Mysql 3

Index

1. JOIN

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

MAKE TEST TABLE & DATA

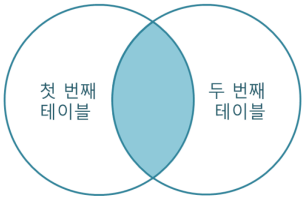
user 테이블

user_id	name
1	Jin
2	Po
3	Alice

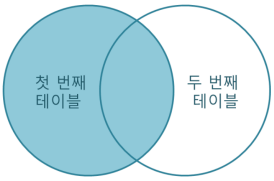
addr 테이블

addr_id	addr	user_id
1	Seoul	1
2	Pusan	2
3	Deagu	4
4	Seoul	5

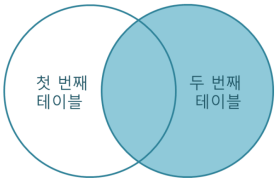
벤다이어그램으로 JOIN의 종류 확인



inner join



left join



right join

Join의 결과

Inner Join

user_id	Name	Addr
1	Jin	Seoul
2	Po	Pusan

Left Join

user_id	Name	Addr
1	Jin	Seoul
2	Po	Pusan
3	Alice	Null

Right Join

user_id	Name	Addr
1	Jin	Seoul
2	Po	Pusan
4	Null	Deagu
5	Null	seoul

 # create table

```
CREATE TABLE user (
        user_id int(11) unsigned NOT NULL AUTO_INCREMENT,
        name varchar(30) DEFAULT NULL,
        PRIMARY KEY (user_id)
)

CREATE TABLE addr (
        addr_id int(11) unsigned NOT NULL AUTO_INCREMENT,
        addr varchar(30) DEFAULT NULL,
        user_id int(11) DEFAULT NULL,
        PRIMARY KEY (addr_id)
)
```

 # insert data

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

INSERT INTO addr(addr, user_id)
VALUES ("seoul", 1),
        ("pusan", 2),
        ("deagu", 4),
        ("seoul", 5);
```

INNER JOIN

두 테이블 사이에 공통된 값이 없는 row는 출력하지 않습니다.

JOIN 쿼리만 실행하면 user와 addr 테이블의 모든 데이터를 매핑하여 출력합니다.

user의 row가 3개, addr의 row가 4개 이므로 총 3*4 = 12개의 row가 출력됩니다.

```
SELECT *
FROM user
JOIN addr;
```

user_id를 기준으로 JOIN하여 결과를 출력하려면 WHERE 절 또는 ON 절에서 user 테이블과 addr 테이블의 같은 user_id 만 출력해야 합니다.

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

ON 대신 WHERE user.user_id = addr.user_id; 를 사용해도 됩니다.

Inner Join의 경우 아래와 같이 간단하게 사용할수도 있습니다.

