'하루 10분 핵심만 골라 마스터하는 SQL' 정리

작성일: 2020.03.11
작성자: 도원진

01. 데이터 조회

- 정렬없이 무작위 배치가 기본

1-1. ORDER BY 옵션

- 오름차순 ASC - 기본값
  - 숫자, 알파벳순서로 정렬 진행
- 내림차순 DESC
- 항시 SELECT 문에서 **맨 마지막에 배치**
- SELECT문에 있는 열이름으로 ORDER BY에 순번으로 대체가능한다
  - SELECT prod_id**,** prod_price**,** prod_name

    FROM products

    ORDER BY 2, 3;
  - **ORDER BY 의 번호는 테이블 컬럼번호가 아닌 SELECT 문의 컬럼번호이다.**

1-2. 필터링

데이터베이스의 필터링은 강력하다. 응용프로그램안에서 필터링하려고 하지 말고

데이터베이스의 필터링을 적극 활요할것!

WHERE 절 연산자	**설명**
< > 또는 !=	같지않음
AND	OR보다 우선순위가 높음. 그래도 괄호를 함께 사용한다.
OR	AND 와 함께 사용할 때는 반드시 괄호를 함께 사용한다.
BETWEEN	두 값 사이(**경계를 포함한다**)
IN	IN (A, B, C) 이렇게 있다.
NOT	조건식을 부정할 때, 조건식 앞에 온다.
IS NULL	NULL값

- IN 연산자
  - OR 연산자 여러번 사용하여 구현가능(IN 연산자가 편리하다)
  - 실행속도가 OR보다 빠르다 / 보기 쉽다 / 이해하기 쉽다.
  - IN안에 SUBQUERY를 사용할 수 있다.

- 와일드카드 와 LIKE
  - 와일드카드를 통해 검색패턴(텍스트 필드 즉, 문자열)을 만들 수 있다
  - (참고) LIKE는 연산자라기 보다 '술어'이다.

와일드카드	설명
%	갯수가 0~多 인 문자열
_(언더바)	문자 1개
[ ]	정규표현식처럼 [ ]으로 문자set을 지원한다, 결국 문자 1개 값이다. MySQL은 지원안함.

  - [ ] 대신 **regexp**

```
SELECT *
FROM table
WHERE column regexp '[a-z].*'
```

  - 주의사항
    - 검색패턴을 만들 때, 왠만하면 `와일드카드`는 뒤에 배치하는 경우만 사용해라
      - 앞에 배치하면 속도가 느려짐.
    - 가급적 쓰지마

2. 계산 필드만들기

- **필드 : SELECT 문에 있는 열**
- 열 : TABLE에 있는 열

2-1. 필드 결합

```
SELECT concat(vend_name, ' (', vend_country,')')
FROM Vendors
ORDER BY vend_name;
```

2-2. 수학적 계산

select문 안 필드에서 수학적 연산이 들어있다. (+, -, *, /)

```
SELECT *FROM quantity * item_price AS expanded_price
FROM orderItems
WHERE order_num = 20008;
```

3. 데이터 조작함수

3-1. SQL 함수 필요에 따라서 적절히

- DBMS간 호환을 기대할 수 없다.
- 하지만, 함수는 적절하게 사용됐을 때 빠르다

텍스트 제어함수

함수	sql	결과	비고
substring	SELECT substring("chongmoa.com", 3, 5)	ongmo	
left	select left('abcde', 2)	ab	
right	select right('abcde', 2)	de	
length	select length('abcde')	5	
soundex	SELECT SOUNDEX('Sure') == SOUNDEX('Shore');	true	

날짜/ 시간 제어함수 MySQL

함수	SQL	result
ADDDATE()	ADDATE('2020-01-01', INTERVAL 31 DAY)	'2020-02-01'
SUBDATE()	SUBDATE('2020-01-01', INTERVAL 1 MONTH)	'2019-12-01'
ADDTIME()	ADDTIME('2020-01-01 23:59:59', '1:1:1')	'2020-01-02 01:01:00'
SUBTIME()	SUBTIME('2020-01-01 23:59:59', '1:1:1')	'2020-01-01 22:58:58'
CURDATE()	다른 함수의 파라미터로 많이 쓰임	yyyy-MM-dd
CURTIME()	다른 함수의 파라미터로 많이 쓰임	00:00:00
NOW(), SYSDATE()	현재 값 , DATETIME 형식이다.	yyyy-MM-dd 00:00:00
DATE(DATETIME형식)	DATETIME형식에서 날짜 추출	yyyy-MM-dd

함수	SQL	result
TIME(DATETIME형식)	DATETIME형식에서 시간추출	00:00:00
DAYOFWEEK( )	DAYOFWEEK(curdate( ))	요일 숫자값
MONTHNAME( )	MONTHNAME(curdate( ))	월 이름

숫자제어함수

함수	result	비고
ABS()	절댓값	
COS()	코사인 값	
EXP(num)	e^num	
SQRT()	지정한 숫자의 제곱근 반환	

4. 데이터 요약

4-1. 집계함수

기본적으로 NULL은 집계에서 제외시칸다.

