1. SQL 기본

SQL(Structured Query Language, 구조화된 질의 언어)문은 데이터베이스에서 사용되는 일종의 공통언어다. DBMS에게는 SQL 문으로 질문하고 명령을 지시해야만 DBMS가 알아듣고 작업을 수행한 후 그 결과 값을 우리에게 준다.

SQL은 일반적으로 대소문자를 구분하지 않는다. 하지만, 여러가지 면에서 전체 대문자 또는 소문자 등으로 통일하는 것이 구문을 읽기 쉽게 하여, MySQL의 성능에도 약간 도움이 된다.

2. <SELECT / INSERT / UPDATE / DELETE>

1. SELECT

데이터베이스 내의 테이블에서 원하는 정보를 추출하는 명령

|  |
| --- |
| SELECT 열이름  FROM 테이블이름  WHERE 조건 |

예시1. user테이블에서 김경호라는 이름 찾기

|  |
| --- |
| SELECT \* FROM user WHERE name=’김경호’; |

예시2. 1970년 이후에 출생하**고**, 신장이 182 이상인 사람의 아이디와 이름 조회

|  |
| --- |
| SELECT id, name FROM user WHERE birth>=1970 **AND** height>=182; |

예시3. 1970년 이후에 출생했**거나** 신장이 182 이상인 사람의 아이디와 이름 조회

|  |
| --- |
| SELECT id, name FROM user WHERE birth>=1970 **OR** height>=182; |

예시4. 키가 180~183인 사람 조회

|  |
| --- |
| SELECT \* FROM user WHERE height>=180 AND height<=183;  SELECT \* FROM user WHERE height BETWEEN 180 AND 183; |

예시5. 지역이 경남, 전북, 경북인 사람의 정보 조회

|  |
| --- |
| SELECT \* FROM user WHERE addr=’경남’ OR addr=’전북’ OR addr=’경북’  SELECT \* FROM user WHERE addr **IN** (‘경남’, ‘전북’, ‘경북’); |

예시6. 김씨 조회

|  |
| --- |
| SELECT \* FROM user WHRER name **LIKE** ’김**%**’; |

% : 그 뒤로는 무엇이든 허용한다는 의미

\_ : 한 글자만 매치됨

예시7. 맨 앞 글자가 한 글자이고, 그 다음이 종신인 사람을 조회

|  |
| --- |
| SELECT \* FROM user WHERE name **LIKE** ‘**\_**종신’; |

예시8. 가입한 순서대로 회원을 출력

|  |
| --- |
| SELECT \* FROM user **ORDER BY** date; |

예시9. 키가 큰 순서대로 정렬하되, 만약 키가 같을 경우에 이름 순으로 정렬

|  |
| --- |
| SELECT \* FROM user **ODER BY** height **DESC**, name **ASC**; |

예시10. 중복 없이 거주지역 출력

|  |
| --- |
| SELECT **DISTINCT** addr FROM user; |

예시11. 입사일이 오래된 직원 5명의 사원번호 검색

|  |
| --- |
| SELECT emp\_no FROM user ODER BY hire\_date ASC **LIMIT 5**; |

GROUP BY절은 말 그대로 그룹으로 묶어주는 역할을 한다.

집계 함수는 주로 GROUP BY절과 함께 쓰이며 데이터를 그룹화 해주는 기능을 한다.

예시 12. **사용자 별**로 구매한 개수를 합쳐서 출력하고, 합친 결과는 ‘총 구매 개수’로 출력

|  |
| --- |
| SELECT userID, **SUM(amount)** **AS ‘총 구매 개수’**  FROM user  **GROUP BY userID**; |

예시13. 구매액의 총합. 구매액은 가격\*수량이다.

|  |
| --- |
| SELECT userID, **SUM(price\*amount)** AS ‘총 구매액’  FROM user  GROUP BY userID; |

집계함수

|  |  |
| --- | --- |
| 함수명 | 설명 |
| AVG() | 평균을 구한다. |
| MIN() | 최소값을 구한다. |
| MAX() | 최대값을 구한다. |
| COUNT() | 행의 개수를 센다. |
| COUNT(DISTINCT) | 행의 개수를 센다.(중복은 1개만 인정) |
| STDEV() | 표준편차를 구한다. |
| VAR\_SAMP() | 분산을 구한다. |

예시14. 전체 구매자가 구매한 물품의 개수의 평균

|  |
| --- |
| SELECT AVG(amount)  FROM user; |

예시15. 각 **사용자 별**로 한 번 구매 시 물건을 평균 몇 개 구매했는지 평균

|  |
| --- |
| SELECT userID, AVG(amount) AS ‘평균 구매 개수;  FROM user  **GROUP BY userID**; |

예시16. 가장 큰 키와 가장 작은 키의 회원 이름과 키를 출력

|  |
| --- |
| SELECT name, height  FROM user  WHERE height = (SELECT MAX(height) FROM user) OR  height = (SELECT MIN(height) FROM user); |

예시17. 휴대폰이 있는 사용자의 수를 카운트

|  |
| --- |
| SELECT COUNT(mobile)  FROM user; |

예시18. 사용자 별 총 구매액 중 1000이상인 사용자

|  |
| --- |
| SELECT userID, SUM(price\*amount) AS ‘총 구매 액’  FROM user  GROUP BY user ID  **HAVING SUM(price\*amount) > 1000**; |

HAVING은 WHERE와 비슷한 개념으로 조건을 제한하는 것이지만, 집계 함수에 대해서 조건을 제한하는 것이라고 생각하면 된다. 그리고 HAVING절은 꼭 GROUP BY절 다음에 나와야 한다. 순서가 바뀌면 안 된다.

예시19. 위 예제에서 총 구매액이 적은 사용자부터 나타내라.

|  |
| --- |
| SELECT userID, SUM(price\*amount) AS ‘총 구매 액’  FROM user  GROUP BY user ID  **HAVING SUM(price\*amount) > 1000**  **ORDER BY SUM(price\*amount) ;** |

1. INSERT

|  |
| --- |
| INSERT INTO 테이블 VALUES(); |

1. UPDATE
2. DELETE