COMP90050: Advanced Database Systems
Semester 2, 2024
Lab 2b: Intention Locking

In this lab session, we delve into Intention Locks, specifically IS and IX locks.

I (Intention) Lock:

For this section, we will be looking at row level locks for specific S and X locks while the relevant table will be under an IS or IX lock as necessary. Generally, databases manage Intention locks themselves including coordinating the sequence of granting Intention locks at coarse granularities with specific locks at fine grained granularities. To enable us to see some of these functionalities more easily, we are going to explicitly setup Intention locks using SQL.

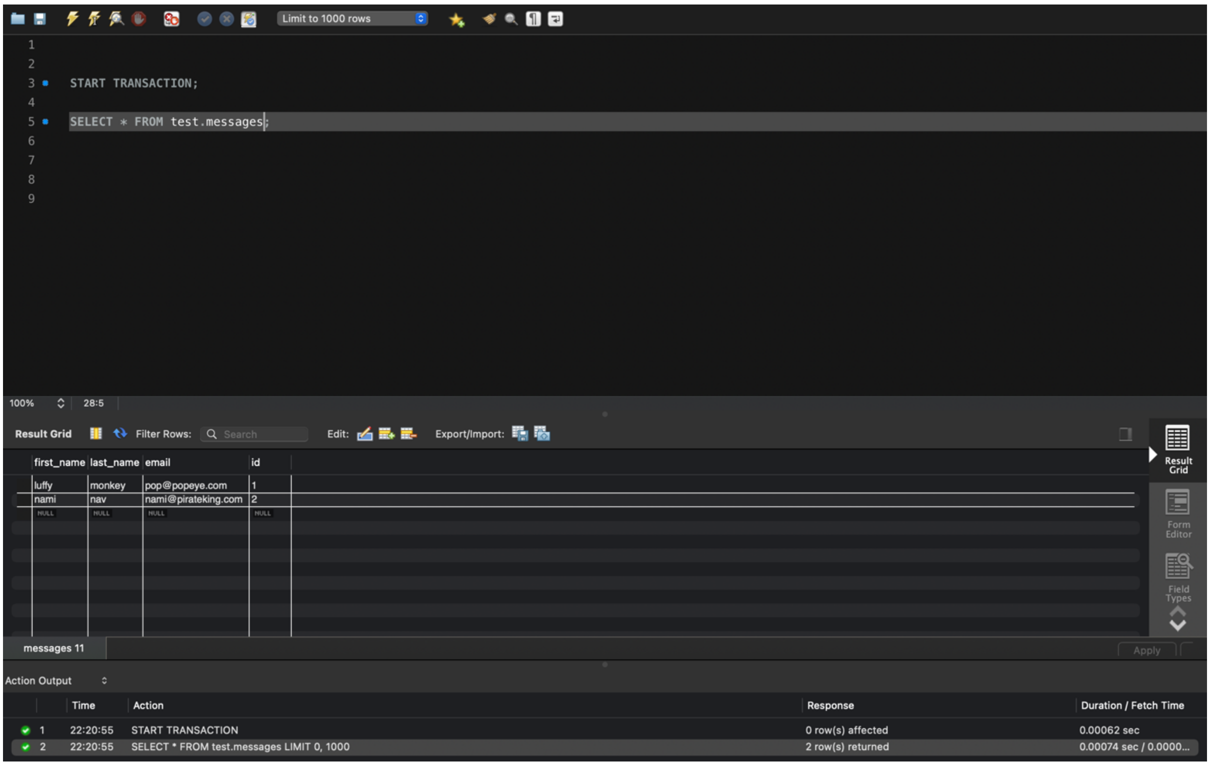
IS Lock:

Note: As part of this Lab, we use the same database table as used in Lab 2a.

The format to explicitly request for an IS lock is as follows:

SELECT …. LOCK IN SHARE MODE;

For this process, we are going to start a transaction to ensure that we retain the lock till we complete the transaction, i.e. till we either COMMIT or ROLLBACK.



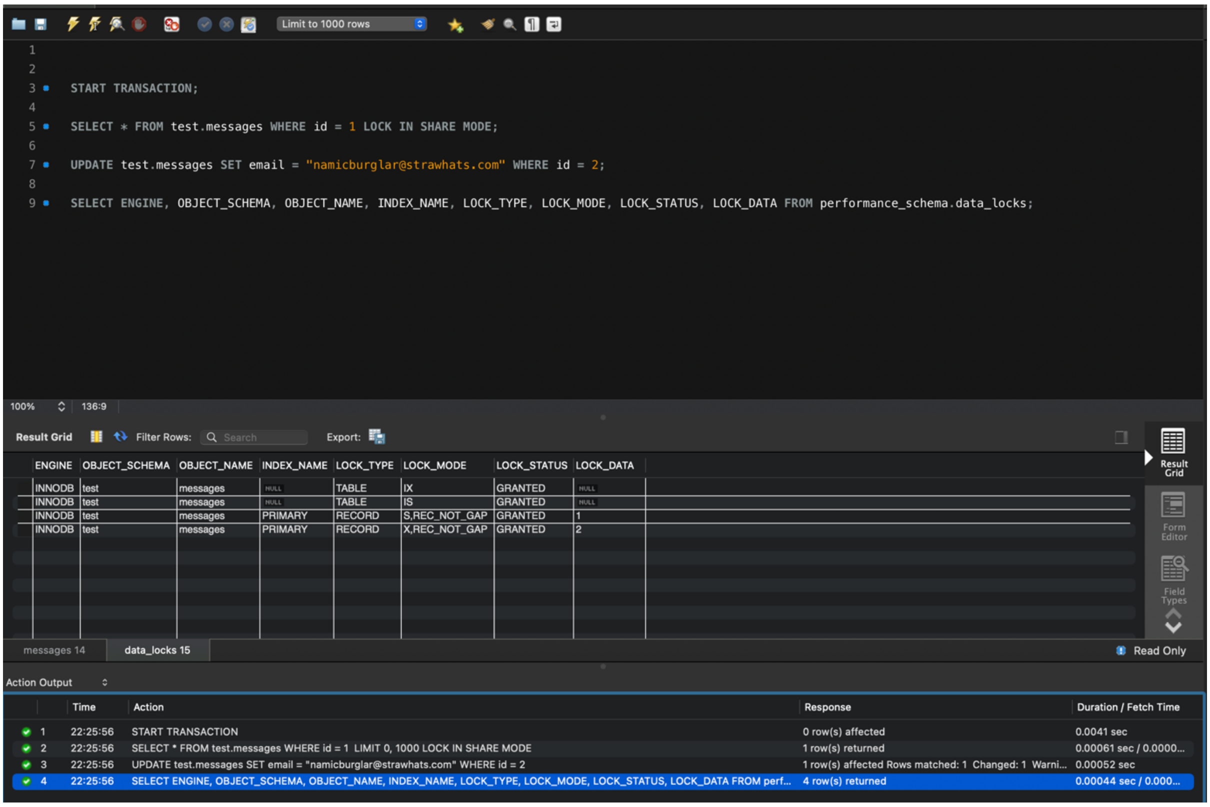
```
START TRANSACTION;

SELECT * FROM test.messages;
```

first_name	last_name	email	id
luffy	monkey	pop@popeye.com	1
nami	nav	nami@pirateking.com	2
NULL	NULL	NULL	NULL

	Time	Action	Response	Duration / Fetch Time
1	22:20:55	START TRANSACTION	0 row(s) affected	0.00062 sec
2	22:20:55	SELECT * FROM test.messages LIMIT 0, 1000	2 row(s) returned	0.00074 sec / 0.0000...

