실행하기
- 2) <연결 시험>에 성공하면 <연결> 버튼을 누르고 메인 화면으로 이동함
 - 편집기 영역에서 여러 개의 SQL 구문을 이어서 작성하는 것이 가능함
 - 하지만 실행할 때는 블록을 지정해 원하는 SQL만 실행하는 것이 좋음

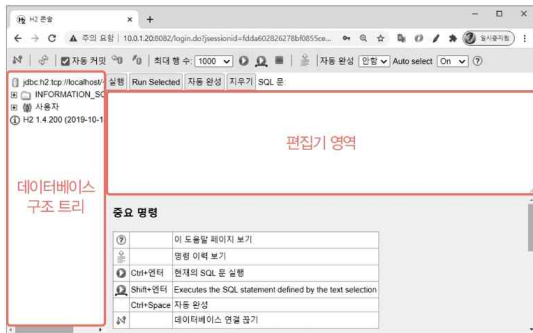


그림 9-9 H2 콘솔 메인 화면

- **CREATE TABLE**

- 1) 우선 학생정보 테이블을 생성하기.

- SQL은 보통 소문자로 작성하지만, 이번 실습에서는 구별을 위해 SQL 키워드는 대문자로 작성함

```
CREATE TABLE student (  
    id INT NOT NULL Primary Key AUTO_INCREMENT,  
    username VARCHAR(20),  
    univ VARCHAR(40),  
    birth DATE,  
    email VARCHAR(40)  
);
```

- NOT NULL: 테이블에 데이터를 추가할 때 반드시 데이터를 넣어야 하는 속성
- Primary Key: 기본키로 설정
- AUTO_INCREMENT: 자동 증가 칼럼(시퀀스)으로 지정하는 속성
 - 중복되지 않는 순차 증가 값으로 기본키 칼럼에 주로 사용됨

- 2) 작성을 완료했다면 편집기 상단의 <실행> 버튼을 눌러 실행하기
 - 이후부터는 <Run Selected> 버튼을 눌러 실행해야 함
- 3) 실행되면 왼쪽 트리에 STUDENT 테이블이 생성된 것을 확인할 수 있음

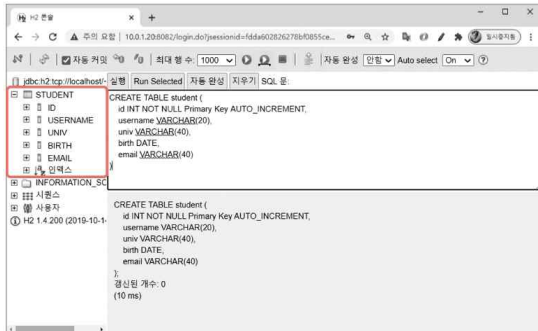


그림 9-10 테이블 생성 확인

- ALTER TABLE

- 4) 생성된 STUDENT 테이블에 tel 칼럼을 추가하고, username의 크기를 20에서 10으로 조정하기. 그리고 id 칼럼에 지정되었던 기본키 제약 조건을 제거해보기

```
ALTER TABLE student ADD(tel VARCHAR(30) NOT NULL);  
ALTER TABLE student ALTER username VARCHAR(10);  
ALTER TABLE student DROP PRIMARY KEY;
```

- 라인 단위로 블록을 지정해 실행해도 되고, SQL 구문 끝에 ';'으로 구분을 해주었다면 차례로 3개의 구문을 한번에 실행할 수도 있음
- 5) 실행 결과 편집기 하단에 성공 메시지가 뜨면 정상 실행된 것
 - 6) 변경된 테이블 구조를 확인하기 위해 SHOW 구문을 실행해 비교하기

```
SHOW COLUMNS FROM student;
```

- **DROP TABLE**

- 7) 테이블을 삭제할 때 삭제하고자 하는 테이블과 연관된 테이블이 있다면, CASCADE를 함께 사용해야 삭제가 가능함
- 이번에는 앞에서 만든 단일 테이블 student를 삭제하기 때문에 CASCADE를 사용하지 않음
 - 테이블이 삭제되면 왼쪽 트리에서도 삭제됨

```
DROP TABLE student;
```

