

Module 12

병행성 제어



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book and NHN Academy was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Copyright © 2022 NHN Academy Corp.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the South Korea.

Module 12: 병행성 제어

데이터베이스 관리 시스템은 조직내의 거의 모든 사용자가 동시에 사용하는 다중 사용자 시스템으로, 개별 사용자 프로그램이 동시에 조작하는 데이터에 대한 무결성을 위한 병행성을 제어할 수 있어야 합니다. 데이터베이스 관리 시스템에서는 트랜잭션을 사용하여 병행성을 제어합니다. 이 단원에서는 데이터베이스 관리 시스템의 병행 제어와 복구를 알아봅니다.

Lab 12-1: 트랜잭션 제어

이 연습에서는 MySQL에서 제공하는 트랜잭션 문법을 사용하여 명시적/암시적 트랜잭션을 제어하는 것을 연습합니다. 아래 절차에 따릅니다.

1. macOS에서는 터미널, Windows에서는 명령 프롬프트를 실행합니다.

2. 아래 명령을 실행하여 MySQL 콘솔에 접근합니다.

```
$ mysql -u root -p
Enter password:
```

3. 아래 명령을 실행하여 데이터베이스 서버의 트랜잭션 커밋 모드를 확인합니다.

```
mysql> show variables like 'autocommit';

+-----+-----+
| Variable_name | Value |
+-----+-----+
| autocommit    | ON    |
+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

암시적 트랜잭션

4. 아래 명령을 수행하여 안중근 승객의 나이를 30으로 업데이트 합니다.

```
mysql> UPDATE passenger SET Age = 30 WHERE PassengerNo = 3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

5. 아래 질의를 수행하여 수정된 안중근 승객의 나이를 확인합니다.

```
mysql> SELECT * FROM passenger WHERE PassengerNo = 3;

+-----+-----+-----+-----+
| PassengerNo | PassengerName | Grade | Age |
+-----+-----+-----+-----+
|          3 | 안중근        | 10    | 30  |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

명시적 트랜잭션

6. 아래 명령을 수행하여 명시적 트랜잭션을 시작합니다.

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

7. 아래 질의를 수행하여 안중근 고객의 나이를 40으로 업데이트 합니다.

```
mysql> UPDATE passenger SET Age = 40 WHERE PassengerNo = 3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

8. 아래 질의를 수행하여 수정된 안중근 승객의 나이를 확인합니다.

```
mysql> SELECT * FROM passenger WHERE PassengerNo = 3;
+-----+-----+-----+-----+
| PassengerNo | PassengerName | Grade | Age |
+-----+-----+-----+-----+
|          3 | 안중근          |    10 |  40 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

9. 아래 질의를 수행하여 명시적으로 시작된 트랜잭션을 Rollback 합니다.

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
```

10. 아래 질의를 수행하여 취소된 트랜잭션을 확인합니다.

```
mysql> SELECT * FROM passenger WHERE PassengerNo = 3;
+-----+-----+-----+-----+
| PassengerNo | PassengerName | Grade | Age |
+-----+-----+-----+-----+
|          3 | 안중근          |    10 |  40 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

트랜잭션 모드 변경

11. 아래 명령을 실행하여 트랜잭션 모드를 변경합니다.

```
mysql> SET Autocommit = 0;
Query OK, 0 rows affected (0.00 sec)
```

12. 아래 명령을 실행하여 데이터베이스 서버의 트랜잭션 커밋 모드를 확인합니다.

```
mysql> show variables like 'autocommit';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| autocommit    | OFF   |
+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

13. 아래 질의를 수행하여 안중근 고객의 나이를 40으로 업데이트 합니다.

```
mysql> UPDATE passenger SET Age = 40 WHERE PassengerNo = 3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

14. 아래 질의를 수행하여 수정된 안중근 승객의 나이를 확인합니다.

```
mysql> SELECT * FROM passenger WHERE PassengerNo = 3;
+-----+-----+-----+-----+
| PassengerNo | PassengerName | Grade | Age |
+-----+-----+-----+-----+
|          3 | 안중근          |    10 |  40 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

15. 아래 질의를 수행하여 트랜잭션을 롤백합니다.

```
mysql> ROLLBACK;  
Query OK, 0 rows affected (0.00 sec)
```

16. 아래 질의를 수행하여 취소된 트랜잭션을 확인합니다.

```
mysql> SELECT * FROM passenger WHERE PassengerNo = 3;  
+-----+-----+-----+-----+  
| PassengerNo | PassengerName | Grade | Age |  
+-----+-----+-----+-----+  
|          3 | 안중근        |    10 |   40 |  
+-----+-----+-----+-----+  
row in set (0.00 sec)
```


Lab 12-2: Deadlock

이 연습에서는 같은 데이터베이스 객체에 두 트랜잭션이 접근하여 잠금 해제를 기다리는 관계에 사이클이 생기는 경우에 발생하는 DeadLock이 발생하는 상황을 시뮬레이션하고 살펴봅니다.

