[Sparta]웹개발의 봄, Spring - 2주차

▼ PDF 파일

[수업 목표]

1. 메모장 프로젝트 구현을 통해 CRUD에 대한 개념을 학습니다.
2. Database와 SQL을 학습합니다.
3. Spring을 사용하여 DB와 소통하는 방법을 학습합니다.

[목차]

메모장
- 01. 메모장 프로젝트 설계
- 02. Create, Read 구현하기
- 03. Update, Delete 구현하기

Database와 SQL
- 04. Database
- 05. SQL
- 06. SQL 연습하기
- 07. JDBC란 무엇일까?
- 08. 2주차 **끝 & 숙제설명**
- HW. 2주차 숙제 답안 코드

모든 토글을 열고 닫는 단축키
Windows : `Ctrl` + `alt` + `t`
Mac : `⌘` + `⌥` + `t`
* `cmd/ctrl` + `shift` + `L` 단축키를 이용하여 다크 모드를 켜고 끌 수 있습니다.

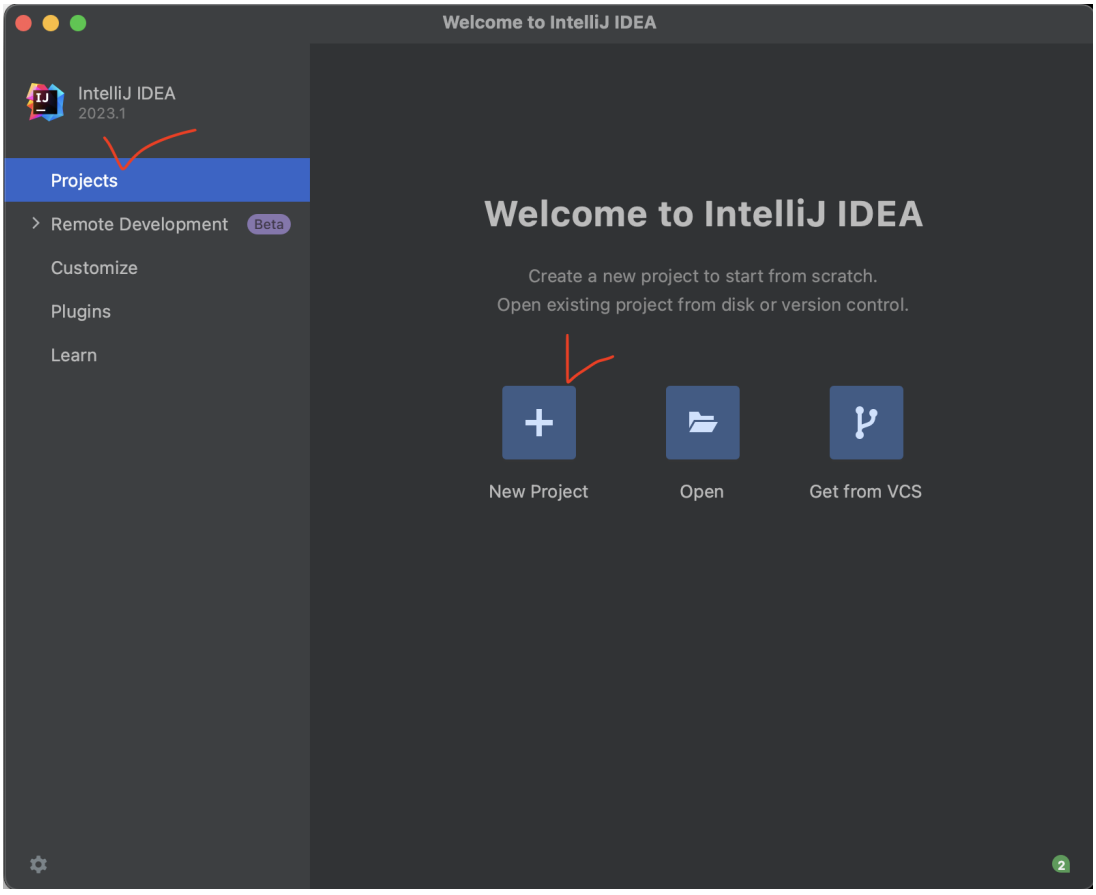
메모장

01. 메모장 프로젝트 설계

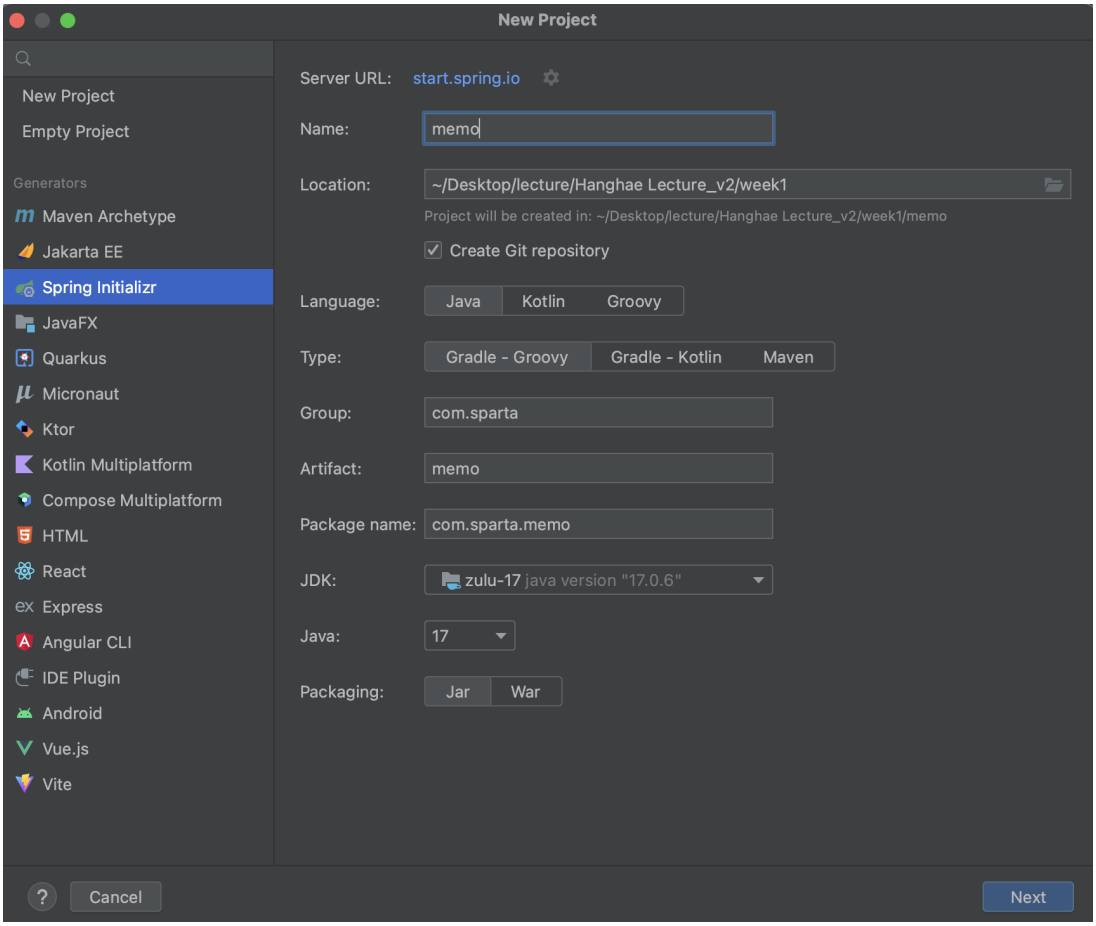
▼ 프로젝트 생성

▼ 프로젝트 준비하기

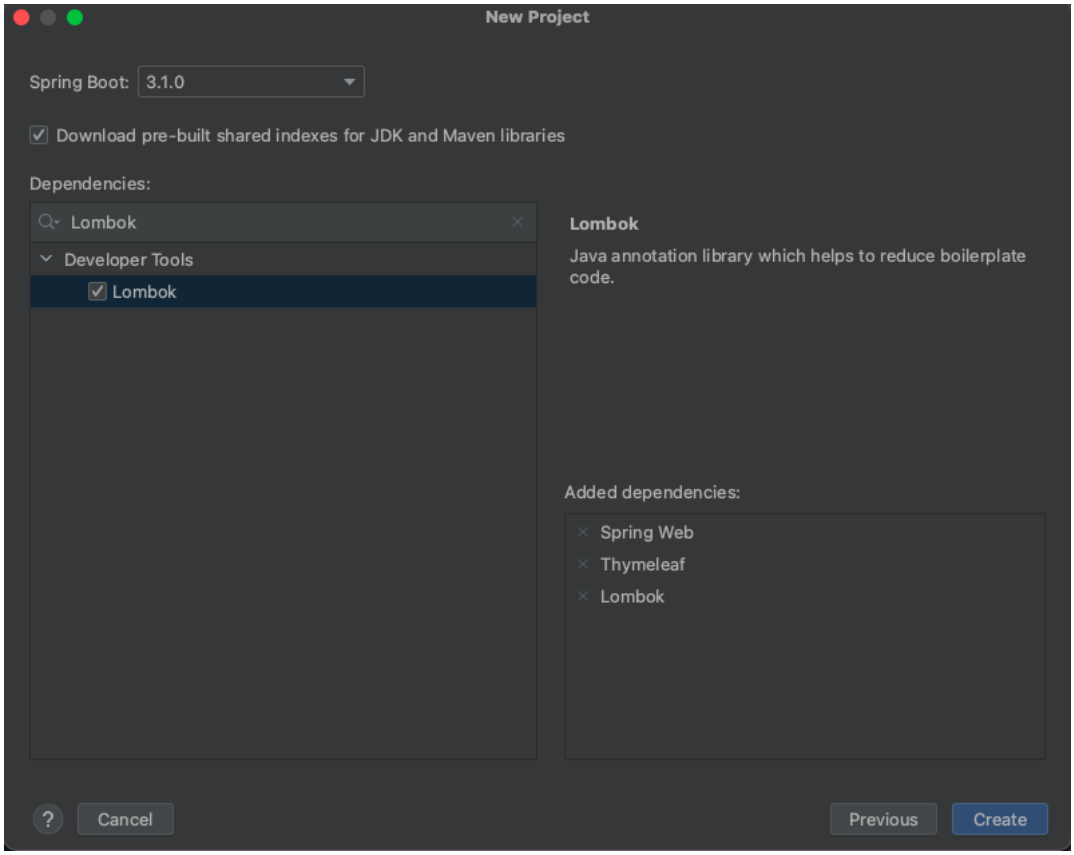
1. IntelliJ를 실행합니다.
2. New Project를 클릭합니다.



3. 왼쪽 메뉴에서 "Spring Initializr"를 클릭하고 아래와 같이 설정합니다.



4. 🚒**[중요] 꼭 다음 사항을 확인해주세요.**
   - Name: memo
   - **Language: Java**
   - **Build system: Gradle - Groovy**
   - Group: com.sparta
   - **JDK: 17**
   - 프로젝트 경로(Location)는 자유롭게 설정하시면 됩니다.
5. Next를 클릭합니다.
6. Dependencies를 아래 사진과 같이 추가해주고 Create를 클릭합니다.



▼ 시작코드

👉 src > main > resources > static 에 만든 index.html 에, 코드스니펫 내용을 넣어주세요.
(혹시 없다면 만들어주세요.)

