

1. トランザクションとは

トランザクションとは、データベース内のデータを更新(INSERT、UPDATE、DELETE)する一連の作業単位のことを言います。
その一連の作業単位は、「データの整合性を為の最低限の処理単位」においてグループ化されます。

例) 社員の給与表

名前	給与額	残業代	合計支給額
山田太郎	100000	0	100000

(処理 1)

ユーザーA は残業代の計算及び合計支給額の計算を担当しているとします。
山田太郎さんの今月の残業代は 20,000 円でした。

ユーザーA はまず以下の SQL 文を発行します。

```
UPDATE 給与表 SET 残業代 = 20000 WHERE 名前 = '山田太郎' ;
```

名前	給与額	残業代	合計支給額
山田太郎	100000	20000	100000

(処理 2)

ユーザーA は、次には合計支給額を変更します。

```
UPDATE 給与表 SET 合計支給額 = 120000 WHERE 名前 = '山田太郎' ;
```

名前	給与額	残業代	合計支給額
山田太郎	100000	20000	120000

※この二つの作業が正常に行われた状態がひとつのトランザクションとなります。

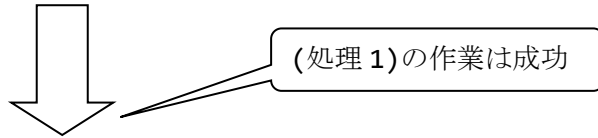
もし、(処理 2)の SQL 文をサーバーへ送信時、回線の異常が発生した場合、
残業代はついているのに合計支給額には反映されていない状態となってしまいます。
この場合、データの「残業代を 0 に戻して再入力」を実行する必要があります。
(※ロールバック)

すなわち、2つの処理〔①残業代更新、②合計支給額〕は、
どちらか片方だけが更新されるのではなく、「両方更新」か「両方更新しない」状態が、
「データの整合性」の観点から正常となります。

ロールバック

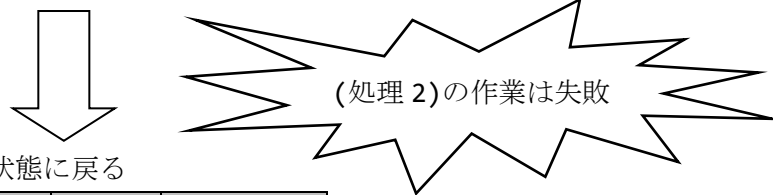
(給与表)

名前	給与額	残業代	合計支給額
山田太郎	100000	0	100000



(処理 1)後の結果

名前	給与額	残業代	合計支給額
山田太郎	100000	20000	100000



(処理 1)の前の状態に戻る

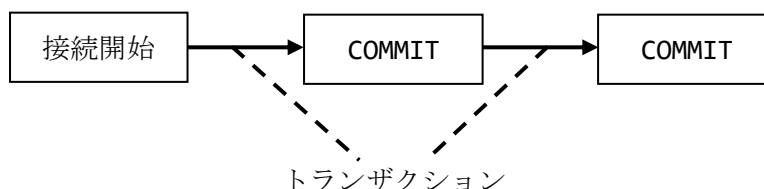
名前	給与額	残業代	合計支給額
山田太郎	100000	0	100000

※トランザクションにおいて、表の内容を変更する SQL 文 [INSERT、UPDATE、DELETE] を実行しただけでは、表の内容は更新されません。
(ファイルの変更を上書き保存しないとファイルの内容が更新されない事と同じです)。

トランザクション (一連の作業) を終了後、**COMMIT** を実行するとデータベースの内容が確定します。
COMMIT は、トランザクション (一連の作業) を表に反映させる命令です。

表の内容を変更している間は、ロールバックセグメントという領域に更新前のデータを保管しています。
変更やエラーの取り消しを行う場合は、ロールバックセグメントの内容を表に戻します。(ROLLBACK)
また **ROLLBACK** のタイミングでロールバックセグメント内の情報を消去します。

