Mysql 3

Index

18. Functions 1 (CONCAT, CEIL, ROUND, TRUNCATE, DATE_FORMAT)

CEIL, ROUND, TRUNCATE는 소수점 올림, 반올림, 버림 함수입니다.

CEIL

CEIL는 실수 데이터를 올림 할 때 사용합니다.

```
# 12.345를 올림하여 정수로 나타냄
SELECT CEIL(12.345)

# 국가별 언어 사용 비율을 소수 첫번째자리에서 올림하여 정수로 나타냄
SELECT CountryCode, Language, Percentage, CEIL(Percentage)
FROM countrylanguage
```

ROUND

ROUND는 실수데이터를 반올림 할 때 사용합니다.

```
# 12.345를 소수 둘째자리까지 나타내고 소수 셋째자리에서 반올림
SELECT ROUND(12.345, 2)

# 국가별 언어 사용 비율을 소수 첫번째자리에서 반올림하여 정수로 나타냄
SELECT CountryCode, Language, Percentage, ROUND(Percentage, 0)
FROM countrylanguage
```

TRUNCATE

TRUNCATE는 실수 데이터를 버림 할 때 사용합니다.

12.345를 소수 둘째자리까지 나타내고 소수 셋째자리에서 버림

SELECT TRUNCATE(12.345, 2)

국가별 언어 사용 비율을 소수 첫번째자리에서 버림하여 정수로 나타냄

SELECT CountryCode, Language, Percentage, TRUNCATE(Percentage, 0)

FROM countrylanguage

SELECT CountryCode, Language, Percentage, ROUND(Percentage, 0),
TRUNCATE(Percentage, 0)

FROM countrylanguage

DATE_FORMAT

DATE_FORMAT은 날짜 데이터에 대한 포멧을 바꿔줍니다.

https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html

sakila 데이터 베이스에서 월별 총 수입

SELECT DATE_FORMAT(payment_date, "%Y-%m") AS monthly, SUM(amount) AS amount

FROM payment

GROUP BY monthly

19. Functions 2 (IF, IFNULL, CASE)

SQL에서도 다른 언어에서 처럼 조건문 사용이 가능합니다. IF, CASE 에 대해서 설명합니다.

IF

IF(조건, 참, 거짓)

도시의 인구가 100만이 넘으면 "big city" 그렇지 않으면 "small city"를 출력하는 city_scale 컬럼을 추가

```
SELECT name, population, IF(population > 1000000, "big city", "small city") AS city_scale
FROM city
```

IFNULL

IFNULL(참, 거짓)

```
# 독립년도가 없는 데이터는 0으로 출력
SELECT IndepYear, IFNULL(IndepYear, 0) as IndepYear
FROM country
```

CASE

```
CASE
    WHEN (조건1) THEN (출력1)
    WHEN (조건2) THEN (출력2)
END AS (컬럼명)
```

```
# 나라별로 인구가 10억 이상, 1억 이상, 1억 이하인 컬럼을 추가하여 출력
SELECT name, population,
    CASE
        WHEN population > 1000000000 THEN "upper 1 bilion"
        WHEN population > 100000000 THEN "upper 100 milion"
        ELSE "below 100 milion"
    END AS result
FROM country
```

20. JOIN

JOIN은 여러개의 테이블에서 데이터를 모아서 보여줄 때 사용됩니다. JOIN에는 INNER JOIN, LEFT JOIN, RIGHT JOIN이 있습니다.

MAKE TEST TABLE & DATA

```
# create table & data

CREATE TABLE user (
        user_id int(11) unsigned NOT NULL AUTO_INCREMENT,
        name varchar(30) DEFAULT NULL,
        PRIMARY KEY (user_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE addr (
        id int(11) unsigned NOT NULL AUTO_INCREMENT,
        addr varchar(30) DEFAULT NULL,
        user_id int(11) DEFAULT NULL,
        PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO user(name)
VALUES ("jin"),
        ("po"),
        ("alice"),
        ("petter");

INSERT INTO addr(addr, user_id)
VALUES ("seoul", 1),
        ("pusan", 2),
```

```
("deajeon", 3),
("deagu", 5),
("seoul", 6);
```

