

SQL을 이용한 데이터 분석

4. JOIN 소개

한기용

keeyonghan@hotmail.com

Contents

1. JOIN이란?
2. 다양한 종류의 JOIN
3. 3일차 숙제 리뷰
4. 숙제

JOIN이란?

JOIN이란?

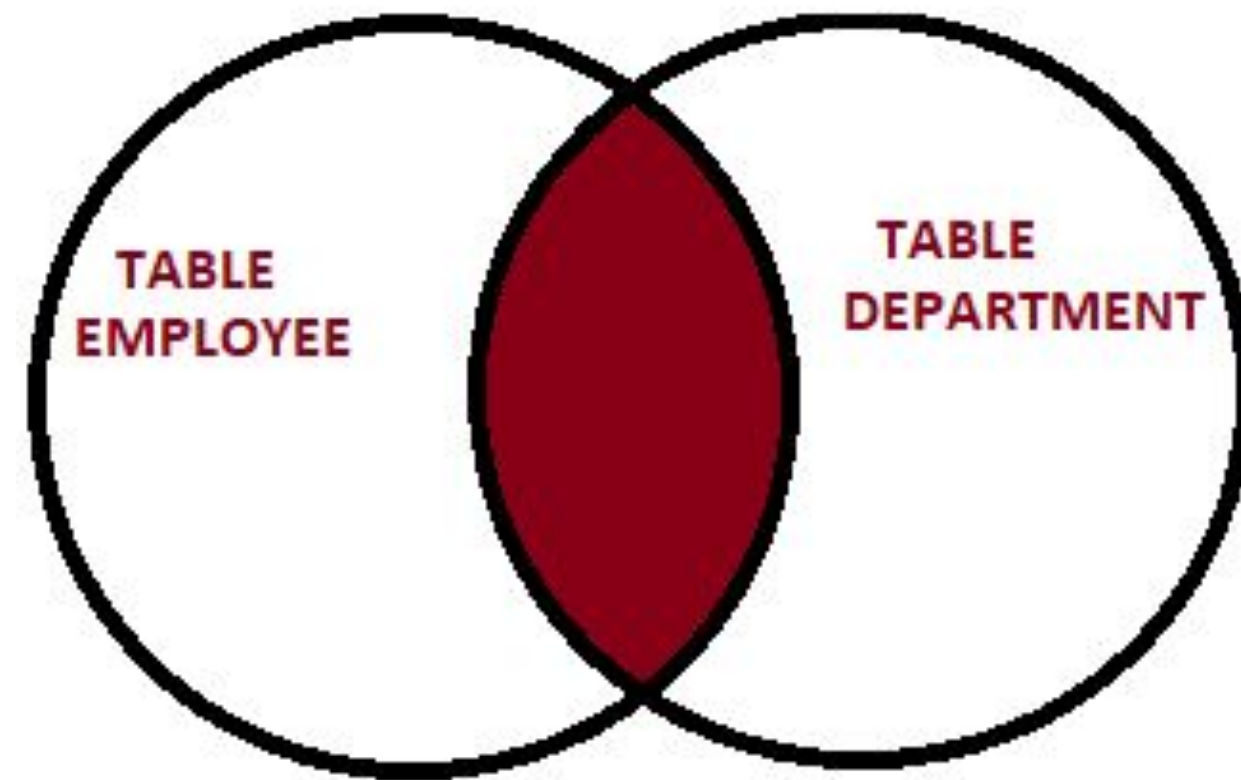
SQL 조인은 두 개 혹은 그 이상의 테이블들을 공통 필드를 가지고 머지하는데 사용된다. 이는 스타 스키마로 구성된 테이블들로 분산되어 있던 정보를 통합하는데 사용된다.

왼쪽 테이블을 **LEFT**라고 하고 오른쪽 테이블을 **RIGHT**이라고 하자. **JOIN**의 결과는 방식에 상관없이 양쪽의 필드를 모두 가진 새로운 테이블을 만들어내게 된다. 조인의 방식에 따라 다음 두 가지가 달라진다:

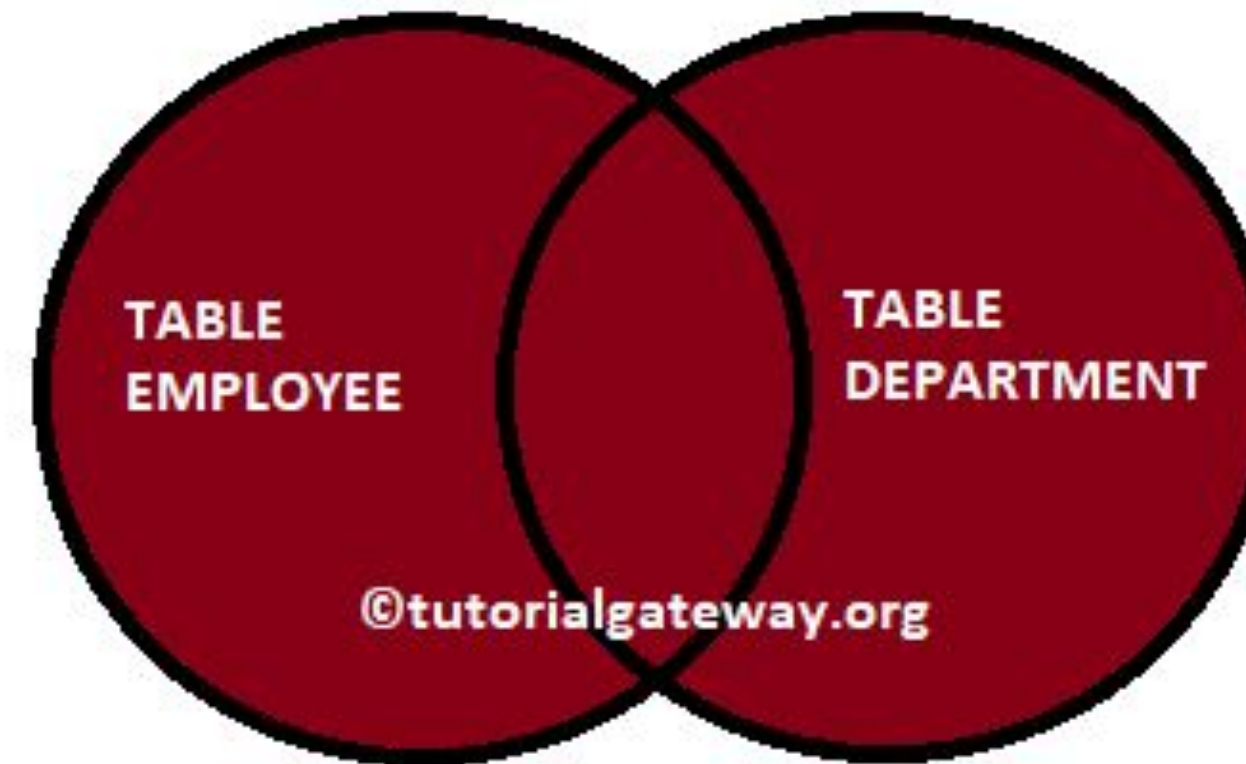
1. 어떤 레코드들이 선택되는지?
2. 어떤 필드들이 채워지는지?