1. macOS에서는 터미널, Windows에서는 명령 프롬프트를 실행합니다. (터미널 1로 호칭합니다)

2. 아래 명령을 실행하여 MySQL 콘솔에 접근합니다.

```
$ mysql -u root -p
Enter password:
```

3. 아래 명령을 수행하여 Deadlock 감지를 수행하지 않도록 설정합니다.

```
mysql> set global innodb_deadlock_detect = off;
Query OK, 0 rows affected (0.01 sec)
```

4. 아래 명령을 수행하여 터미널 1의 프로세스가 가진 잠금을 확인합니다.

```
mysql> SELECT * FROM performance_schema.data_locks;
Empty set (0.00 sec)
```

5. macOS에서는 터미널, Windows에서는 명령 프롬프트를 하나 더 실행합니다. (터미널 2로 호칭합니다)

6. 터미널 2에서 아래 명령을 실행하여 MySQL 콘솔에 접근합니다.

```
$ mysql -u root -p
Enter password:
```

터미널 1에서 잠금 수행

7. 터미널 1에서 아래 명령을 수행하여 트랜잭션을 시작합니다.

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

8. 아래 질의를 시작하여 데이터에 대해 잠금을 수행합니다.

```
mysql> SELECT * FROM Passenger WHERE Grade = 10 FOR UPDATE;
+-----+-----+-----+-----+
| PassengerNo | PassengerName | Grade | Age |
+-----+-----+-----+-----+
|          3 | 안중근        |    10 |   30 |
|          4 | 김영랑        |    10 |   54 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

9. 아래 명령을 수행하여 터미널 1의 프로세스 ID를 확인합니다.

```
mysql> SELECT connection_id();
+-----+
| connection_id() |
+-----+
|          8      |
+-----+
```

```
1 row in set (0.00 sec)
```

10. 아래 명령을 수행하여 터미널 1의 프로세스가 잠근 객체를 확인합니다.

```
mysql> SELECT engine, engine_transaction_ID, Thread_ID, Object_schema, object_name,
Lock_type, Lock_Mode, Lock_data FROM performance_schema.data_locks;
```

engine	engine_transaction_ID	Thread_ID	Object_schema	object_name	Lock_type	Lock_Mode	Lock_data
INNODB	26148	49	module06	passenger	TABLE		
IX	NULL						
INNODB	26148	49	module06	passenger	RECORD	10, 3	
X							
INNODB	26148	49	module06	passenger	RECORD	10, 4	
X							
INNODB	26148	49	module06	passenger	RECORD		
X,REC_NOT_GAP	4						
INNODB	26148	49	module06	passenger	RECORD		
X,REC_NOT_GAP	3						
INNODB	26148	49	module06	passenger	RECORD		
X,GAP	11, 2						

```
6 rows in set (0.00 sec)
```

터미널 2에서 잠금 수행

11. 터미널 2에서, 아래 명령을 수행하여 트랜잭션을 시작합니다.

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

12. 터미널 2에서, 아래 질의를 수행하여 데이터에 대해 잠금을 수행합니다.

```
mysql> SELECT * FROM Passenger WHERE Grade = 9 FOR UPDATE;
```

PassengerNo	PassengerName	Grade	Age
5	김소월	9	45
7	천상병	9	55
8	조지훈	9	43

```
3 rows in set (0.00 sec)
```

13. 아래 명령을 수행하여 터미널 2의 프로세스가 잠근 객체를 확인합니다.

```
mysql> SELECT engine, engine_transaction_ID, Thread_ID, Object_schema, object_name,
Lock_type, Lock_Mode, Lock_data FROM performance_schema.data_locks;
```

```

+-----+-----+-----+-----+-----+-----+-----+
| engine | engine_transaction_ID | Thread_ID | Object_schema | object_name | Lock_type | Lock_Mode | Lock_data |
+-----+-----+-----+-----+-----+-----+-----+
| INNODB | 26149 | 60 | module06 | passenger | TABLE | NULL | |
| IX | 26149 | 60 | module06 | passenger | RECORD | 9, 5 |
| INNODB | 26149 | 60 | module06 | passenger | RECORD | 9, 8 |
| INNODB | 26149 | 60 | module06 | passenger | RECORD | 9, 7 |
| INNODB | 26149 | 60 | module06 | passenger | RECORD | X,REC_NOT_GAP | 5 |
| INNODB | 26149 | 60 | module06 | passenger | RECORD | X,REC_NOT_GAP | 7 |
| INNODB | 26149 | 60 | module06 | passenger | RECORD | X,REC_NOT_GAP | 8 |
| INNODB | 26149 | 60 | module06 | passenger | RECORD | X,GAP | 10, 3 |
| INNODB | 26148 | 49 | module06 | passenger | TABLE | NULL |
| IX | 26148 | 49 | module06 | passenger | RECORD | 10, 3 |
| INNODB | 26148 | 49 | module06 | passenger | RECORD | 10, 4 |
| INNODB | 26148 | 49 | module06 | passenger | RECORD | X,REC_NOT_GAP | 4 |
| INNODB | 26148 | 49 | module06 | passenger | RECORD | X,REC_NOT_GAP | 3 |
| INNODB | 26148 | 49 | module06 | passenger | RECORD | X,GAP | 11, 2 |
+-----+-----+-----+-----+-----+-----+-----+
14 rows in set (0.00 sec)

```

14. 터미널 1에서, 아래 질의를 수행하여 현재 실행중인 트랜잭션을 확인합니다.