▼ [코드 스니펫] index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Memo Service</title>

  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <link href="https://fonts.googleapis.com/css2?family=Noto+Sans+KR:wght@500&display=swap" rel="stylesheet">

  <style>
    @import url(//spoqa.github.io/spoqa-han-sans/css/SpoqaHanSans-kr.css);

    body {
      margin: 0px;
    }

    .area-edit {
      display: none;
    }

    .wrap {
      width: 538px;
      margin: 10px auto;
    }

    #contents {
      width: 538px;
    }
```

```
.area-write {
  position: relative;
  width: 538px;
}

.area-write img {
  cursor: pointer;
  position: absolute;
  width: 22.2px;
  height: 18.7px;
  bottom: 15px;
  right: 17px;
}

.background-header {
  position: fixed;
  z-index: -1;
  top: 0px;
  width: 100%;
  height: 428px;
  background-color: #339af0;
}

.background-body {
  position: fixed;
  z-index: -1;
  top: 428px;
  height: 100%;
  width: 100%;
  background-color: #dee2e6;
}

.header {
  margin-top: 50px;
}

.header h2 {
  /*font-family: 'Noto Sans KR', sans-serif;*/
  height: 33px;
  font-size: 42px;
  font-weight: 500;
  font-stretch: normal;
  font-style: normal;
  line-height: 0.79;
  letter-spacing: -0.5px;
  text-align: center;
  color: #ffffff;
}

.header p {
  margin: 40px auto;
  width: 217px;
  height: 48px;
  font-family: 'Noto Sans KR', sans-serif;
  font-size: 16px;
  font-weight: 500;
  font-stretch: normal;
  font-style: normal;
  line-height: 1.5;
  letter-spacing: -1.12px;
  text-align: center;
  color: #ffffff;
}

textarea.field {
  width: 502px !important;
  height: 146px;
  border-radius: 5px;
  background-color: #ffffff;
  border: none;
  padding: 18px;
  resize: none;
}

textarea.field::placeholder {
  width: 216px;
  height: 16px;
  font-family: 'Noto Sans KR', sans-serif;
  font-size: 16px;
  font-weight: normal;
  font-stretch: normal;
  font-style: normal;
  line-height: 1;
  letter-spacing: -0.96px;
  text-align: left;
  color: #868e96;
}
```

```
.card {
  width: 538px;
  border-radius: 5px;
  background-color: #ffffff;
  margin-bottom: 12px;
}

.card .metadata {
  position: relative;
  display: flex;
  font-family: 'Spoqa Han Sans';
  font-size: 11px;
  font-weight: normal;
  font-stretch: normal;
  font-style: normal;
  line-height: 1;
  letter-spacing: -0.77px;
  text-align: left;
  color: #adb5bd;
  height: 14px;
  padding: 10px 23px;
}

.card .metadata .date {

}

.card .metadata .username {
  margin-left: 20px;
}

.contents {
  padding: 0px 23px;
  word-wrap: break-word;
  word-break: break-all;
}

.contents div.edit {
  display: none;
}

.contents textarea.te-edit {
  border-right: none;
  border-top: none;
  border-left: none;
  resize: none;
  border-bottom: 1px solid #212529;
  width: 100%;
  font-family: 'Spoqa Han Sans';
}

.footer {
  position: relative;
  height: 40px;
}

.footer img.icon-start-edit {
  cursor: pointer;
  position: absolute;
  bottom: 14px;
  right: 55px;
  width: 18px;
  height: 18px;
}

.footer img.icon-end-edit {
  cursor: pointer;
  position: absolute;
  display: none;
  bottom: 14px;
  right: 55px;
  width: 20px;
  height: 15px;
}

.footer img.icon-delete {
  cursor: pointer;
  position: absolute;
  bottom: 12px;
  right: 19px;
  width: 14px;
  height: 18px;
}

#cards-box {
  margin-top: 12px;
```

```
    }
  </style>
  <script>
    // 사용자가 내용을 올바르게 입력하였는지 확인합니다.
    function isValidContents(contents) {
      if (contents == '') {
        alert('내용을 입력해주세요');
        return false;
      }
      if (contents.trim().length > 140) {
        alert('공백 포함 140자 이하로 입력해주세요');
        return false;
      }
      return true;
    }

    // 익명의 username을 만듭니다.
    function genRandomName(length) {
      let result = '';
      let characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
      let charactersLength = characters.length;
      for (let i = 0; i < length; i++) {
        let number = Math.random() * charactersLength;
        let index = Math.floor(number);
        result += characters.charAt(index);
      }
      return result;
    }

    // 수정 버튼을 눌렀을 때, 기존 작성 내용을 textarea 에 전달합니다.
    // 숨길 버튼을 숨기고, 나타낼 버튼을 나타냅니다.
    function editPost(id) {
      showEdits(id);
      let contents = $(`#${id}-contents`).text().trim();
      $(`#${id}-textarea`).val(contents);
    }

    function showEdits(id) {
      $(`#${id}-editarea`).show();
      $(`#${id}-submit`).show();
      $(`#${id}-delete`).show();

      $(`#${id}-contents`).hide();
      $(`#${id}-edit`).hide();
    }

    $(document).ready(function () {
      // HTML 문서를 로드할 때마다 실행합니다.
      getMessages();
    })

    // 메모를 불러와서 보여줍니다.
    function getMessages() {
      // 1. 기존 메모 내용을 지웁니다.
      $('#cards-box').empty();
      // 2. 메모 목록을 불러와서 HTML로 붙입니다.
      $.ajax({
        type: 'GET',
        url: '/api/memos',
        success: function (response) {
          for (let i = 0; i < response.length; i++) {
            let message = response[i];
            let id = message['id'];
            let username = message['username'];
            let contents = message['contents'];
            let modifiedAt = message['modifiedAt'];
            addHTML(id, username, contents, modifiedAt);
          }
        }
      })
    }

    // 메모 하나를 HTML로 만들어서 body 태그 내 원하는 곳에 붙입니다.
    function addHTML(id, username, contents, modifiedAt) {
      // 1. HTML 태그를 만듭니다.
      let tempHtml = `<div class="card">
                <!-- date/username 영역 -->
                <div class="metadata">
                    <div class="date">
                        ${modifiedAt}
                    </div>
                    <div id="${id}-username" class="username">
                        ${username}
                    </div>
                </div>
                <!-- contents 조회/수정 영역-->
                <div class="contents">
```

```
                    <div id="${id}-contents" class="text">
                        ${contents}
                    </div>
                    <div id="${id}-editarea" class="edit">
                        <textarea id="${id}-textarea" class="te-edit" name="" id="" cols="30" rows="5"></textarea>
                    </div>
                </div>
                <!-- 버튼 영역-->
                <div class="footer">
                    <img id="${id}-edit" class="icon-start-edit" src="images/edit.png" alt="" onclick="editPost('${id}')"
                    <img id="${id}-delete" class="icon-delete" src="images/delete.png" alt="" onclick="deleteOne('${id}')
                    <img id="${id}-submit" class="icon-end-edit" src="images/done.png" alt="" onclick="submitEdit('${id}'
                </div>
            </div>`;
      // 2. #cards-box 에 HTML을 붙인다.
      $('#cards-box').append(tempHtml);
    }

    // 메모를 생성합니다.
    function writePost() {
      // 1. 작성한 메모를 불러옵니다.
      let contents = $('#contents').val();

      // 2. 작성한 메모가 올바른지 isValidContents 함수를 통해 확인합니다.
      if (isValidContents(contents) == false) {
        return;
      }
      // 3. genRandomName 함수를 통해 익명의 username을 만듭니다.
      let username = genRandomName(10);

      // 4. 전달할 data JSON으로 만듭니다.
      let data = {'username': username, 'contents': contents};

      // 5. POST /api/memos 에 data를 전달합니다.
      $.ajax({
        type: "POST",
        url: "/api/memos",
        contentType: "application/json",
        data: JSON.stringify(data),
        success: function (response) {
          alert('메시지가 성공적으로 작성되었습니다.');
          window.location.reload();
        }
      });
    }

    // 메모를 수정합니다.
    function submitEdit(id) {
      // 1. 작성 대상 메모의 username과 contents 를 확인합니다.
      let username = $(`#${id}-username`).text().trim();
      let contents = $(`#${id}-textarea`).val().trim();

      // 2. 작성한 메모가 올바른지 isValidContents 함수를 통해 확인합니다.
      if (isValidContents(contents) == false) {
        return;
      }

      // 3. 전달할 data JSON으로 만듭니다.
      let data = {'username': username, 'contents': contents};

      // 4. PUT /api/memos/{id} 에 data를 전달합니다.
      $.ajax({
        type: "PUT",
        url: `/api/memos/${id}`,
        contentType: "application/json",
        data: JSON.stringify(data),
        success: function (response) {
          alert('메시지 변경에 성공하였습니다.');
          window.location.reload();
        }
      });
    }

    // 메모를 삭제합니다.
    function deleteOne(id) {
      // 1. DELETE /api/memos/{id} 에 요청해서 메모를 삭제합니다.
      $.ajax({
        type: "DELETE",
        url: `/api/memos/${id}`,
        success: function (response) {
          alert('메시지 삭제에 성공하였습니다.');
          window.location.reload();
        }
      })
    }
  </script>
</head>
```

```
<body>
<div class="background-header">

</div>
<div class="background-body">

</div>
<div class="wrap">
  <div class="header">
    <h2>Memo</h2>
    <p>
      공유하고 싶은 소식을 입력해주세요.
    </p>
  </div>
  <div class="area-write">
        <textarea class="field" placeholder="공유하고 싶은 소식을 입력해주세요" name="contents" id="contents" cols="30"
                  rows="10"></textarea>
    <!--            <button class="btn btn-danger" onclick="writePost()">작성하기</button>-->
    <img src="images/send.png" alt="" onclick="writePost()">
  </div>
  <div id="cards-box" class="area-read">
    <div class="card">
      <!-- date/username 영역 -->
      <div class="metadata">
        <div class="date">
          October 10, 2020
        </div>
        <div class="username">
          anonymous
        </div>
      </div>
      <!-- contents 조회/수정 영역-->
      <div class="contents">

      </div>
      <!-- 버튼 영역-->
      <div class="footer">
        <img id="1-edit" class="icon-start-edit" src="images/edit.png" alt="" onclick="editPost('1')">
        <img id="1-delete" class="icon-delete" src="images/delete.png" alt="" onclick="deleteOne('1')">
        <img id="1-submit" class="icon-end-edit" src="images/done.png" alt="" onclick="submitEdit('1')">
      </div>
    </div>
  </div>
</div>
</body>

</html>
```

👉 src > main > resources > static 에 images 폴더를 만들고, 아래 이미지 4장을 넣어주세요!
(브라우저에 붙여넣고 엔터를 누르시면 이미지 다운로드가 됩니다.)

▼ [코드 스니펫] delete.png 다운로드

```
https://s3.ap-northeast-2.amazonaws.com/materials.spartacodingclub.kr/spring/week03/delete.png
```

▼ [코드 스니펫] done.png 다운로드

```
https://s3.ap-northeast-2.amazonaws.com/materials.spartacodingclub.kr/spring/week03/done.png
```

▼ [코드 스니펫] edit.png 다운로드

```
https://s3.ap-northeast-2.amazonaws.com/materials.spartacodingclub.kr/spring/week03/edit.png
```

▼ [코드 스니펫] send.png 다운로드

```
https://s3.ap-northeast-2.amazonaws.com/materials.spartacodingclub.kr/spring/week03/send.png
```

▼ 메모장 기능 설계

1. 접속 하자마자 메모 전체 목록 조회하기
    a. GET API 사용해서 메모 목록 불러오기
2. 메모 생성하기
    a. POST API 사용해서 메모 신규 생성하기
    b. 생성된 메모 반환
3. 메모 변경하기
    a. PUT API 사용해서 메모 내용 변경하기
    b. 사용자가 클릭한 메모가 DB에 존재하는지 확인하기
    c. 해당 메모 내용 변경
4. 메모 삭제하기
    a. DELETE API 사용해서 메모 삭제하기
    b. 사용자가 삭제하려는 메모가 DB에 존재하는지 확인하기
    c. DB에서 해당 메모 삭제

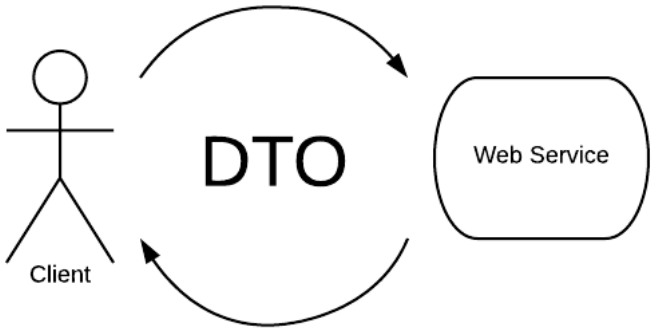
▼ API 테이블

메모장 API

Aa 기능	Method	URL	Return
메모 생성하기	POST	/api/memos	MemoResponseDto
메모 조회하기	GET	/api/memos	List<MemoResponseDto>
메모 변경하기	PUT	/api/memos/{id}	Long
메모 삭제하기	DELETE	/api/memos/{id}	Long

02. Create, Read 구현하기

▼ DTO란 무엇일까?



- 이름에서도 알 수 있듯이 DTO(Data Transfer Object)는 데이터 전송 및 이동을 위해 생성되는 객체를 의미합니다.
- Client에서 보내오는 데이터를 객체로 처리할 때 사용됩니다.
- 또한 서버의 계층간의 이동에도 사용됩니다.

    - 지금은 계층간의 이동이라는 용어에 대해 잘 이해가 되지 않겠지만 이후 강의를 통해 자연스럽게 이해가 되실겁니다.
- 그리고 DB와의 소통을 담당하는 Java 클래스를 그대로 Client에 반환하는 것이 아니라 DTO로 한번 변환한 후 반환할 때도 사용됩니다.
    - 마찬가지로 이후 강의를 통해 자연스럽게 학습할 수 있습니다.

> 📌 Request의 데이터를 처리할 때 사용되는 객체는 RequestDto, Response를 할 때 사용되는 객체는 ResponseDto라는 이름을 붙여 DTO 클래스를 만들 수 있습니다.
>
> - 절대적인 규칙은 아니기 때문에 조직에 따라 규칙이 다를 수 있습니다.
> - 강의에서는 해당 규칙에 따라 DTO 클래스를 제작하도록 하겠습니다.

▼ Create 구현

1. 메모 데이터를 저장할 Memo 클래스 생성

    ▼ [코드 스니펫] entity > Memo