다양한 종류의 조인

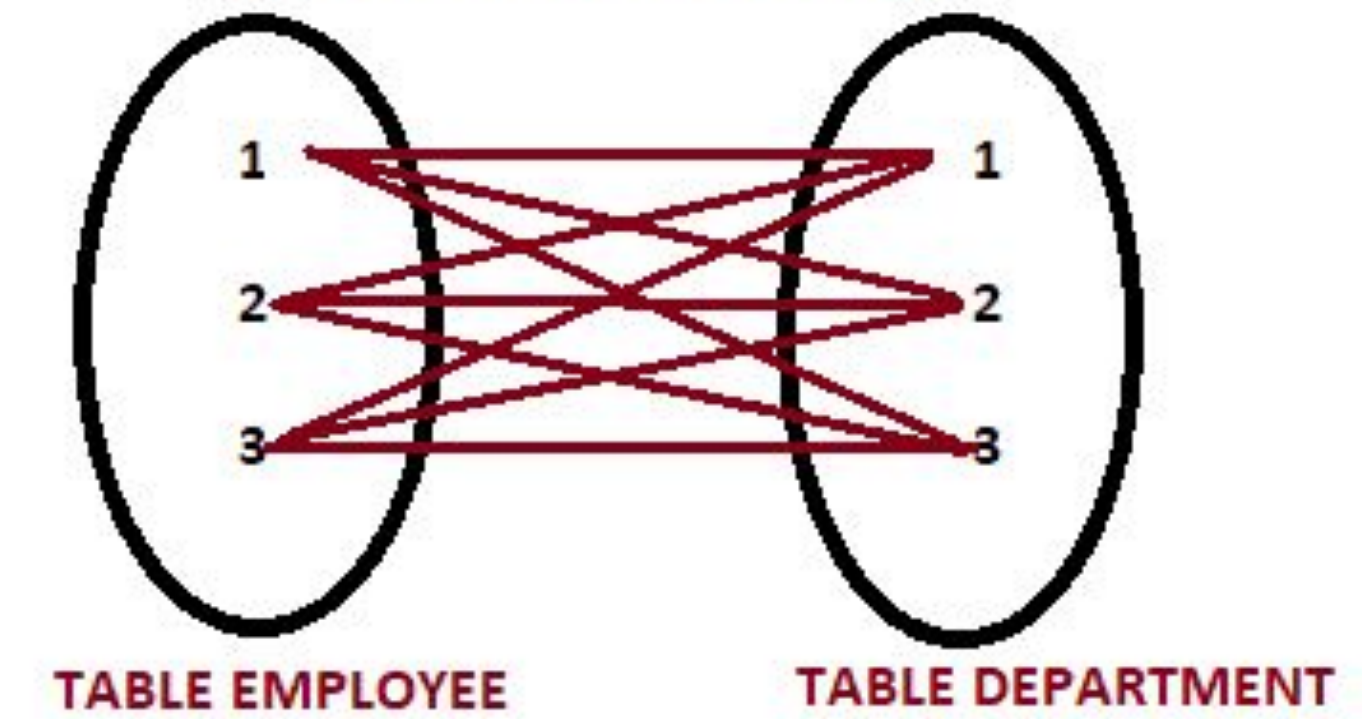
INNER JOIN EXAMPLE



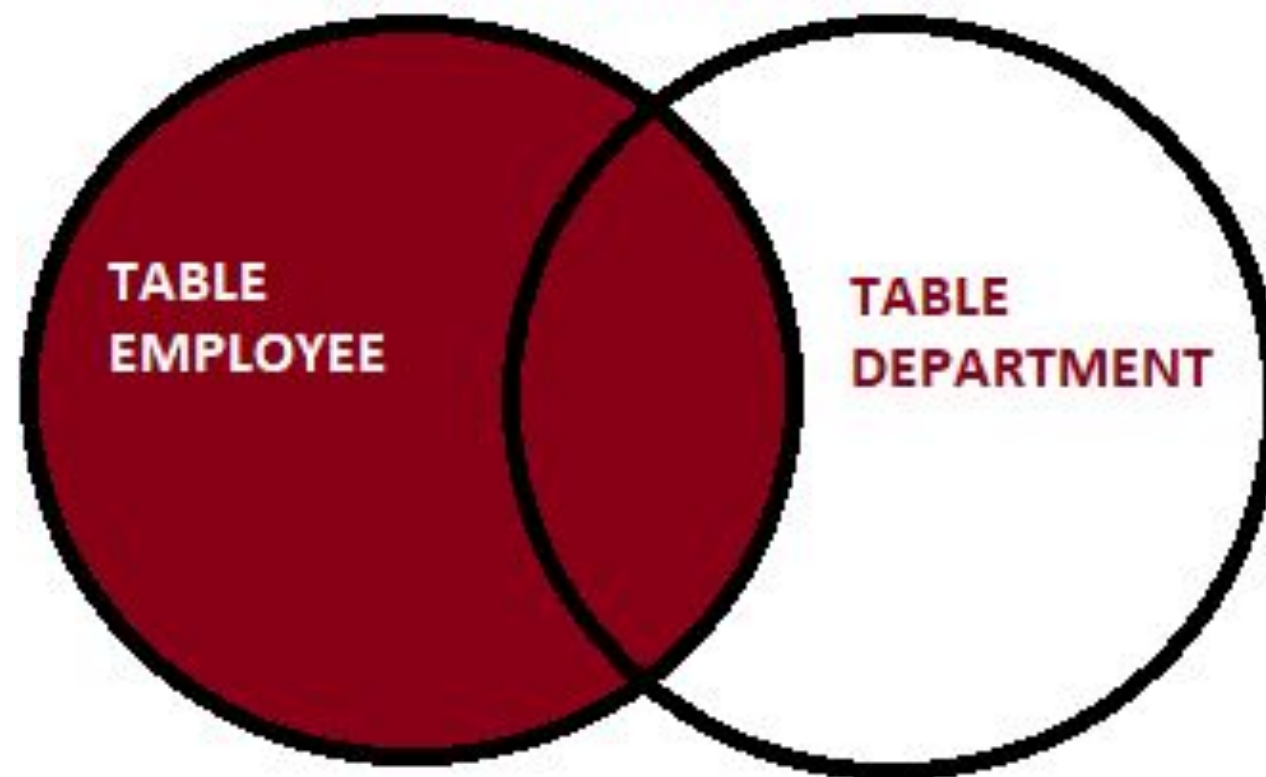
FULL JOIN EXAMPLE



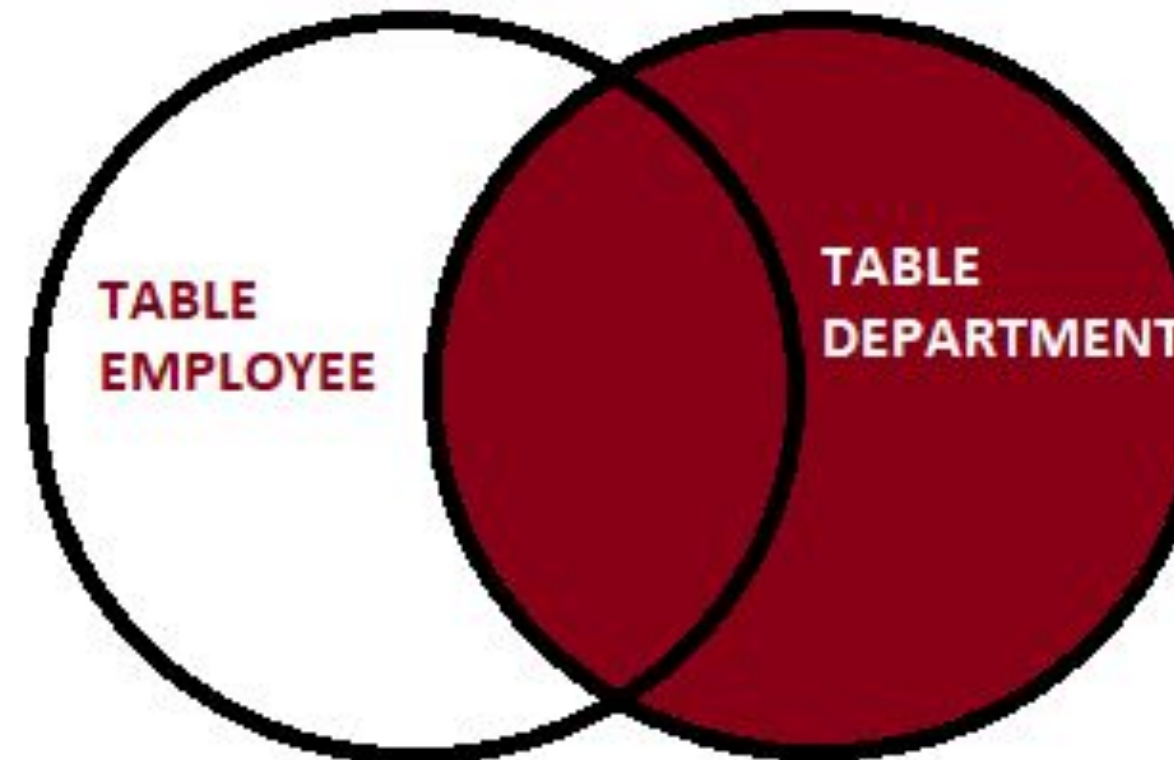
CROSS JOIN EXAMPLE



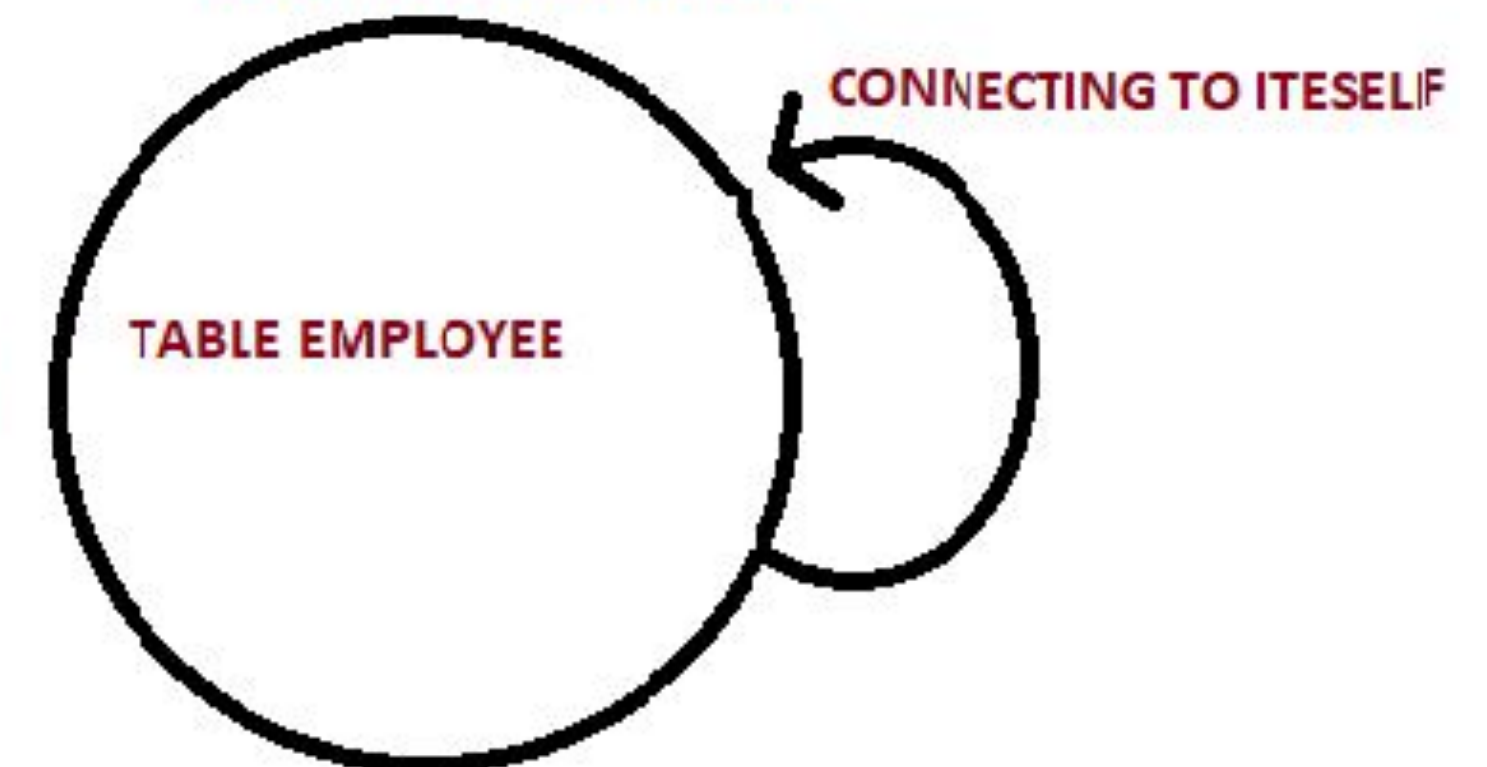
LEFT JOIN EXAMPLE



RIGHT JOIN EXAMPLE



SELF JOIN EXAMPLE



Source: <https://theartofpostgresql.com/blog/2019-09-sql-joins/>

JOIN 문법

```
SELECT A.*, B.*
```

```
FROM raw_data.table1 A
```

```
_____ JOIN raw_data.table2 B ON A.key1 = B.key1 and A.key2 = B.key2
```

```
WHERE A.ts >= '2019-01-01';
```

· INNER, FULL, LEFT, RIGHT, CROSS



JOIN시 고려해야할 점

- 먼저 중복 레코드가 없고 Primary Key의 uniqueness가 보장됨을 체크
 - 아주 중요함!!!
- 조인하는 테이블들간의 관계를 명확하게 정의
 - One to one
 - 완전한 one to one: user_session_channel & session_timestamp
 - 한쪽이 부분집합이 되는 one to one: user_session_channel & session_transaction
 - One to many? (order vs order_items)
 - 이 경우 중복이 더 큰 문제됨 -> 증폭!!
 - Many to one?
 - 방향만 바꾸면 One to many로 보는 것과 사실상 동일.