```

mysql> SELECT trx_id, trx_state, trx_mysql_thread_id, trx_query FROM
information_schema.innodb_trx;

+-----+-----+-----+-----+
| trx_id | trx_state | trx_mysql_thread_id | trx_query |
+-----+-----+-----+-----+
| 26149 | RUNNING | 19 | NULL |
| 26148 | RUNNING | 8 | SELECT trx_id, trx_state,
trx_mysql_thread_id, trx_query FROM information_schema.innodb_trx |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

15. 터미널 2에서, 아래 질의를 수행하여 현재 실행중인 트랜잭션을 확인합니다.

```

mysql> SELECT trx_id, trx_state, trx_mysql_thread_id, trx_query FROM
information_schema.innodb_trx;

```

```

+-----+-----+-----+-----+
|      trx_id      |      trx_state      |      trx_mysql_thread_id      |      trx_query      |
|-----+-----+-----+-----+
|      26149      |      RUNNING      |      19      |      SELECT trx_id, trx_state,
trx_mysql_thread_id, trx_query FROM information_schema.innodb_trx |
|      26148      |      RUNNING      |      8      |      NULL      |
|-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

모니터링을 위한 터미널 실행

16. macOS에서는 터미널, Windows에서는 명령 프롬프트를 하나 더 실행합니다. (터미널 3로 호칭합니다)
17. 터미널 3에서 아래 명령을 실행하여 MySQL 콘솔에 접근합니다.

```

$ mysql -u root -p
Enter password:

```

터미널 1에서 터미널 2가 잠고 있는 데이터에 액세스

18. 터미널 1에서, 아래 쿼리를 수행하여 터미널 2의 프로세스가 잠금중인 객체에 잠금을 시도합니다.

```
mysql> SELECT * FROM Passenger WHERE Grade = 9 FOR UPDATE;
```

DeadLock 발생

19. 터미널 2에서, 아래 쿼리를 수행하여 터미널 1의 프로세스가 잠금중인 객체에 잠금을 시도합니다.

```
mysql> SELECT * FROM Passenger WHERE Grade = 10 FOR UPDATE;
```

20. 터미널 3에서, 아래 쿼리를 수행하여 프로세스를 확인합니다.

```

mysql> show processlist;
+-----+-----+-----+-----+-----+-----+
| Id | User          | Host          | db          | Command | Time | State |
|----+-----+-----+-----+-----+-----+
| 5  | event_scheduler | localhost    | NULL        | Daemon  | 311635 | Waiting on empty queue |
| 8  | root          | localhost:62924 | module06    | Query   | 12    | executing |
| 19 | root          | localhost:55794 | module06    | Query   | 8     | executing |
| 20 | root          | localhost:56150 | NULL        | Query   | 0     | init      |
| show processlist |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

21. 터미널 3에서, 아래 명령을 수행하여 터미널 1에서 수행중인 프로세스를 kill 합니다.

```
mysql> kill 8;
Query OK, 0 rows affected (0.00 sec)
```

22. 터미널 1의 상태를 확인합니다.

```
mysql> SELECT * FROM Passenger WHERE Grade = 9 FOR UPDATE;
ERROR 2013 (HY000): Lost connection to MySQL server during query
No connection. Trying to reconnect...
Connection id: 21
Current database: module06
```

23. 터미널 2의 상태를 확인합니다.

```
mysql> SELECT * FROM Passenger WHERE Grade = 10 FOR UPDATE;

+-----+-----+-----+-----+
| PassengerNo | PassengerName | Grade | Age |
+-----+-----+-----+-----+
|          3 | 안중근        |    10 |  30 |
|          4 | 김영랑        |    10 |  54 |
+-----+-----+-----+-----+

2 rows in set (7.81 sec)
```

Deadlock 감지기능 활성화

24. 아래 명령을 수행하여 Deadlock 감지기능을 활성화 합니다.

```
mysql> set global innodb_deadlock_detect = on;
Query OK, 0 rows affected (0.01 sec)
```

25. 연습을 처음부터 다시 수행하여, Deadlock이 방지되는지 확인합니다.

```
mysql> SELECT * FROM Passenger WHERE Grade = 10 FOR UPDATE;
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction
```

Lab 12-3: 트랜잭션 격리 수준과 잠금

이 연습에서는 트랜잭션 격리 수준에 따른 잠금을 MySQL에서 제공하는 문법을 사용하여 테스트합니다. 아래 절차에 따릅니다.

데이터베이스 트랜잭션 확인

1. macOS에서는 터미널, Windows에서는 명령 프롬프트를 실행합니다.