MySQL におけるトランザクションは、前回の **COMMIT** もしくは接続開始からスタートします。



2. 「 START TRANSACTION 」、 「 SET AUTOCOMMIT = 0 」

MySQL では、初期設定で自動コミットが有効のため、DML 文を発行した場合すぐ反映されます。

トランザクション処理を行うには、『 START TRANSACTION 』 コマンドを発行します。

または、自動コミットを無効にする 『 SET AUTOCOMMIT = 0 』 コマンドを発行します。

再び、自動コミットが有効には、『 SET AUTOCOMMIT = 1 』 コマンドを発行します。

3. 「 COMMIT 」、 「 ROLLBACK 」

トランザクションを確定するには、『 COMMIT; 』 を発行します。

トランザクションを取り消して、前回の COMMIT 文の直後まで戻すには、『 ROLLBACK; 』 を発行します。

ex

データベースに接続し、以下の操作を実行して内容を確認して下さい。

- ① まず、START TRANSACTION コマンドを入力して下さい。

START TRANSACTION;

- ② DEPT 表の全ての列を表示し内容を確認して下さい。

SELECT * FROM DEPT;

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESERCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

- ③ 一行挿入し、DEPT 表の全ての列を表示し内容を確認して下さい。

INSERT INTO DEPT VALUES(50, 'EDUCATION', 'WASHINGTON');

SELECT * FROM DEPT;

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESERCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	EDUCATION	WASHINGTON

- ④ ロールバックして下さい。

ROLLBACK;

⑤ DEPT 表の全ての列を表示し内容を確認して下さい。

```
SELECT * FROM DEPT;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESERCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

※②の状態に戻っている事が確認できました。

⑥ もう一度③の SQL 文を実行し、DEPT 表の全ての列の内容を確認して下さい。

```
INSERT INTO DEPT VALUES(50, 'EDUCATION', 'WASHINGTON');
```

```
SELECT * FROM DEPT;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESERCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	EDUCATION	WASHINGTON

⑦ コミットして下さい。

```
COMMIT;
```

⑧ ロールバックして下さい

```
ROLLBACK;
```

⑨ DEPT 表の全ての列を表示し、⑥と同じ内容であることを確認して下さい。COMMIT 後に ROLLBACK をしても、②の状態に戻すことはできません。

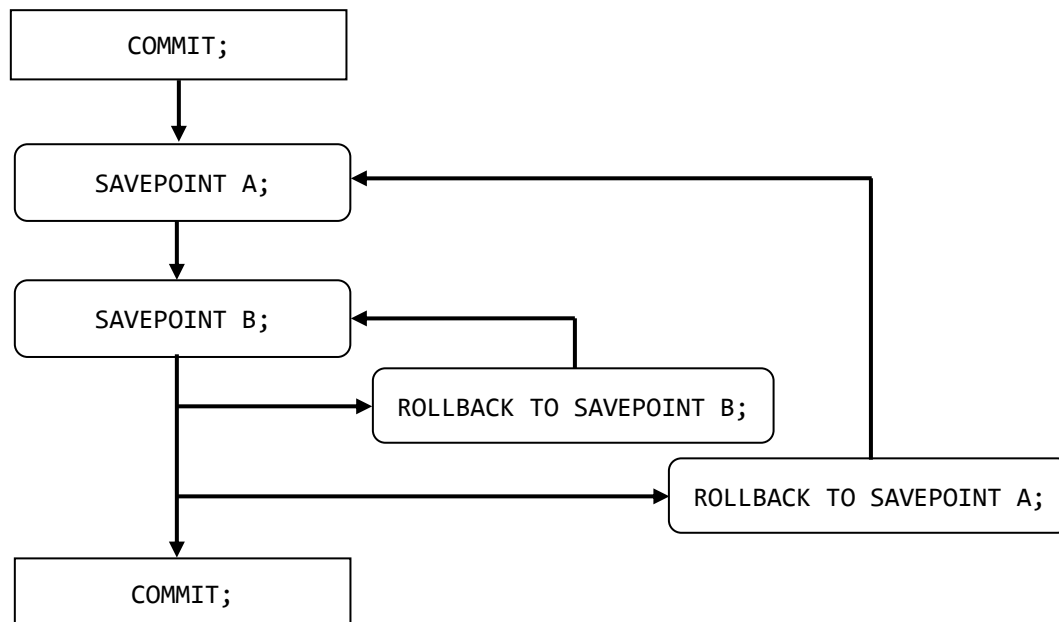
```
SELECT * FROM DEPT;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESERCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	EDUCATION	WASHINGTON

※再度 sql ファイルを使用して、表を元の状態に戻して下さい。

4. セーブポイント

COMMIT と COMMIT の間(トランザクション内)に、いくつかの DML 文が含まれていた時に、何らかの事情でロールバックする必要が出てきた場合、トランザクションは丸ごとロールバックされてしまいます。そこで、長いトランザクションを作る場合には、間にセーブポイントというものを作成しておきます。セーブポイントを作っておくと、ロールバックする際にトランザクションの始めまで戻らなくて済むようになります。



上記の図のように 2 回目の COMMIT をする前に、ROLLBACK をすると、最初の COMMIT までロールバックされます。しかし、途中でセーブポイントを作っておくことで、セーブポイント A やセーブポイント B の位置まで ROLLBACK することができます。

セーブポイントを作るときの構文は以下のとおりです。

SAVEPOINT セーブポイント名 ;

また、セーブポイントまで戻るときの構文は以下のとおりです。

ROLLBACK TO セーブポイント名 ;

ex

データベースに接続し、以下の操作を実行して内容を確認して下さい。

① まず、START TRANSACTION コマンドを入力して下さい。

START TRANSACTION;

② DEPT 表の全ての列を表示し内容を確認して下さい。

SELECT * FROM DEPT;

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESERCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

- ③ DEPTNO が 10 の LOC 列を東京に変更し表の内容を確認して下さい。