집계함수	기능	특징	비고
AVG	평균		
COUNT	행 갯수의 총합	COUNT(*) : 테이블 행갯수, NULL값 포함한다. COUNT(열) : 컬럼의 행갯수, NULL값 포함한다.	
SUM	합계	SUM(item_price * quantity) AS total_price	
MAX, MIN		숫자말고 텍스트에서도 사용된다.	

4-2. 고유값 집계

앞서 집계를 할 때, 동일 값들을 모두 셈하였다. (기본옵션 ALL)

이제, 고유한 값만 셈하겠다.(옵션 DISTINCT)

```
SELECT AVG(DISTINCT prod_price) AS avg_price
FROM products
WHERE vend_id = 'DLL01';
```

- DISTINCT 특성상 COUNT함수에서 사용될 때는 COUNT(컬럼명) 일 때만 사용한다.
- COUNT(*) 에서는 사용불가
    - MAX, MIN에서 사용은 의미없다
- AS 를 통해 테이블 컬럼에 없는 별칭을 지어준다.

5. 데이터 그룹화

```
SELECT COUNT( * ) AS num_prods
FROM Products
WHERE vend_id = 'DLL01'
```

위의 결과로는 **업체('DLL01')하나의 결과만 받는 아쉬움이 있다.**

Q. 여러 업체를 보고싶다면?

1. where 필터링 삭제하고
2. 모든 업체가 조회되면
3. GROUP BY로 그룹화하고
4. COUNT(*) 로 집계하라

```
SELECT vend_id, COUNT( * ) AS num_prods
FROM Products
GROUP BY vend_id; --where 없애고 GROUP BY로
```

5-1. GROUP BY

- **GROUP BY 없이 집계함수만 사용했을 때, 테이블 모든 행들이 하나의 그룹으로 묶임!**
- **GROUP BY를 통해 여러행을 각각의 그룹으로 묶임 이후, 집계 진행**

그룹화 할 기준이 여럿있을 때

- 그룹1 단계 일 때보다 그룹2 단계로 갈 때 세분화됨(나눠지는 정도가 많아짐)
- 세분화가 다 된다면 그룹묶는 것 완료. 이후, 집계진행

유의사항

1. 집계함수(AVG, SUM, COUNT 등등...)을 제외하고 SELECT문의 **모든 컬럼**은 **GROUP BY 절에 있어야 한다.**

   예) SELECT 컬럼1, 컬럼2, **집계함수** GROUP BY **컬럼1, 컬럼2**

2. 원하는 만큼의 열을 넣을 수 있다. 즉, 그룹을 중첩하는 등 세부적으로 제어할 수 있다.

   특히, GROUP BY **맨 마지막에 오는 필드를 기준으로 집계함수가 집계한다**

   예) GROUP BY 필드1, **필드2**

3. SELECT 에 쓰인 컬럼명으로 GROUP BY에 쓰여야함. **별칭은 불가능하다.**
4. SELECT 에 쓰인 **모든 컬럼명**은 반드시 **GROUP BY에서 쓰여야함**.

예) SELECT 컬럼1, 컬럼2 FROM table GROUP BY 컬럼1

에러난다. 이유.

5. SELECT 문의 **열 번호(상대위치)로 지정가능**

   예) GROUP BY 1, 2

6. 그룹화할 기준 열에 NULL값 여럿 있어도 **NULL을 그룹화함.**
7. 키워드 배치 순서
   - SELECT FROM WHERE **GROUP BY HAVING** ORDER BY

5-2. 그룹 필터링

GROUP BY와 함께 쓰이는 HAVING

- **따라서, 그룹에 대해 적용할 수 있는 HAVING BY를 적용하는 것이 옳다.**
- **WHERE vs HAVING**
  - `데이터를 필터링 한다`는 점에서 동일 그러나, 동작원리가 다르다.
  - WHERE
    - 그룹화 전에 동작하여, 그룹에서 제외할 행들을 걸러낸다.
    - 각 행을 대상으로 함.
  - HAVING
    - 그룹화 되고 난 후, 그룹단위로 필터링을 진행하는 것은 HAVING BY다
    - 그룹화된 행을 대상으로 함.
  - 따라서, 각 키워드 WHERE 과 HAVING BY 다음에 오는 조건이 다르다.
    - HAVING BY 뒤에는 집계함수가 온다.
- 문제
  - `주문 상세 내역 테이블에서 지난 12개월 간 2번 이상 주문한 고객을 찾아라`