 - Many to many?
 - 이런 경우는 많지 않으며 이는 one to one이나 one to many로 바꾸는 것이 가능하다면 변환하여 조인하는 것이 덜 위험
- 어느 테이블을 베이스로 잡을지 (From에 사용할지) 결정해야함

다양한 종류의 JOIN

JOIN의 종류

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL OUTER JOIN
- SELF JOIN
- CROSS JOIN

테이블 두 개 설명

UserID	VitalID	Date	Weight
100	1	2020-01-01	75
100	3	2020-01-02	78
101	2	2020-01-01	90
101	4	2020-01-02	95

raw_data.Vital

AlertID	VitalID	AlertType	Date	UserID
1	4	WeightIncrease	2020-01-02	101
2	NULL	MissingVital	2020-01-04	100
3	NULL	MissingVital	2020-01-04	101

raw_data.Alert

INNER JOIN

1. 양쪽 테이블에서 매치가 되는 레코드들만 리턴함
2. 양쪽 테이블의 필드가 모두 채워진 상태로 리턴됨

```
SELECT * FROM raw_data.Vital v  
JOIN raw_data.Alert a ON v.vitalID = a.vitalID;
```

v.UserID	v.VitalID	v.Date	v.Weight	a.AlertID	a.VitalID	a.AlertType	a.Date	a.UserID
101	4	2020-01-02	95	1	4	WeightIncrease	2021-01-02	101

LEFT JOIN

1. 왼쪽 테이블(Base)의 모든 레코드들을 리턴함
2. 오른쪽 테이블의 필드는 왼쪽 레코드와 매칭되는 경우에만 채워진 상태로 리턴됨