- INSERT

- 1) student 테이블의 id 칼럼에는 auto_increment 속성이 부여되어 있기 때문에 다음과 같이 칼럼을 지정한 다음, 데이터를 부분적으로 추가해야 함

```
INSERT INTO student(username, univ, birth, email) VALUES('김길동', 'AA대학교', '1999-10-21', 'kim@aa.com');
```

- 2) 다음 예시를 참고해 4개의 데이터를 더 추가해보기

- 추가된 데이터는 이어지는 SELECT 실습에서 확인함

```
INSERT INTO student(username, univ, birth, email) VALUES('박사랑', 'BB대학교', '2000-1-21', 'park@bb.com');  
INSERT INTO student(username, univ, birth, email) VALUES('나최고', 'CC대학교', '1998-7-11', 'na@cc.com');  
INSERT INTO student(username, univ, birth, email) VALUES('김길동', 'BB대학교', '1999-03-10', 'kim@bb.com');  
INSERT INTO student(username, univ, birth, email) VALUES('홍길동', 'AA대학교', '1999-12-10', 'hong@aa.com');
```

- **SELECT**

3) DML 중에서 가장 많이 사용하는 명령 우선 가장 기본적인 구문을 살펴봄

- 1), 2)번에서 id 칼럼은 입력하지 않았지만 조회할 때 1부터 차례대로 값이 들어가 있는 것을 확인할 수 있음

```
SELECT * FROM student;
```

ID	USERNAME	UNIV	BIRTH	EMAIL
1	김길동	AA대학교	1999-10-21	kim@aa.com
2	박사랑	BB대학교	2000-01-21	park@bb.com
3	나최고	CC대학교	1998-07-11	na@cc.com
4	김길동	BB대학교	1999-03-10	kim@bb.com
5	홍길동	AA대학교	1999-12-10	hong@aa.com

- 특정 칼럼만 지정해서 조회하는 것도 가능함

```
SELECT username, email FROM student;
```


- SELECT

4) WHERE 조건절을 사용해 다양한 조건으로 데이터를 조회하기

- AND, OR를 이용해 조건을 세분화할 수도 있음

```
SELECT * FROM student WHERE username = '김길동';
```

ID	USERNAME	UNIV	BIRTH	EMAIL
1	김길동	AA대학교	1999-10-21	kim@aa.com
4	김길동	BB대학교	1999-03-10	kim@bb.com

```
SELECT * FROM student WHERE username = '김길동' AND univ = 'AA대학교';
```

ID	USERNAME	UNIV	BIRTH	EMAIL
1	김길동	AA대학교	1999-10-21	kim@aa.com

```
SELECT * FROM student WHERE univ = 'AA대학교' OR univ = 'CC대학교';
```

ID	USERNAME	UNIV	BIRTH	EMAIL
1	김길동	AA대학교	1999-10-21	kim@aa.com
3	나희고	CC대학교	1998-07-11	na@cc.com
5	홍길동	AA대학교	1999-12-10	hong@aa.com

// '김'으로 시작하는 모든 데이터 조회

```
SELECT * FROM student WHERE username LIKE '김%';
```

ID	USERNAME	UNIV	BIRTH	EMAIL
1	김길동	AA대학교	1999-10-21	kim@aa.com
4	김길동	BB대학교	1999-03-10	kim@bb.com

- SELECT

5) 정렬을 위한 ORDER BY 사용하기

// 이름을 내림차순으로 정렬

```
SELECT * FROM student ORDER BY username DESC;
```

ID	USERNAME	UNIV	BIRTH	EMAIL
5	홍길동	AA대학교	1999-12-10	hong@aa.com
2	박사랑	BB대학교	2000-01-21	park@bb.com
3	나최고	CC대학교	1998-07-11	na@cc.com
1	김길동	AA대학교	1999-10-21	kim@aa.com
4	김길동	BB대학교	1999-03-10	kim@bb.com

// 학생이름은 오름차순으로 대학은 내림차순으로 정렬

```
SELECT * FROM student ORDER BY username ASC, univ DESC;
```

ID	USERNAME	UNIV	BIRTH	EMAIL
4	김길동	BB대학교	1999-03-10	kim@bb.com
1	김길동	AA대학교	1999-10-21	kim@aa.com
3	나최고	CC대학교	1998-07-11	na@cc.com
2	박사랑	BB대학교	2000-01-21	park@bb.com
5	홍길동	AA대학교	1999-12-10	hong@aa.com