    오답

    ```
    SELECT count(customer_id) -- group by를 쓸 수 없다. 이유. select문에 열이 없음
    ```

    정답.(주문 횟수를 값으로 기대할 수 없는 테이블이므로 집계함수를 사용한다)

    ```
    SELECT customer_id
    FROM order_detail_tbl
    WHERE order_date in (now() - 365, now())
    GROUP BY 1
    HAVING count(*) >= 2
    ```

  - `가격이 4 이상인 제품을 두 개 이상 가진 공급업체를 나열`

    정답

    ```
    SELECT vend_id
    FROM products
    WHERE prod_price >= 4
    GROUP BY vend_id
    HAVING count(*) >= 2;
    ```

- 예시

```
select department_id 부서번호, job_id 직업, manager_id 상사번호,
sum(salary)
from employees
group by department_id, job_id, manager_id
order by department_id;
```

	부서번호	직업	상사번호	SUM(SALARY)
1	10	AD_ASST	101	4400
2	20	MK_MAN	100	13000
3	20	MK_REP	201	6000
4	30	PU_CLERK	114	13900
5	30	PU_MAN	100	11000

6. 서브쿼리

교재 테이블 구조

1. 공급업체
2. 상품
3. 손님
4. 주문
5. 주문상품

6-1. 설명 `RGAN01 라는 상품을 주문한 모든 고객의 목록이 필요하다.`

서브쿼리 전)

1. 5번 테이블에서 **RGAN01 을 이용해** 주문번호를 구한다.
2. 4번 테이블에서 **주문번호를 이용해** 손님ID를 구한다
3. 3번 테이블에서 **손님ID를 이용해** 손님정보를 구한다.

서브쿼리 후)

```
SELECT *FROM customers
WHERE cust_id IN (SELECT cust_id
FROM orders
WHERE order_num IN (SELECT order_num
                    FROM orderitems
                    WHERE prod_id = 'RGAN01'));
```

주의사항

1. 서브쿼리의 결과에서 여러행의 필드갯수는 반드시 1개다. 따라서 **결과값은 N X 1 꼴**이다.
2. **테이블 조인이 더 효율이 좋다.**

6-2. '계산 필드'로서 서브쿼리

```
SELECT cust_name, cust_state, ( SELECT COUNT(*)
                                FROM orders
                               WHERE orders.cust_id = customers.cust_id) AS orders -- 불편하다. 테이블 조인으로 개선!
 FROM customers
 order by cust_name;
```

7. 테이블 조인

- 테이블 소개
  - 제품 TABLE(설명, 가격, 공급업체)
  - 공급업체TABLE(이름, 주소, 연락처)
  - 회사고객 TABLE(번호, 회사이름, 주소, 시, 주, 우편번호, 국가, 연락처이름, 이메일)

7.1. 조인 (내부조인)

- 2가지 방법 모두 가능
  - FROM A, B WHERE
  - FROM A   INNER JOIN B   ON

```
SELECT *FROM vendors; -- 6rows
SELECT *FROM products; -- 9rows

--WHERE 절 없이 조인하면 각 테이블 조회 결과값의 곱만큼 결과가 나옴
--Cross조인 , 곱집합이 나옴
SELECT vend_name, prod_name, prod_price
FROM vendors, products; -- 54rows

--(내부)조인, (inner)join , 가장 일반적인 조인
SELECT vend_name, prod_name, prod_price
FROM vendors, products
WHERE vendors.vend_id = products.vend_id; -- 9 rows

--내부조인은 INNER JOIN 을 명시할 때, ON을 사용(HWERE 대신에)
SELECT vend_name, prod_name, prod_price
FROM vendors INNER JOIN products
WHERE vendors.vend_id = products.vend_id; -- 9 rows
```