2. 아래 명령을 실행하여 MySQL 콘솔에 접근합니다.

```
$ mysql -u root -p
Enter password:
```

3. 아래 질의를 수행하여 현재 세션의 프로세스 ID를 확인합니다.

```
mysql> SELECT connection_id();
+-----+
| connection_id() |
+-----+
|                50 |
+-----+
1 row in set (0.02 sec)
```

4. 아래 질의를 수행하여 현재 세션의 트랜잭션 개수를 확인합니다.

```
mysql> SELECT count(1) FROM information_schema.innodb_trx
-> WHERE trx_mysql_thread_id = CONNECTION_ID();
+-----+
| count(1) |
+-----+
|          0 |
+-----+
1 row in set (0.00 sec)
```

5. 아래 질의를 수행하여 트랜잭션을 시작합니다.

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

6. 아래 질의를 수행하여 안중근 고객의 나이를 40으로 업데이트 합니다.

```
mysql> UPDATE Passenger SET
-> Age = 40
-> WHERE PassengerNo = 3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

7. 아래 질의를 수행하여 현재 세션의 트랜잭션 개수를 확인합니다.

```
mysql> SELECT count(1) FROM information_schema.innodb_trx
-> WHERE trx_mysql_thread_id = CONNECTION_ID();
+-----+
| count(1) |
+-----+
```

```
|          1 |
+-----+
1 row in set (0.00 sec)
```

8. 아래 질의를 수행하여 트랜잭션을 롤백합니다.

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.01 sec)
```

9. 아래 질의를 수행하여 현재 세션의 트랜잭션 개수를 확인합니다.

```
mysql> SELECT count(1) FROM information_schema.innodb_trx
      -> WHERE trx_mysql_thread_id = CONNECTION_ID();
+-----+
| count(1) |
+-----+
|          0 |
+-----+
1 row in set (0.00 sec)
```

예제 데이터베이스 작성

1. MySQL Workbench를 실행하고 localhost 데이터베이스에 접속합니다.

2. 아래 질의를 수행하여 예제 데이터베이스를 작성합니다.

```
CREATE DATABASE Module12;
```

3. 아래 질의를 수행하여 데이터베이스 컨텍스트를 Module12로 변경합니다.

```
USE Module12;
```

4. 아래 질의를 수행하여 테이블을 생성합니다.

```
CREATE TABLE Members (
  Id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
  Name varchar(10) DEFAULT '홍길동',
  City varchar(10) DEFAULT '부산');
```

5. 아래 질의를 수행하여 Members 테이블에 데이터를 삽입하는 저장 프로시저를 생성합니다.

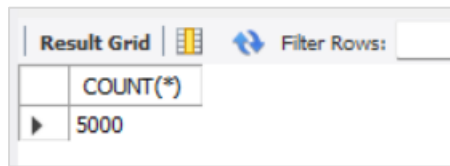
```
DELIMITER $$
CREATE PROCEDURE insertMembers()
BEGIN
  DECLARE i int DEFAULT 1;
  WHILE (i <= 5000) DO
    INSERT INTO members () VALUES ();
    SET i = i + 1;
  END WHILE;
END $$
DELIMITER ;
```

6. 아래 질의를 수행하여 Members 테이블에 데이터를 삽입합니다.

```
CALL insertMembers();
```

7. 아래 질의를 수행하여 Members 테이블의 데이터를 확인합니다.

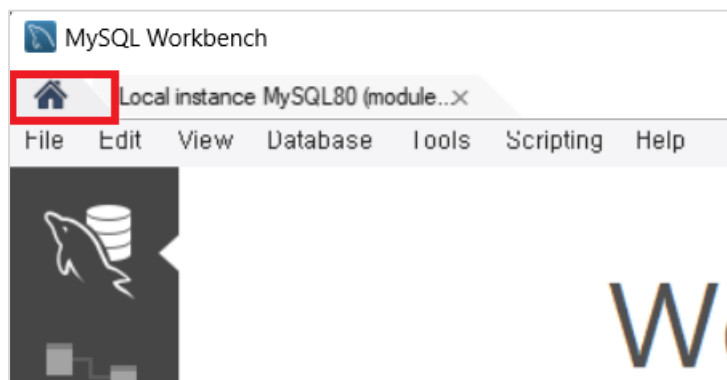
```
SELECT COUNT(*) FROM Members;
```



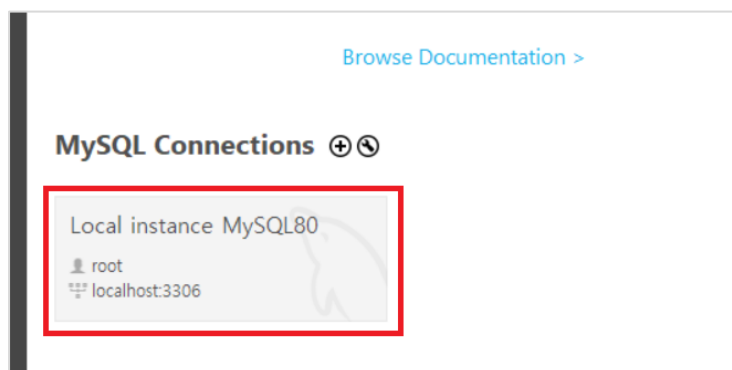
Result Grid
COUNT(*)
5000

프로세스 ID 확인

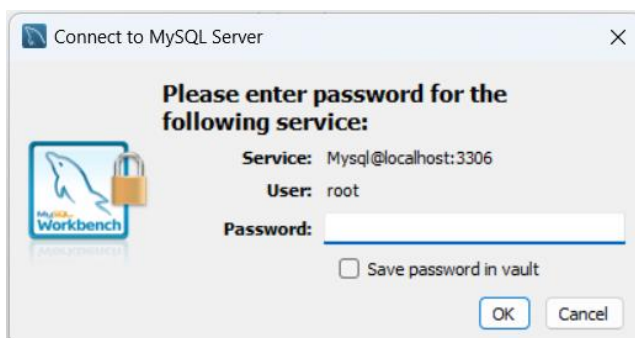
1. MySQL Workbench에서, 왼쪽 위의 HOME 탭을 클릭합니다.