```
UPDATE DEPT SET LOC = '東京' WHERE DEPTNO = 10;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	東京
20	RESERCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

- ④ SAVEPOINT SP1 を設定して下さい。

```
SAVEPOINT SP1;
```

- ⑤ DEPTNO が 20 の LOC 列を大阪に変更し表の内容を確認して下さい。

```
UPDATE DEPT SET LOC = '大阪' WHERE DEPTNO = 20;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	東京
20	RESERCH	大阪
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

- ⑥ セーブポイント SP1 にロールバックして、③と同じ内容であることを確認して下さい。

```
ROLLBACK TO SP1;
```

```
SELECT * FROM DEPT;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	東京
20	RESERCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

※再度 sql ファイルを使用して、表を元の状態に戻しておいて下さい。

5. 暗黙のコミット

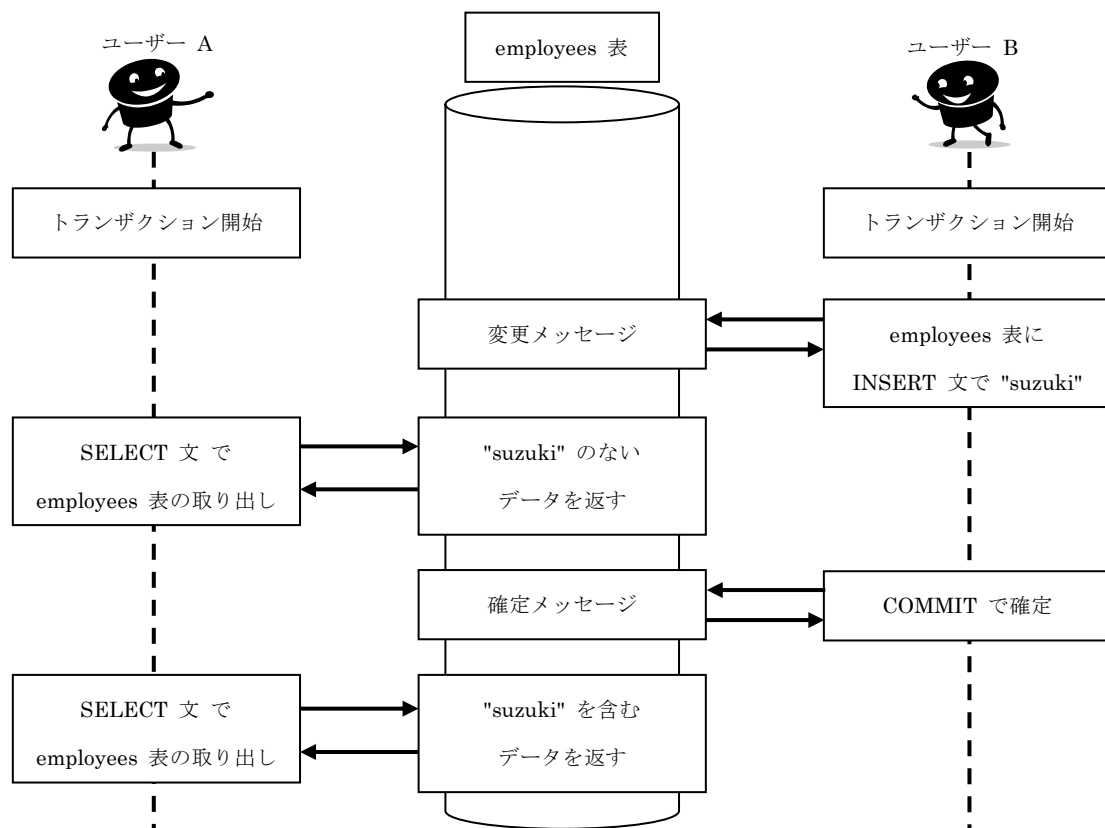
トランザクションでは、**COMMIT** 文でコミットしなくてもトランザクションが終了することがあります。DDL 文を実行すると、トランザクションは自動でコミットされます(DDL 文は **CREATE** 文や **DROP** 文、**ALTER** 文などのデータ定義文です)。

暗黙のコミットでも明示的なコミットでも、コミットには変わりありませんのでロールバックすることはできません。

6. 暗黙のロールバック

ロールバックにも **ROLLBACK** 文で明示的に行うロールバックと、エラー発生時に回復処理のために **MySQL** が自動で行うロールバックがあります。

7. 読み取り一貫性



読み取り一貫性とは、ユーザーBが **INSERT** 文を発行して追加を行っている間に、他のユーザーAが **SELECT** 文を発行した場合でも、ユーザーAは **SELECT** 文を発行した時点で確定されたデータを参照できるように保証することをいいます。

