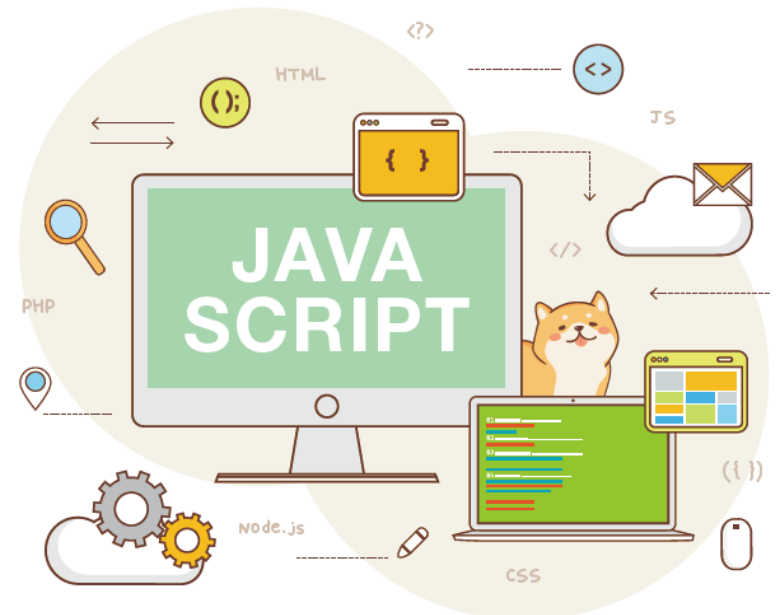




## 스마트헬스 케어 - 12주차

# 학습 내용

1. DB 쿼리 기본
2. DB 연결 노드
3. 간단 웹 애플리케이션



---

# 이론 및 예제 실습

---

# 1\_기본 쿼리 실습

- SELECT는 테이블의 데이터를 읽어 출력

- **SELECT** 필드목록  
**FROM** 테이블  
**WHERE** 조건  
**[ORDER BY 정렬기준]**

- SELECT와 FROM 사이의 필드 목록에 출력할 필드의 이름을 지정하되 \* 기호는 모든 필드 출력
  - SELECT \* FROM city;

- WHERE 절은 읽을 레코드의 조건을 지정한다.

- 필드와 특정값을 비교하는 조건문 형식으로 작성한다.
  - SELECT \* FROM city WHERE Population > 700000;
- 특정 레코드의 특정 필드만 표시
  - SELECT Name, Population FROM city WHERE Population > 700000;
- 문자열이나 날짜는 작은 따옴표로 감싼다.
- **AND**는 두 조건이 모두 참인 레코드를 검색하며 **OR**는 두 조건 중 하나라도 참인 레코드를 검사
  - SELECT \* FROM city WHERE Population >= 100000 AND Population >= 700000;

The screenshot shows a SQL query editor with a query window and a results grid. The query is `select * from city;`. The results grid displays a list of cities with columns: ID, Name, CountryCode, District, and Population. The data is as follows:

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabul	1780000
2	Oandahar	AFG	Oandahar	237500
3	Herat	AFG	Herat	186800
4	Mazar-e-Sharif	AFG	Balkh	127800
5	Amsterdam	NLD	Noord-Holland	731200
6	Rotterdam	NLD	Zuid-Holland	593321
7	Haag	NLD	Zuid-Holland	440900
8	Utrecht	NLD	Utrecht	234323
9	Eindhoven	NLD	Noord-Brabant	201843
10	Tilburg	NLD	Noord-Brabant	193238
11	Groningen	NLD	Groningen	172701
12	Breda	NLD	Noord-Brabant	160398
13	Apeldoorn	NLD	Gelderland	153491
14	Nijmegen	NLD	Gelderland	152463
15	Enschede	NLD	Overijssel	149544
16	Haarlem	NLD	Noord-Holland	148772
17	Almere	NLD	Flevoland	142465
18	Arnhem	NLD	Gelderland	138020

Below the results grid, the 'Action Output' section shows a message: '2000 row(s) returned'.

# 1\_기본 쿼리 실습

- 레코드를 추가하는 명령은 INSERT-INTO-VALUES이다.