```
SELECT * FROM raw_data.Vital v
```

```
LEFT JOIN raw_data.Alert a ON v.vitalID = a.vitalID;
```

v.UserID	v.VitalID	v.Date	v.Weight	a.AlertID	a.VitalID	a.AlertType	a.Date	a.UserID
100	1	2020-01-01	75	NULL	NULL	NULL	NULL	NULL
100	3	2020-01-02	78	NULL	NULL	NULL	NULL	NULL
101	2	2020-01-01	90	NULL	NULL	NULL	NULL	NULL
101	4	2020-01-02	95	1	4	WeightIncrease	2021-01-02	101

FULL JOIN

- 1. 왼쪽 테이블과 오른쪽 테이블의 모든 레코드들을 리턴함
- 2. 매칭되는 경우에만 양쪽 테이블들의 모든 필드들이 채워진 상태로 리턴됨

```
SELECT * FROM raw_data.Vital v
FULL JOIN raw_data.Alert a ON v.vitalID = a.vitalID;
```

v.UserID	v.VitalID	v.Date	v.Weight	a.AlertID	a.VitalID	a.AlertType	a.Date	a.UserID
100	1	2020-01-01	75	NULL	NULL	NULL	NULL	NULL
100	3	2020-01-02	78	NULL	NULL	NULL	NULL	NULL
101	2	2020-01-01	90	NULL	NULL	NULL	NULL	NULL
101	4	2020-01-02	95	1	4	WeightIncrease	2021-01-02	101
NULL	NULL	NULL	NULL	2	NULL	MissingVital	2020-01-04	100
NULL	NULL	NULL	NULL	3	NULL	MissingVital	2020-01-04	101