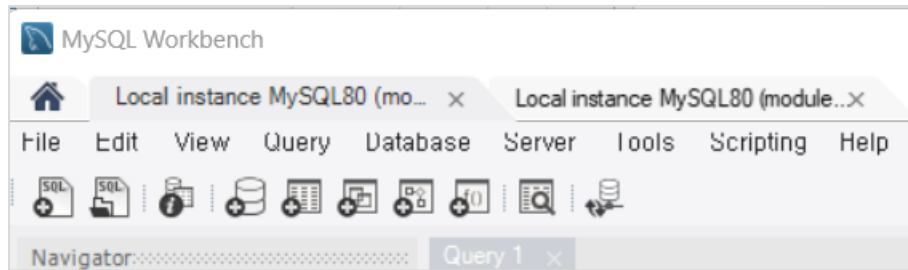
2. MySQL Connections 구역에서 Local instance를 클릭하여 연결을 엽니다.



3. 패스워드를 입력하고 로그인합니다.

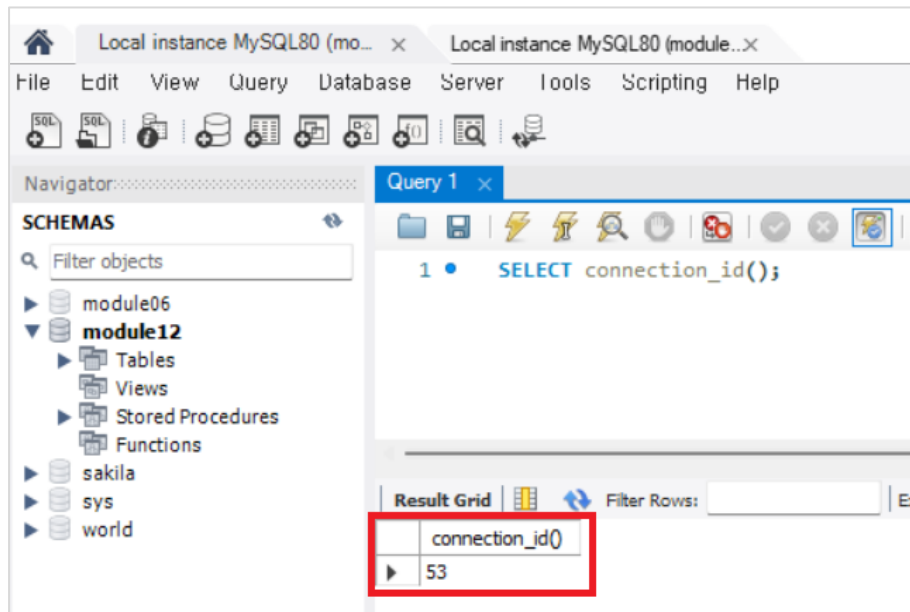


4. MySQL Workbench에서 두 개의 데이터베이스 연결을 확인합니다.



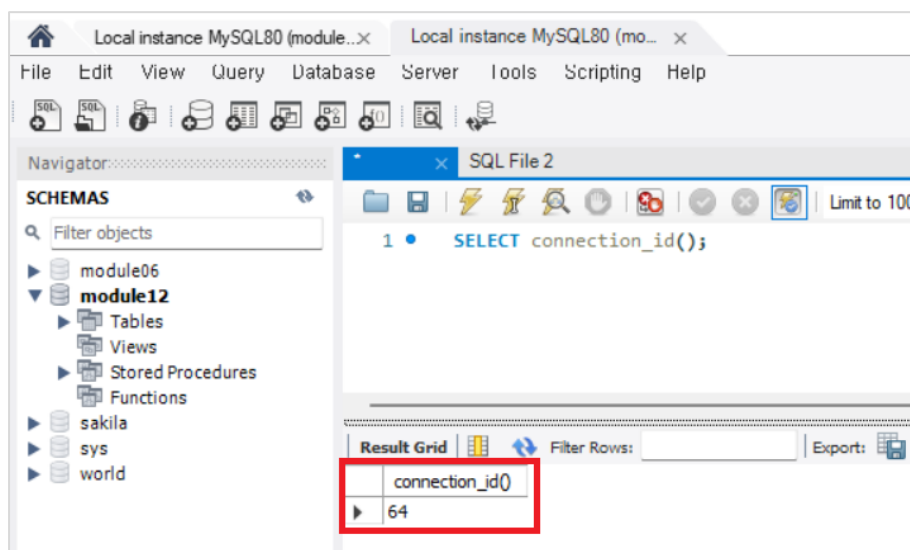
5. 첫 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 프로세스 ID를 확인합니다.

```
SELECT connection_id();
```



6. 두 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 프로세스 ID를 확인합니다.

```
SELECT connection_id();
```



7. 첫 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 전체 프로세스를 확인합니다.