- **SELECT**

6) 중복된 데이터는 하나만 표시하기 위해 DISTINCT를 사용하기

```
// 중복된 이름은 하나만 가져옴
```

```
SELECT DISTINCT username FROM student;
```

USERNAME
김길동
나최고
박사랑
홍길동

```
// 이름과 대학의 조합으로 중복 제거
```

```
SELECT DISTINCT username, univ FROM student;
```

USERNAME	UNIV
김길동	AA대학교
김길동	BB대학교
나최고	CC대학교
박사랑	BB대학교
홍길동	AA대학교

Section 06

JDBC 기본 구조와 API의 이해

■ JDBC의 등장 배경

- 데이터베이스의 종류가 다양하기 때문에 개발에 많은 어려움이 있음
- JDBC는 이러한 문제를 해결하기 위한 자바 애플리케이션에서 표준화된 방법으로 다양한 데이터베이스에 접속할 수 있도록 설계된 인터페이스임
- 따라서 애플리케이션 개발자는 각 데이터베이스에 대해 자세히 알지 못해도 JDBC API만 알면 모든 데이터베이스에서 동작할 수 있는 애플리케이션을 개발할 수 있음
- 응용 프로그램에서는 자바에 기본적으로 포함된 JDBC API(인터페이스로 규격만 정의하고 있음)를 사용해 프로그램 코드를 작성하고 실제 데이터베이스 연결은 각 데이터베이스 회사가 제공하는 JDBC 드라이버(JDBC API 구현 클래스)를 이용해 SQL 문으로 데이터를 조작하는 형태로 동작한다.

JDBC의 개념

■ JDBC의 구조

- JDBC API: 응용 프로그램에서는 자바에 기본적으로 포함된 JDBC API를 사용해 프로그램 코드를 작성함
- JDBC 드라이버(JDBC API 구현 클래스): 실제 데이터베이스 연결은 각 데이터베이스 회사가 제공하는 JDBC 드라이버를 이용함
- SQL 문으로 데이터를 조작하는 형태로 동작함

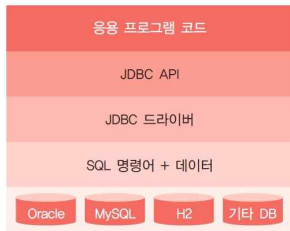


그림 9-11 JDBC 구조

JDBC 드라이버 설치

■ JDBC 드라이버

- 자바 인터페이스로 정의된 일종의 규격이기 때문에 실제 구현된 클래스가 없으면 동작하지 않음
- 이때 실제 구현된 클래스를 JDBC 드라이버라고 부르며 라이브러리와 같은 개념임
- JDBC API를 사용하는 프로그램을 개발하거나 실행하는 과정에 해당 라이브러리가 반드시 필요함
 - 7장에서 Maven 기반으로 프로젝트를 변경했기 때문에 다른 라이브러리와 마찬가지로 pom.xml에 의존성을 추가하면 자동으로 프로젝트에서 참조됨
- JDBC 드라이버는 보통 데이터베이스를 만드는 회사에서 직접 배포함

JDBC 드라이버 설치

- 이클립스에 H2 데이터베이스 드라이버를 추가하기
 - 'pom.xml'의 **<dependencies> ...</dependencies>** 사이에 다음과 같이 H2 데이터베이스 의존성을 추가하기

```
<dependencies>
  <!-- https://mvnrepository.com/artifact/com.h2database/h2 -->
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.4.200</version>
  </dependency>
</dependencies>
```

- 파일을 저장하면 자동으로 h2 라이브러리 다운로드 과정이 백그라운드로 진행됨
- [Java Resources] → [Libraries] → [Maven Dependencies]에 'h2-1.4.200.jar'이 추가되어 있으면 정상적으로 등록된 것임