We can observe that there are two rows in the test.messages table at the moment. We get an IS lock on the test.messages table with a specific lock (Shared lock) on a particular row in that table, with row id = 1. Simultaneously, we also try to update one of the columns (email_id) of row with id = 2. When we run these queries in a transaction, the type of locks are held onto by the transaction till we either COMMIT or ROLLBACK. To observe the type of Locks held on specific resources/hierarchy of the database, we can run a SELECT query on the performance_schema.data_locks table, which provides additional context around type of locks currently held on specific resources.

Further information on the performance_schema.data_locks table can be found here: https://dev.mysql.com/doc/refman/8.0/en/performance-schema-data-locks-table.html

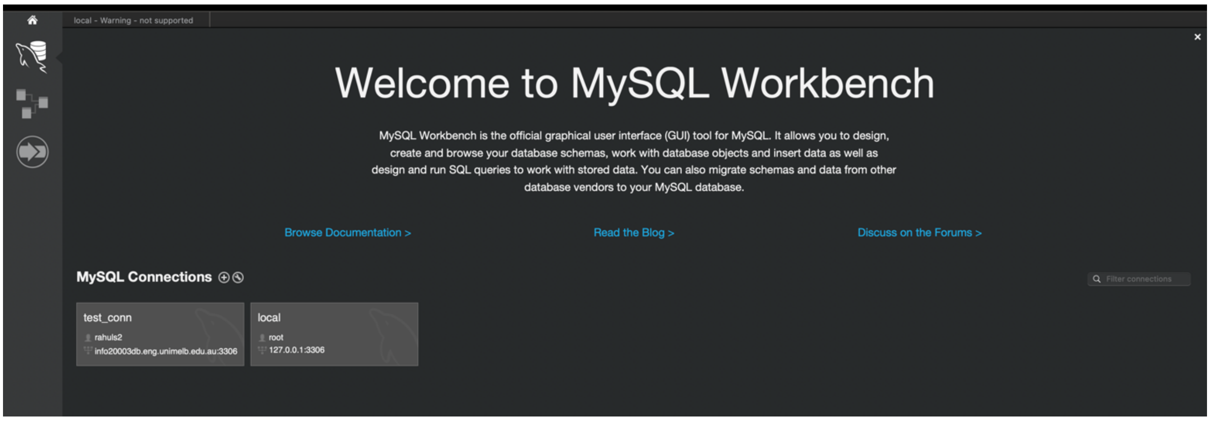


```


START TRANSACTION;

SELECT * FROM test.messages WHERE id = 1 LOCK IN SHARE MODE;

UPDATE test.messages SET email = "namicburglar@strawhats.com" WHERE id = 2;

SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM performance_schema.data_locks;
```

ENGINE	OBJECT_SCHEMA	OBJECT_NAME	INDEX_NAME	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
INNODB	test	messages	NULL	TABLE	IX	GRANTED	NULL
INNODB	test	messages	NULL	TABLE	IS	GRANTED	NULL
INNODB	test	messages	PRIMARY	RECORD	S,REC_NOT_GAP	GRANTED	1
INNODB	test	messages	PRIMARY	RECORD	X,REC_NOT_GAP	GRANTED	2

		Time	Action	Response	Duration / Fetch Time
✓	1	22:25:56	START TRANSACTION	0 row(s) affected	0.0041 sec
✓	2	22:25:56	SELECT * FROM test.messages WHERE id = 1 LIMIT 0, 1000 LOCK IN SHARE MODE	1 row(s) returned	0.00061 sec / 0.0000...
✓	3	22:25:56	UPDATE test.messages SET email = "namicburglar@strawhats.com" WHERE id = 2	1 row(s) affected Rows matched: 1 Changed: 1 Warni...	0.00052 sec
✓	4	22:25:56	SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM perf...	4 row(s) returned	0.00044 sec / 0.000...

We can see that the current session holds an IS and IX lock on the test.messages table, with a specific S lock on row 1 and X lock on row 2. We will now test how these locks persist and work across different transactions. The way we do this is by opening a new Session via the MySQL Workbench home page as follows:

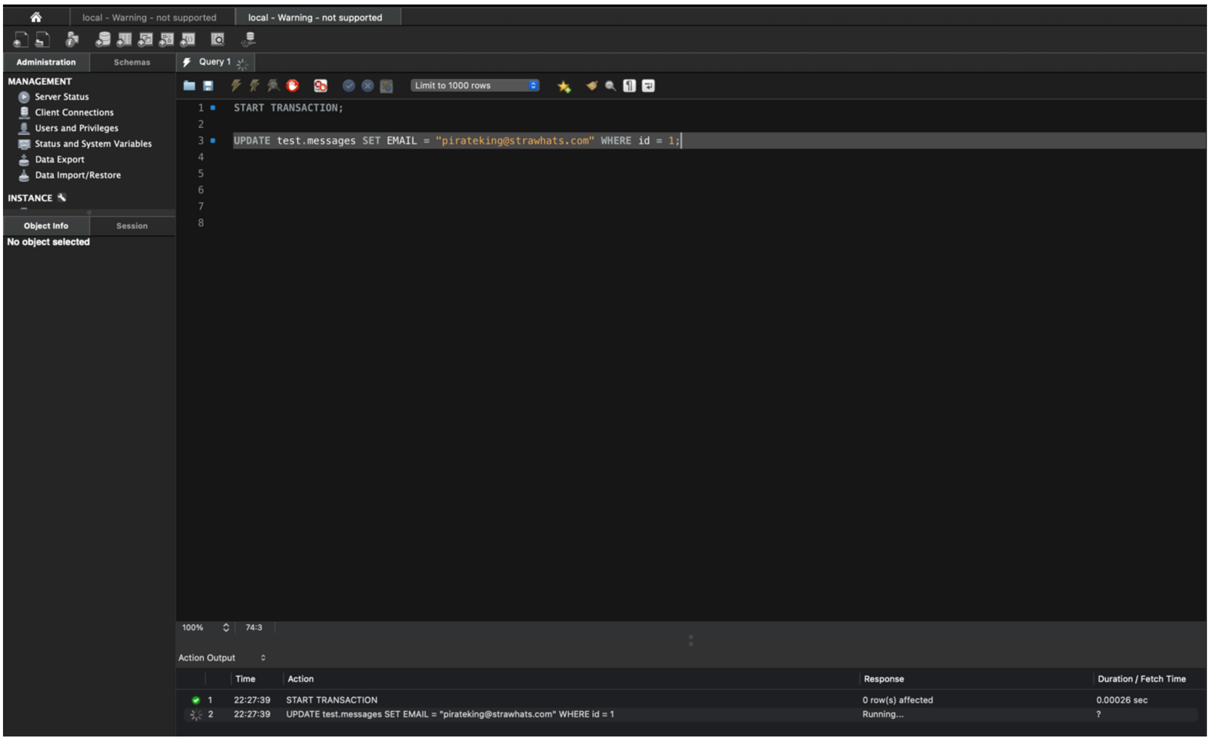


MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.

test_conn
rahuls2
info20003db.eng.unimelb.edu.au:3306

local
root
127.0.0.1:3306

Click on your Database connection to open a new SQL editor tab which will open a new Session with your MySQL server. In your new session, start a transaction and try to update row with id = 1 and execute the queries.