Show full processlist;

Result Grid								
Filter Rows:								
Export: Wrap Cell Content:								
	Id	User	Host	db	Command	Time	State	Info
▶	5	event_scheduler	localhost	NULL	Daemon	785694	Waiting on empty queue	NULL
	53	root	localhost:4074	module12	Query	0	init	show full processlist
	63	root	localhost:1504	module12	Sleep	147		NULL

수행되는 트랜잭션을 두 세션에서 확인

1. 첫 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 트랜잭션을 명시적으로 시작합니다.

```
START TRANSACTION;
```

2. 첫 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 Id가 1000 번에서 1500번 사이의 데이터를 업데이트 합니다.

```
UPDATE Members SET
```

```
City = '광주'
```

```
WHERE Id BETWEEN 1000 AND 1500;
```

3. 첫 번째 탭의 쿼리 창에서, 다음 질의를 수행하여 업데이트 된 데이터를 확인합니다.

```
SELECT * FROM MEMBERS
```

```
WHERE Id BETWEEN 1200 AND 1210;
```

Result Grid			
Filter Rows			
	Id	Name	City
▶	1200	홍길동	광주
	1201	홍길동	광주
	1202	홍길동	광주
	1203	홍길동	광주
	1204	홍길동	광주
	1205	홍길동	광주
	1206	홍길동	광주
	1207	홍길동	광주
	1208	홍길동	광주
	1209	홍길동	광주
	1210	홍길동	광주
*	NULL	NULL	NULL

4. 첫 번째 탭의 쿼리 창에서, 다음 질의를 수행하여 실행중인 트랜잭션을 확인합니다.

```
SELECT count(1) FROM information_schema.innodb_trx
```

```
WHERE trx_mysql_thread_id = CONNECTION_ID();
```

Result Grid	
count(1)	
▶	1

5. 첫 번째 탭의 쿼리 창에서, 다음 질의를 수행하여 트랜잭션이 잠근 데이터를 확인합니다.

```
SELECT * FROM performance_schema.data_locks;
```

ENGINE	ENGINE_LOCK_ID	ENGINE_TRANSACTION_ID	THREAD_ID	EVENT_ID	OBJECT_SCHEMA	OBJECT_NAME	PARTITION_NAME	SUBPARTITION_NAME	INDEX_NAME
INNODB	2264784771792:1104:2264773498904	26925	94	125402	module12	members	NULL	NULL	NULL
INNODB	2264784771792:42:8:61:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:62:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:63:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:64:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:65:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:66:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:67:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:68:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:69:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:70:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:71:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:72:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:73:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:74:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:75:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:76:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:77:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:78:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:79:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:80:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:81:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:82:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:83:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY
INNODB	2264784771792:42:8:84:2264773498904	26925	94	125402	module12	members	NULL	NULL	PRIMARY

6. 두 번째 탭의 쿼리 창에서, 다음 질의를 수행하여 다른 세션에서 업데이트한 데이터를 확인합니다.

```
SELECT * FROM MEMBERS
WHERE Id BETWEEN 1200 AND 1210;
```

	Id	Name	City
▶	1200	홍길동	부산
	1201	홍길동	부산
	1202	홍길동	부산
	1203	홍길동	부산
	1204	홍길동	부산
	1205	홍길동	부산
	1206	홍길동	부산
	1207	홍길동	부산
	1208	홍길동	부산
	1209	홍길동	부산
	1210	홍길동	부산
●	NULL	NULL	NULL

7. 첫 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 트랜잭션을 커밋합니다.

```
COMMIT;
```

8. 첫 번째 탭의 쿼리 창에서, 다음 질의를 수행하여 실행중인 트랜잭션을 확인합니다.

```
SELECT count(1) FROM information_schema.innodb_trx
WHERE trx_mysql_thread_id = CONNECTION_ID();
```

	count(1)
▶	0

9. 두 번째 탭의 쿼리 창에서, 다음 질의를 수행하여 다른 세션에서 업데이트한 데이터를 확인합니다.

```
SELECT * FROM MEMBERS
WHERE Id BETWEEN 1200 AND 1210;
```

Result Grid			
	Id	Name	City
▶	1200	홍길동	광주
	1201	홍길동	광주
	1202	홍길동	광주
	1203	홍길동	광주
	1204	홍길동	광주
	1205	홍길동	광주
	1206	홍길동	광주
	1207	홍길동	광주
	1208	홍길동	광주
	1209	홍길동	광주
	1210	홍길동	광주
•	NULL	NULL	NULL

10. 첫 번째 탭의 쿼리 창에서, 아래 명령을 수행하여 현재 세션의 트랜잭션 격리 수준을 확인합니다.

```
SHOW variables LIKE 'transaction_isolation';
```

