Kt ds 신입사원 기술 온보딩

Database with MySQL

강사 : 백인욱

Database with MySQL

1.DBMS개요와 MySQL

2.SQL STATEMENT

3.SQL FUNCTION

4.JOIN & SUBQUERY

5.TABLE & VIEW

6.INDEX

DBMS개요와 MySQL

Database란?

데이터베이스

‘데이터의 집합’

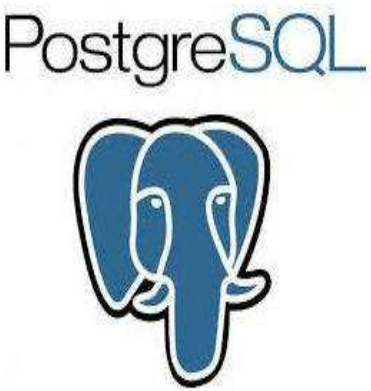
여러 명의 사용자나 응용프로그램이 공유하는 데이터들

동시에 접근 가능해야

‘데이터의 저장 공간’ 자체

DBMS

데이터베이스를 관리·운영하는 역할



DBMS의 주요기능

- **트렌잭션관리**
  - 데이터베이스 트랜잭션을 관리하여 데이터의 일관성과 무결성을 보장
- **무결성 제약 조건**
  - 데이터의 정확성과 신뢰성을 유지하기 위해 다양한 제약 조건을 제공
- **접근제어 및 보안**
  - 사용자 권한 관리와 접근 제어를 통해 보안 강화
- **데이터 백업 및 복구**
  - 데이터 손실을 방지하고 복구할 수 있는 기능을 제공
- **정규화**
  - 데이터 중복을 최소화하고 저장 공간을 효율적으로 사용하도록 데이터베이스 설계

DBMS의 종류

- **관계형 DBMS (RDBMS)**
  - MySQL. PostgreSQL, Oracle
- **객체지향 DBMS(OODBMS)**
  - PostgreSQL, Oracle
- **NoSQL DBMS**
  - MongoDB, Cassandra, Redis
- **객체지향 DBMS(OODBMS)**
  - PostgreSQL, Oracle

SQL과 PL/SQL

▶DDL (Data Definition Language)

- ▷ 데이터베이스 객체를 생성, 삭제, 변경하는 언어
- ▷ CREATE : 테이블이나 인덱스, 뷰 등 데이터베이스 객체를 생성
- ▷ DROP : 생성된 데이터베이스 객체를 영구히 삭제
- ▷ ALTER : 기 생성된 데이터베이스 객체를 수정
- ▷ TRUNCATE : 테이블이나 클러스터의 데이터를 통째로 삭제

SQL과 PL/SQL

DML (Data Manipulation Language)

- 데이터베이스 객체를 조작하는 언어
- SELECT : 테이블이나 뷰에 있는 데이터를 조회
- INSERT : 데이터를 신규로 생성
- UPDATE : 기 생성된 데이터를 수정
- DELETE : 데이터 삭제
- COMMIT : 변경된 데이터를 최종 적용
- ROLLBACK : 변경된 데이터를 적용하지 않고 이전 상태로 되돌림.

SQL과 PL/SQL

PL/SQL

- SQL과는 달리 절차적 특징을 가진 언어
- 다른 프로그래밍 언어처럼 변수 선언과 할당, 함수 작성이 가능
- PL/SQL 코드 내에서 SQL을 사용 가능
- 앞으로 이 책에서 다룰 내용의 상당 부분이 PL/SQL에 해당됨

SQL Statement

SELECT

- 테이블이나 뷰에 있는 데이터를 선택(조회)할 때 사용
- 구문

```
SELECT  * 혹은 컬럼
FROM [스키마.]테이블명 혹은 [스키마.]뷰명
WHERE 조건
ORDER BY 컬럼;
```

- SELECT : 선택하고자 하는 컬럼명, 모든 컬럼을 조회하고 싶다면 *
- FROM : 선택할 테이블이나 뷰 명
- WHERE : 선택 조건, 여러 조건 기술 시에는 AND, OR로 연결
- ORDER BY : 조회 데이터 정렬 시, 정렬하고자 하는 컬럼명 기술