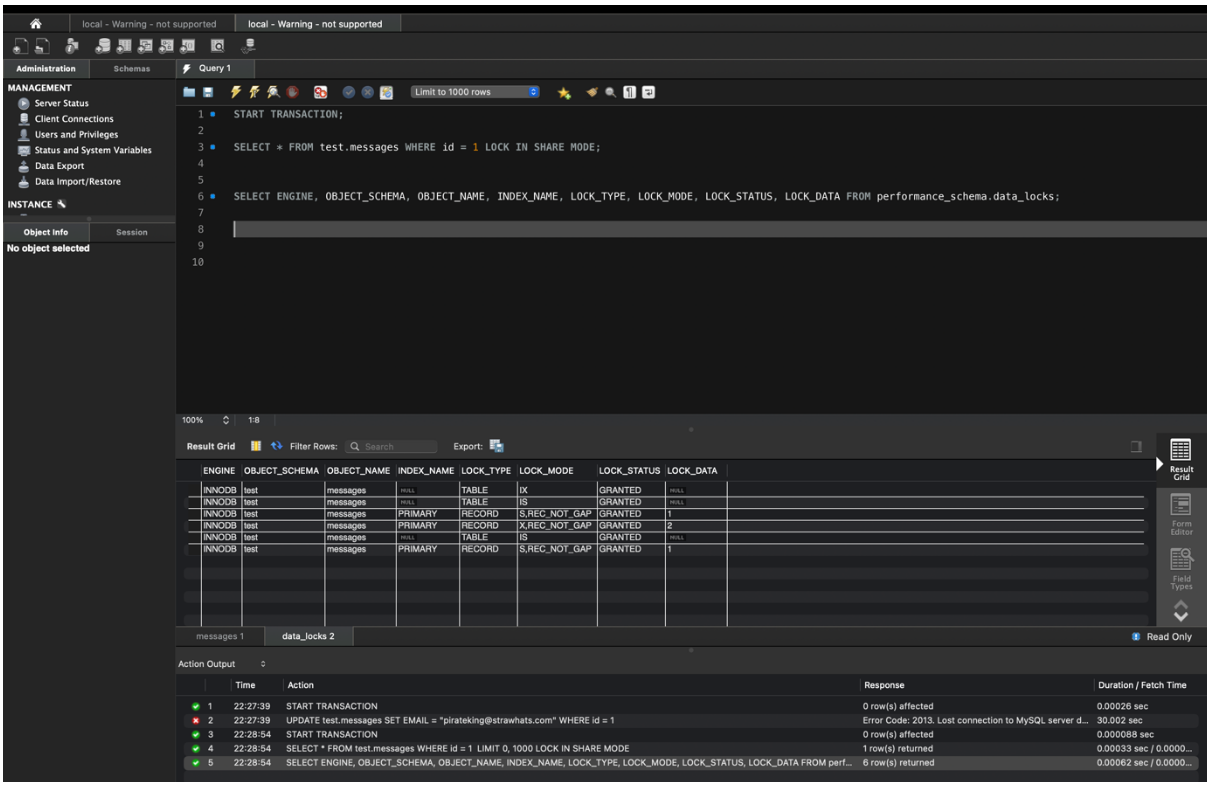


```
START TRANSACTION;

UPDATE test.messages SET EMAIL = "pirateking@strawhats.com" WHERE id = 1;
```

```
     Time      Action                                                            Response            Duration / Fetch Time
  1  22:27:39  START TRANSACTION                                                 0 row(s) affected   0.00026 sec
  2  22:27:39  UPDATE test.messages SET EMAIL = "pirateking@strawhats.com" WHERE id = 1   Running...   ?
```

You will observe that the UPDATE query hangs and ends up with an error that the query cannot be executed. This occurs because the previous Session holds a S lock on row with id = 1, which is not compatible with an UPDATE query that tries to get an X lock on it.

However, if we try to get an IS lock on the test.messages table with a specific S lock on row with id = 1, then we will observe that this lock will be granted to the transaction in the new Session as follows:



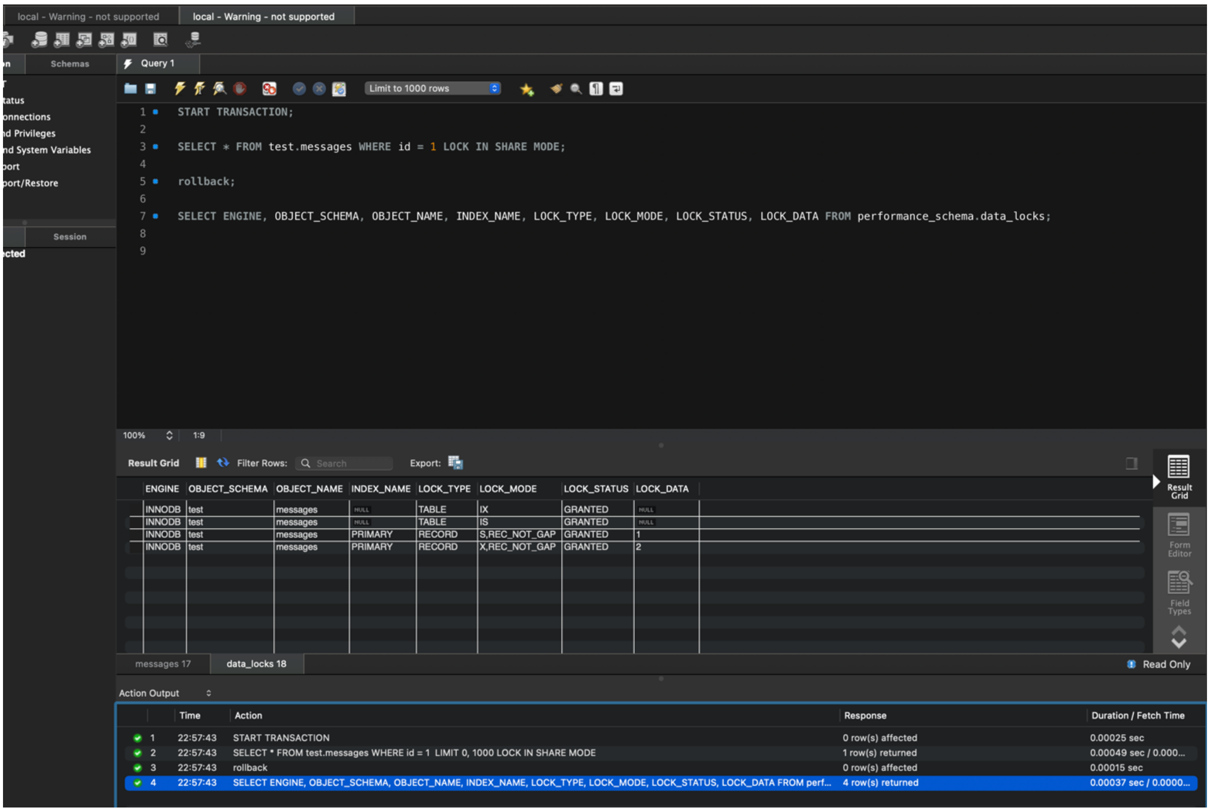
```
START TRANSACTION;

SELECT * FROM test.messages WHERE id = 1 LOCK IN SHARE MODE;


SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM performance_schema.data_locks;
```