CROSS JOIN

1. 왼쪽 테이블과 오른쪽 테이블의 모든 레코드들의 조합을 리턴함

```
SELECT * FROM raw_data.Vital v CROSS JOIN raw_data.Alert a;
```

v.UserID	v.VitalID	v.Date	v.Weight	a.AlertID	a.VitalID	a.AlertType	a.Date	a.UserID
100	1	2020-01-01	75	1	4	WeightIncrease	2020-01-01	101
100	3	2020-01-02	78	1	4	WeightIncrease	2020-01-01	101
101	2	2020-01-01	90	1	4	WeightIncrease	2020-01-01	101
101	4	2020-01-02	95	1	4	WeightIncrease	2020-01-01	101
100	1	2020-01-01	75	2		MissingVital	2020-01-04	100
100	3	2020-01-02	78	2		MissingVital	2020-01-04	100
101	2	2020-01-01	90	2		MissingVital	2020-01-04	100
101	4	2020-01-02	95	2		MissingVital	2020-01-04	100
100	1	2020-01-01	75	3		MissingVital	2020-01-04	101
100	3	2020-01-02	78	3		MissingVital	2020-01-04	101
101	2	2020-01-01	90	3		MissingVital	2020-01-04	101
101	4	2020-01-02	95	3		MissingVital	2020-01-04	101

SELF JOIN

1. 동일한 테이블을 **alias**를 달리해서 자기 자신과 조인함

```
SELECT * FROM raw_data.Vital v1
```

```
JOIN raw_data.Vital v2 ON v1.vitalID = v2.vitalID;
```

v1.UserID	v1.VitalID	v1.Date	v1.Weight	v2.UserID	v2.VitalID	v2.Date	v2.Weight
100	1	2020-01-01	75	100	1	2020-01-01	75
100	3	2020-01-02	78	100	3	2020-01-02	78
101	2	2020-01-01	90	101	2	2020-01-01	90
101	4	2020-01-02	95	101	4	2020-01-02	95

◆ SQL 실습

❖ 구글 Colab 실습 링크:

- https://colab.research.google.com/drive/15_DN3jvdjL5GDBrseGM75pv4GyzMPsV1?usp=sharing

3일차 숙제 리뷰

◆ BOOLEAN 타입 처리

❖ True or False

❖ 다음 2개는 동일한 표현

- flag = True
- flag is True

❖ 다음 2개는 동일한 표현인가?

- flag is True
- flag is not False

flag
True
False
True
NULL
False

raw_data.boolean_test

SELECT

COUNT(CASE WHEN flag = True THEN 1 END) true_cnt1,

COUNT(CASE WHEN flag is True THEN 1 END) true_cnt2,

COUNT(CASE WHEN flag is not False THEN 1 END) not_false_cnt

FROM raw_data.boolean_test;

◆ NULL 비교

- ❖ NULL 비교는 항상 IS 혹은 IS NOT으로 수행
- ❖ NULL 비교를 = 혹은 != 혹은 <>으로 수행하면 잘못된 결과가 나옴

```
SELECT COUNT(1)
FROM raw_data.boolean_test
WHERE flag is NULL;
```

```
SELECT COUNT(1)
FROM raw_data.boolean_test
WHERE flag = NULL;
```

flag
True
False
True
NULL
False

raw_data.boolean_test

◆ 채널별 월별 매출액 테이블 만들기

❖ session_timestamp, user_session_channel, session_transaction 사용

- 아래와 같은 필드로 구성

- month
- channel
- uniqueUsers (총방문 사용자)
- paidUsers (구매 사용자: refund한 경우도 판매로 고려)
- conversionRate (구매사용자 / 총방문 사용자)
- grossRevenue (refund 포함)
- netRevenue (refund 제외)

속제풀이 (1)

- 채널별 월 매출액 테이블 만들기 - 먼저 유일한 사용자 수부터 세보자

```
SELECT LEFT(ts, 7) "month",  
       usc.channel,  
       COUNT(DISTINCT userid) uniqueUsers  
FROM raw_data.user_session_channel usc  
JOIN raw_data.session_timestamp t ON t.sessionid = usc.sessionid  
GROUP BY 1, 2  
ORDER BY 1, 2;
```

복잡한 JOIN시 먼저 JOIN 전략부터 수립

1. raw_data.user_session_channel
2. raw_data.session_timestamp
3. raw_data.session_transaction

- 위의 3개 테이블 모두 sessionid를 기반으로 조인을 해야함
- user_session_channel과 session_timestamp는 일대일로 조인가능: INNER JOIN
- 하지만 session_transaction의 경우에는 모든 sessionid가 존재하지 않음
 - LEFT JOIN (혹은 RIGHT JOIN)
 - FROM에 사용하는 테이블은 user_session_channel 혹은 session_timestamp가 되어야함

속제 풀이 (2)