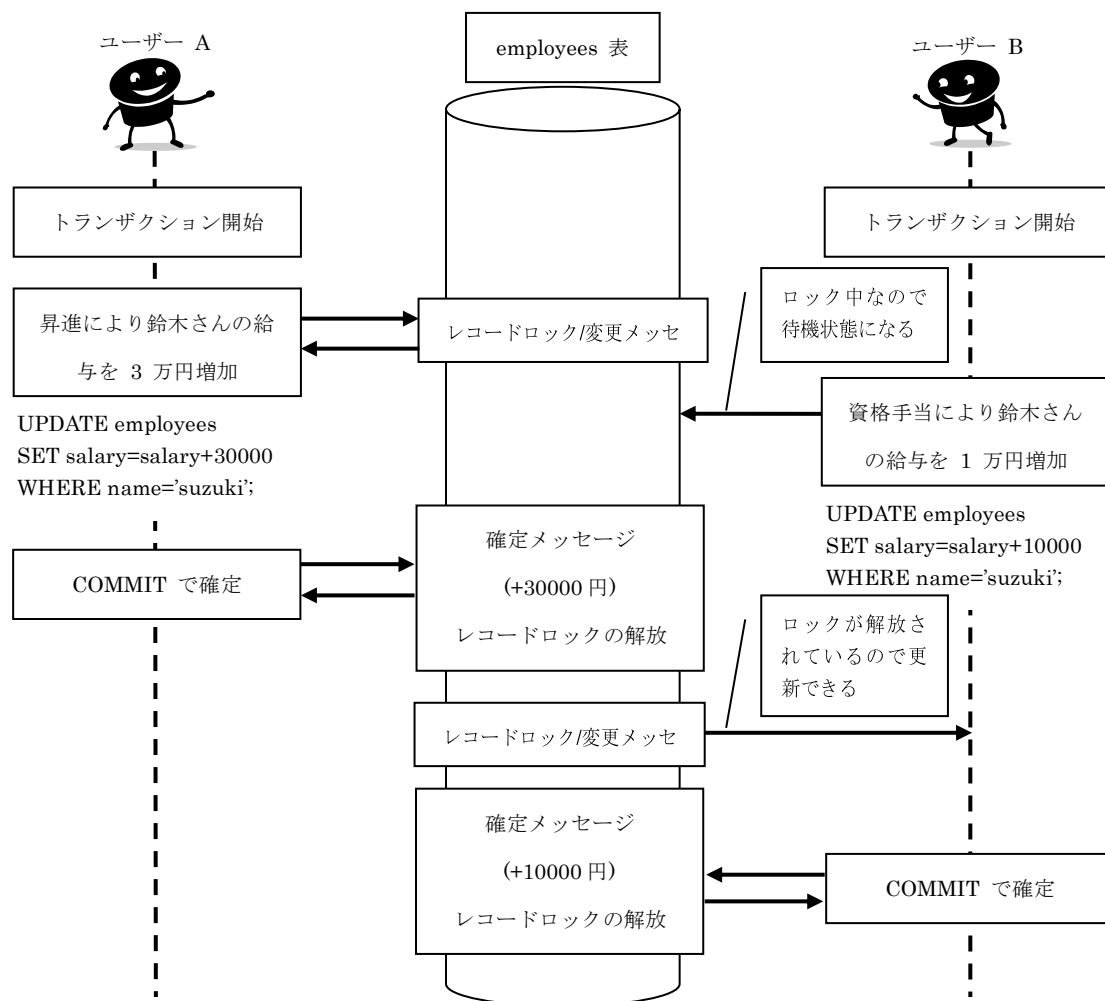
例えば、ユーザーBがデータを変更中だとしても、ユーザーAは以前の値を見続けるような仕組みになっています。ここでいう以前の値とは、他のユーザーが変更/コミットする前の値ということです。

では、どういうメカニズムで読み取り一貫性は保証されるのでしょうか？

まず、ユーザーBに関して前回のコミット以降の内容は全てロールバックセグメントにコピーされます。変更を開始する前のデータはロールバックセグメントに保存されているわけですが、全ての読み取りユーザーにはこのロールバックセグメントのデータが表示されます。ユーザーBが変更をコミットしたとしても、その変更以前から **SELECT** 文を発行しているユーザーにはそのロールバックセグメントのコピーデータが表示されるというわけです。

読み取り一貫性の保証により、**SELECT** 文の結果に矛盾が起こらないようにしています。

8. ロック