ENGINE	OBJECT_SCHEMA	OBJECT_NAME	INDEX_NAME	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
INNODB	test	messages	NULL	TABLE	IX	GRANTED	NULL
INNODB	test	messages	NULL	TABLE	IS	GRANTED	NULL
INNODB	test	messages	PRIMARY	RECORD	S,REC_NOT_GAP	GRANTED	1
INNODB	test	messages	PRIMARY	RECORD	X,REC_NOT_GAP	GRANTED	2
INNODB	test	messages	NULL	TABLE	IS	GRANTED	NULL
INNODB	test	messages	PRIMARY	RECORD	S,REC_NOT_GAP	GRANTED	1

```
     Time      Action                                                                                                   Response                                            Duration / Fetch Time
  1  22:27:39  START TRANSACTION                                                                                        0 row(s) affected                                   0.00026 sec
  2  22:27:39  UPDATE test.messages SET EMAIL = "pirateking@strawhats.com" WHERE id = 1                                 Error Code: 2013. Lost connection to MySQL server d...   30.002 sec
  3  22:28:54  START TRANSACTION                                                                                        0 row(s) affected                                   0.000088 sec
  4  22:28:54  SELECT * FROM test.messages WHERE id = 1  LIMIT 0, 1000 LOCK IN SHARE MODE                               1 row(s) returned                                   0.00033 sec / 0.0000...
  5  22:28:54  SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM perf...   6 row(s) returned   0.00062 sec / 0.0000...
```

Since the read based Intention and specific locks are compatible, we observe that the new sessions transaction will be granted the IS and S lock.

If we rollback the new transaction then we will observe that the IS and S locks held by this transaction will be released. Only the first session will be maintaining its locks now.

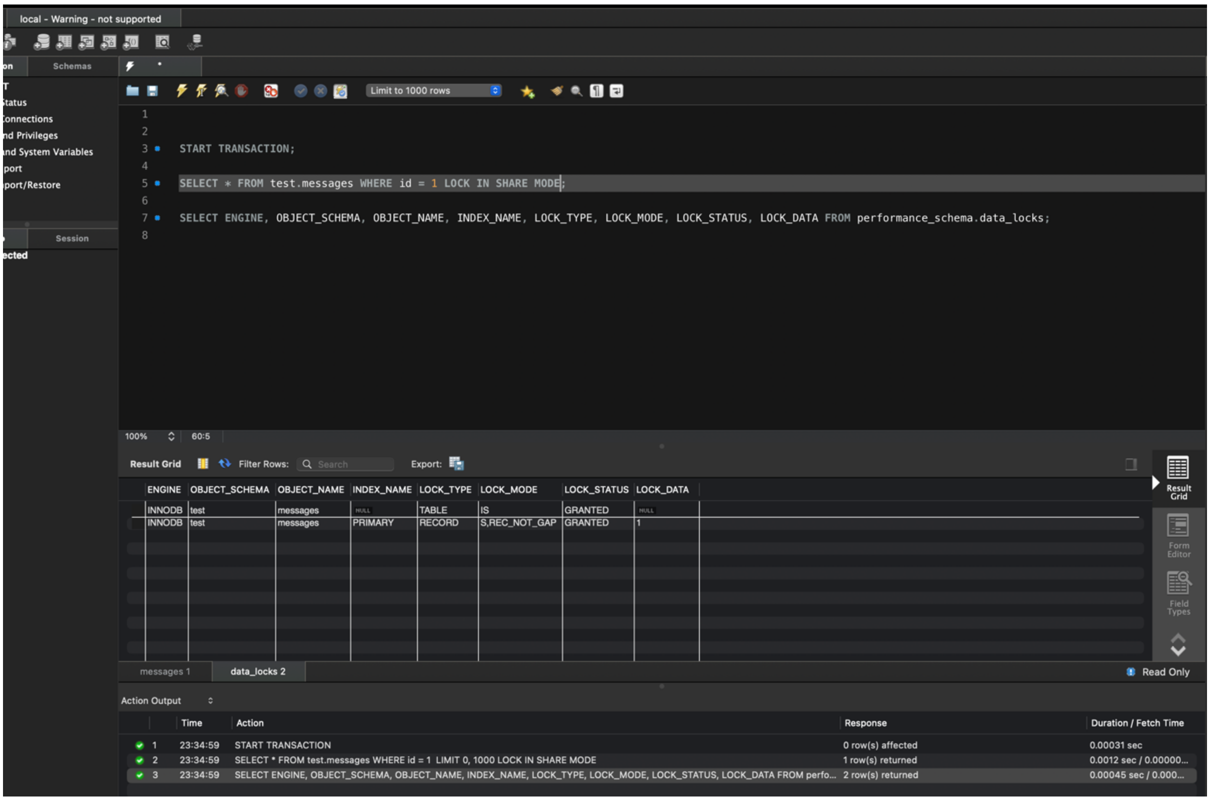


```
START TRANSACTION;

SELECT * FROM test.messages WHERE id = 1 LOCK IN SHARE MODE;

rollback;

SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM performance_schema.data_locks;
```

ENGINE	OBJECT_SCHEMA	OBJECT_NAME	INDEX_NAME	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
INNODB	test	messages	NULL	TABLE	IX	GRANTED	NULL
INNODB	test	messages	NULL	TABLE	IS	GRANTED	NULL
INNODB	test	messages	PRIMARY	RECORD	S,REC_NOT_GAP	GRANTED	1
INNODB	test	messages	PRIMARY	RECORD	X,REC_NOT_GAP	GRANTED	2

	Time	Action	Response	Duration / Fetch Time
1	22:57:43	START TRANSACTION	0 row(s) affected	0.00025 sec
2	22:57:43	SELECT * FROM test.messages WHERE id = 1 LIMIT 0, 1000 LOCK IN SHARE MODE	1 row(s) returned	0.00049 sec / 0.000...
3	22:57:43	rollback	0 row(s) affected	0.00015 sec
4	22:57:43	SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM perf...	4 row(s) returned	0.00037 sec / 0.0000...

We can further understand whether an IS lock is compatible with an X lock by getting an IS lock on the test.messages table in Session 1 as shown below.



```
START TRANSACTION;

SELECT * FROM test.messages WHERE id = 1 LOCK IN SHARE MODE;

SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM performance_schema.data_locks;
```