INSERT

▷ 새로 데이터를 입력해 넣을 때 사용하는 문장

▶ 기본형태

▷ 구문

```
INSERT INTO [스키마.]테이블명 (컬럼1, 컬럼2, …)
VALUES (값1, 값2, …);
```

- 테이블명 다음 컬럼 순서, 타입이 VALUES 다음 괄호 안의 값 순서와 타입이 일치해야 함

INSERT

▶ 컬럼명 기술 생략

▷ 구문

```
INSERT INTO [스키마.]테이블명
VALUES (값1, 값2, …);
```

- 테이블명 다음에 컬럼을 명시하지 않았으므로 테이블에 있는 모든 컬럼에 값을 넣는다는 의미
- VALUES 다음 값은 테이블에 있는 모든 컬럼 순서에 맞춰 넣어야 한다
- 컬럼 순서는 테이블 생성 시 명시한 컬럼 순서

INSERT

INSERT ~ SELECT 형태

구문

```
INSERT INTO [스키마.]테이블명 (컬럼1, 컬럼2, …)
SELECT 문;
```

- VALUES 절과 함께 값을 일일이 명시하는 대신 SELECT 문을 사용
- 테이블명 다음 컬럼 순서와 SELECT 다음의 컬럼 순서, 타입이 맞아야 함
- 테이블명 다음 컬럼 리스트를 생략할 경우 이 테이블의 모든 컬럼에 값을 넣는다는 의미

UPDATE

- 테이블에 있는 기존 데이터를 수정하는 문장
- 구문

```
UPDATE [스키마.]테이블명
SET 컬럼1 = 변경값1,
      컬럼2 = 변경값2,
….
WHERE 조건;
```

- SET : 변경하고자 하는 컬럼과 그 값을 명시, 여러 컬럼을 갱신할 때는 콤마로 분리
- WHERE : 데이터를 갱신하는 조건, 이 조건에 맞는 데이터만 변경됨. WHERE 조건을 생략하면 테이블에 있는 모든 데이터가 변경된다.

DELETE

- 테이블에 있는 데이터를 삭제하는 문장
- 구문

① 일반 구문

```
DELETE [FROM] [스키마.]테이블명
WHERE delete조건;
```

② 특정 파티션만 삭제할 경우의 구문

```
DELETE [FROM] [스키마.]테이블명 PARTITION (파티션명)
WHERE delete조건;
```

TRANSACTION

COMMIT

- 변경한 데이터를 데이터베이스에 최종적으로 반영
- 구문

```
COMMIT [WORK] [TO SAVEPOINT 세이브포인트명] ;
```

ROLLBACK

- 변경한 데이터를 변경 전 상태로 되돌림
- 구문

```
ROLLBACK [WORK] [TO SAVEPOINT 세이브포인트명] ;
```

OPERATOR

- 수식연산자 : +, -, *, /
- 문자연산자 : ||
- 논리연산자 : >, <, >=, <=, =, <>, !=, ^=
- 집합연산자 : UNION, UNION ALL, INTERSECT, MINUS
- 계층형 쿼리 연산자 : PRIOR, CONNECT_BY_ROOT

EXPRESSION

- 한 개 이상의 값과 연산자 그리고 SQL 함수 등이 결합된 식

CASE 표현식

- 구문

```
CASE WHEN 조건1 THEN 값1
       WHEN 조건2 THEN 값2
        …
       ELSE 기타값
END
```

CONDITION

- 조건 혹은 조건식(Condition)은 한 개 이상의 표현식과 논리 연산자가 결합된 식
- TRUE, FALSE, UNKNOWN 3가지 타입을 반환

비교 조건식

- 논리 연산자나 ANY, SOME, ALL 키워드로 비교하는 조건식

논리 조건식

- AND, OR, NOT을 사용하는 조건식

NULL 조건식

- 특정 값이 NULL인지 여부를 체크하는 조건식
- IS NULL, IS NOT NULL

CONDITION

- **BETWEEN AND 조건식**
  - 범위에 해당되는 값을 찾을 때 사용
- **IN 조건식**
  - 조건절에 명시한 값이 포함된 건을 반환, ANY와 비슷
- **EXISTS 조건식**
  - IN과 비슷하지만 후행 조건절로 값의 리스트가 아닌 서브쿼리만 올 수 있다
  - 또한 서브쿼리 내에서 조인조건이 있어야 한다