- INSERT INTO 테이블 (필드목록) VALUES (값목록)
- INSERT INTO city (Name, CountryCode, Population) VALUES ('Seoul', 'KOR', 10000000);

- 명령문과 필드 목록은 딱 한 번만 밝히고 실제 삽입할 데이터만 나열할 수 있다.

- INSERT INTO city (Name, CountryCode, Population) VALUES ('Busan', 'KOR', 5000000), ('Daejeon', 'KOR', 4000000);

- 레코드를 삭제할 때는 DELETE-FROM-WHERE 명령을 사용한다.

- DELETE FROM 테이블 WHERE 조건
- DELETE FROM city WHERE Name = 'Busan';

- 레코드 값을 변경할 때는 UPDATE-SET-WHERE 명령을 사용한다.

SET 키워드 뒤에 필드에 값을 대입하는 대입문이 옴 (복수개의 필드를 한꺼번에 변경가능)

- UPDATE 테이블 SET 필드=값 [,필드=값] WHERE 조건
- 서울의 인구를 1100만명으로 변경  
UPDATE city SET Population = 11000000 WHERE Name = 'Seoul';

# 1\_기본 쿼리 실습

## ■ Workbench 사용

The screenshot displays the MySQL Workbench interface. The top pane shows a SQL query editor with a list of queries. The second query, `SELECT * FROM city;`, is highlighted with a red box. The bottom pane shows the 'Result Grid' for this query, also highlighted with a red box. The grid contains 10 columns: ID, Name, CountryCode, District, and Population. The data is sorted by ID, showing cities from Odessa to Seoul. The bottom pane also shows the 'Output' tab with a log of executed queries and their results.

ID	Name	CountryCode	District	Population
4064	Odessa	USA	Texas	89293
4065	Carson	USA	California	89089
4066	Charleston	USA	South Carolina	89063
4067	Charlotte Amalie	VIR	St Thomas	13000
4068	Harare	ZWE	Harare	1410000
4069	Bulawavo	ZWE	Bulawavo	621742
4070	Chitungwiza	ZWE	Harare	274912
4071	Mount Darwin	ZWE	Harare	164362
4072	Mutare	ZWE	Manicaland	131367
4073	Gweru	ZWE	Midlands	128037
4074	Gaza	PSE	Gaza	353632
4075	Khan Yunis	PSE	Khan Yunis	123175
4076	Hebron	PSE	Hebron	119401
4077	Jabalva	PSE	North Gaza	113901
4078	Nablus	PSE	Nablus	100231
4079	Rafah	PSE	Rafah	92020
4080	Seoul	KOR		10000000
NULL	NULL	NULL	NULL	NULL

city 17 x

Output

Action Output

#	Time	Action	Message
✓ 12	21:30:22	SELECT * FROM city LIMIT 0, 100	100 row(s) returned
✓ 13	21:30:49	SELECT DISTINCT CountryCode FROM city LIMIT 0, 100	100 row(s) returned
✓ 14	21:31:12	SELECT count(DISTINCT CountryCode) FROM city LIMIT 0, 100	1 row(s) returned
✓ 15	21:33:36	SELECT * FROM city LIMIT 0, 100	100 row(s) returned
✓ 16	21:34:40	SELECT COUNT(*) as 'CityCount' CountryCode FROM city GROUP BY CountryCode LIMIT 0, 100	100 row(s) returned

# 1\_기본 쿼리 실습

```
SELECT * FROM city WHERE Population > 700000;
```

```
SELECT Name, Population FROM city WHERE Population > 700000;
```

```
SELECT * FROM city WHERE Population >= 100000 AND Population >= 700000 ;
```

```
SELECT * FROM city WHERE name LIKE 'Over%';
```

```
SELECT * FROM city ORDER BY population DESC LIMIT 4;
```

```
INSERT INTO city (Name, CountryCode, Population) VALUES ('Seoul', 'KOR', 10000000);
```

```
INSERT INTO city (Name, CountryCode, Population) VALUES  
('Busan', 'KOR', 5000000),  
('Daejeon', 'KOR', 4000000);
```

```
DELETE FROM city WHERE Name = 'Busan';
```

```
UPDATE city SET Population = 11000000 WHERE Name = 'Seoul';
```

# 1\_기본 쿼리 실습

**쿼리 문제** : 다음 조건을 검색할 수 있는 쿼리를 만드시오

1. 'user' 테이블에서 userId가 'user3'이고 userPassword가 'bigdata123' 인 것 검색
2. 'countrylanguage' 테이블에서 언어(Language 필드)를 'Dutch'로 쓰는 나라의 국가코드(CountryCode 필드)를 검색하시오.
3. 'country' 테이블에서 'Australia'(Name 필드)의 GNP(GNP 필드)가 얼마인지 찾으시오.
4. 'city' 테이블에서 'Seoul' ('Name' 필드)의 인구(Population 필드)가 얼마인지 검색하시오.