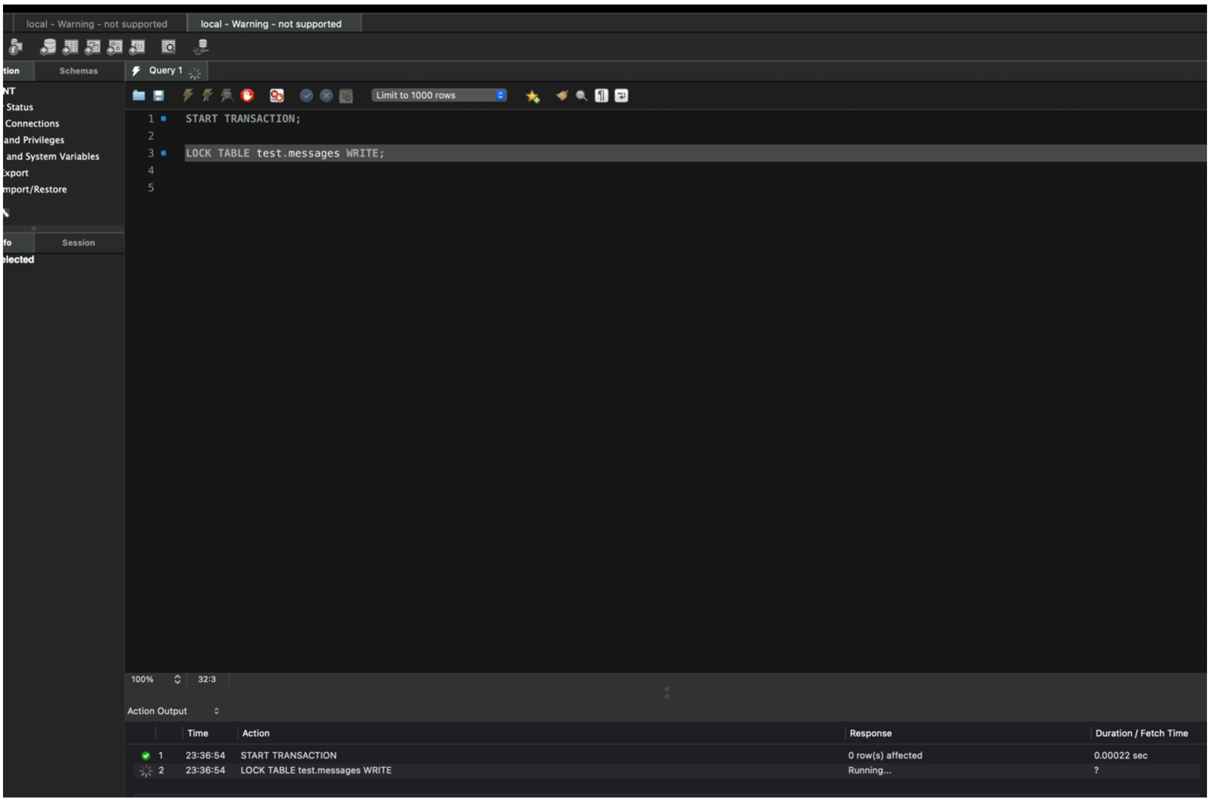
ENGINE	OBJECT_SCHEMA	OBJECT_NAME	INDEX_NAME	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
INNODB	test	messages	NULL	TABLE	IS	GRANTED	NULL
INNODB	test	messages	PRIMARY	RECORD	S,REC_NOT_GAP	GRANTED	1

messages 1 | data_locks 2

Action Output

```
   Time      Action                                                                                                    Response            Duration / Fetch Time
1  23:34:59  START TRANSACTION                                                                                         0 row(s) affected   0.00031 sec
2  23:34:59  SELECT * FROM test.messages WHERE id = 1  LIMIT 0, 1000 LOCK IN SHARE MODE                                1 row(s) returned   0.0012 sec / 0.00000...
3  23:34:59  SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM perfo...  2 row(s) returned   0.00045 sec / 0.000...
```

Following this, in Session 2 we try to get a table level X lock using the LOCK TABLE… WRITE query as seen in Lab 2a.


local - Warning - not supported
local - Warning - not supported
Schemas
Query 1
Session
Limit to 1000 rows

```
START TRANSACTION;

LOCK TABLE test.messages WRITE;
```

Action Output

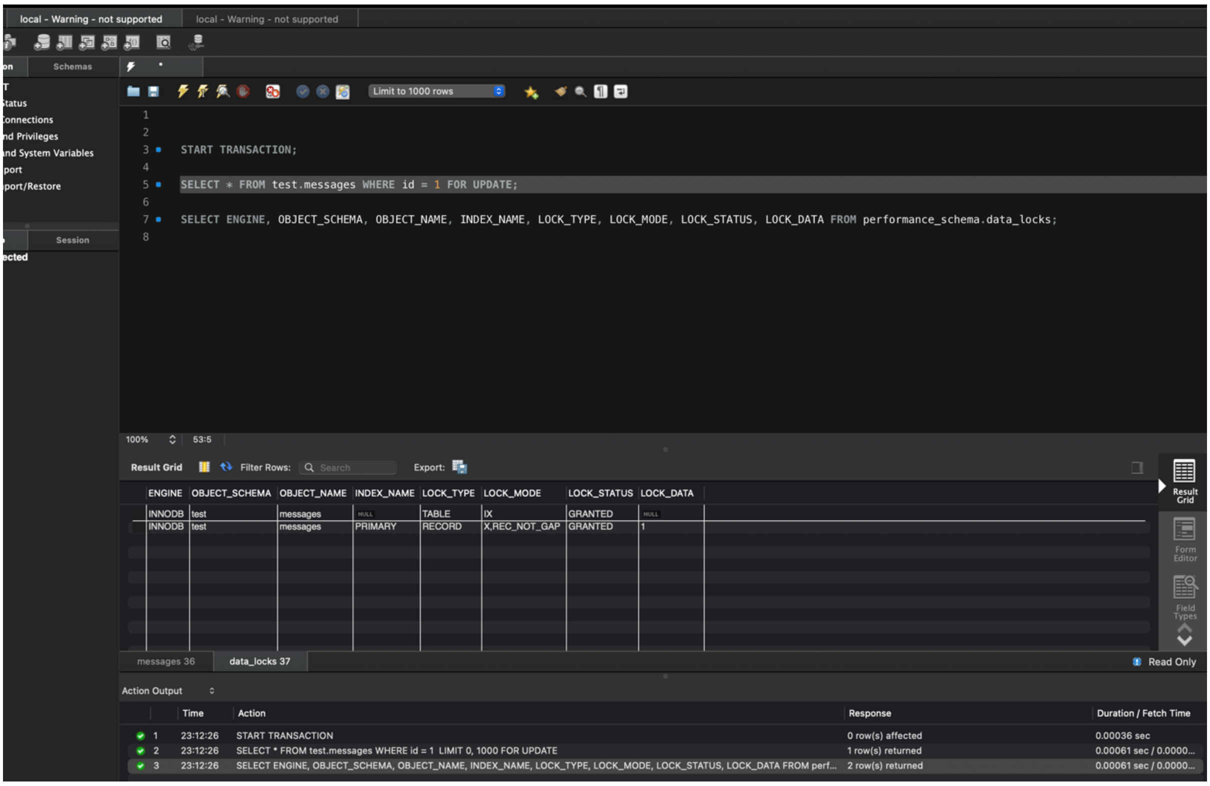
```
   Time      Action                          Response            Duration / Fetch Time
1  23:36:54  START TRANSACTION               0 row(s) affected   0.00022 sec
2  23:36:54  LOCK TABLE test.messages WRITE  Running...          ?
```

When we try to get the X lock on test.messages (currently under an IS lock by Session 1) from Session 2, we observe that the request for an X lock will be stuck waiting and will eventually time out. This is because IS locks are not compatible with an X lock. If we try the reverse (trying to get an IS lock on a table with an X lock already), we will observe the same issue.