```
package com.sparta.memo.entity;

import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@NoArgsConstructor
public class Memo {
    private Long id;
    private String username;
    private String contents;
}
```

2. 메모 생성하기 API를 받을 수 있는 Controller와 메서드 생성

```
@RestController
@RequestMapping("/api")
public class MemoController {

    @PostMapping("/memos")
    public MemoResponseDto createMemo(@RequestBody MemoRequestDto requestDto) {
        return null;
    }

}
```

3. Client에 데이터를 반환할 때 사용할 MemoResponseDto 클래스 생성

    ▼ [코드 스니펫] MemoResponseDto

```
package com.sparta.memo.dto;

import lombok.Getter;

@Getter
public class MemoResponseDto {
    private Long id;
    private String username;
    private String contents;
}
```

3. Client의 요청 데이터를 받아줄 MemoRequestDto 클래스 생성

    ▼ [코드 스니펫] MemoRequestDto

```
package com.sparta.memo.dto;
```

```
import lombok.Getter;

@Getter
public class MemoRequestDto {
    private String username;
    private String contents;
}
```

5. DB와 연결을 하지 않았기 때문에 메모 데이터를 저장할 Java 컬렉션 생성

```
private final Map<Long, Memo> memoList = new HashMap<>();
```

6. 메모 생성 로직 작성

```
@PostMapping("/memos")
public MemoResponseDto createMemo(@RequestBody MemoRequestDto requestDto) {
    // RequestDto -> Entity
    Memo memo = new Memo(requestDto);

    // Memo Max ID Check
    Long maxId = memoList.size() > 0 ? Collections.max(memoList.keySet()) + 1 : 1;
    memo.setId(maxId);

    // DB 저장
    memoList.put(memo.getId(), memo);

    // Entity -> ResponseDto
    MemoResponseDto memoResponseDto = new MemoResponseDto(memo);

    return memoResponseDto;
}
```

▼ Read 구현

```
@GetMapping("/memos")
public List<MemoResponseDto> getMemos() {
    // Map To List
    List<MemoResponseDto> responseList = memoList.values().stream()
            .map(MemoResponseDto::new).toList();

    return responseList;
}
```

- DB 역할을 하는 memoList를 조회하여 List<MemoResponseDto>로 변환한 후 반환합니다.

▼ 구현 완료 코드

▼ [코드 스니펫] MemoController

```
package com.sparta.memo.controller;

import com.sparta.memo.dto.MemoRequestDto;
import com.sparta.memo.dto.MemoResponseDto;
import com.sparta.memo.entity.Memo;
import org.springframework.web.bind.annotation.*;

import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@RestController
@RequestMapping("/api")
public class MemoController {

    private final Map<Long, Memo> memoList = new HashMap<>();

    @PostMapping("/memos")
    public MemoResponseDto createMemo(@RequestBody MemoRequestDto requestDto) {
        // RequestDto -> Entity
        Memo memo = new Memo(requestDto);
```

```
        // Memo Max ID Check
        Long maxId = memoList.size() > 0 ? Collections.max(memoList.keySet()) + 1 : 1;
        memo.setId(maxId);

        // DB 저장
        memoList.put(memo.getId(), memo);

        // Entity -> ResponseDto
        MemoResponseDto memoResponseDto = new MemoResponseDto(memo);

        return memoResponseDto;
    }

    @GetMapping("/memos")
    public List<MemoResponseDto> getMemos() {
        // Map To List
        List<MemoResponseDto> responseList = memoList.values().stream()
                .map(MemoResponseDto::new).toList();

        return responseList;
    }
}
```

03. Update, Delete 구현하기

▼ Update 구현

```
@PutMapping("/memos/{id}")
public Long updateMemo(@PathVariable Long id, @RequestBody MemoRequestDto requestDto) {
    // 해당 메모가 DB에 존재하는지 확인
    if(memoList.containsKey(id)) {
        // 해당 메모 가져오기
        Memo memo = memoList.get(id);

        // memo 수정
        memo.update(requestDto);
        return memo.getId();
    } else {
        throw new IllegalArgumentException("선택한 메모는 존재하지 않습니다.");
    }
}
```

- Memo 클래스에 수정하는 기능의 메서드 update 생성

```
public void update(MemoRequestDto requestDto) {
    this.username = requestDto.getUsername();
    this.contents = requestDto.getContents();
}
```

▼ Delete 구현

```
@DeleteMapping("/memos/{id}")
public Long deleteMemo(@PathVariable Long id) {
    // 해당 메모가 DB에 존재하는지 확인
    if(memoList.containsKey(id)) {
        // 해당 메모 삭제하기
        memoList.remove(id);
        return id;
    } else {
        throw new IllegalArgumentException("선택한 메모는 존재하지 않습니다.");
    }
}
```

▼ 구현 완료 코드

▼ [코드 스니펫] MemoController

```
package com.sparta.memo.controller;

import com.sparta.memo.dto.MemoRequestDto;
import com.sparta.memo.dto.MemoResponseDto;
import com.sparta.memo.entity.Memo;
import org.springframework.web.bind.annotation.*;

import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@RestController
@RequestMapping("/api")
public class MemoController {

    private final Map<Long, Memo> memoList = new HashMap<>();

    @PostMapping("/memos")
    public MemoResponseDto createMemo(@RequestBody MemoRequestDto requestDto) {
        // RequestDto -> Entity
        Memo memo = new Memo(requestDto);

        // Memo Max ID Check
        Long maxId = memoList.size() > 0 ? Collections.max(memoList.keySet()) + 1 : 1;
        memo.setId(maxId);

        // DB 저장
        memoList.put(memo.getId(), memo);

        // Entity -> ResponseDto
        MemoResponseDto memoResponseDto = new MemoResponseDto(memo);

        return memoResponseDto;
    }

    @GetMapping("/memos")
    public List<MemoResponseDto> getMemos() {
        // Map To List
        List<MemoResponseDto> responseList = memoList.values().stream()
                .map(MemoResponseDto::new).toList();

        return responseList;
    }

    @PutMapping("/memos/{id}")
    public Long updateMemo(@PathVariable Long id, @RequestBody MemoRequestDto requestDto) {
        // 해당 메모가 DB에 존재하는지 확인
        if(memoList.containsKey(id)) {
            // 해당 메모 가져오기
            Memo memo = memoList.get(id);

            // memo 수정
            memo.update(requestDto);
            return memo.getId();
        } else {
            throw new IllegalArgumentException("선택한 메모는 존재하지 않습니다.");
        }
    }

    @DeleteMapping("/memos/{id}")
    public Long deleteMemo(@PathVariable Long id) {
        // 해당 메모가 DB에 존재하는지 확인
        if(memoList.containsKey(id)) {
            // 해당 메모 삭제하기
            memoList.remove(id);
            return id;
        } else {
            throw new IllegalArgumentException("선택한 메모는 존재하지 않습니다.");
        }
    }
}
```

Database와 SQL

04. Database

▼ 개념

💡 Database를 한 마디로 정의하면 '***데이터의 집합***' 이라고 할 수 있습니다.
DB는 우리가 매일 사용하는 카톡 메시지, 인스타그램의 사진등의 정보를 저장하고 관리해 줍니다.

▼ 용어 정리

💡 DBMS

- DBMS 는 '***Database Management System***' 의 약자로 Database를 관리하고 운영하는 소프트웨어를 의미합니다.

💡 RDBMS

- RDBMS는 '***Relational DBMS***'의 약자로 관계형 데이터베이스라고 불립니다.
- RDBMS는 테이블(table)이라는 최소 단위로 구성되며, 이 테이블은 열(column)과 행(row)으로 이루어져 있습니다.
- 테이블간 FK(Foreign Key)를 통해 다른 데이터를 조합해서 함께 볼수 있다라는 장점이 있습니다.

아이디	이름	전화번호	그룹	column 명
ka123	카즈하	010-7777-7777	르세라핌	1 row
kim123	김채원	010-6666-6666	르세라핌	2 row
sa123	사쿠라	010-8888-8888	르세라핌	3 row
heo123	허윤진	010-0000-0000	르세라핌	4 row
hong123	홍은채	010-1111-1111	르세라핌	5 row

▼ RDBMS의 종류

💡 각 제품 간 차이가 크지 않아서 사실 어떤 걸 사용하든 좋습니다. 유료인 Oracle을 제외하고 보통 MySQL, PostgreSQL 중에서 많이 고르는 편인데, 우리는 MySQL을 사용할 것입니다.

- MySQL

💡 MySQL

- MySQL은 우리가 서비스를 배포할 때 사용할 데이터베이스입니다.
- AWS RDS 라는 서비스를 사용해 붙여볼 예정입니다.
- Spring과 궁합이 좋아서 많은 회사에서 사용하고 있습니다.

05. SQL

▼ 개념

💡 SQL은 '***Structured Query Language***' 의 약자로 RDBMS에서 사용되는 언어입니다.
수 많은 정보를 Database에서 조작하고 관리하기 위해서는 SQL 언어를 사용해야합니다.

- 국제표준화기구에서 SQL에 대한 표준을 정해서 발표하고 있습니다. 하지만...
- DBMS를 만드는 회사가 여러 곳이기 때문에 DBMS 마다 표준 SQL을 준수하되, 각 제품의 특성을 반영하기 위한 약간의 차이가 존재합니다.

▼ DDL

💡 '***Data Definition Language'*** 의 약자로 테이블이나 관계의 구조를 생성하는데 사용합니다

- CREATE : 새로운 데이터베이스 및 테이블을 생성해 줍니다.

```
CREATE DATABASE 데이터베이스이름;
CREATE TABLE 테이블이름
(
    필드이름1 필드타입1,
    필드이름2 필드타입2,
    ...
);
```

- ALTER : 데이터베이스와 테이블의 내용을 수정할 수 있습니다.

```
ALTER TABLE 테이블이름 ADD 필드이름 필드타입;
ALTER TABLE 테이블이름 DROP 필드이름;
ALTER TABLE 테이블이름 MODIFY COLUMN 필드이름 필드타입;
```

- DROP : 데이터베이스와 테이블을 삭제할 수 있습니다. 데이터 및 테이블 전체를 삭제합니다.

```
DROP DATABASE 데이터베이스이름;
DROP TABLE 테이블이름;
```

- TRUNCATE : 데이터베이스와 테이블을 삭제할 수 있습니다. 최초 테이블이 만들어졌던 상태 즉, 컬럼값만 남깁니다.

```
TRUNCATE DATABASE 데이터베이스이름;
TRUNCATE TABLE 테이블이름;
```

▼ DCL

💡 '***Data Control Language***' 의 약자로 데이터의 사용 권한을 관리하는데 사용합니다.

- GRANT : 사용자 또는 ROLE에 대해 권한을 부여할 수 있습니다.

```
GRANT [객체권한명] (컬럼)
ON [객체명]
TO { 유저명 | 롤명 | PUBLC} [WITH GRANT OPTION];

//ex
GRANT SELECT ,INSERT
ON mp
TO scott WITH GRANT OPTION;
```

- REVOKE : 사용자 또는 ROLE에 부여한 권한을 회수할 수 있습니다.

```
REVOKE { 권한명 [, 권한명...] ALL}
ON 객체명
FROM {유저명 [, 유저명...] | 롤명(ROLE) | PUBLIC}
[CASCADE CONSTRAINTS];

//ex
REVOKE SELECT , INSERT
ON emp
FROM scott
[CASCADE CONSTRAINTS];
```

▼ DML

💡 '***Data Manipulation Language***' 의 약자로 테이블에 데이터를 검색, 삽입, 수정, 삭제하는데 사용합니다.

- INSERT : 테이블에 새로운 row를 추가할 수 있습니다.

```
INSERT INTO 테이블이름(필드이름1, 필드이름2, 필드이름3, ...) VALUES(데이터값1, 데이터값2, 데이터값3, ...);
INSERT INTO 테이블이름 VALUES(데이터값1, 데이터값2, 데이터값3, ...);
```

- SELECT : 테이블의 row를 선택할 수 있습니다.

```
SELECT 필드이름 FROM 테이블이름 [WHERE 조건];
```

- UPDATE : 테이블의 row의 내용을 수정할 수 있습니다.