## 2\_Database 노드 만들기

데이터 베이스 연결 노드 추가 (커스텀 노드 추가)

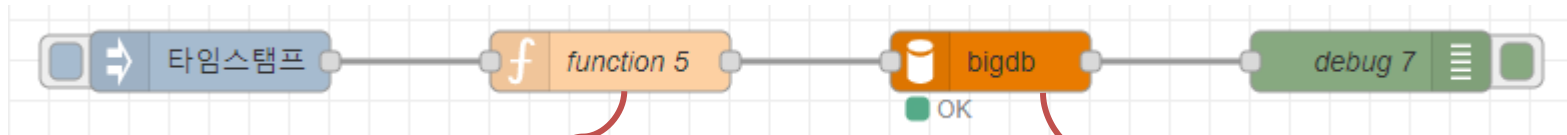
- 1) 메뉴에서 '팔레트 관리' 선택
- 2) '설치가 가능한 노드'에서 mysql로 검색
- 3) 추가 노드 설치 → 저장 팔레트에 'mysql'노드 추가 됨

The image shows a two-part process in the Node-RED web interface. On the left, a dark sidebar menu is open, with '팔레트 관리' (Palette Management) highlighted and circled in red. A red arrow points from this menu item to the right-hand panel. The right-hand panel, titled '사용자 설정' (User Settings), has tabs for '설치된 노드' (Installed Nodes) and '설치 가능한 노드' (Installable Nodes). The '설치 가능한 노드' tab is active. A search bar in the '팔레트' (Palette) section contains the text 'mysql'. Below the search bar, two nodes are listed: 'node-red-node-mysql' and 'node-red-node-mysql-test'. The 'node-red-node-mysql' node has a red box around its '설치됨' (Installed) button. At the bottom right, a '저장' (Save) panel shows a list of nodes in a palette, with 'mysql' highlighted and circled in red.

## 2\_Database 노드 만들기

### [1] 데이터 베이스 연결 노드 만들기

- 1) 쿼리 노드 만들기 : 함수 노드 → msg.topic에 SQL 쿼리를 입력
- 2) 데이터베이스 노드에 연결할 데이터베이스 설정을 입력



속성

이름: function 5

Setup On Start 코드 On Stop

```
1 msg.payload = {};  
2 msg.topic =  
3 "SELECT * FROM city limit 10 ;";  
4  
5 return msg;
```

\* msg.topic 에 쿼리를 담음

```
let param = msg.payload;  
  
msg.topic =  
"SELECT * FROM city limit 10 ;";  
  
msg.payload = {};  
  
return msg;
```

속성

Database: bigdb

이름: 이름

속성

Host: 127.0.0.1

Port: 3306

User: user1

Password: .....