ロックは同一のデータに対する複数の更新を防止する機能です。

MySQL は複数の利用者が同時にデータにアクセスできるようになっています。しかし、同時にデータにアクセスできるが故に、とても大きな問題点が浮上してきます。それは、「2 人の人が同時に同じデータを更新してしまう可能性があること」です。これが行われると多くの矛盾が生まれることになります。そのため、ロックという機能を使って複数のトランザクションが同時に一つのデータを更新することを防止します。

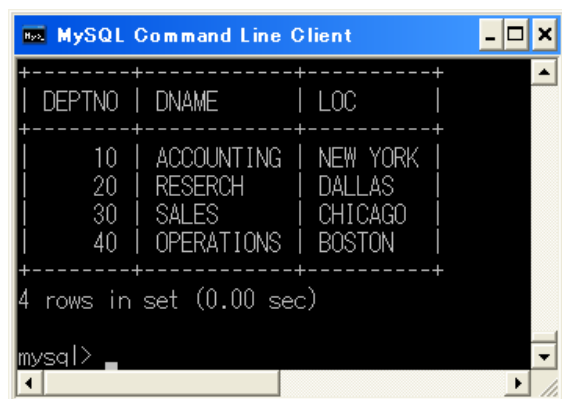
ロック機能とは同じデータに対して更新を行う際に、先に更新をはじめたほうが他からの更新ができないようにしてしまうことです。MySQL においては、ロックは明示的に行う必要はありません。トランザクションがある行に対して更新をかけようとする、MySQL は自動でその行にロックをかけます。他のユーザー(もしくはトランザクション)は、そのロックが解除されるまで更新はできなくなります。

