programmers

실리콘밸리에서 날아온 데이터베이스

4. SQL 고급 다지기 (JOIN)

keeyonghan@hotmail.com

한기용

Harmonize, Inc

programmers

Contents

programmers

INSERT/UPDATE/DELETE 설명

SELECT 이외의 다른 DML 명령인
INSERT/UPDATE/DELETE에 대해 배워보자

◆ MySQL에서 지원하는 컬럼 타입 (1)

❖ Numeric Type

- INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT
- DECIMAL, NUMERIC
- FLOAT, DOUBLE, BIT

❖ Date and Time Type

- DATE, DATETIME, TIMESTAMP,TIME, YEAR

◆ MySQL에서 지원하는 컬럼 타입 (1)

❖ String Type

- CHAR, VARCHAR, BINARY, VARBINARY, BLOB, TEXT, ENUM, SET

❖ JSON Type

- 다양한 JSON 조작함수를 제공함

❖ Spatial Type

- 위도와 경도를 중심으로한 위치 관련 타입

◆ INSERT (1)

❖ 뒤의 조인에서 사용할 테이블 2개를 생성하고 INSERT로 추가해보자

```
CREATE TABLE prod.vital (
    user_id int not null,
    vital_id int primary key,
    date timestamp not null,
    weight int not null
);

CREATE TABLE prod.alert (
    alert_id int primary key,
    vital_id int,
    alert_type varchar(32),
    date timestamp,
    user_id int
);
```

◆ INSERT (2)

❖ 뒤의 조인에서 사용할 테이블 2개를 생성하고 INSERT로 추가해보자

```
INSERT INTO prod.vital(user_id, vital_id, date, weight) VALUES(100, 1, '2020-01-01', 75);
INSERT INTO prod.vital(user_id, vital_id, date, weight) VALUES(100, 3, '2020-01-02', 78);
INSERT INTO prod.vital(user_id, vital_id, date, weight) VALUES(101, 2, '2020-01-01', 90);
INSERT INTO prod.vital(user_id, vital_id, date, weight) VALUES(101, 4, '2020-01-02', 95);
INSERT INTO prod.vital(user_id, vital_id, date, weight) VALUES(999, 5, '2020-01-02', -1);
INSERT INTO prod.vital(user_id, vital_id, date, weight) VALUES(999, 5, '2020-01-02', 10);
```

```
INSERT INTO prod.alert VALUES(1, 4, 'WeightIncrease', '2020-01-02', 101);
INSERT INTO prod.alert VALUES(2, NULL, 'MissingVital', '2020-01-04', 100);
INSERT INTO prod.alert VALUES(3, NULL, 'MissingVital', '2020-01-04', 101);
```

◆ DELETE

❖ 조건을 기반으로 테이블에서 레코드 삭제 혹은 모든 레코드 삭제

- 후자의 경우에도 테이블은 계속 존재

❖ DELETE FROM vs. TRUNCATE

- 차이점을 이해하는 것이 중요
- TRUNCATE은 조건없이 모든 레코드 삭제. 속도가 빠른 대신 트랜잭션 사용시 롤백 불가

```
DELETE FROM prod.vital WHERE weight <= 0;
DELETE FROM prod.vital;
SELECT * FROM prod.vital;
```

◆ UPDATE

❖ 조건을 기반으로 테이블에서 특정 레코드(들)의 필드 값 수정 가능

❖ 예: vital_id가 4인 레코드의 weight를 92로 변경

```
SELECT * FROM prod.vital WHERE vital_id = 4;

UPDATE prod.vital
SET weight = 92
WHERE vital_id = 4;

SELECT * FROM prod.vital WHERE vital_id = 4;
```

programmers

INSERT/UPDATE/DELETE 실습

◆ SQL 실습

❖ MySQL Workbench로 실습

- 실습 SQL 링크

programmers

다양한 JOIN 살펴보기

INNER JOIN
LEFT/RIGHT JOIN
~~OUTER JOIN~~
CROSS JOIN
SELF JOIN

◆ JOIN이란?