- 채널별 월 매출액 테이블 만들기 - 이제 session_transaction 테이블을 추가해보자

```
SELECT LEFT(ts, 7) "month",  
       usc.channel,  
       COUNT(DISTINCT userid) uniqueUsers  
FROM raw_data.user_session_channel usc  
JOIN raw_data.session_timestamp t ON t.sessionid = usc.sessionid  
LEFT JOIN raw_data.session_transaction st ON st.sessionid = usc.sessionid  
GROUP BY 1, 2  
ORDER BY 1, 2;
```

속제풀이 (3)

- 채널별 월 매출액 테이블 만들기 - 이제 paidUsers를 추가해보자

```
SELECT LEFT(ts, 7) "month",  
       usc.channel,  
       COUNT(DISTINCT userid) uniqueUsers,  
       COUNT(DISTINCT CASE WHEN amount > 0 THEN usc.userid END) paidUsers,  
FROM raw_data.user_session_channel usc  
JOIN raw_data.session_timestamp t ON t.sessionid = usc.sessionid  
LEFT JOIN raw_data.session_transaction st ON st.sessionid = usc.sessionid  
GROUP BY 1, 2  
ORDER BY 1, 2;
```


속제풀이 (4)

- 채널별 월 매출액 테이블 만들기 - 이제 `conversionRate`을 추가해보자
 - 첫 번째 시도:
 - `paidUsers/uniqueUsers AS conversionRate`
 - 두 번째 시도:
 - `paidUsers::float/uniqueUsers AS conversionRate`
 - 세 번째 시도:
 - `ROUND(paidUsers*100.0/uniqueUsers, 2) AS conversionRate`
 - 네 번째 시도:
 - `ROUND(paidUsers*100.0/NULLIF(uniqueUsers, 0), 2) AS conversionRate`

◆ NULLIF

❖ paidUsers/uniqueUsers

- 0으로 나누는 경우 divide by 0 에러 발생
- 이를 어떻게 방지할까? NULLIF를 사용하여 0을 NULL로 변경
 - `paidUsers/NULLIF(uniqueUsers, 0)`
 - 다시 한번 사칙연산에 NULL이 들어가면 결과도 NULL이 됨을 기억!

속제풀이 (5)

- 채널별 월 매출액 테이블 만들기

```
SELECT LEFT(ts, 7) "month", -- "year month"
       channel,
       COUNT(DISTINCT usc.userid) uniqueUsers,
       COUNT(DISTINCT CASE WHEN amount > 0 THEN usc.userid END) paidUsers,
       ROUND(paidUsers::float*100/NULLIF(uniqueUsers, 0),2) conversionRate,
       SUM(amount) grossRevenue,
       SUM(CASE WHEN refunded is False THEN amount END) netRevenue
FROM raw_data.user_session_channel usc
LEFT JOIN raw_data.session_timestamp t ON t.sessionid = usc.sessionid
LEFT JOIN raw_data.session_transaction st ON st.sessionid = usc.sessionid
GROUP BY 1, 2
ORDER BY 1, 2;
```

◆ COALESCE

❖ NULL 값을 다른 값으로 바꿔주는 함수

- 즉 NULL대신에 다른 백업값을 리턴해주는 함수

❖ COALESCE(exp1, exp2, exp3, ...)

- exp1부터 인자를 하나씩 살펴서 NULL이 아닌 값이 나오면 그것을 리턴
- 끝까지 갔는데도 모두 NULL이면 최종적으로 NULL을 리턴

value
NULL
1
1
0
0
4
3

SELECT

value,

COALESCE(value, 0) -- value가 NULL이면 0을 리턴

FROM raw_data.count_test;

raw_data.count_test

◆ 공백 혹은 예약키워드를 필드 이름으로 사용하려면?

❖ ""로 둘러싸서 사용

```
CREATE TABLE keeyong.test (  
    group int primary key,  
    'mailing address' varchar(32)  
);
```

속제풀이 (6)

- 채널별 월 매출액 테이블 만들기

```
DROP TABLE IF EXISTS adhoc.keeyong_monthly_channel_summary;  
CREATE TABLE adhoc.keeyong_monthly_channel_summary AS  
SELECT LEFT(ts, 7) "month",  
       channel,  
       COUNT(DISTINCT usc.userid) uniqueUsers,  
       COUNT(DISTINCT CASE WHEN amount > 0 THEN usc.userid END) paidUsers,  
       ROUND(paidUsers::float*100/NULLIF(uniqueUsers, 0),2) conversionRate,  
       SUM(amount) grossRevenue,  
       SUM(CASE WHEN refunded is False THEN amount END) netRevenue  
FROM raw_data.user_session_channel usc  
LEFT JOIN raw_data.session_timestamp t ON t.sessionid = usc.sessionid  
LEFT JOIN raw_data.session_transaction st ON st.sessionid = usc.sessionid  
GROUP BY 1, 2;
```