Database: bigdb

Timezone: ±hh:mm

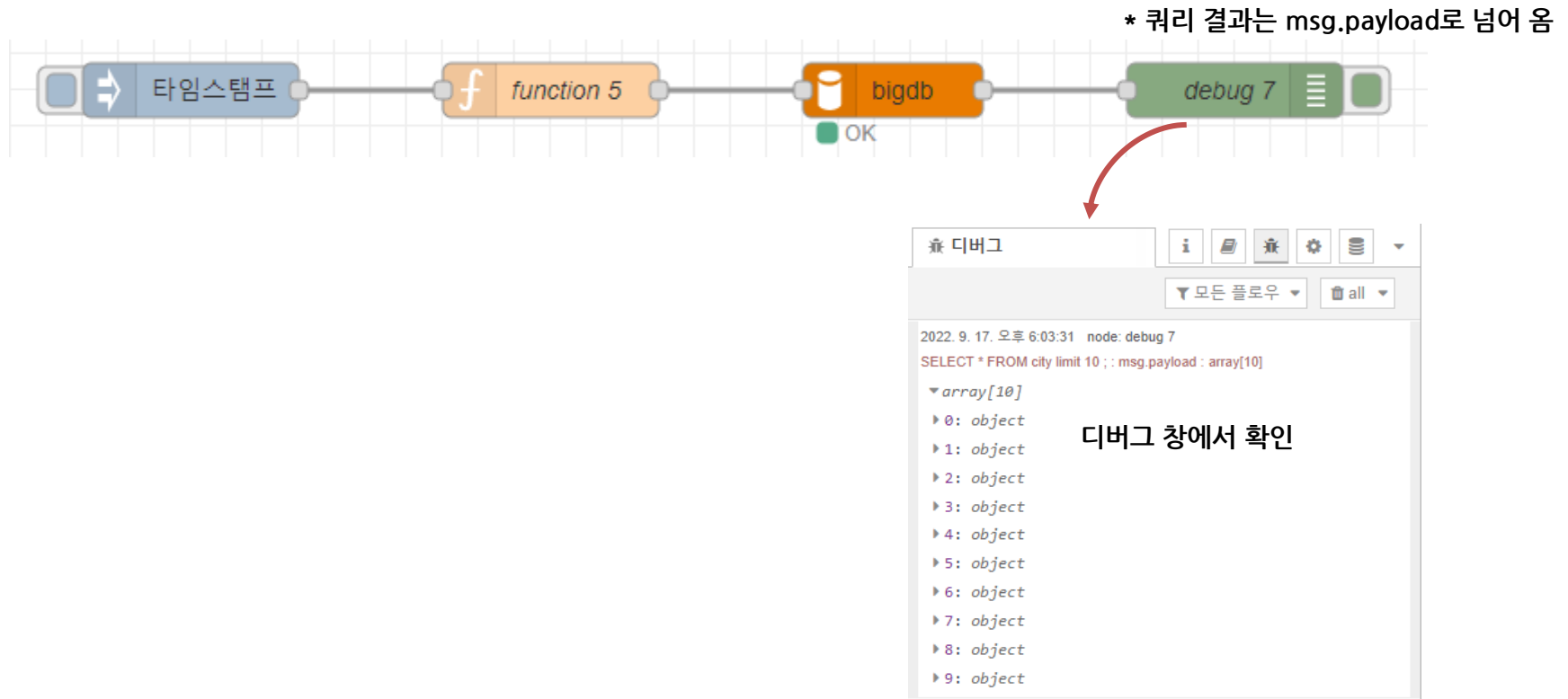
Charset: UTF8

\* Database 접속정보 입력

## 2\_Database 노드 만들기

[1] 데이터 베이스 연결 노드 만들기

3) 로그로 출력 : 쿼리 결과가 msg.payload에 담겨있음



## 2\_Database 노드 만들기

### [2] 데이터 베이스 연결 API 만들기

- 1) 쿼리 노드 만들기 : 함수 노드 → msg.topic에 SQL 쿼리를 입력
- 2) 데이터베이스 노드에 연결할 데이터베이스 설정을 입력
- 3) http-in과 http-response 연결



localhost:1880/getdata

00\_관리용 02\_연구개발관련 03\_프로젝트관련

```
{
  "ID": 1,
  "Name": "Kabul",
  "CountryCode": "AFG",
  "District": "Kabul",
  "Population": 1780000
}
```

속성

이름 function 5

Setup On Start 코드 On Stop

```
1 msg.payload = {};  
2 msg.topic =  
3 "SELECT * FROM city limit 10 ;";  
4  
5 return msg;
```

속성

Database bigdb

이름 이름

속성

메소드 GET

URL /getdata

이름 이름

속성

Host 127.0.0.1

Port 3306

User user1

Password .....