Result Grid		
	Variable_name	Value
▶	transaction_isolation	REPEATABLE-READ

11. 두 번째 탭의 쿼리 창에서, 아래 명령을 수행하여 현재 세션의 트랜잭션 격리 수준을 확인합니다.

```
SHOW variables LIKE 'transaction_isolation';
```

Result Grid		
	Variable_name	Value
▶	transaction_isolation	REPEATABLE-READ

READ COMMITTED 트랜잭션 격리 수준

1. 첫 번째 탭의 쿼리 창에서, 아래 명령을 수행하여 현재 세션의 트랜잭션 격리 수준을 READ COMMITTED로 설정합니다.

```
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

2. 첫 번째 탭의 쿼리 창에서, 아래 명령을 수행하여 현재 세션의 트랜잭션 격리 수준을 확인합니다.

```
SHOW variables LIKE 'transaction_isolation';
```

Result Grid		
	Variable_name	Value
▶	transaction_isolation	READ-COMMITTED

3. 첫 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 트랜잭션을 명시적으로 시작합니다.

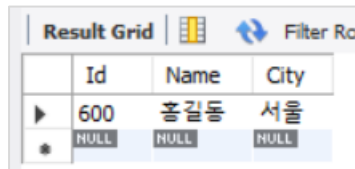
```
START TRANSACTION;
```

4. 첫 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 Id가 500번에서 1000번 사이의 데이터를 업데이트 합니다.

```
UPDATE Members SET
City = '서울'
WHERE Id BETWEEN 500 AND 1000;
```

5. 첫 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 업데이트 중인 데이터를 읽습니다.

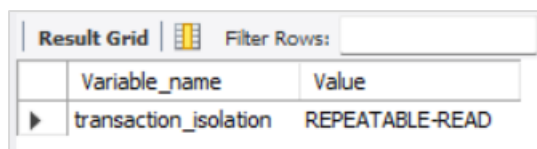
```
SELECT * FROM Members WHERE Id = 600;
```



	Id	Name	City
▶	600	홍길동	서울
★	NULL	NULL	NULL

6. 두 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 트랜잭션 격리 수준을 확인합니다.

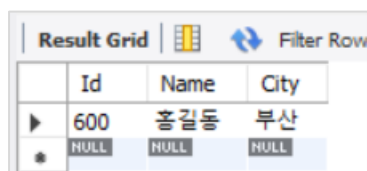
```
SHOW variables LIKE 'transaction_isolation';
```



	Variable_name	Value
▶	transaction_isolation	REPEATABLE-READ

7. 두 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 업데이트 중인 데이터를 읽습니다.

```
SELECT * FROM Members WHERE Id = 600;
```



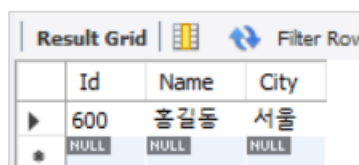
	Id	Name	City
▶	600	홍길동	부산
★	NULL	NULL	NULL

8. 두 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 트랜잭션 격리 수준을 READ UNCOMMITTED로 변경합니다.

```
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

9. 두 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 업데이트 중인 데이터를 읽습니다.

```
SELECT * FROM Members WHERE Id = 600;
```



	Id	Name	City
▶	600	홍길동	서울
★	NULL	NULL	NULL

10. 두 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 트랜잭션 격리 수준을 READ COMMITTED로 변경합니다.

```
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

11. 두 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 업데이트 중인 데이터를 읽습니다.

```
SELECT * FROM Members WHERE Id = 600;
```

Result Grid			
	Id	Name	City
▶	600	홍길동	부산
✱	NULL	NULL	NULL

주의 이 데이터는 실제 테이블의 데이터가 아닌, UNDO 영역의 백업된 레코드에서 가져온 결과입니다. READ COMMITTED 격리 수준에서는 어떤 트랜잭션에서 변경된 내용이 커밋되기 전까지는 다른 트랜잭션에서 변경 내역을 읽을 수 없습니다.

12. 두 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 트랜잭션 격리 수준을 REPEATABLE READ로 변경합니다.

```
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

13. 두 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 업데이트 중인 데이터를 읽습니다.

```
SELECT * FROM Members WHERE Id = 600;
```

Result Grid			
	Id	Name	City
▶	600	홍길동	부산
✱	NULL	NULL	NULL

주의 이 데이터는 실제 테이블의 데이터가 아닌, UNDO 영역의 백업된 레코드에서 가져온 결과입니다. REPEATABLE READ 격리 수준에서는 어떤 트랜잭션에서 변경된 내용이 커밋되기 전까지는 다른 트랜잭션에서 변경 내역을 읽을 수 없습니다.

14. 두 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 트랜잭션 격리 수준을 SERIALIZABLE로 변경합니다.

```
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

15. 두 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 업데이트 중인 데이터를 읽습니다.

```
SELECT * FROM Members WHERE Id = 600;
```

Result Grid			
	Id	Name	City
▶	600	홍길동	부산
✱	NULL	NULL	NULL

16. 첫 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 트랜잭션을 Commit 합니다.

```
COMMIT;
```

17. 두 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 업데이트 중인 데이터를 읽습니다.