■ JDBC 프로그래밍의 기본 단계와 사용 클래스

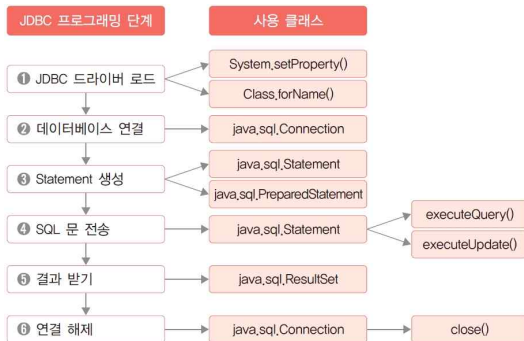


그림 9-12 JDBC 프로그래밍 과정

■ 1단계: JDBC 드라이버 로드

- 먼저 해당 데이터베이스의 JDBC 드라이버를 로드해야 함
- JDBC 드라이버를 로드하는 방법
 - ① jdbc.drivers라는 시스템 환경 변수에 등록된 내용으로 하는 방법
 - ② Class.forName() 메서드를 이용해서 직접 해당 클래스를 로드하는 방법
- 대부분의 경우 다음 코드와 같이 Class.forName() 메서드를 사용함

```
Class.forName("org.h2.Driver");
```

- 드라이버 로드에서 사용되는 org.h2.Driver는 H2 데이터베이스의 드라이버 클래스로 이름을 잘못 명시하면 오류 발생함

■ 2단계: 데이터베이스 연결

- 드라이버가 로드되면 해당 데이터베이스의 JDBC 드라이버를 이용해 프로그램을 작성할 수 있는 상태가 된 것을 의미함
- 실제 데이터베이스와 연결하려면 Connection 클래스의 인스턴스가 필요함
- DriverManager.getConnection() 메서드를 이용해서 레퍼런스를 가져올 수 있음
- 또한 JDBC URL, DB 사용자 아이디/비밀번호가 필요함
- JDBC URL: 데이터베이스에 대한 다양한 정보를 포함함
- JDBC URL 구조

```
jdbc:하위 프로토콜:데이터 원본 식별자
```

■ 2단계: 데이터베이스 연결

- H2의 경우 DB 실행 방식에 따라 다음과 같은 형식을 취함

```
// 임베디드 모드(파일 직접 접속)
jdbc:h2:~/test————윈도우 사용자 홈디렉터리에 'test.mv.db' 파일 사용
jdbc:h2:e:/Dev/test————e드라이브의 [Dev] 폴더에 'test.mv.db' 파일 사용

// 네트워크 서버 모드
jdbc:h2:tcp://localhost/~/test————네트워크로 접속 허용, 데이터 파일은 사용자 홈디렉터리에 위치

// 메모리 DB 모드(종료 시 데이터 소멸됨)
jdbc:h2:mem:test_mem
```

■ 2단계: 데이터베이스 연결

■ Connection 클래스 인스턴스 레퍼런스 얻기

- JDBC 클래스 로딩과 URL이 준비되었으면 실제 데이터베이스와의 연결을 만들기 위한 코드를 작성하기
- DriverManager의 getConnection() 메서드를 사용함

```
Connection conn = DriverManager.getConnection(JDBC_URL, "아이디", "비밀번호");
```

- JDBC_URL: 해당 데이터베이스에 맞게 미리 정의되어 있는 문자열
- 아이디와 비밀번호: 데이터베이스에 등록된 계정

■ 3단계: Statement 생성

- 데이터베이스와 연결을 한번 완료하면 이후 연동부터는 SQL 문을 통해 이루어짐
- 이때 문자열로 이루어진 SQL 문을 JDBC에서 처리할 수 있는 객체로 변환해야 하는데, 이때 사용되는 것이 Statement 객체임
- 하지만 보통 SQL 문과 데이터를 조합하기 때문에 일반 Statement보다는 Statement를 상속받는 PreparedStatement를 사용하는 것이 좋음
- PreparedStatement
 - SQL 문을 미리 만들어두고 변수를 따로 입력하는 방식
 - 효율성이나 유지보수 측면에서 유리한 구조임
 - 기본적으로 Statement 클래스를 상속받기 때문에 Statement 클래스 메서드를 모두 사용할 수 있음

```
PreparedStatement pstmt = conn.prepareStatement("insert into test values(?,?)");  
pstmt.setString(1,request.getParameter("username");  
pstmt.setString(2,request.getParameter("email");
```