```
UPDATE 테이블이름 SET 필드이름1=데이터값1, 필드이름2=데이터값2, ... WHERE 필드이름=데이터값;
```

- DELETE : 테이블의 row를 삭제할 수 있습니다.

```
DELETE FROM 테이블이름 WHERE 필드이름=데이터값;
```

06. SQL 연습하기

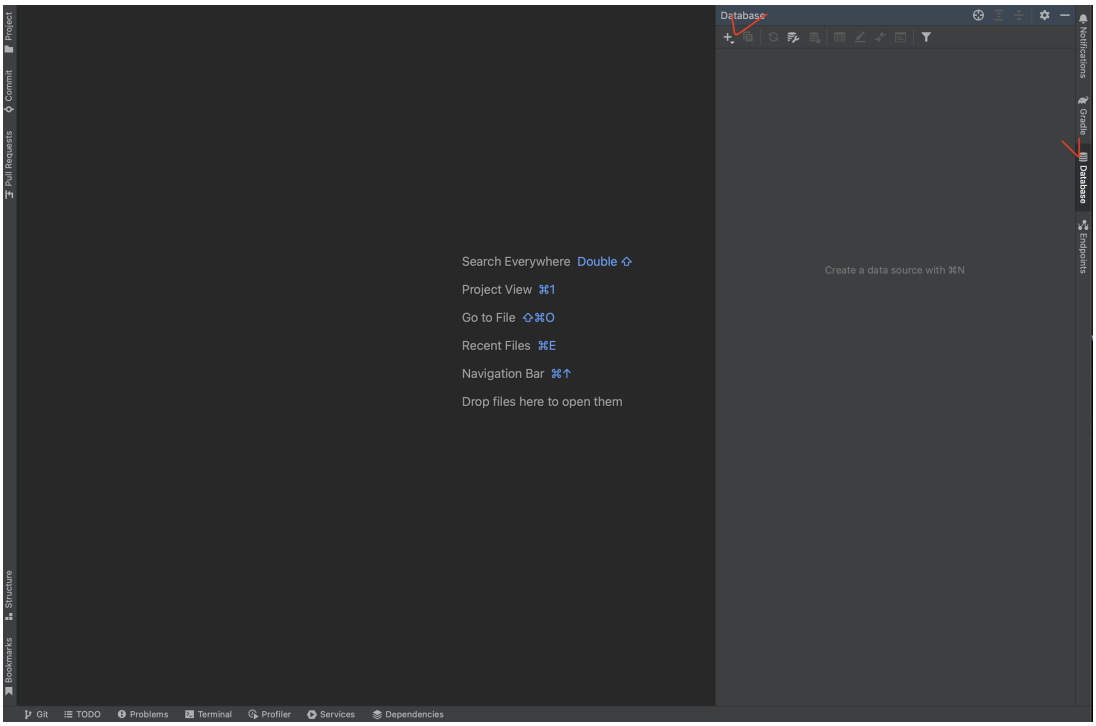
▼ Intellij Database 연동

▼ CREATE DATABASE

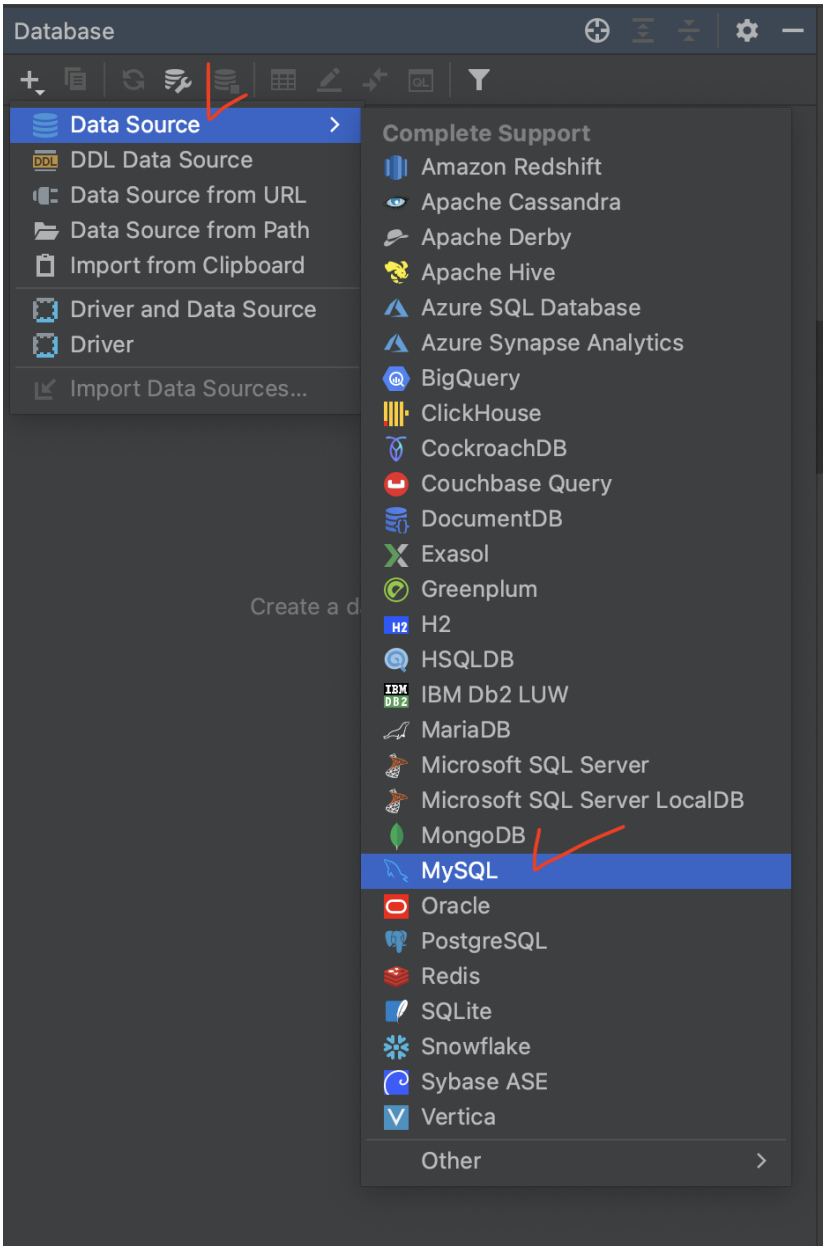
```
CREATE DATABASE academy;
```

▼ 연동 순서

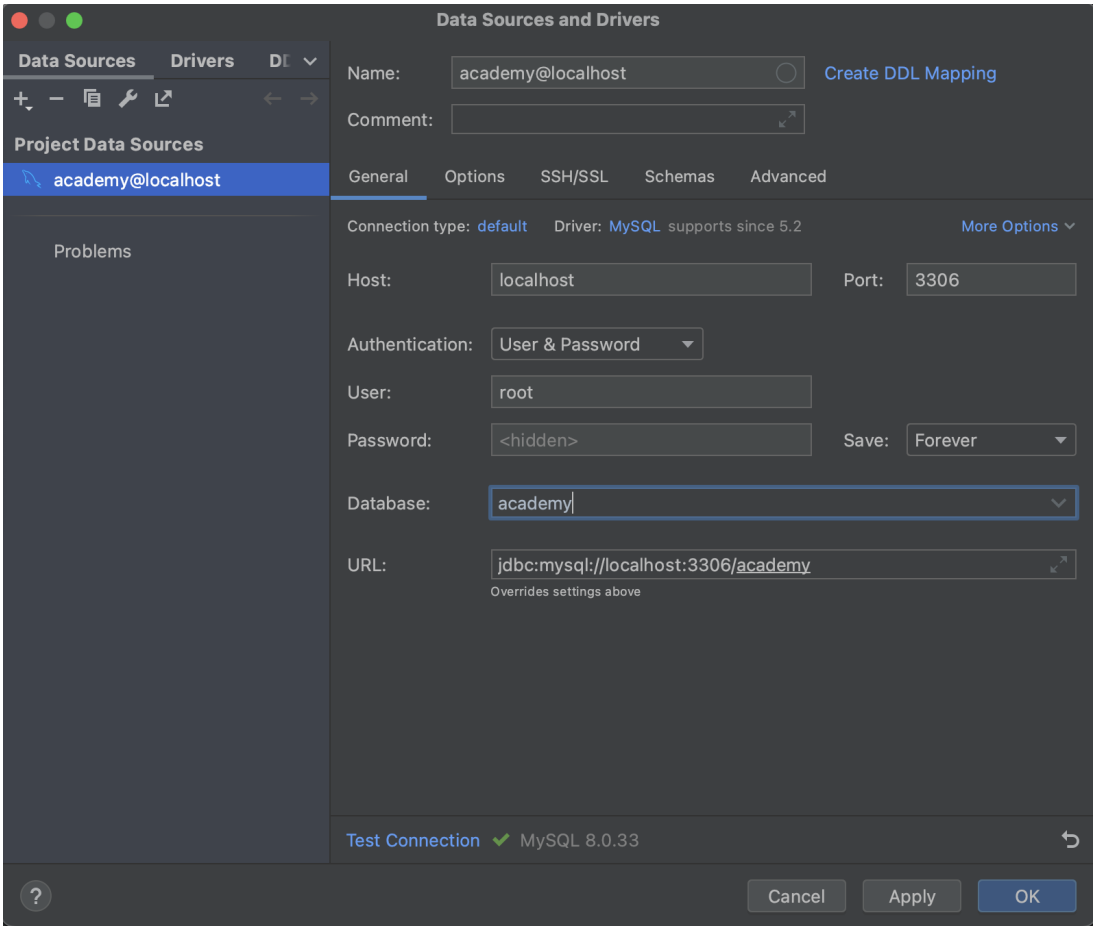
1. Database 탭을 클릭하시고 + 버튼을 누릅니다.



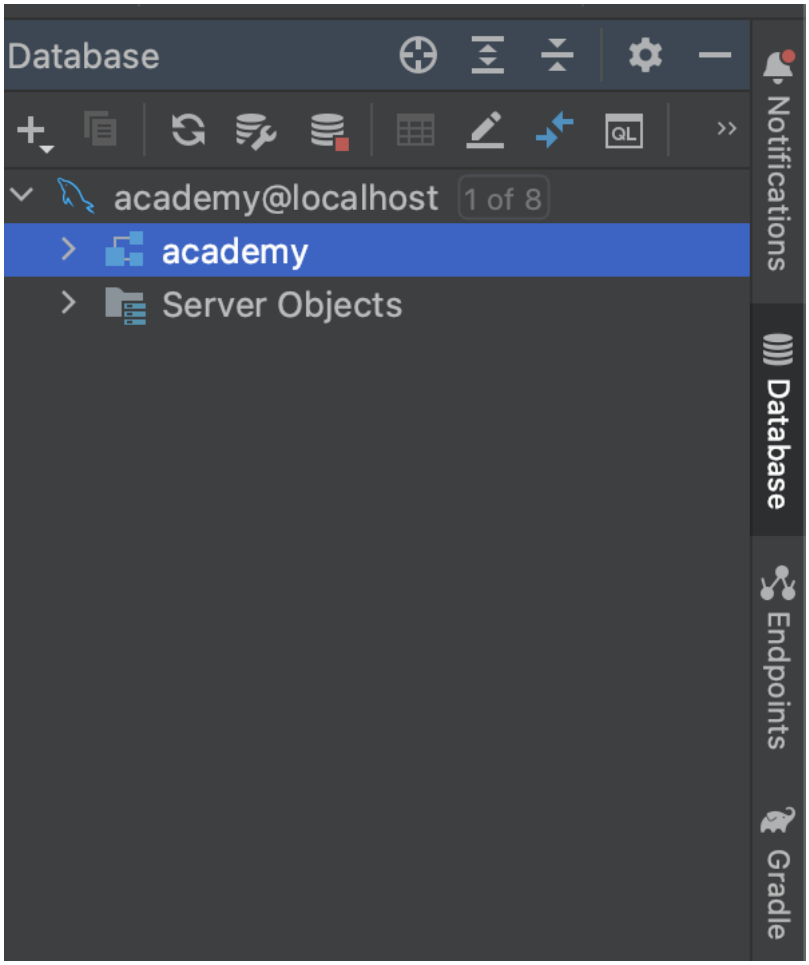
2. Data Source > MySQL 를 클릭합니다.



3. User, Password, Database 정보를 추가한 후 Ok를 클릭합니다.



4. MySQL Database에 연결이 완료되었습니다.



▼ CREATE

▼ 제약조건

▼ AUTO_INCREMENT : 컬럼의 값이 중복되지 않게 1씩 자동으로 증가하게 해줘 고유번호를 생성해 줍니다.