```
SELECT user.user_id, user.name, addr.addr
FROM user, addr
WHERE user.user_id = addr.user_id;
```

LEFT JOIN

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

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;

 # JOIN 실습 문제

world 데이터베이스에서 도시이름과 국가이름, 국가인구, 도시이름, 도시인구 출력

country_name	country_population	city_name	city_population
Aruba	103000	Oranjestad	29034
Afghanistan	22720000	Kabul	1780000
Afghanistan	22720000	Qandahar	237500
Afghanistan	22720000	Herat	186800
Afghanistan	22720000	Mazar-e-Sharif	127800
Angola	12878000	Luanda	2022000
Angola	12878000	Huambo	163100
Angola	12878000	Lobito	130000
Angola	12878000	Benguela	128300

world 데이터베이스에서 도시인구가 900만명 이상인 도시이름, 국가이름, 국가인구, 도시인구를 국가인구 순으로 내림차순 정렬하여 출력

* HAVING, ORDER BY 사용

country_name	country_population	city_name	city_population
China	1277558000	Shanghai	9696300
India	1013662000	Mumbai (Bombay)	10500000
Indonesia	212107000	Jakarta	9604900
Brazil	170115000	São Paulo	9968485
Pakistan	156483000	Karachi	9269265
South Korea	46844000	Seoul	9981619

sakila 데이터베이스에서 payment와 staff 테이블을 JOIN하여 스테프 아이디, 스테프 전체 이름, 매출액을 출력하세요.

* CONCAT 사용

staff_id	staff_name	amount
1	Mike Hillyer	6.99
1	Mike Hillyer	1.99
1	Mike Hillyer	4.99
1	Mike Hillyer	1.99
2	Jon Step…	0.99
2	Jon Step…	9.99
2	Jon Step…	0.99
2	Jon Step…	5.99
2	Jon Step…	2.99

테이블 세개 조인하기

국가별, 도시별, 언어의 사용율을 출력

country_name	city_name	language	percentage
Aruba	Oranjestad	Dutch	5.3
Aruba	Oranjestad	English	9.5
Aruba	Oranjestad	Papiamento	76.7
Aruba	Oranjestad	Spanish	7.4
Afghanistan	Kabul	Balochi	0.9
Afghanistan	Qandahar	Balochi	0.9
Afghanistan	Herat	Balochi	0.9
Afghanistan	Mazar-e-Sharif	Balochi	0.9

위의 결과에서 도시 인구기준으로 해당 언어를 사용하는 인구수 컬럼 출력

country_name	city_name	language	percentage	city_population	language_population
Aruba	Oranjestad	Dutch	5.3	29034	1539
Aruba	Oranjestad	English	9.5	29034	2758
Aruba	Oranjestad	Papiamento	76.7	29034	22269
Aruba	Oranjestad	Spanish	7.4	29034	2149
Afghanistan	Kabul	Balochi	0.9	1780000	16020
Afghanistan	Kabul	Dari	32.1	1780000	571380
Afghanistan	Kabul	Pashto	52.4	1780000	932720
Afghanistan	Kabul	Turkmenian	1.9	1780000	33820

2. 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 합쳐서 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
```

3. 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, 국가이름, 도시이름, 도시인구수를 출력 ( FROM 절에 사용 )

SELECT city.countrycode, country.name, city.name, city.population

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;


# 위의 문제를 Sub Query를 사용하지 않고 HAVING을 사용해서 출력

SELECT city.countrycode, country.name as country_name, city.name as city_name,
city.population

FROM city

JOIN country

ON city.countrycode = country.code

HAVING city.population > 8000000;
```

 # Sub Query, HAVING 의 비교

* Sub Query 사용
  * JOIN으로 만드는 데이터의 수 : city 테이블 데이터 10개 * country 테이블 데이터 239개 = 2390개
* Having 사용
  * JOIN으로 만드는 데이터의 수 : city 테이블 데이터 4079개 * country 테이블 데이터 239개 = 974881개

> 위의 쿼리에서 쿼리 실행의 결과는 같지만 Sub Query를 사용하는것이 JOIN 했을때 생성되는 데이터의 수가 작으므로 더 효율적인 쿼리가 됩니다.

```
# 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
)
```

 # ANY / ALL

WHERE 절에 결과가 여러개인경우 OR나 AND 조건으로 결과를 출력할때 사용합니다.

```
# 한국이나 브라질보다 인구수가 많은 국가코드, 국가이름, 인구수를 출력
SELECT code, name, population
FROM country
WHERE population >
	ANY (select population FROM country WHERE code in ("KOR", "BRA"));

# 한국과 브라질보다 인구수가 많은 국가코드, 국가이름, 인구수를 출력
SELECT code, name, population
FROM country
WHERE population >
	ALL (select population FROM country WHERE code in ("KOR", "BRA"));

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

 # Sub Query 실습 문제

위의 쿼리를 활용하여 대륙과 지역별 사용하는 언어의 수 출력

* GROUP BY, COUNT 함수 사용, From 절에 위의 쿼리를 활용

region	continent	count
Caribbean	North America	10
Southern and Central Asia	Asia	54
Central Africa	Africa	47
Southern Europe	Europe	22
Middle East	Asia	21
South America	South America	21
Polynesia	Oceania	15

4. VIEW

가상 테이블로 특정한 쿼리를 실행한 결과 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 인덱스 설정이 불가능 합니다.