SQL FUNCTION

NUMERIC FUNCTION

▷ **ABS**(n) : n의 절대값 반환

예) ABS(3) ➔ 3, ABS(-3) ➔ 3

▷ **CEIL**(n) : n과 같거나 가장 큰 정수 반환

예) CEIL(10.123) ➔ 11, CEIL(10.541) ➔ 11

▷ **FLOOR**(n) : n보다 작거나 가장 큰 정수 반환

예) FLOOR(10.123) ➔ 10, FLOOR(10.541) ➔ 10

▷ **ROUND**(n, i) : n을 소수점 기준 (i+1) 번째에서 반올림한 결과 반환

예) ROUND(10.154) ➔ 10, ROUND(10.154, 2) ➔ 10.15

NUMERIC FUNCTION

▷ **TRUNC**(n1, n2) : n1을 소수점 기준 n2자리에서 무조건 잘라낸 결과를 반환

예) TRUNC(115.155) ➔ 115, TRUNC(115.155, 1) ➔ 115.1

▷ **POWER**(n2, n1) : n2를 n1 제곱한 결과를 반환, n2가 음수이면 n1은 반드시 정수

예) POWER(3, 2) ➔ 9, POWER(3, 3) ➔ 27

▷ **SQRT**(n) : n의 제곱근 반환

예) SQRT(2) ➔ 1.41421356, SQRT(5) ➔ 2.23606798

▷ **MOD**(n2, n1) : n2를 n1으로 나눈 나머지 값을 반환

예) MOD(19,4) ➔ 3, MOD(19.123, 4.2) ➔ 2.323

NUMERIC FUNCTION

▷ **ABS**(n) : n의 절대값 반환

예) ABS(3) ➔ 3, ABS(-3) ➔ 3

▷ **CEIL**(n) : n과 같거나 가장 큰 정수 반환

예) CEIL(10.123) ➔ 11, CEIL(10.541) ➔ 11

▷ **FLOOR**(n) : n보다 작거나 가장 큰 정수 반환

예) FLOOR(10.123) ➔ 10, FLOOR(10.541) ➔ 10

▷ **ROUND**(n, i) : n을 소수점 기준 (i+1) 번째에서 반올림한 결과 반환

예) ROUND(10.154) ➔ 10, ROUND(10.154, 2) ➔ 10.15

STRING FUNCTION

▷ **INITCAP**(char) : char의 첫 문자는 대문자로, 나머지는 소문자로 반환
첫 문자 인식 기준은 공백 그리고 알파벳과 숫자를 제외한 문자