```
CREATE TABLE 테이블이름
(
    필드이름 필드타입 AUTO_INCREMENT,
    // id bigint AUTO_INCREMENT,
    ...
);
```

▼ NOT NULL : 해당 필드는 NULL 값을 저장할 수 없게 됩니다.

```
CREATE TABLE 테이블이름
(
    필드이름 필드타입 NOT NULL,
    ...
);
```

▼ UNIQUE : 해당 필드는 서로 다른 값을 가져야만 합니다.

```
CREATE TABLE 테이블이름
(
```

```
    필드이름 필드타입 UNIQUE,
    ...
);
```

▼ PRIMARY KEY : 해당 필드가 NOT NULL과 UNIQUE 제약 조건의 특징을 모두 가지게 됩니다.

```
CREATE TABLE 테이블이름
(
    필드이름 필드타입 PRIMARY KEY,
    ...
);
```

💡 PRIMARY KEY 이해하기

- PRIMARY KEY 즉, **기본 키**는 테이블 내에서 '**유일하게 존재하는 값의 조합**'을 설정해서 중복된 데이터가 테이블에 삽입되는 것을 방지하는 제약조건입니다.

❓기본 키를 사용하는 이유?

1️⃣ **데이터의 중복을 방지합니다.**

- 만약 기본키를 설정하지 않고서 회원을 관리하는 테이블이 있을 때 기존에 있던 회원이 예전에 가입한 것을 잊고서 다시 회원가입을 하는데
  최근에 번호 및 주소가 바뀌었다면?
- 우리는 둘 중에 어느 정보가 정확한 정보인지 판단하기 어렵습니다. 즉, **데이터의 무결성**이 깨집니다.

2️⃣ **데이터를 매우 빠르게 찾을 수 있게 됩니다.**

- 기본 키를 설정하면 DBMS는 **인덱스**를 만듭니다.
- 이 인덱스는 해당 데이터를 빨리 찾을 수 있게 도와주는 일종의 목차라고 생각하시면 좋습니다.
- 만약 주민번호 컬럼에 기본 키가 설정 되어있지 않는다면 ?
- 우리는 주민번호가 절대 중복될 수 없다는 것을 알지만 DBMS는 그것을 모르기 때문에 주민번호가 중복될 수 있는 것을 가정하여 5000만 row를 전부 확인 합니다.
- 이럴 때 주민번호에 기본 키 설정이 되어있으면 row를 전부 확인하지 않고 1개만 찾으면 바로 해당 데이터를 반환합니다.

▼ FOREIGN KEY : 하나의 테이블을 다른 테이블에 의존하게 만들며 데이터의 무결성을 보장해 줍니다.

- FK 를 가지는 테이블이 참조하는 기준 테이블의 열은 반드시 PK, UNIQUE 제약조건이 설정되어 있어야 합니다.

```
CREATE TABLE 테이블이름
(
    필드이름 필드타입,
    ...
    FOREIGN KEY(필드이름)
    REFERENCES 테이블이름(필드이름)
);
```

❗ FOREIGN KEY 이해하기

- FOREIGN KEY 즉, 외래 키는 두개의 테이블을 연결하는 다리 역할을 해주는 키입니다.

💡 예를 들어보겠습니다.

- 손님이 주문을 했을 때 해당 데이터를 관리하기 위해 DB에 저장을 합니다. 근데 이 때 해당 데이터들을 하나의 테이블로 관리한다면?

<회원주문 테이블>

회원_id	회원_이름	주문_id	상품_이름	상품_수량
u_1	userA	o_1	item_1	5
u_1	userA	o_2	item_2	3
u_1	userA	o_3	item_1	10
u_2	userB	o_4	item_3	7
u_2	userB	o_5	item_2	3

- 이런식으로 중복된 데이터가 들어갈 것입니다.

❓그러면 2개로 테이블을 분리하면 되지 않을까?

<회원 테이블>

회원_id	회원_이름
u_1	userA
u_2	userB

<주문 테이블>

주문_id	상품_이름	상품_수량
o_1	item_1	5
o_2	item_2	3
o_3	item_1	10
o_4	item_3	7
o_5	item_2	3

- 이러면 중복된 데이터는 없어지지만 어느 사용자가 무슨 주문을 했는지를 알 수 없게 됩니다.

❓이 때 외래 키를 사용한다면?

<회원 테이블>

회원_id	회원_이름
u_1	userA
u_2	userB

<주문 테이블>

주문_id	상품_이름	상품_수량	회원_id
o_1	item_1	5	u_1
o_2	item_2	3	u_1
o_3	item_1	10	u_1
o_4	item_3	7	u_2
o_5	item_2	3	u_2

- 이렇게 외래 키를 사용하면 중복되는 데이터를 없애고 주문 테이블에서 외래 키를 사용해서 사용자테이블에 접근해 주문을 한 사용자의 정보도 가져올 수 있게 됩니다.

- 추가로 기본 키가 하나의 테이블에서 중복된 데이터가 삽입되는 것을 방지하는 역할을 해주는데 외래 키 역시 비슷한 문제를 방지하는 역할을 수행합니다.
- 외래 키는 데이터가 새롭게 추가될 때 외래 키에 해당하는 값이 외래 키가 참조하는 테이블에 존재하는지를 확인합니다.

▼ CASCADE : FOREIGN KEY 로 연관된 데이터를 삭제,변경할 수 있습니다.

```
CREATE TABLE 테이블이름
(
    필드이름 필드타입,
    ...
    FOREIGN KEY(필드이름)
    REFERENCES 테이블이름(필드이름) ON DELETE CASCADE
                              //ON UPDATE CASCADE
);
```

▼ MAJOR 테이블

▼ [코드 스니펫] MAJOR 테이블

```
CREATE TABLE IF NOT EXISTS MAJOR
(
  major_code varchar(100) primary key comment '주특기코드',
  major_name varchar(100) not null comment '주특기명',
  tutor_name varchar(100) not null comment '튜터'
);
```

▼ STUDENT 테이블

▼ [코드 스니펫] STUDENT 테이블

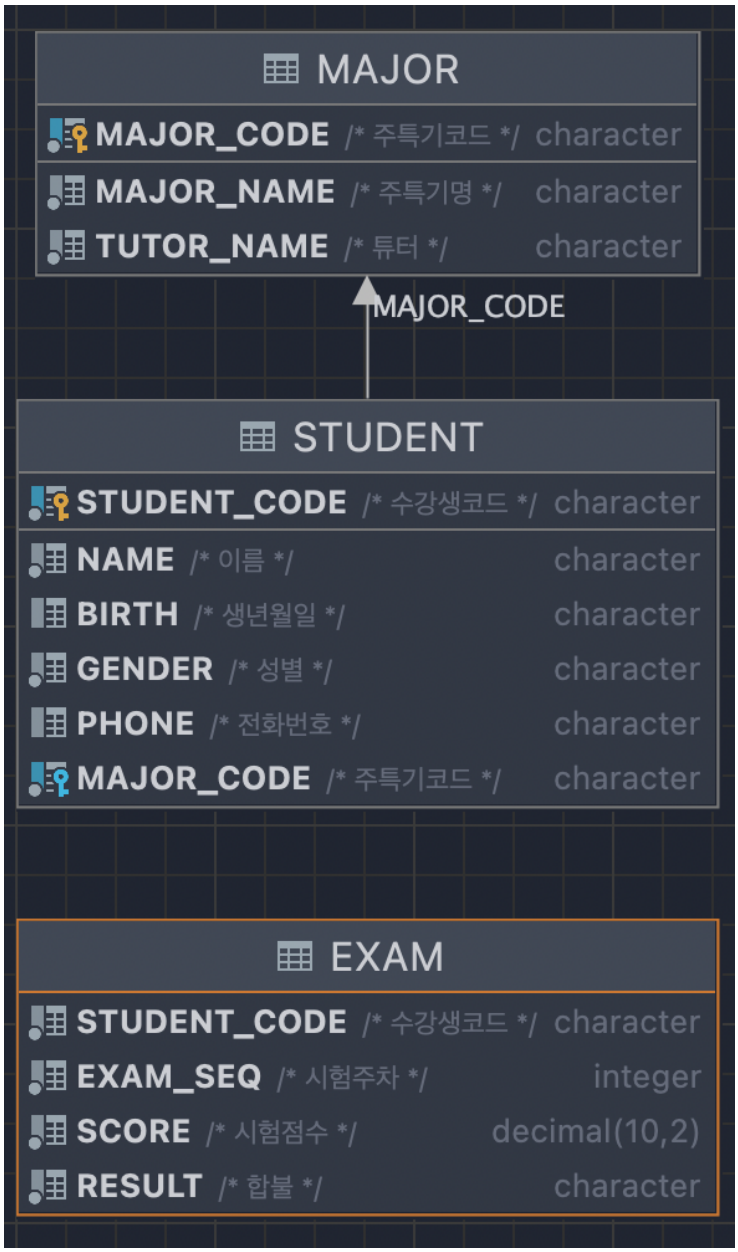
```
CREATE TABLE IF NOT EXISTS STUDENT
(
  student_code varchar(100) primary key comment '수강생코드',
  name varchar(100) not null comment '이름',
  birth varchar(8) null comment '생년월일',
  gender varchar(1) not null comment '성별',
  phone varchar(11) null comment '전화번호',
  major_code varchar(100) not null comment '주특기코드',
  foreign key(major_code) references major(major_code)
);
```

▼ EXAM 테이블

▼ [코드 스니펫] EXAM 테이블

```
CREATE TABLE IF NOT EXISTS EXAM
(
  student_code varchar(100) not null comment '수강생코드',
  exam_seq int not null comment '시험주차',
  score decimal(10,2) not null comment '시험점수',
  result varchar(1) not null comment '합불'
);
```

▼ ERD

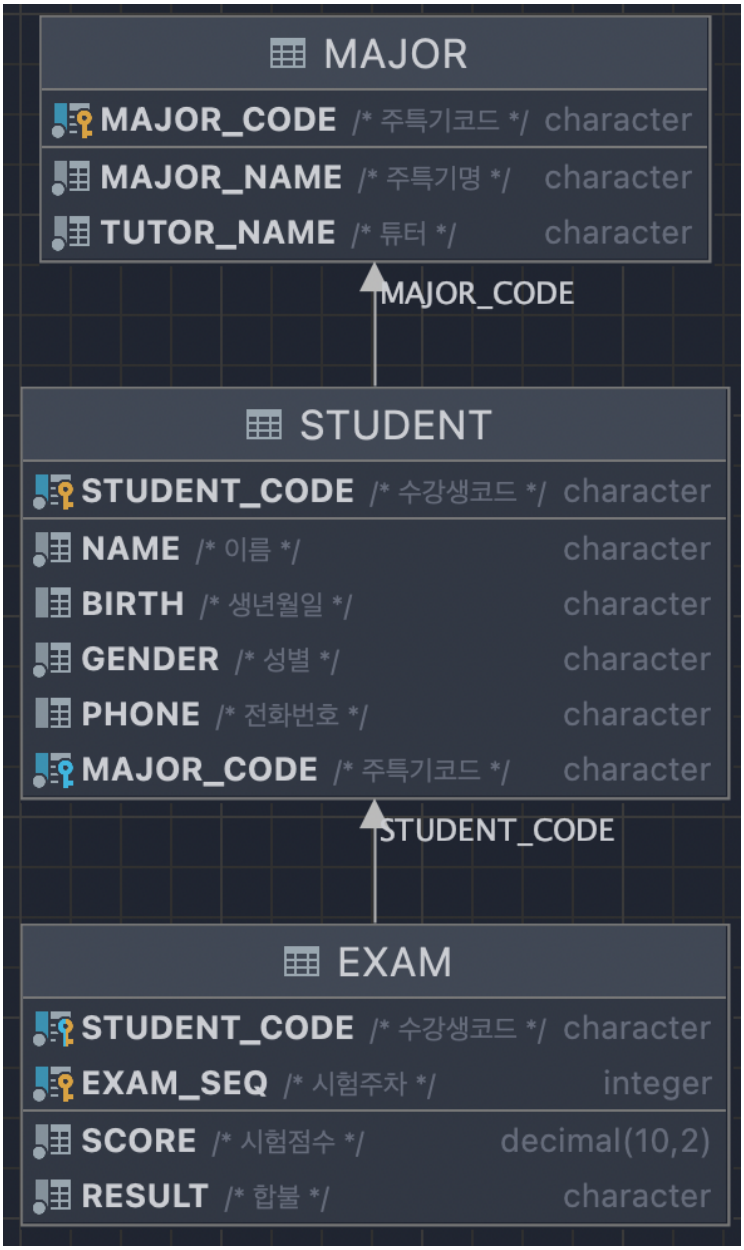


▼ ALTER

▼ [코드 스니펫] ALTER