뷰 생성

CREATE VIEW <뷰이름> AS

(QUERY)

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

 # View 실습 문제

step 1. 한국의 인구수보다 많은 국가의 국가코드, 국가이름, 국가 인구수, 도시이름, 도시 인구수를 출력하고 도시 인구수 순으로 정렬하여 출력

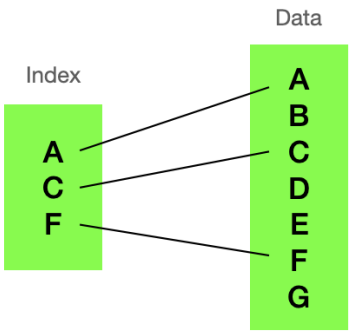
step 2. 한국의 인구수보다 많은 국가의 국가코드, 국가이름, 인구수를 저장하는 뷰를 생성

step 3. 생성한 뷰를 활용하여 step 1의 결과를 출력

code	name	population	name	population
IND	India	1013662000	Mumbai (Bombay)	10500000
KOR	South Korea	46844000	Seoul	9981619
BRA	Brazil	170115000	São Paulo	9968485
CHN	China	1277558000	Shanghai	9696300
IDN	Indonesia	212107000	Jakarta	9604900
PAK	Pakistan	156483000	Karachi	9269265
TUR	Turkey	66591000	Istanbul	8787958
MEX	Mexico	98881000	Ciudad de México	8591309
RUS	Russian Federation	146934000	Moscow	8389200

5. 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.

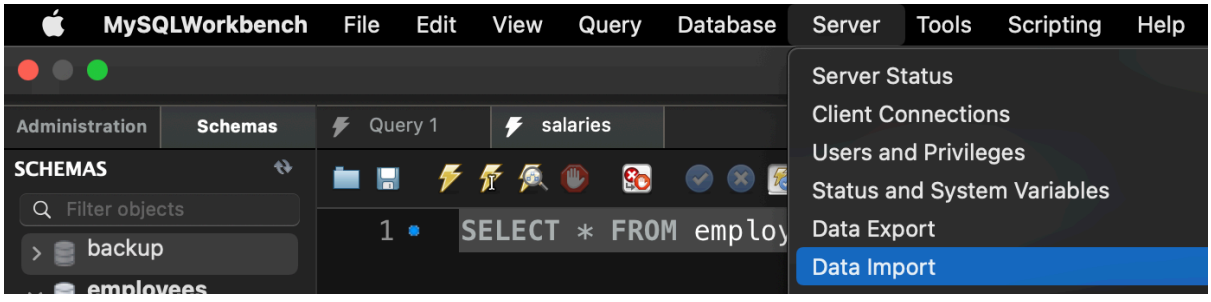
내부 작동 원리 (B-Tree)

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

 # employees 데이터 저장하기

sql> create database employees;