■ 4단계: SQL 문 전송

- 3단계 과정을 통해 PreparedStatement 객체가 준비되었고 실제 쿼리의 실행은 SQL 문 종류에 따라 executeQuery() 혹은 executeUpdate()를 사용하게 됨
- executeQuery()
 - SELECT 문을 수행할 때 사용함
 - 반환값은 ResultSet 클래스 타입으로, 해당 SELECT 문의 결과에 해당하는 데이터에 접근할 수 있는 방법을 제공함
- executeUpdate()
 - UPDATE, DELETE와 같은 문을 수행할 때 사용함
 - 반환값은 INT 값으로, 처리된 데이터의 수를 반환함

■ 5단계: 결과 받기

- 데이터베이스에서 데이터 결과를 받으려면 Statement나 PreparedStatement의 execute Query()를 사용함
- 입력, 수정, 삭제와 달리 데이터를 갖고 오는 경우에는 가져온 결과 데이터를 처리하기 위한 ResultSet 객체가 필요함
- ResultSet은 조회한 결과값에 순차적으로 접근할 수 있는 커서를 다룰 수 있게 함

```
ResultSet rs = pstmt.executeQuery();
```


■ 6단계: 연결 해제

- 데이터베이스 사용이 종료되면 기본적으로 연결을 해제해야 함
- 데이터베이스는 동시에 여러 연결을 지원하지만 동시 연결 수에 따라 라이선스 비용이 증가하기도 하고, 동시 연결 가능 수가 적은 경우 대기 시간이 길어지는 문제가 발생하기도 함
- 따라서 사용이 끝난 데이터베이스와의 연결은 해제해주는 것이 좋으며 하나의 연결에서 발생하는 여러 Statement, ResultSet 같은 객체도 종료해주는 것이 좋음.

```
rs.close();  
pstmt.close();  
conn.close();
```

Section 07

[실습 9-2] JDBC 종합 실습 : 학생정보 조회와 등록

- 이번 실습의 개요

- [실습 9-1]에서 생성한 학생정보 테이블을 이용해 데이터를 저장하고 전체 데이터를 보여주는 예제를 JDBC를 사용해 실습함
 - [그림 9-4]를 기반으로 생성한 테이블
- 또한 8장에서 배운 MVC 패턴으로 구현함
 - 우선 [src/main/java] 폴더에 [ch09] 패키지를 생성
 - [ch09]에는 Student 클래스, StudentDAO 클래스, StudentController 서블릿이 들어감

번호(PK)	이름	대학	생년월일	이메일
1	김길동	AA대학교	1999-10-21	kim@aa.com
2	박사랑	BB대학교	2000-01-21	park@bb.com
3	나최고	CC대학교	1998-07-11	na@cc.com
4	김길동	BB대학교	1999-03-10	kim@bb.com
5	홍길동	AA 대학교	1999-12-10	hong@aa.com

그림 9-4 기본키가 추가된 학생정보 테이블

1) [ch09] 패키지에 Student 클래스를 생성하기

- Student 클래스는 DO 클래스에 해당됨
- getter/setter 메서드는 8장을 참고

예제 9-1

Student.java

```
1 package ch09;
2 import java.sql.Date;
3
4 public class Student {
5     private int id;
6     private String username;
7     private String univ;
8     private Date birth;
9     private String email;
10
11     // getter/setter 생략
12 }
```

- 클래스 기본 구조 생성

- 2) [ch09] 패키지에 StudentDAO 클래스를 생성하기

- StudentDAO 클래스는 데이터베이스 연동을 위한 DAO 클래스에 해당됨
- 데이터베이스 연결/종료, CRUD 지원을 위한 메서드로 구성됨
 - 여기서는 등록과 목록 보기 기능에 필요한 메서드만 구현함
- 주의) JDBC API 클래스는 java.sql 패키지를 사용하므로 패키지를 import할 때 java.sql인지 확인하여 선택하기

```
package ch09;
// import 생략

public class StudentDAO {
    Connection conn = null;
    PreparedStatement pstmt;

    final String JDBC_DRIVER = "org.h2.Driver";
    final String JDBC_URL = "jdbc:h2:tcp://localhost/~jwbookdb";
}
```

- 연결/종료 메서드 구현
- 3) DB 관련 작업 요청 시 데이터베이스에 연결하고, 종료 시 연결도 종료하는 형식으로 구성하기
- 연결/종료를 위한 메서드 생성

```
public void open(){
    try {
        Class.forName(JDBC_DRIVER);
        conn = DriverManager.getConnection(JDBC_URL,"jwbook","1234");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
public void close() {
    try {
        pstmt.close();
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

- 학생 등록 메서드 구현

4) 학생 등록을 위해서는 Student 객체를 인자로 받아 SQL과 데이터를 조합하는 과정이 필요함.

- PreparedStatement를 이용해 '?'에 해당하는 데이터를 매핑하기