```
ALTER TABLE EXAM ADD PRIMARY KEY(student_code, exam_seq);
ALTER TABLE EXAM ADD CONSTRAINT exam_fk_student_code FOREIGN KEY(student_code) REFERENCES STUDENT(student_code);
```

▼ ERD



▼ INSERT

▼ MAJOR 테이블

▼ [코드 스니펫] MAJOR INSERT

```
INSERT INTO MAJOR VALUES('m1', '스프링', '남병관');
INSERT INTO MAJOR VALUES('m2', '노드', '강승현');
INSERT INTO MAJOR VALUES('m3', '플라스크', '이범규');
INSERT INTO MAJOR VALUES('m4', '루비온레일즈', '차은서');
INSERT INTO MAJOR VALUES('m5', '라라벨', '구름');
INSERT INTO MAJOR VALUES('m6', '리엑트', '임민영');
INSERT INTO MAJOR VALUES('m7', '뷰', '김서영');
INSERT INTO MAJOR VALUES('m8', '엥귤러', '한현아');
```

▼ STUDENT 테이블

▼ [코드 스니펫] STUDENT INSERT

```
INSERT INTO STUDENT VALUES('s1', '최원빈', '20220331', 'M', '01000000001', 'm1');
INSERT INTO STUDENT VALUES('s2', '강준규', '20220501', 'M', '01000000002', 'm1');
INSERT INTO STUDENT VALUES('s3', '김영철', '20220711', 'M', '01000000003', 'm1');
INSERT INTO STUDENT VALUES('s4', '예상기', '20220408', 'M', '01000000004', 'm6');
INSERT INTO STUDENT VALUES('s5', '안지현', '20220921', 'F', '01000000005', 'm6');
INSERT INTO STUDENT VALUES('s6', '이대호', '20221111', 'M', '01000000006', 'm7');
INSERT INTO STUDENT VALUES('s7', '정주혜', '20221117', 'F', '01000000007', 'm8');
INSERT INTO STUDENT VALUES('s8', '고미송', '20220623', 'F', '01000000008', 'm6');
INSERT INTO STUDENT VALUES('s9', '이용우', '20220511', 'M', '01000000009', 'm2');
INSERT INTO STUDENT VALUES('s10', '심선아', '20220504', 'F', '01000000010', 'm8');
INSERT INTO STUDENT VALUES('s11', '변정섭', '20220222', 'M', '01000000020', 'm2');
INSERT INTO STUDENT(student_code, name, gender, major_code) VALUES('s12', '권오빈', 'M', 'm3');
INSERT INTO STUDENT VALUES('s13', '김가은', '20220121', 'F', '01000000030', 'm1');
INSERT INTO STUDENT(student_code, name, gender, major_code) VALUES('s14', '김동현', 'M', 'm4');
INSERT INTO STUDENT VALUES('s15', '박은진', '20221101', 'F', '01000000040', 'm1');
INSERT INTO STUDENT(student_code, name, birth, gender, phone, major_code) VALUES('s16', '정영호', '20221105', 'M', '0100000
INSERT INTO STUDENT(student_code, name, gender, major_code) VALUES('s17', '박가현', 'F', 'm7');
INSERT INTO STUDENT(student_code, name, birth, gender, phone, major_code) VALUES('s18', '박용태', '20220508', 'M', '0100000
INSERT INTO STUDENT VALUES('s19', '김예지', '20220505', 'F', '01000000070', 'm2');
INSERT INTO STUDENT VALUES('s20', '윤지용', '20220909', 'M', '01000000080', 'm3');
INSERT INTO STUDENT VALUES('s21', '손윤주', '20220303', 'F', '01000000090', 'm6');
```

▼ EXAM 테이블

▼ [코드 스니펫] EXAM INSERT

```
INSERT INTO EXAM VALUES('s1', 1, 8.5, 'P');
INSERT INTO EXAM VALUES('s1', 2, 9.5, 'P');
INSERT INTO EXAM VALUES('s1', 3, 3.5, 'F');
INSERT INTO EXAM VALUES('s2', 1, 8.2, 'P');
INSERT INTO EXAM VALUES('s2', 2, 9.5, 'P');
INSERT INTO EXAM VALUES('s2', 3, 7.5, 'P');
INSERT INTO EXAM VALUES('s3', 1, 9.3, 'P');
INSERT INTO EXAM VALUES('s3', 2, 5.3, 'F');
INSERT INTO EXAM VALUES('s3', 3, 9.9, 'P');
INSERT INTO EXAM VALUES('s4', 1, 8.4, 'P');
INSERT INTO EXAM VALUES('s5', 1, 9.5, 'P');
INSERT INTO EXAM VALUES('s5', 2, 3.5, 'F');
INSERT INTO EXAM VALUES('s6', 1, 8.3, 'P');
INSERT INTO EXAM VALUES('s7', 1, 9.2, 'P');
INSERT INTO EXAM VALUES('s7', 2, 9.9, 'P');
INSERT INTO EXAM VALUES('s7', 3, 3.6, 'F');
INSERT INTO EXAM VALUES('s8', 1, 8.4, 'P');
INSERT INTO EXAM VALUES('s9', 1, 9.7, 'P');
INSERT INTO EXAM VALUES('s10', 1, 8.4, 'P');
INSERT INTO EXAM VALUES('s10', 2, 9.8, 'P');
INSERT INTO EXAM VALUES('s10', 3, 8.4, 'P');
INSERT INTO EXAM VALUES('s11', 1, 8.6, 'P');
INSERT INTO EXAM VALUES('s12', 1, 9.2, 'P');
INSERT INTO EXAM VALUES('s13', 1, 8.1, 'P');
INSERT INTO EXAM VALUES('s13', 2, 9.5, 'P');
INSERT INTO EXAM VALUES('s13', 3, 2.1, 'F');
INSERT INTO EXAM VALUES('s14', 1, 9.2, 'P');
INSERT INTO EXAM VALUES('s15', 1, 9.7, 'P');
INSERT INTO EXAM VALUES('s15', 2, 1.7, 'F');
INSERT INTO EXAM VALUES('s16', 1, 8.4, 'P');
INSERT INTO EXAM VALUES('s17', 1, 9.3, 'P');
INSERT INTO EXAM VALUES('s17', 2, 9.9, 'P');
INSERT INTO EXAM VALUES('s17', 3, 1.3, 'F');
INSERT INTO EXAM VALUES('s18', 1, 9.9, 'P');
INSERT INTO EXAM VALUES('s19', 1, 9.4, 'P');
INSERT INTO EXAM VALUES('s19', 2, 8.9, 'P');
INSERT INTO EXAM VALUES('s19', 3, 7.4, 'F');
INSERT INTO EXAM VALUES('s20', 1, 8.1, 'P');
INSERT INTO EXAM VALUES('s20', 2, 6.4, 'F');
INSERT INTO EXAM VALUES('s21', 1, 9.5, 'P');
INSERT INTO EXAM VALUES('s21', 2, 8.8, 'P');
INSERT INTO EXAM VALUES('s21', 3, 8.2, 'P');
```

▼ UPDATE

▼ [코드 스니펫] UPDATE

```
INSERT INTO STUDENT VALUES('s0', '수강생', '20220331', 'M', '01000000005', 'm1');
UPDATE STUDENT SET major_code= 'm2' where student_code= 's0';
```

▼ DELETE

▼ [코드 스니펫] DELETE

```
DELETE FROM STUDENT WHERE student_code = 's0';
```

▼ SELECT

▼ [코드 스니펫] SELECT

```
SELECT * FROM STUDENT;
SELECT * FROM STUDENT WHERE STUDENT_CODE = 's1';
SELECT name, major_code FROM STUDENT WHERE student_code = 's1';
```

▼ JOIN

▼ [코드 스니펫] JOIN