Server > Data Import

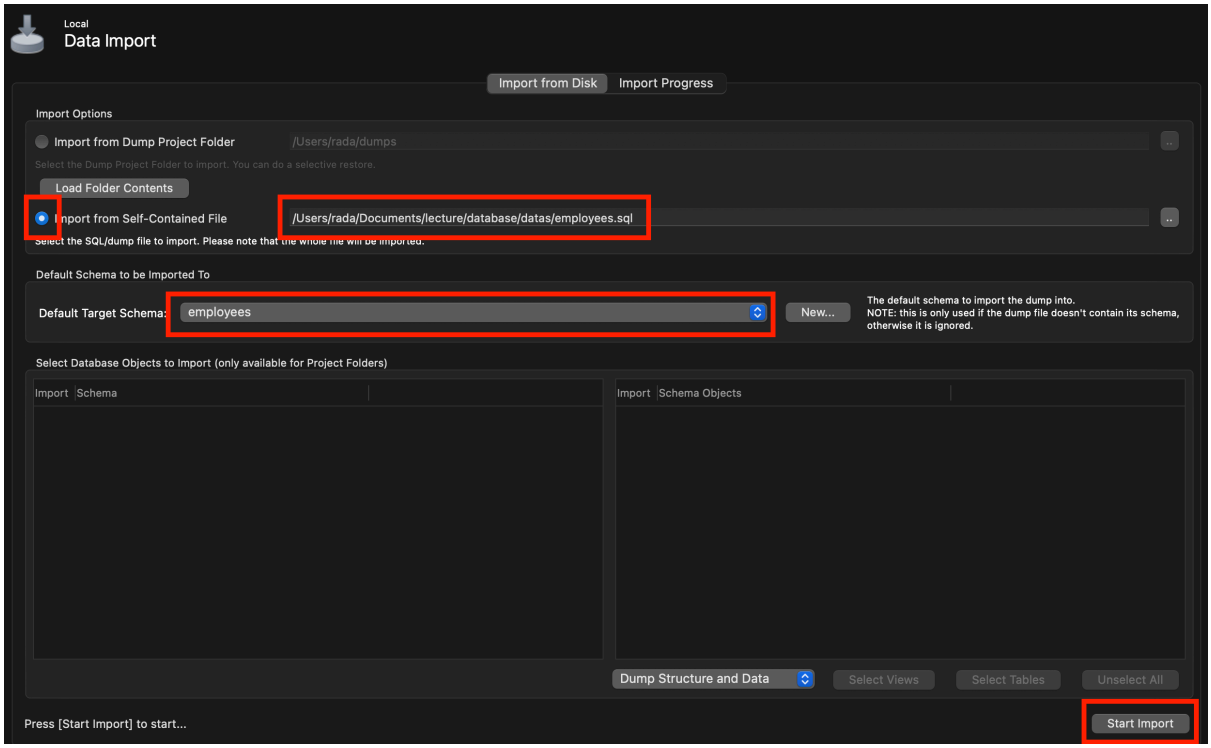


1. Import from Self-Contained File 선택

2. employees.sql 파일을 파일 시스템에서 찾아서 선택

3. Default Target Schema를 미리 만들어 놓은 employees 데이터 베이스 선택

4. Start Import 버튼 클릭하여 실행



employees 데이터 베이스에서 실행

USE employees;

인덱스 확인

SHOW INDEX FROM salaries;

* Primary key로 설정된 컬럼은 클러스터형 인덱스가 자동으로 생성됩니다.

인덱스의 종류에는 클러스터형(clustered) 인덱스와 보조(secondary) 인덱스가 있습니다.

클러스터형 인덱스 : 데이터 자체가 인덱스로 사용되어 검색 속도 향상에 큰 영향을 주지 않습니다. 행 데이터를 자신의 열을 기준으로 정렬해주는 기능을 합니다.

보조 인덱스 : 일부 데이터만 추출하여 사용하기 때문에 검색속도가 향상됩니다.

```
# SQL 쿼리 실행으로 SQL 쿼리의 실행 속도 확인
# workbench 사용시 데이터 출력 속도가 느려 인덱스의 성능을 정확하게 확인할수 없습니다.
# sequal pro 또는 heidisql 을 사용해서 확인해보시면 좋습니다.
SELECT *
FROM salaries
WHERE from_date < "1986-01-01"

SELECT *
FROM salaries
WHERE to_date < "1986-01-01"

# SQL 쿼리의 실행 계획은 확인합니다.
# index를 사용하지 않는것을 확인할수 있습니다.
EXPLAIN
SELECT *
FROM salaries
WHERE from_date < "1986-01-01"

EXPLAIN
SELECT *
FROM salaries
WHERE to_date < "1986-01-01"

# 인덱스를 생성하고 확인합니다.
CREATE INDEX fdate ON salaries (from_date);
CREATE INDEX tdate ON salaries (to_date);
SHOW INDEX FROM salaries;
```

```
# 실행계획을 확인하여 인덱스를 사용하는지 확인합니다.
EXPLAIN
SELECT *
FROM salaries
WHERE from_date < "1986-01-01"

EXPLAIN
SELECT *
FROM salaries
WHERE to_date < "1986-01-01"

# SELECT 쿼리를 실행하여 속도를 확인합니다.
SELECT *
FROM salaries
WHERE from_date < "1986-01-01"

SELECT *
FROM salaries
WHERE to_date < "1986-01-01"

# 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";
```

6. TRIGGER

특정 테이블을 감시하고 있다가 설정한 조건에 감지되면 지정해 놓은 쿼리가 자동으로 실행되도록 하는 방법입니다.