예) INITCAP('never say goodbye') ➔ Never Say Goodbye

▷ **LOWER**(char) : 소문자 변환 후 반환

예) LOWER('NEVER SAY GOODBYE') ➔ never say goodbye

▷ **UPPER**(char) : 대문자 변환 후 반환

예) UPPER('never say goodbye') ➔ NEVER SAY GOODBYE

▷ **CONCAT**(char1, char2) : 두 문자를 붙여 반환

예) CONCAT('I Have', ' A Dream') ➔ I Have A Dream

STRING FUNCTION

- **SUBSTR**(char, pos, len) : char의 pos번째 문자부터 len 길이만큼 잘라낸 결과를 반환

예) SUBSTR('ABCDEFG', 1, 4) ➔ ABCD, SUBSTR('ABCDEFG', -1, 4) ➔ G

- **SUBSTRB**(char, pos, len) : SUBSTR과 같으나 문자 개수가 아닌 바이트 수 단위

예) SUBSTRB('ABCDEFG', 1, 4) ➔ ABCD, SUBSTRB('가나다라마바사', 1, 4) ➔ 가나

- **LTRIM**(char, set) : char에서 set으로 지정된 문자열을 왼쪽 끝에서 제거 후 나머지 문자열

예) LTRIM('ABCDEFGABC', 'ABC') ➔ DEFGABC

- **RTRIM**(char, set) : LTRIM과 반대로 오른쪽 끝에서 제거한 뒤 나머지 문자열을 반환

예) RTRIM('ABCDEFGABC', 'ABC') ➔ ABCDEFG

STRING FUNCTION

▷ **LPAD**(expr1, n, expr2) : expr2 문자열을 n자리만큼 왼쪽부터 채워 expr1을 반환

예) LPAD('111-1111', 12, '(02)') ➔ (02)111-1111

▷ **RPAD**(expr1, n, expr2) : LPAD와는 반대로 오른쪽에 해당 문자열을 채워 반환

예) RPAD('111-1111', 12, '(02)') ➔ 111-1111(02)

▷ **REPLACE**(char, search_str, replace_str) : char에서 search_str을 찾아 이를 replace_str로 대체한 결과를 반환

예) REPLACE('나는 너를 모르는데', '나', '너') ➔ 너는 너를 모르는데

▷ **TRANSLATE**(expr, from_str, to_str) : expr에서 from_str에 해당하는 문자를 찾아 to_str로 한 글자씩 바꾼 결과 반환

예) TRANSLATE('나는 너를 모르는데', '나는', '너를') ➔ 너를 너를 모르를데

STRING FUNCTION

- **INSTR**(str, substr, pos, occur) : str에서 substr과 일치하는 위치를 반환, pos는 시작위치, occur은 몇 번째 일치하는지를 명시

예) LPAD('111-1111', 12, '(02)') ➔ (02)111-1111

- **LENGTH**(chr) : 문자열의 길이 반환

예) LENGTH('대한민국') ➔ 4

- **LENGTHB**(chr) : 문자열의 BYTE수 반환

예) LENGTHB('대한민국') ➔ 8

DATE & TIME FUNCTION

▷ 날짜 및 시간 함수

```
DATE( ), TIME( )
     DATETIME 형식에서 연-월-일 및 시 : 분 : 초만 추출
DATEDIFF(날짜1, 날짜2), TIMEDIFF(날짜1 또는 시간1, 날짜1 또는 시간2)
     DATEDIFF( )는 날짜1-날짜2의 일수를 결과로 구함
DAYOFWEEK(날짜), MONTHNAME( ), DAYOFYEAR(날짜)
     요일(1:일, 2:월~7:토) 및 1년 중 몇 번째 날짜인지 구함
LAST_DAY(날짜)
     주어진 날짜의 마지막 날짜를 구함

MAKEDATE(연도, 정수)
     연도에서 정수만큼 지난 날짜 구함
MAKETIME(시, 분, 초)
     시, 분, 초를 이용해서 '시 : 분 : 초'의 TIME 형식 만듦
PERIOD_ADD(연월, 개월수), PERIOD_DIFF(연월1, 연월2)
     PERIOD_ADD( )는 연월에서 개월만큼의 개월이 지난 연월 구함
     PERIOD_DIFF( )는 연월1-연월2의 개월수 구함
QUARTER(날짜)
     날짜가 4분기 중에서 몇 분기인지를 구함
TIME_TO_SEC(시간)
     시간을 초 단위로 구함
```

GROUP FUNCTION

- **COUNT** : 쿼리 결과건 수, 로우 수 반환
- **SUM**(expr) : expr의 전체 합계
- **AVG**(expr) : expr의 평균
- **MIN**(expr) : expr의 최소값
- **MAX**(expr) : expr의 최대값
- **VARIANCE**(expr) : expr의 분산
- **STDDEV**(expr) : expr의 표준편차

GROUP FUNCTION

GROUP BY 절

- 특정 그룹으로 묶어 데이터 집계 시 사용
- WHERE와 ORDER BY절 사이에 위치
- 집계함수와 함께 사용
- SELECT 리스트에서 집계함수를 제외한 모든 컬럼과 표현식은 GROUP BY 절에 명시해야 함

HAVING 절

- GROUP BY절 다음에 위치해 GROUP BY한 결과를 대상으로 다시 필터를 거는 역할
- HAVING 다음에는 SELECT 리스트에 사용했던 집계함수를 이용한 조건을 명시

ROLLUP & CUBE

ROLLUP (expr1, expr2, ...)