```
SELECT * FROM Members WHERE Id = 600;
```

Result Grid			
	Id	Name	City
▶	600	홍길동	서울
*	NULL	NULL	NULL

REPEATABLE READ 격리 수준

1. 첫 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 잠금 수준을 REPEATABLE READ로 변경합니다.

```
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

2. 첫 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 잠금 수준을 확인합니다.

```
SHOW Variables LIKE 'transaction_isolation';
```

Result Grid		
	Variable_name	Value
▶	transaction_isolation	REPEATABLE-READ

3. 첫 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 트랜잭션 수를 확인합니다.

```
SELECT count(1) FROM information_schema.innodb_trx
WHERE trx_mysql_thread_id = CONNECTION_ID();
```

Result Grid	
	count(1)
▶	0

4. 첫 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 트랜잭션을 명시적으로 시작합니다.

```
START TRANSACTION;
```

5. 아래 질의를 수행하여 번호가 1000번인 회원을 읽습니다.

```
SELECT * FROM Members WHERE Id = 1000;
```

Result Grid			
	Id	Name	City
▶	1000	홍길동	서울
*	NULL	NULL	NULL

6. 두 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 회원 번호가 1000인 회원의 이름을 이순신으로 업데이트 합니다.

```
update members set
name = '이순신'
where id = 1000;
```

7. 두 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 회원 번호가 1000인 회원의 데이터를 확인합니다.

```
SELECT * FROM members where id = 1000;
```

Result Grid			
	Id	Name	City
▶	1000	이순신	서울
•	NULL	NULL	NULL

8. 첫 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 회원 번호가 1000인 회원의 데이터를 읽습니다. 두 번째 탭의 업데이트가 반영되지 않습니다.

```
SELECT * FROM Members WHERE Id = 1000;
```

Result Grid			
	Id	Name	City
▶	1000	홍길동	서울
•	NULL	NULL	NULL

9. 첫 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 트랜잭션을 커밋합니다.

```
COMMIT;
```

10. 첫 번째 탭의 쿼리 창에서, 아래 질의를 수행하여 회원 번호가 1000인 회원의 데이터를 읽습니다. 두 번째 탭의 업데이트가 적용된 것을 확인합니다.

```
SELECT * FROM Members WHERE Id = 1000;
```

Result Grid			
	Id	Name	City
▶	1000	홍길동	서울
•	NULL	NULL	NULL

SERIALIZABLE 격리 수준

1. 첫 번째 탭의 쿼리 창에서, 아래 명령을 수행하여 트랜잭션 격리 수준을 SERIALIZABLE로 변경합니다.

```
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

2. 첫 번째 탭의 쿼리 창에서 아래 명령을 수행하여 트랜잭션 격리 수준을 확인합니다.

```
SHOW Variables LIKE 'transaction_isolation';
```

Result Grid		
	Variable_name	Value
▶	transaction_isolation	SERIALIZABLE

3. 첫 번째 탭의 쿼리 창에서 아래 명령을 수행하여 수행중인 트랜잭션의 수를 확인합니다.

```
SELECT count(1) FROM information_schema.innodb_trx
WHERE trx_mysql_thread_id = CONNECTION_ID();
```


Result Grid	
	count(1)
▶	0

4. 첫 번째 탭의 쿼리 창에서, 아래 명령을 수행하여 트랜잭션을 명시적으로 시작합니다.

```
START TRANSACTION;
```

5. 첫 번째 탭의 쿼리 창에서, 아래 명령을 수행하여 광주에 사는 회원의 수를 확인합니다.

```
SELECT COUNT(*) FROM Members WHERE City = '광주';
```

Result Grid	
	COUNT(*)
▶	500

6. 두 번째 탭의 쿼리 창에서, 아래 명령을 수행하여 광주에 사는 회원을 추가합니다.

```
INSERT INTO Members (city) VALUES('광주');
```

상태를 확인합니다.

4	13	22:32:25	INSERT INTO Members (city) VALUES('광주')	Running...
---	----	----------	---	------------

잠시 후, 쿼리가 실패하는 것을 확인합니다.

✖	12	22:30:46	INSERT INTO Members (city) VALUES('광주')	Error Code: 2013. Lost connection to MySQL server during query
---	----	----------	---	--

7. 두 번째 탭의 쿼리 창에서, 아래 명령을 수행하여 광주에 사는 회원을 추가합니다.

```
INSERT INTO Members (city) VALUES('광주');
```

8. 쿼리가 수행되는 도중, 첫 번째 탭의 쿼리 창에서 아래 명령을 수행하여 트랜잭션을 커밋합니다.

```
COMMIT;
```

9. 두 번째 탭에서 쿼리가 성공적으로 수행되었음을 확인합니다.

✔	13	22:32:25	INSERT INTO Members (city) VALUES('광주')	1 row(s) affected
---	----	----------	---	-------------------

10. 첫 번째 탭의 쿼리 창에서, 광주에 사는 회원의 수를 확인합니다.

```
SELECT COUNT(*) FROM Members WHERE City = '광주';
```

Result Grid	
	COUNT(*)
▶	501