문법

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | UPDATE| DELETE }
ON table_name FOR EACH ROW
BEGINE
        trigger_body;
END;
```

```
# 데이터를 삭제하면 다른 테이블에 백업하는 트리거 생성
create database tr;
use tr;

create table chat(
    id VARCHAR(32),
    answer VARCHAR(32) NOT NULL
);
```

```
create table chatBackup(
    idBackup VARCHAR(32),
    answerBackup VARCHAR(32) NOT NULL,
    backupDate timestamp DEFAULT CURRENT_TIMESTAMP
);

# 트리거 설정
delimiter |
        create trigger backup
        before delete on chat
        for each row begin
                insert into chatBackup(idBackup, answerBackup)
                values(old.id, old.answer);
        end |

# 트리거 확인
show triggers;

# 데이터 추가
insert into chat(id, answer)
values (1, "hello"), (2, "mysql"), (3, "hello");

# 데이터 확인
select * from chat;
select * from chatBackup;

# 삭제 쿼리 실행
delete from chat
where answer = "hello"
limit 10;
select * from chat;
```

```
# 트리거가 실행되었는지 확인
select * from chatBackup;
```

7. BACKUP

(1) Backup의 종류

Hot Backup

데이터 베이스를 중지하지 않은 상태로 데이터 백업

- 백업하는 동안 서비스가 실행
- 백업하는 동안 데이터가 변경되어 완전한 백업이 안될수 있음

Cold Backup

데이터 베이스를 중지한 상태로 데이터 백업

- 안정적으로 백업이 가능
- 백업하는 동안 서비스가 중단되어야 함

Logical Backup

SQL 문으로 백업

- 느린 속도의 백업과 복원
- 디스크 용량을 적게 사용
- 작업시 시스템 자원을 많이 사용
- 문제 발생에 대한 파악이 쉬움
- 서버 OS 호환이 잘됨

Physical Backup

파일 차체를 백업

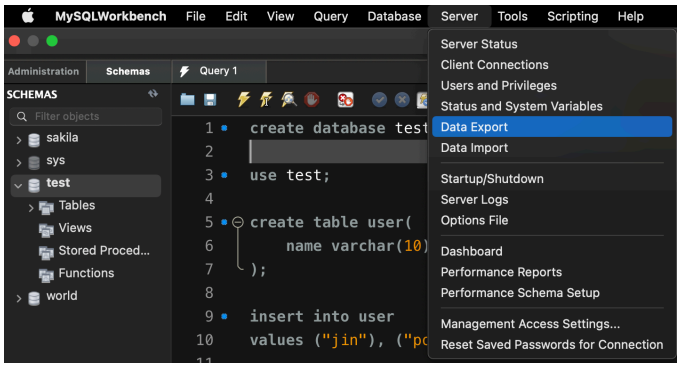
- 빠른 속도의 백업과 복원
- 디스크 용량 많이 사용
- 작업시 시스템 자원을 적게 사용
- 문제 발생에 대한 파악과 검토가 어려움
- 서버 OS 호환이 잘안될수 있음

 # Hot Logical Backup 실습

MySQL Workbench를 이용

 # SQL 파일로 백업

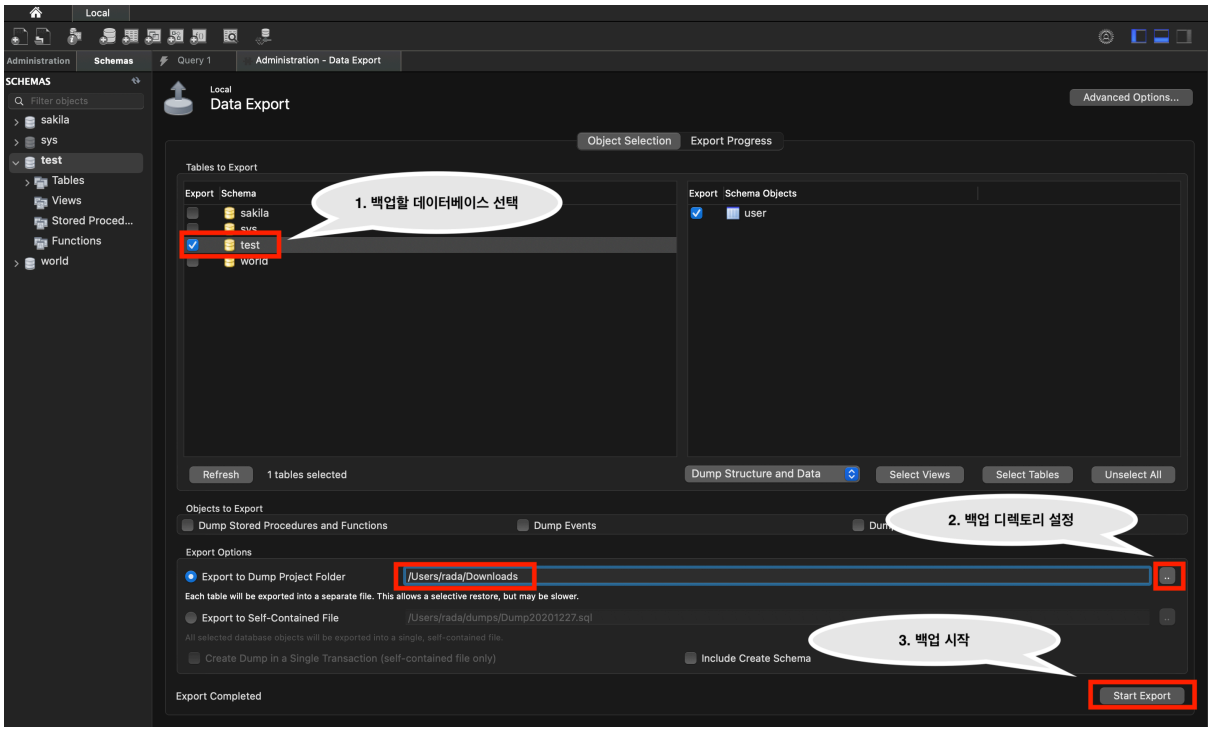