- GROUP BY 절에서 사용됨
- expr로 명시한 표현식을 기준으로 집계한 결과, 추가 정보 집계
- expr로 명시한 표현식 수와 순서에 따라 레벨 별로 집계
- expr 개수가 n개 라면, n+1 레벨까지, 하위에서 상위 레벨 순으로 집계

CUBE (expr1, expr2, ...)

- GROUP BY 절에서 사용됨
- 명시한 표현식 개수에 따라 가능한 모든 조합별로 집계
- expr 개수가 3이면 2의 3승, 즉 총 8가지 종류로 집계됨

SET FUNCTION

UNION

- 집합의 합집합 개념
- 두 개 이상의 개별 SELECT 쿼리를 연결
- 개별 SELECT 쿼리 반환 결과가 중복될 경우 UNION 연산 결과는 한 로우만 반환됨

UNION ALL

- UNION과 유사
- 개별 SELECT 쿼리 반환 결과가 중복될 경우, 중복되는 건까지 모두 반환

SET FUNCTION

INTERSECT

- 집합의 교집합 개념
- 두 개 이상의 개별 SELECT 쿼리를 연결
- 개별 SELECT 쿼리 반환 결과 중 공통된 항목만 추출

MINUS

- 집합의 차집합 개념
- 두 개 이상의 개별 SELECT 쿼리를 연결
- 개별 SELECT 쿼리 반환 결과 중 중복된 건을 제외한 선행 쿼리 결과 추출

SET FUNCTION

집합 연산자 제한사항

- 개별 SELECT 쿼리의 SELECT 리스트 개수와 데이터 타입이 일치해야 함
- ORDER BY 절은 맨 마지막 개별 SELECT 쿼리에만 명시 가능함
- BLOB, CLOB, BFILE 같은 LOB 타입 컬럼은 집합 연산자 사용 불가
- UNION, INTERSECT, MINUS 연산자는 LONG형 컬럼에는 사용 불가

JOIN & SUB QUERY

JOIN

- **조인이란?**
  - 테이블(혹은 뷰)간의 관계를 맺는 방법
- **조인의 종류**
  - 조인 연산자에 따른 구분 : INNER JOIN, OUTER JOIN, CROSS JOIN
  - 조인대상에 따른 구분 : **SELF JOIN**

JOIN

내부조인

동등 조인(EQUI JOIN)

- 가장 기본적이고 일반적인 조인 방법
- WHERE 절에서 등호('=')연산자를 사용해 2개 이상의 테이블이나 뷰를 연결한 조인 ➔ 조인조건
- 컬럼 단위로 조인조건 기술
-
```
SELECT   *
  FROM   TAB1 a, TAB2 b
  WHERE  a.col1 = b.col1
  ….
```

JOIN

▷ 세미 조인(SEMI JOIN)

- 서브쿼리를 사용해 서브쿼리에 존재하는 데이터만 메인 쿼리에서 추출하는 조인
- WHERE 절에서 **IN** 과 **EXISTS** 연산자를 사용한 조인방법
-
```
SELECT   *
  FROM   TAB1 a
  WHERE  EXISTS ( SELECT 1
                  FROM   TAB2 b
                  WHERE  a.col1 = b.col1
                  ….
                );
```

-
```
SELECT   *
  FROM   TAB1 a
  WHERE  a.col1 IN ( SELECT b.col1
                     FROM   TAB2 b
                     WHERE  ….
                     ….
                     );
```

- 세미 조인은 서브쿼리에 존재하는 메인쿼리 데이터가 여러 건 존재하더라도 최종 반환되는 메인쿼리 데이터에는 중복되는 건이 없다 ➔ 일반 조인과의 차이점

JOIN

▷ 안티 조인(ANTI JOIN)