7-2. 자체조인

요구사항: `Jim Jones라는 연락처가 있는 회사의 모든 직원들 정보`

```
SELECT cust_id, cust_name, cust_contact
FROM customers
WHERE cust_name = (SELECT cust_name
                   FROM customers
                   WHERE cust_contact = 'Jim Jones')); -- 서브쿼리

SELECT c1.cust_id, c1.cust_name, c1.cust_contact
FROM customers AS c1, customers AS c2
WHERE c1.cust_name = c2.cust_name AND c2.cust_contact = 'Jim Jones'; -- 자체조인
-- c2 테이블에서 `Jim Jones'라는 contact를 조건을 만족하는 c2.cust_name을 찾고
-- c1 테이블과 매칭한다.
```

7-3. 외부조인

```
SELECT c.cust_id, o.order_num
FROM customers c INNER JOIN orders o
ON c.cust_id = o.cust_id; -- 내부조인
```

```
SELECT c.cust_id, o.order_num
FROM customers c LEFT OUTER JOIN orders o -- FROM 구분에서 left에 올 테이블, right
에 올 테이블이 결정
ON c.cust_id = o.cust_id;

SELECT c.cust_id, o.order_num
FROM customers c right OUTER JOIN orders o-- FROM 구분에서 left에 올 테이블, right
에 올 테이블이 결정
ON c.cust_id = o.cust_id;

--FULL OUTTER JOIN은 MySQL은 지원하지 않음.
```

- **해당 Side 반대편에 있는 컬럼에서 값이 0 이거나 NULL 인 row들을 모두 반환함**
- A RIGHT OUTTER JOIN B
  - B 테이블에 있는 컬럼의 모든 행들을 표기한다.
  - 따라서, A테이블의 컬럼에 NULL 이나 0 이 표시될 수 있다.
- A LEFT OUTTER JOIN B
  - A 테이블에 있는 컬럼의 모든 행들을 표기한다.
  - 따라서, B테이블의 컬럼에 NULL 이나 0 이 표시될 수 있다.

7-4. 집계함수와 함께 쓰일 수 있음.

여러 테이블과 JOIN을 통해 새로운 테이블이 만들어 지면 집계함수로 sum, max, min....등을 처리할 수 있고,

GROUP BY 를 통해 그룹화하여 집계할 수 있다.

8. 쿼리결합 UNION

여러 SELECT 문을 결합하여 한번의 SQL문으로 만든 것.

```
SELECT cust_name, cust_contact, cust_email
FROM customers
WHERE cust_state IN ('IL', 'IN', 'MT');

SELECT cust_name, cust_contact, cust_email
FROM customers
WHERE cust_name = 'Fun4All';
```

```
SELECT cust_name, cust_contact, cust_email
FROM customers
WHERE cust_state IN ('IL', 'IN', 'MT')
UNION
SELECT cust_name, cust_contact, cust_email
FROM customers
WHERE cust_name = 'Fun4All';--UNION

SELECT cust_name, cust_contact, cust_email
FROM customers
WHERE cust_state IN ('IL', 'IN', 'MT')
OR cust_name = 'Fun4All'; --OR 은 UNION과 동일한 결과임
```

9. 데이터 입력 INSERT

- 유형
  - 모든 열을 입력
  - 부분 열을 입력
  - 쿼리 결과(여러 행)을 입력

```
INSERT INTO customer VALUES('10000006', 'Toy Land', '123 Any Street', 'New
York', 'NY', , ,);
-- 안전하지 않음. 테이블 순서를 지켜서 컬럼을 입력해야하며, 테이블의 컬럼위치가 변경됐을
때, 쿼리를 보장X

-- 열목록을 지정하면 좋다.
INSERT INTO customer (cust_id, cust_name, cust_address, cust_city, ,,,,)
            VALUES('1000000006', 'Toy Land', '123 Any Street', 'New York', 'NY',
, ,);--보다 안전함
```

- 열의 생략
  - NULL이 허용된 열
  - 기본값이 정해진 열
- SELECT 문 결과 입력
  - SELECT문의 열이름은 신경쓰지 않아도 좋다. 다만, 순서는 중요하다.

  ```
  INSERT INTO TABLE (c1, c2, c3) SELECT c1, c2, c3 FROM TABLE
  ```

- 테이블B 생성하기(단, 테이블B에 SELECT 결과를 INSERT하기)
  - 복사본 만들기 딱 좋음

```
SELECT c1, c2 INTO tableB FROM tableA -- 다른 DBMS
CREATE TABLE tableB AS SELECT c1, c2, c3 FROM tableA -- MySQL, ORACLE
```

10. 데이터 UPDATE, DELETE

10-1. UPDATE

```
UPDATE table SET c1 = 'value1', c2 = 'value2' WHERE c0 = 'value';
UPDATE table SET c1 = 'value1', c2 = 'value2' WHERE c0 IN SELECT c FROM table;
UPDATE table SET c1 = 'value1', c2 = (SELECT문) WHERE c0 IN SELECT c FROM table;
```