Server > Data Export 선택



1. 백업할 데이터베이스 선택

2. 백업 파일을 저장할 디렉토리 선택

3. 백업시작



 # 복원하기

1. 데이터 베이스 선택

USE <데이터 베이스 이름>

2. 스크립트 실행해서 테이블과 데이터 추가

File > Open SQL Script 메뉴를 이용하여 스크립트를 열고 실행

 # CSV 파일로 백업

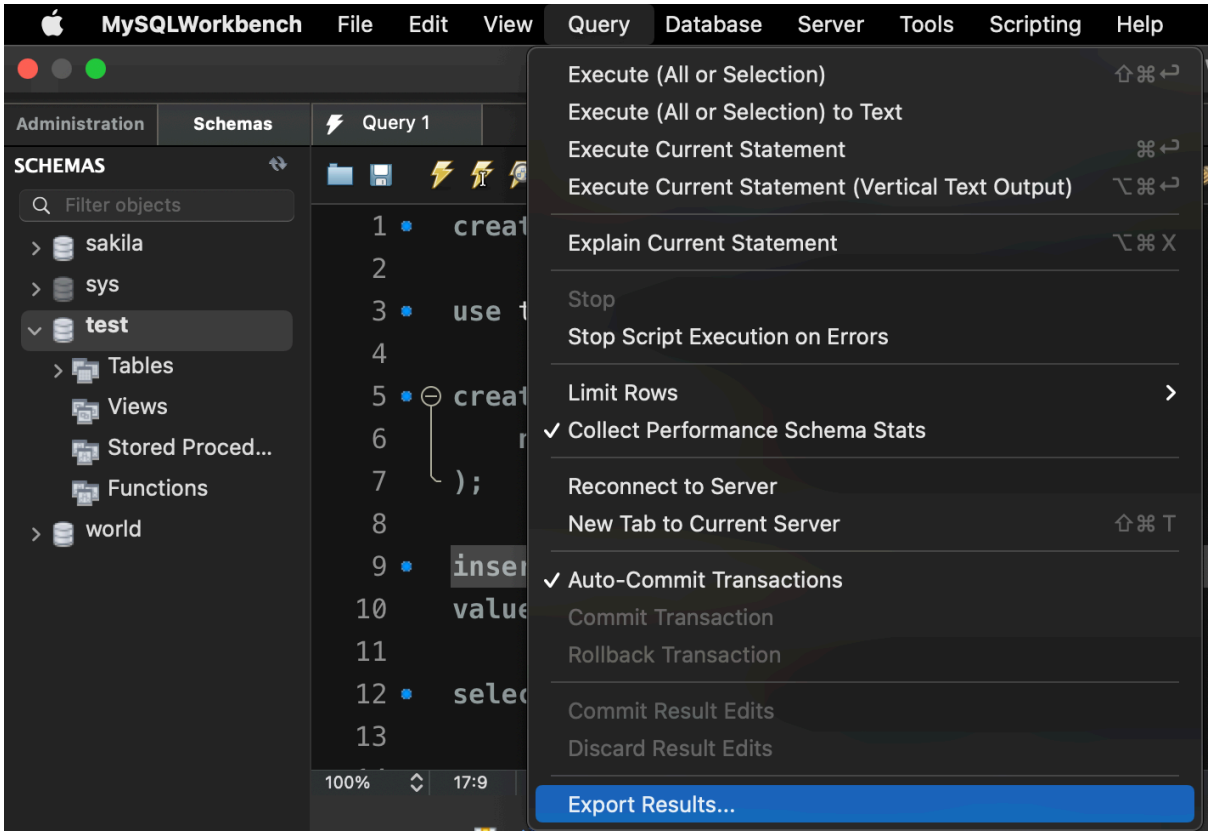
CSV 파일은 엑셀로 열어서 데이터를 볼수 있습니다.

저장하고자 하는 데이터를 출력하는 쿼리 실행

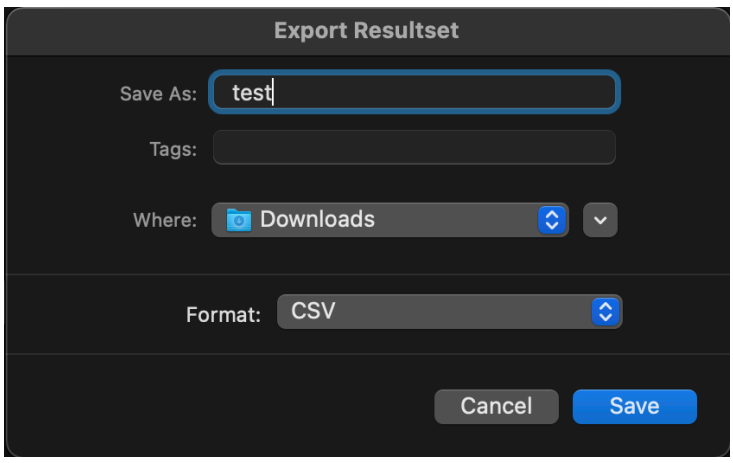
USE world;

SELECT * FROM country;

Query > Export Results



저장할 데이터의 파일 및 디렉토리 설정 후 Save 버튼 클릭



 # Backup 실습 문제

(1) ssac 이름의 데이터 베이스를 생성

(2) 테이블 생성

Field	Type	Null	Key	Default	Extra
user_id	int	NO	PRI	NULL	auto_increment
name	varchar(50)	YES		NULL	
age	int	YES		NULL	

(3) 데이터 추가

user_id	name	age
1	andy	23
2	peter	35

(4) ssac 데이터 베이스를 sql 파일로 백업

(5) backup 데이터 베이스를 만들어 백업 받은 sql 파일을 복원