INNER JOIN

두 테이블 사이에 공통된 값이 없는 row는 출력하지 않는다.

```
# 두 테이블을 합쳐 id, name, addr 출력
SELECT id, user.name, addr.addr
FROM user
JOIN addr
ON user.user_id = addr.user_id

# world 데이터베이스에서 도시이름과 국가이름을 출력
SELECT city.name AS country_name, country.name AS city_name
FROM city
JOIN country
ON city.CountryCode = country.code

# 아래와 같이 사용할수도 있다.
SELECT city.name AS country_name, country.name AS city_name
FROM city, country
WHERE city.CountryCode = country.code
```

LEFT JOIN

왼쪽 테이블을 기준으로 왼쪽 테이블의 모든 데이터가 출력되고 매핑되는 키값이 없으면 NULL로 출력된다.

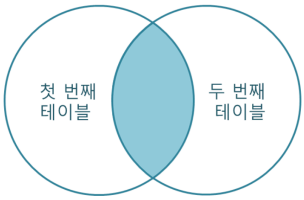
```
# 두 테이블을 합쳐 id, name, addr 출력
```

```
SELECT id, user.name, addr.addr
FROM user
LEFT JOIN addr
ON user.user_id = addr.user_id
```

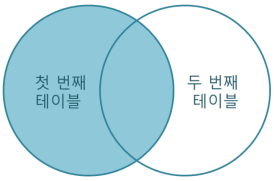
RIGHT JOIN

오른쪽 테이블을 기준으로 왼쪽 테이블의 모든 데이터가 출력되고 매핑되는 키값이 없으면 NULL로 출력된다.

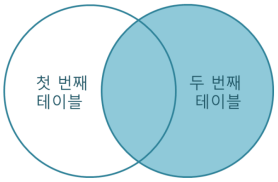
```
# 두 테이블을 합쳐 id, name, addr 출력
SELECT id, user.name, addr.addr
FROM user
RIGHT JOIN addr
ON user.user_id = addr.user_id
```



inner join



left join



right join

```
# JOIN과 DISDINCT의 사용
# 지역과 대륙별 사용하는 언어 출력
SELECT DISTINCT country.Region, country.continent, countrylanguage.Language
FROM country
JOIN countrylanguage
ON countrylanguage.CountryCode = country.Code
```

```
# 대륙과 지역별 사용하는 언어의 수 출력
SELECT sub1.region, sub1.continent, count(*) as count
FROM (
    SELECT DISTINCT country.Region, country.continent, countrylanguage.Language
    FROM country
    JOIN countrylanguage
    ON countrylanguage.CountryCode = country.Code
) AS sub1
GROUP BY sub1.region, sub1.continent

# 테이블 세개 조인하기
# 국가별, 도시별, 언어의 사용율
SELECT country.name as country_name, city.name as city_name,
countrylanguage.language, countrylanguage.percentage
FROM country
JOIN city
ON country.code = city.countrycode
JOIN countrylanguage
ON country.code = countrylanguage.countrycode

SELECT country.name as country_name, city.name as city_name,
countrylanguage.language, countrylanguage.percentage
FROM country, city, countrylanguage
WHERE country.code = city.countrycode and country.code = countrylanguage.countrycode
```

21. UNION

UNION은 SELECT 문의 결과 데이터를 하나로 합쳐서 출력합니다. 컬럼의 갯수와 타입, 순서가 같아야 합니다.

UNION은 자동으로 distinct를 하여 중복을 제거해 줍니다. 중복제거를 안하고 컬럼 데이터를 합치고 싶으면 UNION ALL을 사용합니다.

또한 UNION을 이용하면 Full Outer Join을 구현할수 있습니다.