```
public void insert(Student s) {  
    open();  
    String sql = "INSERT INTO student(username, univ, birth, email) values(?,?,?,?)";  
  
    try {  
        pstmt = conn.prepareStatement(sql);  
        pstmt.setString(1, s.getUsername());  
        pstmt.setString(2, s.getUniv());  
        pstmt.setDate(3, s.getBirth());  
        pstmt.setString(4, s.getEmail());  
  
        pstmt.executeUpdate();  
    } catch (Exception e) {  
        e.printStackTrace();  
    } finally {  
        close();  
    }  
}
```

- 학생 목록 메서드 구현

- 5) 전체 학생 목록은 단순히 SELECT 문을 사용해 가져올 수 있음. 이때 ResultSet을 이용해 Student 객체에 데이터를 매핑하고 List에 추가한 다음 모든 데이터가 담긴 List 타입을 리턴해야 함

```
public List<Student> getAll() {
    open();
    List<Student> students = new ArrayList<>();

    try {
        pstmt = conn.prepareStatement("select * from student");
        ResultSet rs = pstmt.executeQuery();

        while(rs.next()) {
            Student s = new Student();
            s.setId(rs.getInt("id"));
            s.setUsername(rs.getString("username"));
            s.setUniv(rs.getString("univ"));
            s.setBirth(rs.getDate("birth"));
            s.setEmail(rs.getString("email"));

            students.add(s);
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        close();
    }
    return students;
}
```


6) SudentDAO.java의 전체 코드

예제 9-2

StudentDAO.java

```
1 package ch09;
2 import java.sql.Connection; // java.sql 패키지를 import해야 함
3 // 이하 import 문 생략
4
5 public class StudentDAO {
6     Connection conn = null;
7     PreparedStatement pstmt;
8
9     final String JDBC_DRIVER = "org.h2.Driver";
10    final String JDBC_URL = "jdbc:h2:tcp://localhost/~jwbookdb";
11
12    public void open(){
13        try {
14            Class.forName(JDBC_DRIVER);
15            conn = DriverManager.getConnection(JDBC_URL,"jwbook","1234");
16        } catch (Exception e) { e.printStackTrace(); }
17    }
18
19    public void close() {
20        try {
21            pstmt.close();
22            conn.close();
23        } catch (SQLException e) { e.printStackTrace(); }
24    }
25 }
```

```
26 public void insert(Student s) {
27     open();
28     String sql =
29         "INSERT INTO student(username, univ, birth, email) values(?,?,?,?)";
30
31     try {
32         pstmt = conn.prepareStatement(sql);
33         pstmt.setString(1, s.getUsername());
34         pstmt.setString(2, s.getUniv());
35         pstmt.setDate(3, s.getBirth());
36         pstmt.setString(4, s.getEmail());
37
38         pstmt.executeUpdate();
39     } catch(Exception e) { e.printStackTrace(); }
40     finally { close(); }
41 }
42
43 public List<Student> getAll() {
44     open();
45     List<Student> students = new ArrayList<>();
46
47     try {
48         pstmt = conn.prepareStatement("select * from student");
49         ResultSet rs = pstmt.executeQuery();
50
```

```
51     while(rs.next()) {
52         Student s = new Student();
53         s.setId(rs.getInt("id"));
54         s.setUsername(rs.getString("username"));
55         s.setUniv(rs.getString("univ"));
56         s.setBirth(rs.getDate("birth"));
57         s.setEmail(rs.getString("email"));
58
59         students.add(s);
60     }
61 } catch (Exception e) { e.printStackTrace(); }
62     finally { close(); }
63     return students;
64 }
65 }
```

- 7) [ch09] 패키지에서 StudentController 서블릿을 생성하기. 이때 URL은 studentControl로 지정함
- 컨트롤러 서블릿의 구조는 [실습 8-2]에서 만든 것을 재사용함
 - 다음과 같이 기본 구조를 만든 다음 action 요청에 따라 필요한 메서드를 추가함
 - 여기서는 list, insert의 2개의 action 요청을 구현함