IX Locks:

Similarly, we can retrieve an IX lock explicitly using the format:

SELECT …. FOR UPDATE;

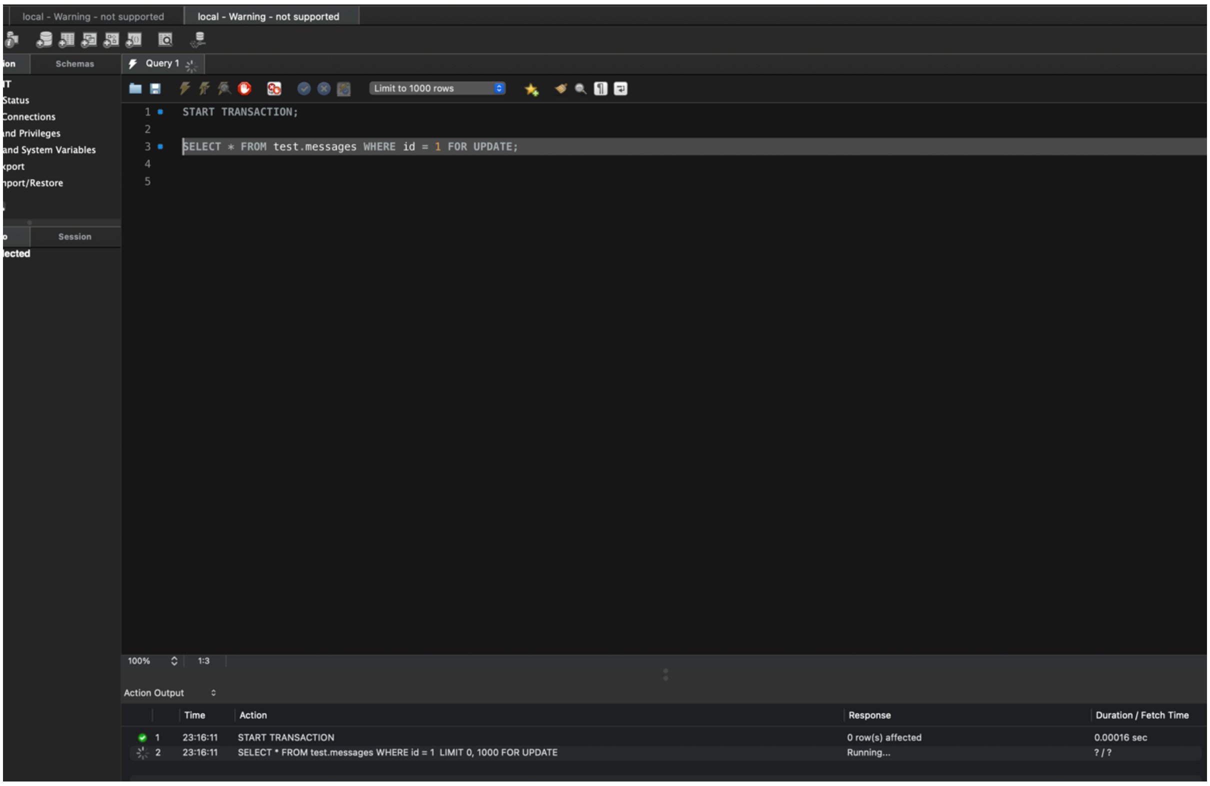


```
START TRANSACTION;

SELECT * FROM test.messages WHERE id = 1 FOR UPDATE;

SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM performance_schema.data_locks;
```

ENGINE	OBJECT_SCHEMA	OBJECT_NAME	INDEX_NAME	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
INNODB	test	messages	NULL	TABLE	IX	GRANTED	NULL
INNODB	test	messages	PRIMARY	RECORD	X,REC_NOT_GAP	GRANTED	1

	Time	Action	Response	Duration / Fetch Time
1	23:12:26	START TRANSACTION	0 row(s) affected	0.00036 sec
2	23:12:26	SELECT * FROM test.messages WHERE id = 1 LIMIT 0, 1000 FOR UPDATE	1 row(s) returned	0.00061 sec / 0.0000...
3	23:12:26	SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM perf...	2 row(s) returned	0.00061 sec / 0.0000...

In the above sequence, we have acquired an IX lock on the test.messages table and a specific X lock on the row with id = 1 in the first session. If we open a new session and try to get an X lock on the row with id = 1, the X lock would be in the waiting state and eventually timeout since only 1 session would be allowed to have an X lock on a data object (in our case, a row of data) at one time. This can be seen in the below screenshot:

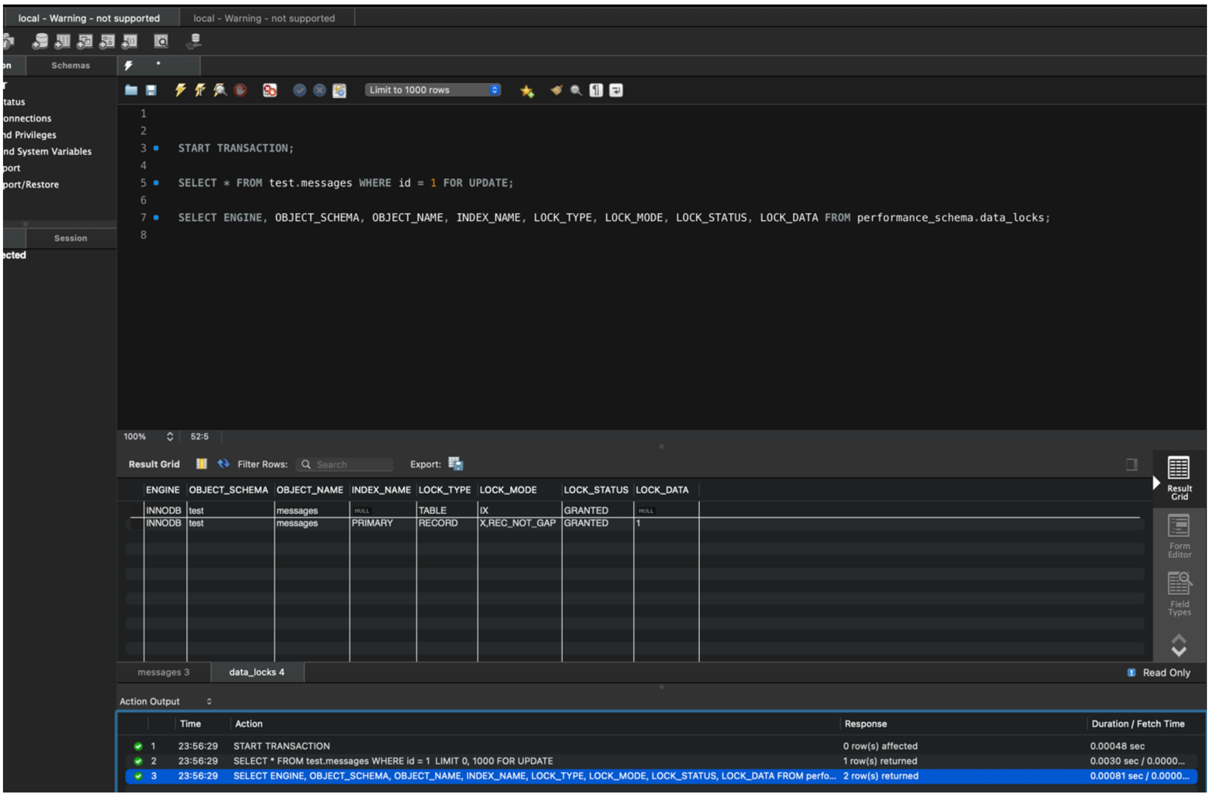


```sql
START TRANSACTION;

SELECT * FROM test.messages WHERE id = 1 FOR UPDATE;
```

		Time	Action	Response	Duration / Fetch Time
✓	1	23:16:11	START TRANSACTION	0 row(s) affected	0.00016 sec
	2	23:16:11	SELECT * FROM test.messages WHERE id = 1 LIMIT 0, 1000 FOR UPDATE	Running...	? / ?