- 서브쿼리 테이블에는 없는, 메인쿼리 테이블의 데이터만 추출하는 조인 방법
- WHERE 절에서 **NOT IN** 과 **NOT EXISTS** 연산자를 사용한 조인방법
-
```
SELECT   *
  FROM   TAB1 a
  WHERE  NOT EXISTS ( SELECT 1
                      FROM   TAB2 b
                      WHERE  a.col1 = b.col1
                       ….
                     );
```

-
```
SELECT   *
  FROM   TAB1 a
  WHERE  a.col1 NOT IN ( SELECT b.col1
                         FROM   TAB2 b
                         WHERE  ….
                          ….
                        );
```

JOIN

▷ 셀프 조인(SELF JOIN)

- 서로 다른 두 테이블이 아닌 동일한 한 테이블을 사용해 조인

- ```
  SELECT   *
    FROM   TAB1 a,  TAB1 b
    WHERE a.col1 = b.col1
    ….
    ;
  ```
  ```

JOIN

외부조인

외부 조인(OUTER JOIN)

- 일반 조인을 확장한 개념
- 조인 조건에 만족하는 데이터 뿐만 아니라, 어느 한 쪽 테이블에 조인 조건에 명시된 컬럼에 값이 없거나(NULL 이더라도) 해당 로우가 아예 없더라도 데이터를 모두 추출
- 조인조건에서 데이터가 없는 쪽 테이블의 컬럼 끝에 (+)를 붙인다
- 조인조건이 여러 개일 경우 모든 조인조건에 (+)를 붙여야 한다

- SELECT a.department_id, a.department_name, b.job_id, b.department_id
  FROM departments a, job_history b
  WHERE a.department_id = b.department_id;

  ➔ 일반조인, 결과는 10건

- SELECT a.department_id, a.department_name, b.job_id, b.department_id
  FROM departments a, job_history b
  WHERE a.department_id = b.department_id(+);

  ➔ 외부조인, 결과는 31건

JOIN

▷ 외부 조인(OUTER JOIN)

- 한 번에 한 테이블에만 외부 조인

  A와 B 테이블을 외부 조인으로 연결했다면, 동시에 A와 C 테이블에 외부 조인을 걸 수는 없다.

- (+)연산자가 붙은 조건과 OR를 같이 사용할 수 없다
- (+)연산자가 붙은 조건에는 IN 연산자를 같이 사용할 수 없다. 단 IN절에 포함되는 값이

  1개인 때는 사용 가능하다.

JOIN

ANSI SQL 문법을 사용한 조인

ANSI 내부조인

```
SELECT A.컬럼1, A.컬럼2, B.컬럼1, B.컬럼2 …
  FROM 테이블 A
  INNER JOIN 테이블 B
  ON ( A.컬럼1 = B.컬럼1)  -- 조인 조건
  WHERE ….;
```

- 내부조인의 경우 FROM 절에 **INNER JOIN**을 명시
- 조인조건은 **ON** 절에 명시
- 테이블간 조인조건 외의 다른 조건은 WHERE 절에 명시한다.

SUB QUERY

▷ 서브쿼리란?

- 한 SQL 문장 안에서 보조로 사용되는 또 다른 SELECT문
- 메인 쿼리를 기준으로 보조 역할을 한다
- SELECT, FROM, WHERE절 뿐만 아니라, INSERT, UPDATE, MERGE, DELETE 문에서도

  서브쿼리를 사용할 수 있다

▷ 서브쿼리의 종류

- 메인쿼리와의 연관성에 따라

  - 연관성 없는(Noncorrelated) 서브쿼리
  - 연관성 있는 서브쿼리

- 형태에 따라

  - 일반 서브쿼리 (SELECT 절)
  - 인라인 뷰 (FROM 절)
  - 중첩쿼리 (WHERE 절)

SUB QUERY

▷ 인라인 뷰

- FROM절에 사용하는 서브쿼리를 말한다