```
@WebServlet("/studentControl")
public class StudentController extends HttpServlet {
    private static final long serialVersionUID = 1L;

    StudentDAO dao;

    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        dao = new StudentDAO();
    }
}
```

```
protected void service(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    String action = request.getParameter("action");
    String view = "";
    if(request.getParameter("action") == null) {
        getServletContext().getRequestDispatcher("/studentControl?action=list")
        .forward(request, response);
    }else{
        switch(action) {
            case "list": view = list(request, response); break;
            case "insert": view = insert(request, response); break;
        }
        getServletContext().getRequestDispatcher("/ch09/"+view).forward(request,
response);    }
    }

    public String list(HttpServletRequest request, HttpServletResponse response) {
        return "";
    }

    public String insert(HttpServletRequest request, HttpServletResponse response) {
        return "";
    }
}
```

8) 먼저 전체 데이터 목록 요청을 위한 list 메서드를 살펴봄

- dao.getAll() 메서드를 호출한 결과를 request scope object의 속성에 'students'라는 이름으로 저장하고 뷰의 이름을 리턴함
- 여기서는 하나의 뷰(studentInfo.jsp)에서 목록 보기와 데이터 추가를 모두 구현함

```
public String list(HttpServletRequest request, HttpServletResponse response) {  
    request.setAttribute("students", dao.getAll());  
    return "studentInfo.jsp";  
}
```

9) 새로운 학생을 추가하는 insert 메서드 생성하기

- request.getParameter()를 이용해 모든 입력값을 읽어와 Student 객체로 매핑한 다음 dao.insert() 메서드를 호출하고, 뷰 페이지로 되돌아가야 함
- 이때 입력값을 일일이 매핑하기보다는 Apache Commons Bean Utils를 이용하는 것임
 - 따라서 먼저 pom.xml에 다음과 같이 BeanUtils 라이브러리 의존성을 추가해야 함

```
<!-- https://mvnrepository.com/artifact/commons-beanutils/commons-beanutils -->
<dependency>
  <groupId>commons-beanutils</groupId>
  <artifactId>commons-beanutils</artifactId>
  <version>1.9.4</version>
</dependency>
```

10) insert 메서드 내부 구현하기

- Student 객체를 생성한 다음 BeanUtils.populate() 메서드로 파라미터로 전달된 name 속성과 일치하는 Student 클래스의 멤버 변수를 찾아 값이 전달되도록 함
- 그 후 dao.insert() 메서드로 새로운 데이터를 저장한 다음 다시 새로운 목록으로 화면을 표시하기 위해 list() 메서드를 호출한 결과를 리턴함

```
public String insert(HttpServletRequest request, HttpServletResponse response) {  
    Student s = new Student();  
    try {  
        BeanUtils.populate(s, request.getParameterMap());  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    dao.insert(s);  
    return list(request, response);  
}
```


11) StudentController의 전체 코드

예제 9-3

StudentController.java

```
1 package ch09;
2 import java.io.IOException;
3 import org.apache.commons.beanutils.BeanUtils;
4
5 @WebServlet("/studentControl")
6 public class StudentController extends HttpServlet {
7     private static final long serialVersionUID = 1L;
8
9     StudentDAO dao;
10    public void init(ServletConfig config) throws ServletException {
11        super.init(config);
12        dao = new StudentDAO();
13    }
14
15    protected void service(HttpServletRequest request, HttpServletResponse
16 response) throws ServletException, IOException {
17        request.setCharacterEncoding("utf-8");
18        String action = request.getParameter("action");
19        String view = "";
20
```

```
21     if(action == null) {
22         getServletContext().getRequestDispatcher("/studentControl?action=list")
23             .forward(request, response);
24     } else {
25         switch(action) {
26             case "list": view = list(request, response); break;
27             case "insert": view = insert(request, response); break;
28         }
29         getServletContext().getRequestDispatcher("/ch09/vview")
30             .forward(request, response);
31     }
32 }
33
34 public String list(HttpServletRequest request, HttpServletResponse response)
35 {
36     request.setAttribute("students", dao.getAll());
37     return "studentInfo.jsp";
38 }
39
40 public String insert(HttpServletRequest request, HttpServletResponse
41 response) {
42     Student s = new Student();
43     try {
44         BeanUtils.populate(s, request.getParameterMap());
45     } catch (Exception e) { e.printStackTrace(); }
46     dao.insert(s);
47     return list(request, response);
48 }
49 }
```

12) 뷰 구현하기

- 목록 화면과 입력 화면을 따로 만들어도 되지만 간단한 JDBC 구현을 알아보기 위해 하나의 화면을 사용함
- [webapp]에 [ch09] 폴더를 만들고 studentInfo.jsp를 생성하기

13) jstl을 사용하기 때문에 taglib을 추가해주고 목록은 다음과 같이 c:forEach를 사용함