Database bigdb

Timezone ±hh:mm

Charset UTF8

\* Database 접속정보 입력



## 2\_Database 노드 만들기

### [2] 데이터 베이스 연결 API 만들기

#### 4) Ajax로 호출 해 보기

```
<script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
<script>
$.ajax({
    type : "GET",
    url  : "http://localhost:1880/getdata",
    data : {},
    dataType : "text",
    success : function (data) {
        $('body').append('<div>' + data + '</div>');
        console.log(JSON.parse(data));
    },
    error : function () {
        alert("통신실패!!!!");
    }
});
</script>
```

---

## 응용 예제

---

# 실습-1: 간단 웹애플리케이션 만들기

## 로그인 웹페이지를 만들고 정보를 확인하여 로그인 결과를 알려주는 프로그램 작성

1. 로그인 페이지 작성
  - id 입력, pw 입력, '로그인' 버튼 작성
2. Node-Red로 로그인 결과를 확인해 주는 API 작성
  - API 이름은 login, 타입은 'get' 방식
  - 클라이언트에서 보내온 파라미터에서 id/pw를 받고 아래 값과 비교하여 로그인 성공여부 판단  
user : "user1"  
password : "bigdata123"
  - 유효한 사용자가 있고 비번이 맞으면 { result : "OK" } 로 응답  
없거나 비번이 틀리면 { result : "NO" }로 응답
3. 로그인 페이지에서 호출 Ajax 작성
  - : 2번에서 작성한 API와 연결하여 로그인 결과를 받고 성공여부를 메시지 박스(alert)로 출력



웹 페이지

- AJAX로 로그인 여부 확인후  
결과 알림



데이터 가공  
API

- 로그인 확인 API (login)  
: 로그인 확인 결과를 http로 실어 보냄

# 실습-1: 간단 웹애플리케이션 만들기

## 로그인 웹페이지

```
<body>
  <div>
    <div>
      USER :
      <input type="text" id="idText" style="width: 300px" />
    </div>
    <div>
      PASSWORD :
      <input type="password" id="pwText" style="width: 300px" />
    </div>
    <input type="button" id="loginBtn" style="width: 100px" value="로그인" />
  </div>

  <script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
  <script>
    const loginObj = document.querySelector("#loginBtn");

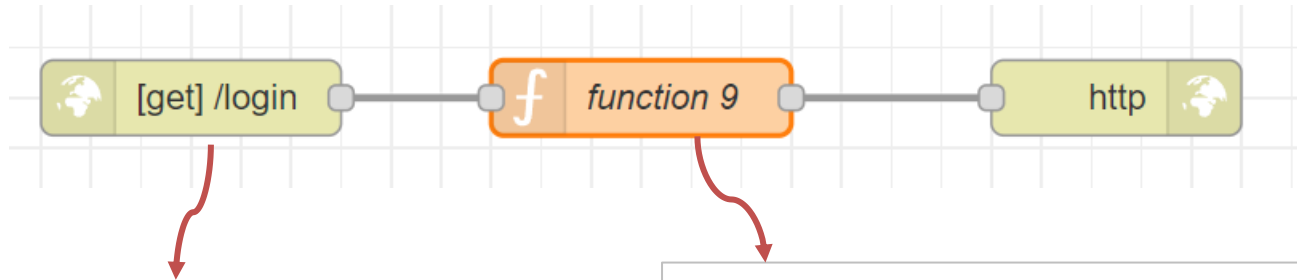
    window.onload = function () {
      loginObj.onclick = function (event){
        login();
      };
    };

  </script>
</body>
```



# 실습-1: 간단 웹애플리케이션 만들기

## 로그인 API 작성



http in의 노드 수정

속성

메소드 GET

URL /login

이름 이름

```
let param = msg.payload;
console.log(param);

let result = "NO";
if (param.id == "user1" &&
    param.password == "bigdata123") {
    result = "OK";
}

let senddata = {
    result : result
};

msg.payload = senddata;
return msg;
```

# 실습-1: 간단 웹애플리케이션 만들기

## 로그인 API 호출 Ajax

```
function login() {  
    let id = document.querySelector("#idText").value;  
    let pw = document.querySelector("#pwText").value;  
  
    $.ajax({  
        type: "GET",  
        url: "http://127.0.0.1:1880/login",  
        data: { id: id, password: pw },  
        dataType: "json",  
        success: function (data) {  
            console.log(data.result);  
            alert(data.result);  
        },  
        error: function () {  
            alert("통신실패!!!!");  
        },  
    });  
}
```

---

# 팀 프로젝트

---