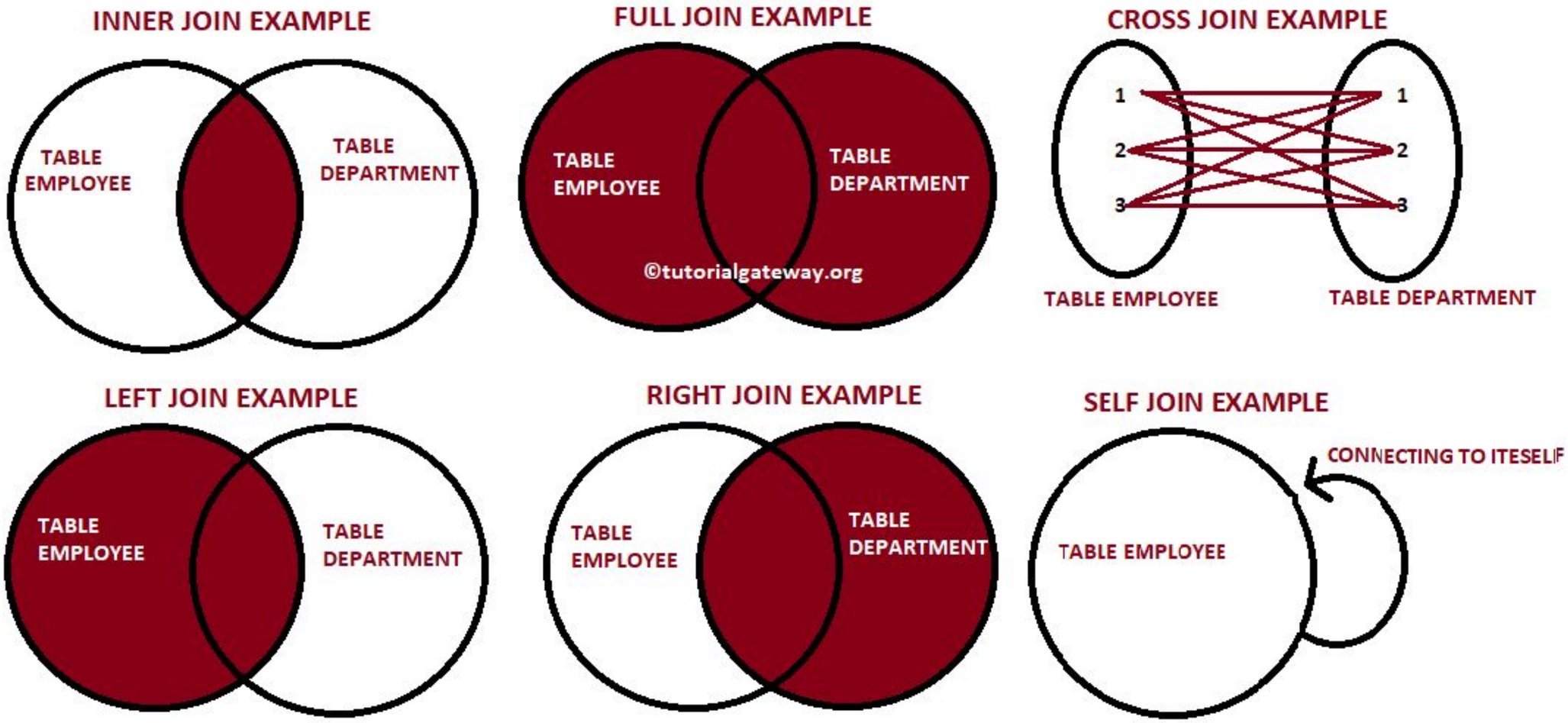
❖ SQL 조인은 두 개 이상의 테이블들을 공통 필드를 가지고 통합

- 스타 스키마로 구성된 테이블들로 분산되어 있던 정보를 통합하는데 사용

❖ JOIN의 결과로 양쪽의 필드를 모두 가진 새로운 테이블을 만들어짐

- 조인의 방식에 따라 다음 두 가지가 달라짐:
  - 어떤 레코드들이 선택되는지?
  - 어떤 필드들이 채워지는지?

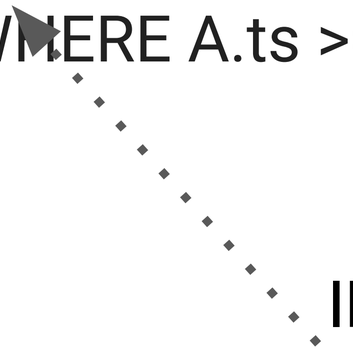
◆ 다양한 종류의 JOIN



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, LEFT, RIGHT, CROSS
MySQL은 FULL 조인을 지원하지 않음

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: session & session_channel
    - 한쪽이 부분집합이 되는 one to one
  - 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?
    - 이는 one to one이나 one to many로 바꾸는 것이 가능하다면 변환하여 조인하는 것이 덜 위험
- 어느 테이블을 베이스로 잡을지 (From에 사용할지) 결정해야함

JOIN의 종류

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL OUTER JOIN:
  - MySQL은 이를 지원하지 않음 LEFT JOIN과 RIGHT JOIN을 UNION하는 것으로 이를 대신할 수 있음
- 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

prod.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

prod.alert

INNER JOIN

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

```
SELECT * FROM prod.vital v
JOIN prod.alert a ON v.vital_id = a.vital_id;
```

v.user_id	v.vital_id	v.date	v.weight	a.alert_id	a.vital_id	a.alerttype	a.date	a.user_id
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 prod.vital v
LEFT JOIN prod.alert a ON v.vital_id = a.vital_id;
```

v.user_id	v.vital_id	v.date	v.weight	a.alert_id	a.vital_id	a.alerttype	a.date	a.user_id
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 prod.vital v
LEFT JOIN prod.alert a ON v.vital_id = a.vital_id
UNION  -- vs. UNION ALL
SELECT * FROM prod.vital v
RIGHT JOIN prod.alert a ON v.vital_id = a.vital_id;
```

v.user_id	v.vital_id	v.date	v.weight	a.alert_id	a.vital_id	a.alerttype	a.date	a.user_id
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 prod.vital v CROSS JOIN prod.alert a;

v.user_id	v.vital_id	v.date	v.weight	a.alert_id	a.vital_id	a.alerttype	a.date	a.user_id
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 alias를 달리해서 자기 자신과 조인함

```
SELECT * FROM prod.vital v1
JOIN prod.vital v2 ON v1.vital_id = v2.vital_id;
```

v1.user_id	v1.vital_id	v1.date	v1.weight	v2.user_id	v2.vital_id	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

programmers

JOIN 실습

◆ SQL 실습

❖ MySQL Workbench로 실습

- 실습 SQL 링크

강의 요약

MySQL 지원 타입 설명
INSERT/UPDATE/DELETE 설명
다양한 JOIN 설명