UNION

```
# user 테이블의 name 컬럼과 addr 테이블의 addr 컬럼의 데이터를 하나로 함쳐서 출력
SELECT name
FROM user
UNION
SELECT addr
FROM addr
```

UNION ALL

```
# 중복데이터를 제거하지 않고 결과 데이터 합쳐서 출력
SELECT name
FROM user
UNION ALL
SELECT addr
FROM addr
```

FULL OUTER JOIN

```
# union을 이용하여 full outer join 구현
SELECT id, user.name, addr.addr
FROM user
```

```
LEFT JOIN addr
ON user.user_id = addr.user_id
UNION
SELECT id, user.name, addr.addr
FROM user
RIGHT JOIN addr
ON user.user_id = addr.user_id
```

22. Sub Query

sub query는 query 문 안에 있는 query를 의미합니다. SELECT절 FROM절, WHERE 등에 사용이 가능합니다.

```
# 전체 나라수, 전체 도시수, 전체 언어수를 출력 ( SELECT 절에 사용 )
SELECT
        (SELECT count(name) FROM city) AS total_city,
        (SELECT count(name) FROM country) AS total_country,
        (SELECT count(DISTINCT(Language)) FROM countrylanguage) AS total_language
FROM DUAL

# 800만 이상되는 도시의 국가코드, 이름, 도시인구수를 출력 ( FROM 절에 사용 )
SELECT *
FROM
        (SELECT countrycode, name, population
        FROM city
        WHERE population > 8000000) AS city
JOIN
        (SELECT code, name
        FROM country) AS country
ON city.countrycode = country.code
```

```
# 800만 이상 도시의 국가코드, 국가이름, 대통령이름을 출력( WHERE 절에 사용 )
SELECT code, name, HeadOfState
FROM country
WHERE code IN (
	SELECT DISTINCT(countrycode) FROM city WHERE population > 8000000
)
```

23. VIEW

가상 테이블로 특 정수로 데이터만 보고자 할때 사용합니다. 실제 데이터를 저장하고 있지는 않습니다. 한마디로 특정 컬럼의 데이터를 보여주는 역할만 합니다. 뷰를 사용 함으로 쿼리를 더 단순하게 만들수 있습니다. 한번 생성된 뷰는 수정이 불가능 하며 인덱스설정이 불가능 합니다.

8.1 syntax

```
CREATE VIEW <뷰이름> AS
(QUERY)
```

8.2 example

```
# 국가코드와 국가이름이 있는 뷰 생성
CREATE VIEW code_name AS
SELECT code, name
FROM country

# city 테이블에 국가 이름 추가
SELECT *
FROM city
JOIN code_name
```

ON city.countrycode = code_name.code

24. INDEX

테이블에서 데이터를 검색할때 빠르게 찾을수 있도록 해주는 기능입니다.

장점

검색속도가 빨라짐

단점

저장공간을 10% 정도 더 많이 차지

INSERT, DELETE, UPDATE 할때 속도가 느려짐

사용법

SELECT시 WHERE 절에 들어가는 컬럼을 Index로 설정하면 좋다.

내부 작동 원리 (B-Tree)

루트노드와 리프노드의 계층적 구조로 루트노드를 이용하여 리프노드에서의 데이터를 빠르게 찾을수 있는 자료구조 알고리즘.

employees 데이터 베이스에서 실행

실행계획을 확인하여 인덱스를 사용하는지 확인

EXPLAIN

SELECT *

FROM salaries

WHERE from_date < "1986-01-01"

EXPLAIN

SELECT *

```
FROM salaries
WHERE to_date < "1986-01-01"

# 인덱스 확인
SHOW INDEX FROM salaries;

# 인덱스 생성
CREATE INDEX fdate
ON salaries (from_date)

CREATE INDEX tdate
ON salaries (to_date)

# 여러개의 컬럼을 가지는 인덱스 생성도 가능
CREATE INDEX ftdate
ON salaries (from_date, to_date)

# 인덱스 삭제
DROP INDEX fdate
ON salaries

DROP INDEX tdate
ON salaries

DROP INDEX ftdate
ON salaries

# 여러개의 컬럼을 조건으로 WHERE절에 사용하는 경우 인덱스 확인
# 인덱스가 하나의 컬럼에 있을때 보다 둘다 있을때가 더 빠름
EXPLAIN
```

```
SELECT *
FROM salaries
WHERE from_date < "1986-01-01" AND to_date < "1986-01-01"
```