```
<table border="1">
  <tr><th>id</th><th>이름</th><th>대학</th><th>생일</th><th>이메일</th></tr>
  <c:forEach items="{students}" var="s">
    <tr>
      <td>${s.id}</td> <td>${s.username}</td> <td>${s.univ}</td>
      <td>${s.birth}</td> <td>${s.email}</td>
    </tr>
  </c:forEach>
</table>
```

14) 데이터 입력 양식은 HTML <form> 태그를 사용하고 action에는 컨트롤러 호출 url에 action=insert를 넣음

- 주의) <input>의 name 속성값을 Student 클래스의 멤버 변수 이름과 동일하게 작성해야 함

```
<form method="post" action="/jwbook/studentControl?action=insert">
  <label>이름</label>
  <input type="text" name="username"><br>
  <label>대학</label>
  <input type="text" name="univ"><br>
  <label>생일</label>
  <input type="text" name="birth"><br>
  <label>이메일</label>
  <input type="text" name="email"><br>
  <button type="submit">등록</button>
</form>
```

15) studentInfo.jsp의 전체 코드

예제 9-4

studentInfo.jsp

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
4
5 <!DOCTYPE html>
6 <html>
7 <head>
8 <meta charset="UTF-8">
9 <title>학생정보</title>
10 </head>
11 <body>
12 <h2>학생정보</h2>[<a href="/jwbook/studentControl">새로고침</a>]
13 <hr>
14 <table border="1">
15 <tr><th>id</th><th>이름</th><th>대학</th><th>생일</th><th>이메일</th></tr>
16 <c:forEach items="${students}" var="s">
17 <tr>
18 <td>${s.id}</td> <td>${s.username}</td> <td>${s.univ}</td>
19 <td>${s.birth}</td> <td>${s.email}</td>
20 </tr>
21 </c:forEach>
22 </table>
23 </body>
```

```
24 <h2>학생 추가</h2>
25 <hr>
26 <form method="post" action="/jwbook/studentControl?action=insert">
27   <label>이름</label>
28   <input type="text" name="username"><br>
29   <label>대학</label>
30   <input type="text" name="univ"><br>
31   <label>생일</label>
32   <input type="text" name="birth"><br>
33   <label>이메일</label>
34   <input type="text" name="email"><br>
35   <button type="submit">등록</button>
36 </form>
37 </body>
38 </html>
```

16) StudentController 서블릿을 실행하면 자동으로 'action=list' 인자를 붙여 다시 호출하므로 학생정보 목록이 바로 나타남

- [실습 9-1]을 진행하며 5개의 데이터를 등록했기 때문에 5명의 학생정보가 나타남

학생 정보

[\[새로고침\]](#)

id	이름	대학	생일	이메일
1	김길동	AA대학교	1999-10-21	kim@aa.com
2	박사병	BB대학교	2000-01-21	park@bb.com
3	나최고	CC대학교	1998-07-11	na@cc.com
4	김길동	BB대학교	1999-03-10	kim@bb.com
5	홍길동	AA대학교	1999-12-10	hong@aa.com

학생 추가

이름

대학

생일

이메일

17) 새로운 데이터를 등록하면 추가된 결과를 확인할 수 있음

학생 정보

[새로고침]

id	이름	대학	생일	이메일
1	김길동	AA대학교	1999-10-21	kim@aa.com
2	박사랑	BB대학교	2000-01-21	park@bb.com
3	나최고	CC대학교	1998-07-11	na@cc.com
4	김길동	BB대학교	1999-03-10	kim@bb.com
5	홍길동	AA대학교	1999-12-10	hong@aa.com
33	김사랑	CC대학교	2000-02-10	love@cc.com

학생 추가

이름

대학

생일

이메일

Thank you!