```
SELECT a.employee_id, a.emp_name, b.department_id, b.department_name
  FROM employees a,
       departments b,
       ( SELECT AVG(c.salary) AS avg_salary
           FROM departments b,
                employees c
          WHERE b.parent_id = 90  -- 기획부
            AND b.department_id = c.department_id ) d
 WHERE a.department_id = b.department_id
   AND a.salary > d.avg_salary;
```

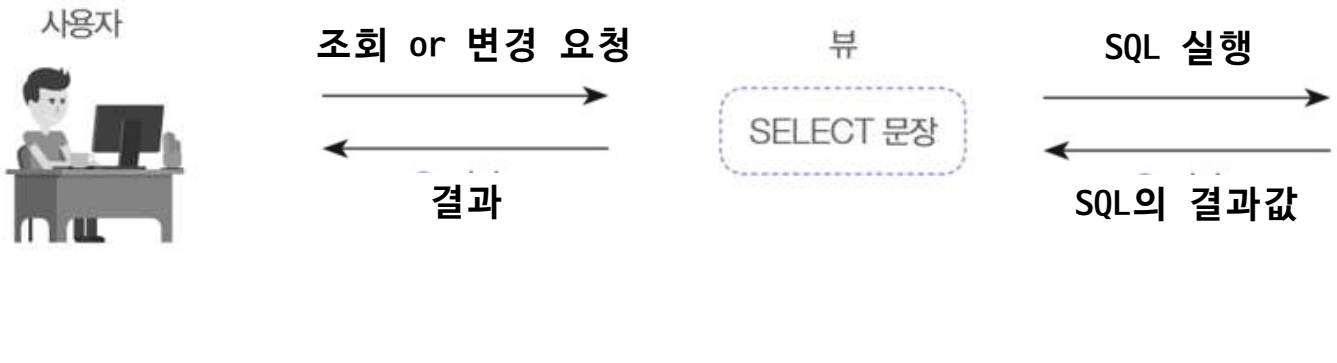
TABLE & VIEW

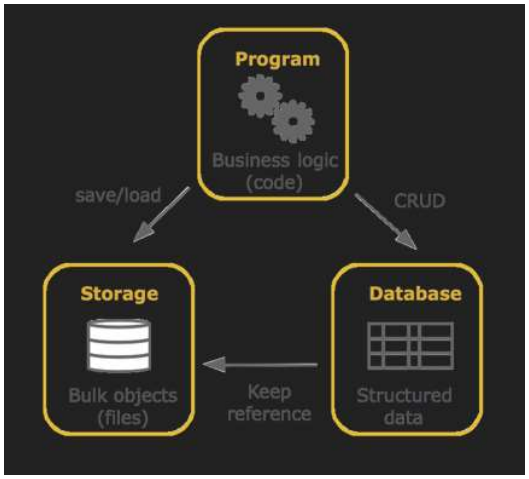
VIEW

VIEW의 개념

▷ 일반 사용자 입장에서 테이블과 동일하게 사용하는 객체

VIEW의 작동방식





VIEW

CREATE VIEW

SIMPLE VIEW

```
CREATE VIEW V_USER
AS
SELECT USERID, NAME, ADDR FROM USERTBL;


SELECT * FROM V_USER;
```

COMPLEX VIEW

```
CREATE VIEW V_USER_PROD
AS
SELECT U.USERID, U.NAME, B.PRODNAME, U.ADDR, CONCAT(U.MOBILE1, U.MOBILE2) AS MOBILE_PHONE
FROM USERTBL U
     INNER JOIN PROD_TBL B
        ON U.USERID = B.USERID


SELECT * FROM V_USER_PROD WHERE USERID='USER01';
```

INDEX

INDEX

INDEX의 개념

데이터를 좀 더 빠르게 찾을 수 있도록 해 주는 도구

INDEX의 장단점

장점

- 검색속도의 향상(항상 그런 것은 아님)
- 쿼리의 부하감소로 시스템 전체의 성능 향상

단점

- 인덱스가 공간을 차지해서 추가적인 공간 필요, 대략 데이터베이스크기의 10%정도의 추가공간 필요
- 처음 인덱스를 생성하는데 시간소요
- 데이터의 변경작업(INSERT, UPDATE, DELETE)이 자주 일어나는 경우 성능이 나빠질 수 있음

INDEX

CREATE INDEX

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```