```
SELECT s.name, s.major_code, m.major_name FROM STUDENT s JOIN MAJOR m ON s.major_code = m.major_code;
SELECT s.name, s.major_code, m.major_name FROM STUDENT s, MAJOR m WHERE s.major_code = m.major_code;
```

📌 JOIN 이해하기

- JOIN은 나누어진 테이블을 하나로 합치기 위해 데이터베이스가 제공하는 기능입니다.
- JOIN 은 ON 이라는 키워드를 통해 기준이 되는 컬럼을 선택하여 2개의 테이블을 합쳐 줍니다.
- JOIN을 할 때에는 적어도 하나의 컬럼을 서로 공유하고 있어야 하기 때문에 테이블에 외래 키가 설정 되어 있다면 해당 컬럼을 통해 JOIN을 하면 조건을 충족할 수 있습니다.

❗**다만 JOIN을 하기 위해 외래 키를 설정하는 것이 항상 좋은 선택이 아닐 수도 있습니다.**

- 외래 키를 설정하면 데이터 무결성을 확인하는 추가 연산이 발생합니다.
- 또한 무결성을 지켜야하기 때문에 상황에 따라 개발하는데 불편할 수 있습니다.

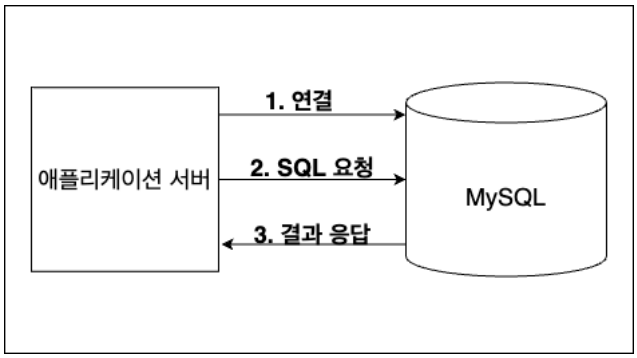
👉 결론은 항상 테이블에 모든 제약조건을 걸어야 하는 것은 아닙니다. 프로젝트의 상황에 따라 가장 효율적인 제약조건을 테이블에 적용해야합니다.

07. JDBC란 무엇일까?

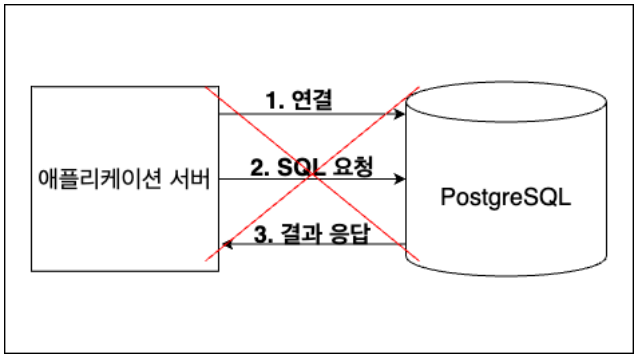
📌 애플리케이션 서버와 데이터베이스는 어떻게 소통해야 할까요?

- 우리는 애플리케이션 서버에서 요청을 받고, 해당 요청을 처리하기 위해 데이터베이스와 소통을 해야합니다.
- 따라서 실제로 서버가 데이터베이스와 어떠한 방법을 통해 소통하고 있는지 알아보려고 합니다.

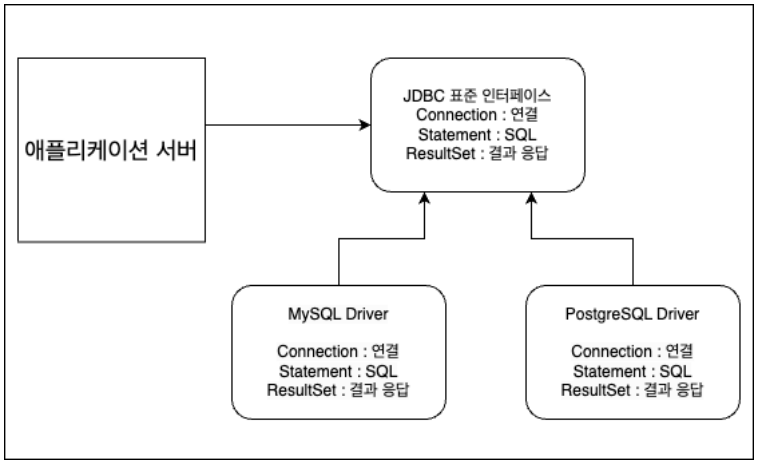
▼ JDBC의 등장배경



- 애플리케이션 서버에서 DB에 접근 하기 위해서는 여러가지 작업이 필요합니다.
  1. 우선 DB에 연결하기 위해 커넥션을 연결해야합니다.
  2. SQL을 작성한 후 커넥션을 통해 SQL을 요청합니다.
  3. 요청한 SQL에 대한 결과를 응답 받습니다.



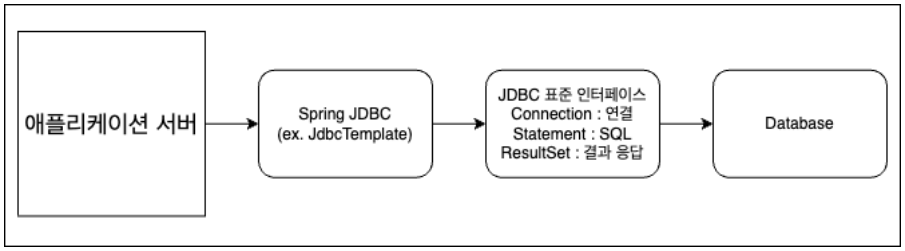
- 기존에 사용하던 MySQL 서버를 PostgreSQL 서버로 변경한다면 무슨일이 발생할까요?
- MySQL과 PostgreSQL은 커넥션을 연결하는 방법, SQL을 전달하는 방법, 결과를 응답받는 방법 모두 다를 수 있습니다.
- 따라서 애플리케이션 서버에서 작성했던 DB 연결 로직들을 전부 수정해야합니다.



- 이러한 문제를 해결하기위해 JDBC 표준 인터페이스가 등장했습니다.
- JDBC는 Java Database Connectivity로 DB에 접근할 수 있도록 Java에서 제공하는 API입니다.

- JDBC에 연결해야하는 DB의 JDBC 드라이버를 제공하면 DB 연결 로직을 변경할 필요없이 DB 변경이 가능합니다.
    - DB 회사들은 자신들의 DB에 맞도록 JDBC 인터페이스를 구현한 후 라이브러리로 제공하는데 이를 JDBC 드라이버라 부릅니다.
- 따라서, MySQL 드라이버를 사용해 DB에 연결을 하다 PostgreSQL 서버로 변경이 필요할 때 드라이버만 교체하면 손쉽게 DB 변경이 가능합니다.

▼ JdbcTemplate이란?



- JDBC의 등장으로 손쉽게 DB교체가 가능해졌지만 아직도 DB에 연결하기 위해 여러가지 작업 로직들을 직접 작성해야한다는 불편함이 남았습니다.
- 이러한 불편함을 해결하기 위해 커넥션 연결, statement 준비 및 실행, 커넥션 종료 등의 반복적이고 중복되는 작업들을 대신 처리해주는 JdbcTemplate이 등장했습니다.

▼ JdbcTemplate 사용방법

1. application.properties에 DB에 접근하기 위한 정보를 작성합니다.

    ▼ [코드 스니펫] application.properties

    ```
    spring.datasource.url=jdbc:mysql://localhost:3306/memo
    spring.datasource.username=root
    spring.datasource.password={비밀번호}
    spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
    ```

2. build.gradle에 JDBC 라이브러리와 MySQL을 등록합니다.

    ▼ [코드 스니펫] build.gradle : JDBC, MySQL 추가

    ```
    // MySQL
    implementation 'mysql:mysql-connector-java:8.0.28'
    implementation 'org.springframework.boot:spring-boot-starter-data-jdbc'
    ```

    ▼ 프로젝트 설정

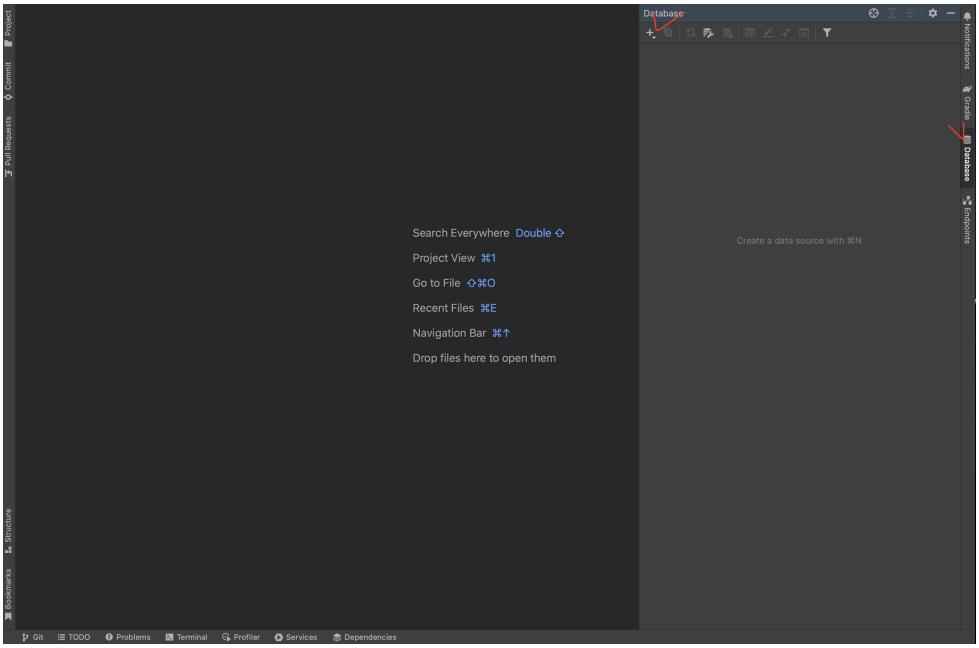
    - Intellij Database 연동

        ▼ CREATE DATABASE

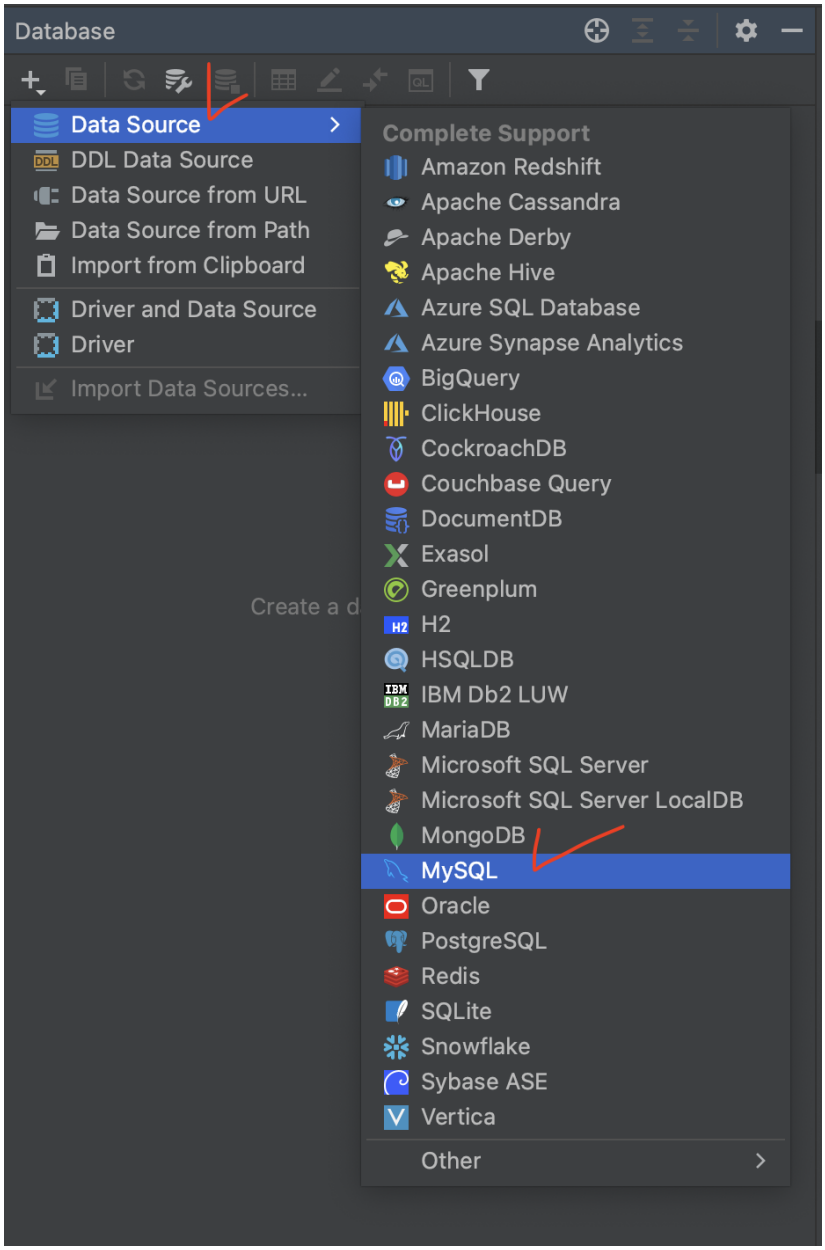
        ```
        CREATE DATABASE memo;
        ```

        ▼ 연동 순서

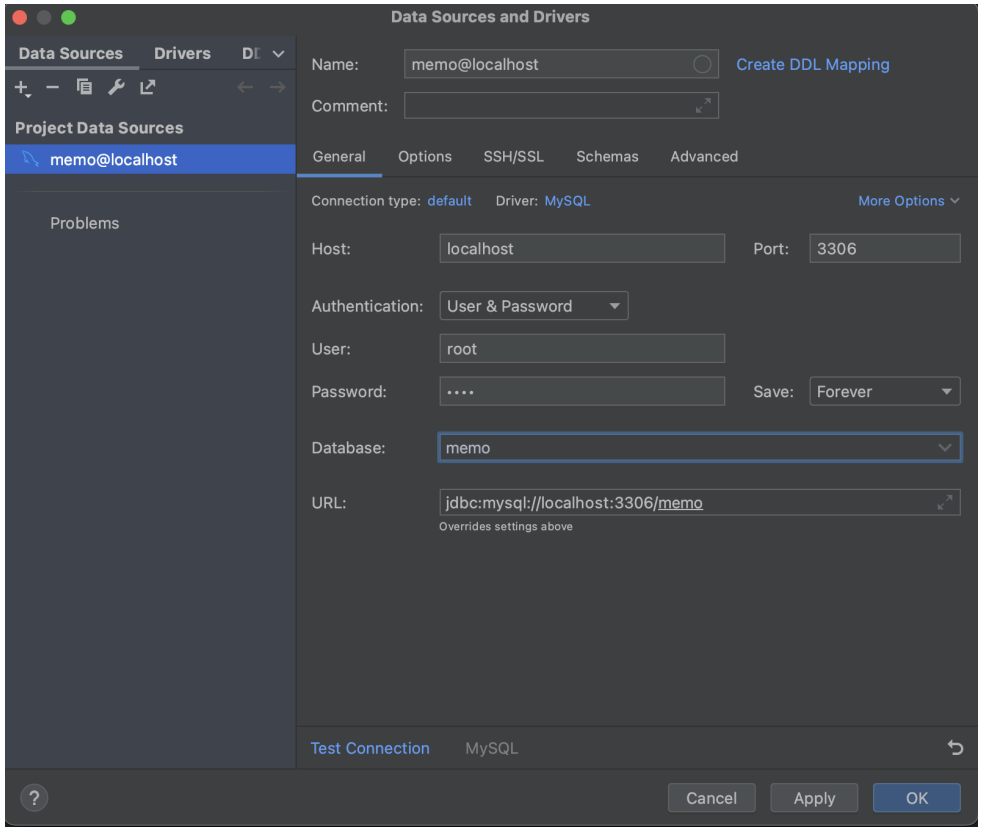
        1. Database 탭을 클릭하시고 + 버튼을 누릅니다.



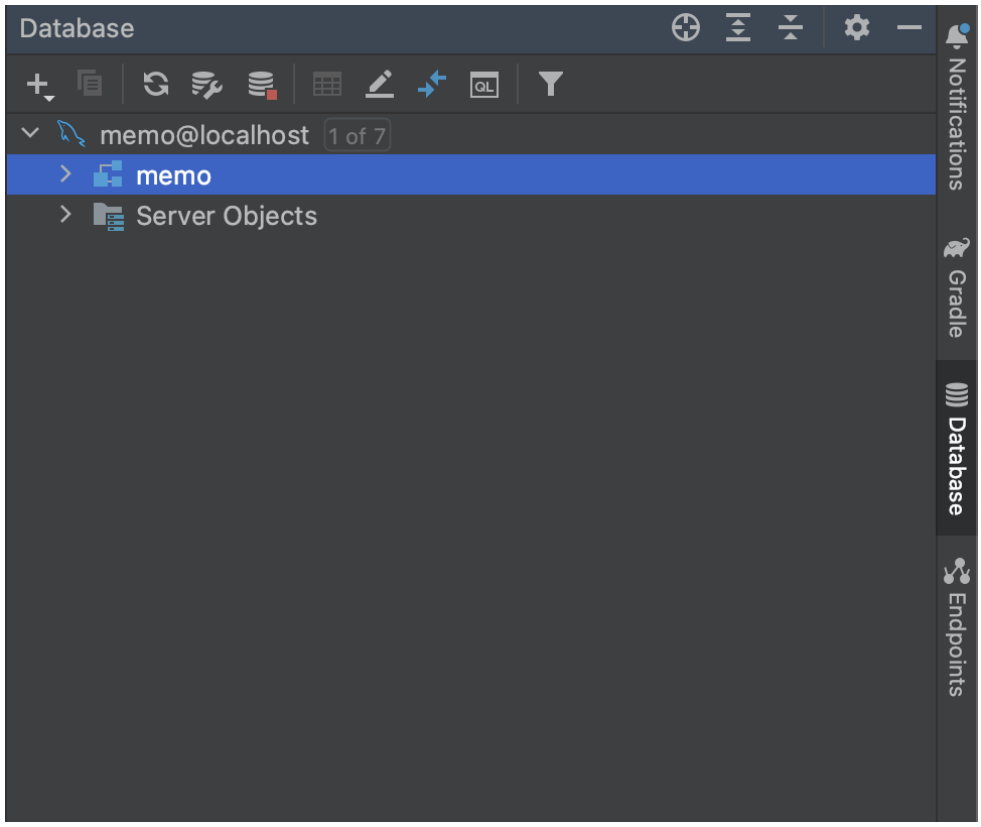
2. Data Source > MySQL 를 클릭합니다.



3. User, Password, Database 정보를 추가한 후 Ok를 클릭합니다.



4. MySQL Database에 연결이 완료되었습니다.



▼ [코드 스니펫] create memo table

```
create table memo
(
    id       bigint       not null auto_increment,
    contents varchar(500) not null,
    username varchar(255) not null,
    primary key (id)
);
```

2. DB연결이 필요한 곳에서 JdbcTemplate을 주입받아와 사용합니다.