- 특정 row 1줄을 업데이트
  - WHERE절 함께 쓰임
- 모든 row 를 업데이트

10-2. DELETE

- 방법
  - 테이블의 행 1개 삭제 (WHERE 절과 함께 쓰임)
    - WHERE절은 필수
    - pk를 조건절에 사용한다.
  - 테이블의 모든 행 삭제
    - `TRUNCATE TABLE`이 성능이 더 좋음

```
DELETE FROM tableA WHERE c0 = 'value'; -- FROM 을 꼭 명시하자
```

11.테이블 생성

11-1. 생성

```
CREATE TABLE tableA(
    c1  CHAR(10)    NOT NULL,
    c2  CHAR(10)    NOT NULL,
    c3  CHAR(10)    NOT NULL,
    c4  text(1000)  NULL, -- NULL 생략가능함

);
```

- NULL 기본값
  - 지정하지 않으면 NULL 허용
  - 반드시 입력값이 이어야 하면, NOT NULL을 꼭 명시해야함

11-2. 기본값 지정

```
CREATE TABLE tableA(
    c1  INTEGER     NOT NULL DEFAULT 1, -- 기본값 1로 지정
    c2  char(10)    NOT NULL
);
```

- 날짜 기본값
    - CURRENT_DATE()

11-3. 테이블 업데이트

```
ALTER TABLE tableA
ADD column CHAR(20); -- 컬럼 추가

ALTER TABLE tableB
DROP COLUMN column; --컬럼 제거
```

- 데이터가 이미 INSERT된 테이블은 구조를 바꾸지 않는것이 원칙
- 열 추가
    - 모든 DBMS가 허용
    - 추가할 수 있는 데이터 형식, NULL, DEFAULT 는 제약이 있다
- 열 이름 변경가능

11-4. 테이블 삭제

```
DROP TABLE tableA;
```

- 실수로 삭제해도 되돌릴 수 없다.
    - 경고도 없기 때문에 조심!
    - 다른 테이블과 관계규칙을 맺으면 , 관계삭제전까지 테이블삭제가 불가능해진다. 더 안전한 방법

11-5. 테이블 이름바꾸기

```
RENAME TABLE old_table TO new_table; -- RENAME 이용
ALTER TABLE old_table RENAME new_table; -- ALTER 이용
RENAME TABLE old_table1 TO new_table1,  -- 다중 테이블 이름 변경
             old_table2 TO new_table2,
             old_table3 TO new_table3;
RENAME TABLE current_db.table_name TO other_db.table_name; -- 테이블을 다른 스키마로 이동
```

12. 뷰

12-1. 뷰의 이해

1. 복잡한 SQL문을 재작성할 필요가 없다.

2. SQL문 결과를 임시테이블로 만들어 낸다. (단, 복사본이 아닌 원본과 참조연결이 된 데이터다)
3. 포멧팅하여 반환된 값을 사용할 수 있다.

유의사항

1. 고유한 뷰이름
2. ViewA를 통해 뷰ViewB를 만들 수 있다.
3. 인덱스, 트리거, 기본값 지정불가능

12-2 뷰 생성

```
-- view 생성
CREATE VIEW viewA AS
SELECT c1, c2, c3
FROM tableA, tableB, tableC
WHERE [join조건]

-- view 조회
SELECT *
FROM viewA;
```

View는 업데이트 기능이 없다. 따라서 DROP하고 다시 CREATE해야함.

12-3 데이터 포맷팅을 위한 뷰

```
-- 2장 필드결합
SELECT concat(vend_name, ' (', vend_country,')')
FROM Vendors
ORDER BY vend_name;

-- 위 SQL이 자주 필요하다고 하면, VIEW를 생성하자
CREATE VIEW viewA AS
SELECT concat(vend_name, ' (', vend_country,')') AS newName -- AS로 view의 컬럼
이름을 꼭 지정해준다.
FROM Vendors; -- view생성할 땐 OREDER BY 없음
```

12-4 데이터 필터링/ 계산필드 를 위한 뷰

데이터 필터링

```
-- 필터링한 값만 View에 담을 때
CREATE VIEW viewA AS
SELECT *
FROM
WHERE [view에 담을 데이터의 조건]
```

```
-- view 조회
SELECT *
FROM viewA
WHERE [데이터가 담긴 view에서 조회할 때 필터링할 조건]
```

계산필드

```
CREATE VIEW viewA AS
SELECT a, b, a*b AS result
FROM
```