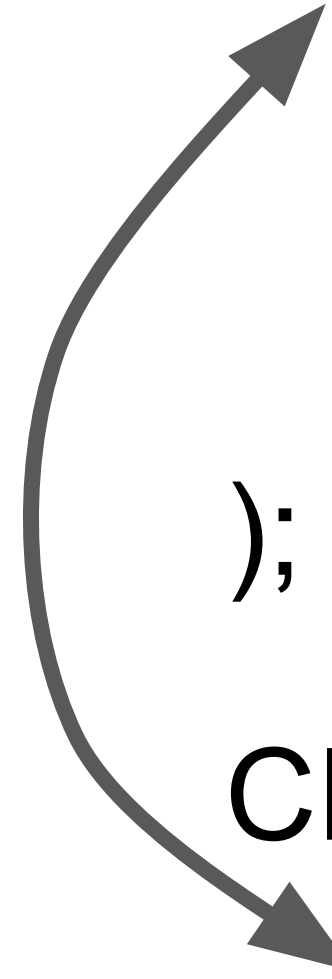
기타 JOIN 문제 풀이

가상 이커머스 사이트의 인기 상품 찾기 (주문 수량/금액 기준)

- 보통 온라인 주문 하나에는 여러 개의 아이템이 들어갈 수 있고 또 아이템의 경우 하나만 주문하는 것이 아니라 다수가 주문이 될 수 있음
- 주문 수량과 금액이란 관점에서 가장 인기가 있는 상품은 무엇일까?
- 주문은 `raw_data.order`라는 테이블에 들어있고 주문에 들어가는 아이템은 `raw_data.order_item` 테이블에 들어가 있음

Order Schema Tables

```
CREATE TABLE raw_data.order (  
  order_id character varying(32) PRIMARY KEY,  
  ordered_at timestamp without time zone,  
  refunded_at timestamp without time zone  
);
```



```
CREATE TABLE raw_data.order_item (  
  order_id character varying(32),  
  product_name character varying(32),  
  quantity smallint,  
  amount double precision  
);
```

- 두 개의 테이블은 `order_id`라는 필드로 조인이 가능함
- 이 두 테이블간의 관계는?
 - one to one
 - one to many
 - many to many
- 데이터 품질을 어떻게 체크해볼 것인각?

데이터 품질 검사



문제 1. 환불되지 않은 주문들만 봤을 때 주문수량 (quantity)
기준으로 가장 많이 주문된 Top 5 상품의 이름과 총
주문수량을 찾아보세요



문제 2. 환불되지 않은 주문들만 봤을 때 주문금액
($\text{amount} \times \text{quantity}$) 기준으로 가장 많이 주문된 Top 5 상품의
이름과 총 주문금액을 찾아보세요.

◆ SQL 실습

❖ 구글 Colab 실습 링크:

- https://colab.research.google.com/drive/1ig4v_NMeI4rD3a9MHYNCkjNgVp-izJul?usp=sharing

속제

속제 1: 사용자별로 처음 채널과 마지막 채널 알아내기

- ROW_NUMBER vs. FIRST_VALUE/LAST_VALUE
- 사용자 251번의 시간순으로 봤을 때 첫 번째 채널과 마지막 채널은 무엇인가?
 - 노가다를 하자면 아래 쿼리를 실행해서 처음과 마지막 채널을 보면 된다.

```
SELECT ts, channel
FROM raw_data.user_session_channel usc
JOIN raw_data.session_timestamp st ON usc.sessionid = st.sessionid
WHERE userid = 251
ORDER BY 1
```

- ROW_NUMBER를 이용해서 해보자
 - ROW_NUMBER() OVER (PARTITION BY field1 ORDER BY field2) nn

ROW_NUMBER 설명

3. ROW_NUMBER를 쓰면 2를 구현 가능
ROW_NUMBER OVER (partition by userid
order by ts) seq

userid	ts	channel
10	2021-01-01	google
11	2021-01-03	facebook
11	2021-01-01	naver
10	2021-01-02	facebook
11	2021-01-04	google
10	2021-01-03	youtube

userid	ts	channel	seq
10	2021-01-01	google	1
10	2021-01-02	facebook	2
10	2021-01-03	youtube	3
11	2021-01-01	naver	1
11	2021-01-03	facebook	2
11	2021-01-04	google	3

1. 사용자별로 시간순으로 일련번호를
매기고 싶다면?

2. 새로운 컬럼 추가!!

- 사용자별로 레코드를 모으고 그 안에서
시간순으로 소팅한 후 사용자별로 1부터 번호
부여

속제 2: Gross Revenue가 가장 큰 UserID 10개 찾기

- user_session_channel과 session_transaction과 session_timestamp 테이블을 사용
- Gross revenue: Refund 포함한 매출

○

속제 3: raw_data.nps 테이블을 바탕으로 월별 NPS 계산

- 고객들이 0 (의향 없음) 에서 10 (의향 아주 높음)
- **detractor** (비추천자) : 0 에서 6
- **passive** (소극자) : 7이나 8점
- **promoter** (홍보자) : 9나 10점
- $NPS = \text{promoter 퍼센트} - \text{detractor 퍼센트}$