```
private final JdbcTemplate jdbctemplate;

public MemoRepository(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
}
```

- 생성자의 파라미터를 통해 JdbcTemplate 객체가 자동으로 넘어와 jdbctemplate 변수에 저장됩니다.

- INSERT

```
String sql = "INSERT INTO memo (username, contents) VALUES (?, ?)";
jdbcTemplate.update(sql, "Robbie", "오늘 하루도 화이팅!");
```

  - INSERT SQL을 작성해 String 변수에 저장합니다.
    - 이때, 넣고자 하는 데이터 부분에 ?를 사용하면 유동적으로 데이터를 넣어줄 수 있습니다.
  - `jdbcTemplate.update()` 메서드는 INSERT, UPDATE, DELETE 와 같이 생성, 수정, 삭제에 사용될 수 있는데 첫 번째 파라미터로 SQL을 받고 그 이후에는 ?에 들어갈 값을 받습니다.
- UPDATE

```
String sql = "UPDATE memo SET username = ? WHERE id = ?";
jdbcTemplate.update(sql, "Robbert", 1);
```

  - UPDATE SQL을 작성해 String 변수에 저장한 후 `update()` 메서드 첫 번째 파라미터에 넣어줍니다.
    - 이때, 넣고자 하는 데이터 부분에 ?를 사용하면 유동적으로 데이터를 넣어줄 수 있습니다.
- DELETE

```
String sql = "DELETE FROM memo WHERE id = ?";
jdbcTemplate.update(sql, 1);
```

  - DELETE SQL을 작성해 String 변수에 저장한 후 `update()` 메서드 첫 번째 파라미터에 넣어줍니다.
    - 이때, 넣고자 하는 데이터 부분에 ?를 사용하면 유동적으로 데이터를 넣어줄 수 있습니다.
- SELECT

```
String sql = "SELECT * FROM memo";
return jdbcTemplate.query(sql, new RowMapper<MemoResponseDto>() {
    @Override
    public MemoResponseDto mapRow(ResultSet rs, int rowNum) throws SQLException {
        // SQL 의 결과로 받아온 Memo 데이터들을 MemoResponseDto 타입으로 변환해줄 메서드
        Long id = rs.getLong("id");
        String username = rs.getString("username");
        String contents = rs.getString("contents");
        return new MemoResponseDto(id, username, contents);
    }
});
```

- SELECT SQL을 작성해 String 변수에 저장한 후 `query()` 메서드 첫 번째 파라미터에 넣어줍니다.
- SELECT의 경우 결과가 여러 줄로 넘어오기 때문에 RowMapper를 사용하여 한 줄씩 처리 할 수 있습니다.
  - RowMapper는 인터페이스 이기 때문에 익명 클래스를 구현하여 처리합니다.
  - 오버라이딩 된 mapRow 메서드는 제네릭스에 선언한 MemoResponseDto 타입으로 데이터 한 줄을 변환하는 작업을 수행합니다.

▼ [코드 스니펫] MemoController

```
package com.sparta.memo.controller;

import com.sparta.memo.dto.MemoRequestDto;
import com.sparta.memo.dto.MemoResponseDto;
import com.sparta.memo.entity.Memo;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.support.GeneratedKeyHolder;
import org.springframework.jdbc.support.KeyHolder;
import org.springframework.web.bind.annotation.*;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.List;

@RestController
@RequestMapping("/api")
public class MemoController {

    private final JdbcTemplate jdbcTemplate;

    public MemoController(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    @PostMapping("/memos")
    public MemoResponseDto createMemo(@RequestBody MemoRequestDto requestDto) {
        // RequestDto -> Entity
        Memo memo = new Memo(requestDto);

        // DB 저장
        KeyHolder keyHolder = new GeneratedKeyHolder(); // 기본 키를 반환받기 위한 객체

        String sql = "INSERT INTO memo (username, contents) VALUES (?, ?)";
        jdbcTemplate.update( con -> {
                    PreparedStatement preparedStatement = con.prepareStatement(sql,
                            Statement.RETURN_GENERATED_KEYS);

                    preparedStatement.setString(1, memo.getUsername());
                    preparedStatement.setString(2, memo.getContents());
                    return preparedStatement;
                },
                keyHolder);

        // DB Insert 후 받아온 기본키 확인
        Long id = keyHolder.getKey().longValue();
        memo.setId(id);

        // Entity -> ResponseDto
        MemoResponseDto memoResponseDto = new MemoResponseDto(memo);

        return memoResponseDto;
    }

    @GetMapping("/memos")
    public List<MemoResponseDto> getMemos() {
        // DB 조회
        String sql = "SELECT * FROM memo";

        return jdbcTemplate.query(sql, new RowMapper<MemoResponseDto>() {
            @Override
            public MemoResponseDto mapRow(ResultSet rs, int rowNum) throws SQLException {
                // SQL 의 결과로 받아온 Memo 데이터들을 MemoResponseDto 타입으로 변환해줄 메서드
                Long id = rs.getLong("id");
                String username = rs.getString("username");
                String contents = rs.getString("contents");
                return new MemoResponseDto(id, username, contents);
            }
        });
```

```
    }

    @PutMapping("/memos/{id}")
    public Long updateMemo(@PathVariable Long id, @RequestBody MemoRequestDto requestDto) {
        // 해당 메모가 DB에 존재하는지 확인
        Memo memo = findById(id);
        if(memo != null) {
            // memo 내용 수정
            String sql = "UPDATE memo SET username = ?, contents = ? WHERE id = ?";
            jdbcTemplate.update(sql, requestDto.getUsername(), requestDto.getContents(), id);

            return id;
        } else {
            throw new IllegalArgumentException("선택한 메모는 존재하지 않습니다.");
        }
    }

    @DeleteMapping("/memos/{id}")
    public Long deleteMemo(@PathVariable Long id) {
        // 해당 메모가 DB에 존재하는지 확인
        Memo memo = findById(id);
        if(memo != null) {
            // memo 삭제
            String sql = "DELETE FROM memo WHERE id = ?";
            jdbcTemplate.update(sql, id);

            return id;
        } else {
            throw new IllegalArgumentException("선택한 메모는 존재하지 않습니다.");
        }
    }

    private Memo findById(Long id) {
        // DB 조회
        String sql = "SELECT * FROM memo WHERE id = ?";

        return jdbcTemplate.query(sql, resultSet -> {
            if(resultSet.next()) {
                Memo memo = new Memo();
                memo.setUsername(resultSet.getString("username"));
                memo.setContents(resultSet.getString("contents"));
                return memo;
            } else {
                return null;
            }
        }, id);
    }
}
```

💡 혹시! JdbcTemplate을 사용하여 DB와 소통하는 방법이 어렵고 복잡하다고 느껴지셨나요?
물론 JdbcTemplate이 JDBC를 직접 사용할 때 발생하는 불편함을 해결해 주었지만 아직도 복잡하고 사용하기 까다로운 것은 분명한 사실입니다.
다행히도 Java 개발자들을 위해 DB와 객체를 매핑하여 소통할 수 있는 ORM이라는 기술이 등장했습니다.
따라서 JdbcTemplate 사용방법은 가볍게 보고 넘어가도록 하겠습니다.

08. 2주차 끝 & 숙제설명

▼ 숙제 설명

❓ SQL 문제를 풀어봅니다.
SQL 연습하기 강의와 이어지는 문제입니다.

▼ 문제 1

👉 수강생을 관리하는 MANAGER 테이블을 만들어보세요.

- 컬럼은 총 id, name, student_code 입니다.
- id는 bigint 타입이며 PK입니다.
- name은 최소 2자 이상, varchar 타입, not null 입니다.
- student_code는 STUDENT 테이블을 참조하는 FK이며 not null 입니다.
- FK는 CONSTRAINT 이름을 'manager_fk_student_code' 로 지정해야합니다.

▼ 문제 2

👉 ALTER, MODIFY를 이용하여 MANAGER 테이블의 id 컬럼에 AUTO_INCREMENT 기능을 부여하세요.

▼ 문제 3

👉 INSERT를 이용하여 수강생 s1, s2, s3, s4, s5를 관리하는 managerA와 s6, s7, s8, s9를 관리하는 managerB를 추가하세요.

- AUTO_INCREMENT 기능을 활용하세요

▼ 문제 4

👉 JOIN을 사용하여 managerA가 관리하는 수강생들의 이름과 시험 주차 별 성적을 가져오세요.

▼ 문제 5

👉 STUDENT 테이블에서 s1 수강생을 삭제했을 때 EXAM에 있는 s1수강생의 시험성적과 MANAGER의 managerA가 관리하는 수강생 목록에 자동으로 삭제될 수 있도록 하세요.

- ALTER, DROP, MODIFY, CASCADE 를 사용하여 EXAM, MANAGER 테이블을 수정합니다.

- 제출 파일
  - 문제에 대한 답안 SQL 전체

HW. 2주차 숙제 답안 코드

▼ [코드 스니펫] - 1주차 숙제 답안 코드

▼ 문제 1 답안

```
CREATE TABLE IF NOT EXISTS MANAGER
(
    id bigint primary key,
    name varchar(100) not null,
    student_code varchar(100) not null,
```

```
    CONSTRAINT manager_fk_student_code foreign key(student_code) references student(student_code)
);
```

▼ 문제 2 답안

```
ALTER TABLE MANAGER MODIFY COLUMN id bigint auto_increment;
```

▼ 문제 3 답안

```
INSERT INTO MANAGER(name, student_code) VALUES('managerA', 's1');
INSERT INTO MANAGER(name, student_code) VALUES('managerA', 's2');
INSERT INTO MANAGER(name, student_code) VALUES('managerA', 's3');
INSERT INTO MANAGER(name, student_code) VALUES('managerA', 's4');
INSERT INTO MANAGER(name, student_code) VALUES('managerA', 's5');

INSERT INTO MANAGER(name, student_code) VALUES('managerB', 's6');
INSERT INTO MANAGER(name, student_code) VALUES('managerB', 's7');
INSERT INTO MANAGER(name, student_code) VALUES('managerB', 's8');
INSERT INTO MANAGER(name, student_code) VALUES('managerB', 's9');
```

▼ 문제 4 답안

```
SELECT s.name, e.exam_seq, e.score
FROM MANAGER m JOIN STUDENT S on m.student_code  = s.student_code
JOIN EXAM e on m.student_code  = e.student_code WHERE m.name = 'managerA';
```

▼ 문제 5 답안

```
ALTER TABLE EXAM DROP CONSTRAINT exam_fk_student_code;
ALTER TABLE EXAM ADD CONSTRAINT exam_fk_student_code FOREIGN KEY(student_code) REFERENCES STUDENT(student_code) ON DELETE CAS
ALTER TABLE MANAGER DROP CONSTRAINT manager_fk_student_code;
ALTER TABLE MANAGER ADD CONSTRAINT manager_fk_student_code FOREIGN KEY(student_code) REFERENCES STUDENT(student_code) ON DELE

DELETE FROM STUDENT WHERE student_code = 's1';
```