Similarly, if we try to get an IS lock on the same table with a specific S lock on the same row, the query will still timeout despite IX and IS locks being compatible. This is because S locks and X locks on the same resource are not compatible. However, if we try to get a lock on a separate row (id other than 1), the lock would be granted.

We can verify whether an IX lock is compatible with another IX lock by first getting an IX lock on the test.messages table from Session 1.

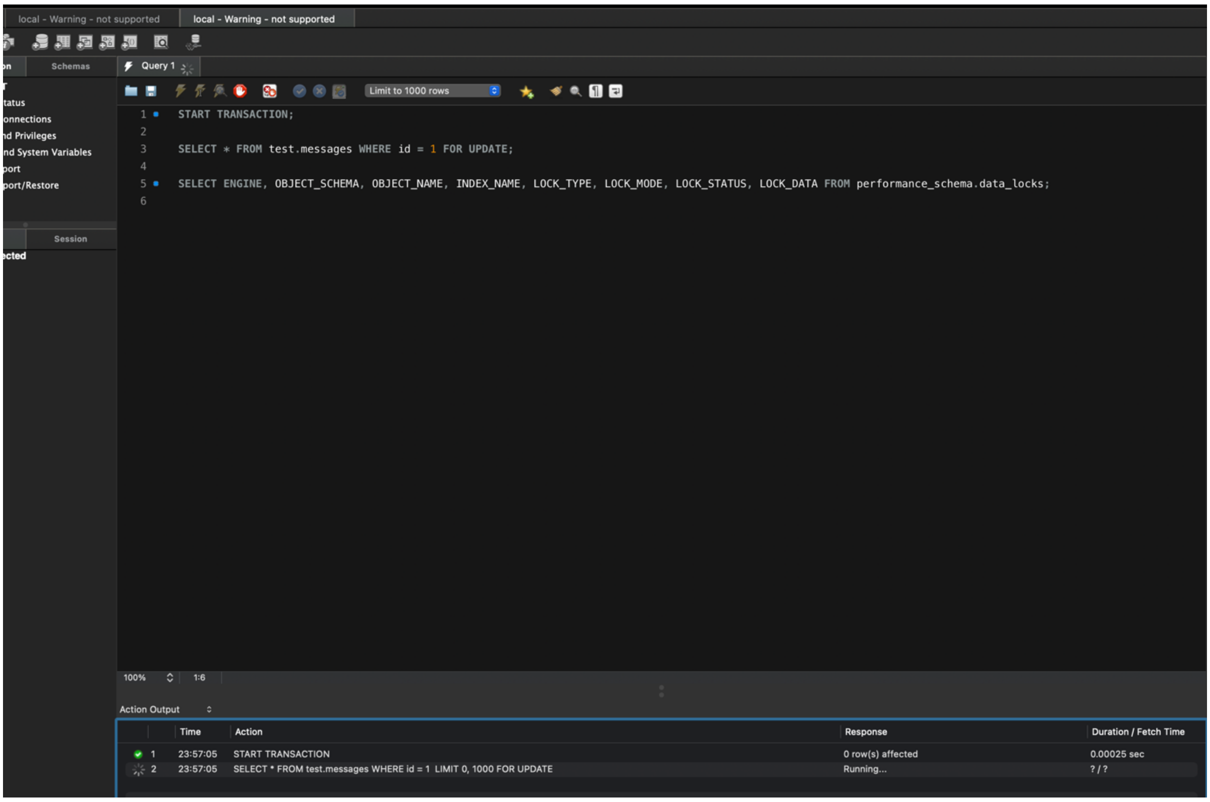


```
START TRANSACTION;

SELECT * FROM test.messages WHERE id = 1 FOR UPDATE;

SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM performance_schema.data_locks;
```

ENGINE	OBJECT_SCHEMA	OBJECT_NAME	INDEX_NAME	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
INNODB	test	messages	NULL	TABLE	IX	GRANTED	NULL
INNODB	test	messages	PRIMARY	RECORD	X,REC_NOT_GAP	GRANTED	1

	Time	Action	Response	Duration / Fetch Time
1	23:56:29	START TRANSACTION	0 row(s) affected	0.00048 sec
2	23:56:29	SELECT * FROM test.messages WHERE id = 1 LIMIT 0, 1000 FOR UPDATE	1 row(s) returned	0.0030 sec / 0.0000...
3	23:56:29	SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM perfo...	2 row(s) returned	0.00081 sec / 0.0000...

As seen in the output of the performance_schema.data_locks table, we have gotten an IX lock on the test.messages table from Session 1. Next, we try to get an IX lock on the test.messages table from Session 2 as shown below.

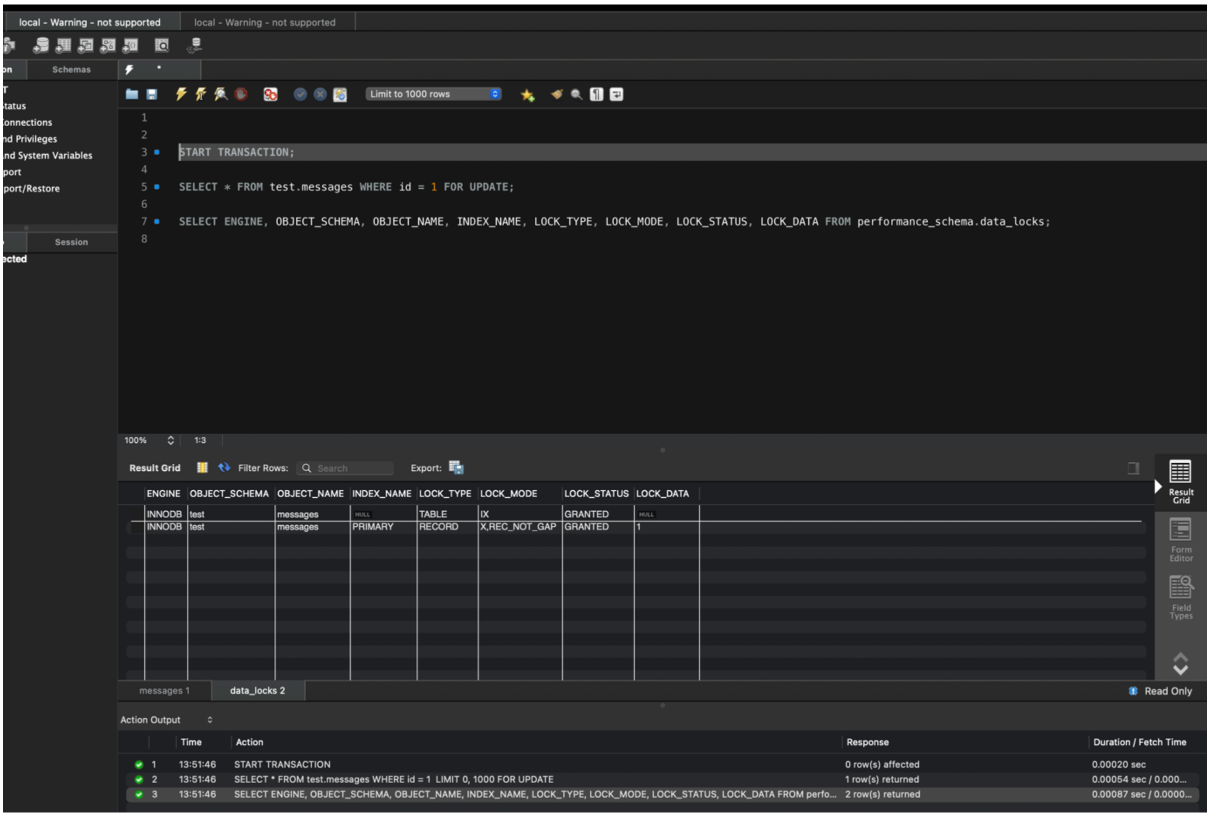


```
START TRANSACTION;

SELECT * FROM test.messages WHERE id = 1 FOR UPDATE;

SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM performance_schema.data_locks;
```

	Time	Action	Response	Duration / Fetch Time
1	23:57:05	START TRANSACTION	0 row(s) affected	0.00025 sec
2	23:57:05	SELECT * FROM test.messages WHERE id = 1 LIMIT 0, 1000 FOR UPDATE	Running...	? / ?

We can observe in the logs for Session 2 that the request for getting a specific X lock on the same row that the transaction in Session 1 holds will be blocked, even if the IX lock in Session 1 is compatible with the IX lock in Session 2 at the table level. This is because an X lock from one transaction is not compatible with an X lock requested by another transaction.

Next we can verify whether an IX of one transaction is compatible with an IX lock from another transaction with specific locks being granted at different database resources at lower level granularities. For this, we first retrieve a specific X lock on the row with id = 1 and an IX lock on the test.messages table as seen below.

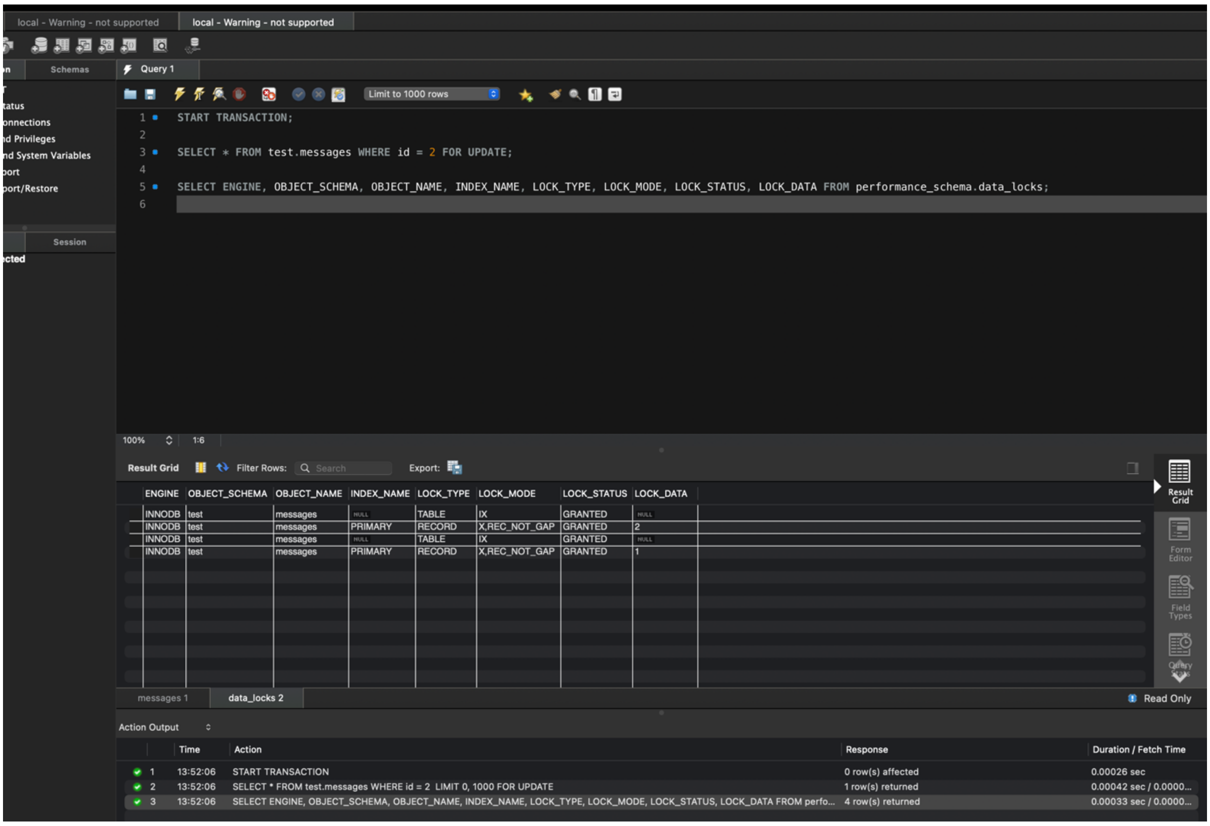


```
START TRANSACTION;

SELECT * FROM test.messages WHERE id = 1 FOR UPDATE;

SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM performance_schema.data_locks;
```

ENGINE	OBJECT_SCHEMA	OBJECT_NAME	INDEX_NAME	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
INNODB	test	messages	NULL	TABLE	IX	GRANTED	NULL
INNODB	test	messages	PRIMARY	RECORD	X,REC_NOT_GAP	GRANTED	1

```
Action Output
    Time      Action                                                                                                  Response             Duration / Fetch Time
1   13:51:46  START TRANSACTION                                                                                       0 row(s) affected    0.00020 sec
2   13:51:46  SELECT * FROM test.messages WHERE id = 1  LIMIT 0, 1000 FOR UPDATE                                      1 row(s) returned    0.00054 sec / 0.000...
3   13:51:46  SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM perfo...  2 row(s) returned    0.00087 sec / 0.0000...
```

In another session, we try to retrieve a specific X lock on a row with id = 2 with an IX lock on the test.messages table as seen below.



```sql
START TRANSACTION;

SELECT * FROM test.messages WHERE id = 2 FOR UPDATE;

SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM performance_schema.data_locks;
```

ENGINE	OBJECT_SCHEMA	OBJECT_NAME	INDEX_NAME	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
INNODB	test	messages	NULL	TABLE	IX	GRANTED	NULL
INNODB	test	messages	PRIMARY	RECORD	X,REC_NOT_GAP	GRANTED	2
INNODB	test	messages	NULL	TABLE	IX	GRANTED	NULL
INNODB	test	messages	PRIMARY	RECORD	X,REC_NOT_GAP	GRANTED	1

	Time	Action	Response	Duration / Fetch Time
1	13:52:06	START TRANSACTION	0 row(s) affected	0.00026 sec
2	13:52:06	SELECT * FROM test.messages WHERE id = 2 LIMIT 0, 1000 FOR UPDATE	1 row(s) returned	0.00042 sec / 0.0000...
3	13:52:06	SELECT ENGINE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM perfo...	4 row(s) returned	0.00033 sec / 0.0000...

We can observe that both transactions are able to get a specific X lock on different rows with IX locks on the test.messages table. This is because IX locks are compatible with requests for an IX lock from other transactions.

There are other types of locks as well in MySQL which can be read about here: https://dev.mysql.com/doc/refman/8.0/en/innodb-locking.html

Try running some of the lock types described in the link above while running READ and WRITE based operations.