ところで、ロックの範囲は矛盾が起きないために最小限にかけの方が優れています。ロックがかかっている間は、他のユーザーがその範囲に対して変更を加えることができないからです。前述したように MySQL のロックは行レベルでかけることができます。これによりロックの影響を最小限に抑えているのです。

ex

2つのMySQL Command Line Clientでデータベースに接続し、DEPT表を表示して、以下の操作を実行して内容を確認して下さい。それぞれユーザーA、ユーザーBとします。

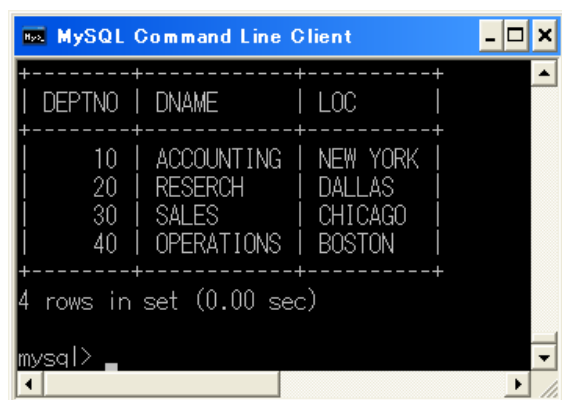
ユーザーA



```
MySQL Command Line Client
+-----+-----+-----+
| DEPTNO | DNAME      | LOC      |
+-----+-----+-----+
| 10     | ACCOUNTING | NEW YORK |
| 20     | RESERCH    | DALLAS   |
| 30     | SALES       | CHICAGO  |
| 40     | OPERATIONS | BOSTON   |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

ユーザーB



```
MySQL Command Line Client
+-----+-----+-----+
| DEPTNO | DNAME      | LOC      |
+-----+-----+-----+
| 10     | ACCOUNTING | NEW YORK |
| 20     | RESERCH    | DALLAS   |
| 30     | SALES       | CHICAGO  |
| 40     | OPERATIONS | BOSTON   |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

- ① ユーザーA で START TRANSACTION コマンドを入力して下さい。

START TRANSACTION;

- ② ユーザーA で DEPTNO が 30 の DNAME 列を ENGINEER に変更し、DEPT 表のすべての列を表示し確認して下さい。

UPDATE DEPT SET DNAME = 'ENGINEER' WHERE DEPTNO = 30;

SELECT * FROM DEPT;

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESERCH	DALLAS
30	ENGINEER	CHICAGO
40	OPERATIONS	BOSTON

時間を置くとタイムアップになるので注意！

- ③ ユーザーB で DEPT 表を表示し、DEPTNO が 30 の DNAME が変更されていないことを確認して下さい。

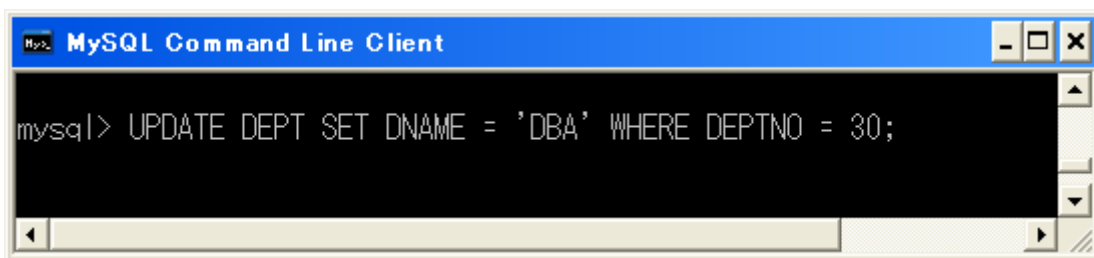
SELECT * FROM DEPT;

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESERCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

ユーザーA がまだ COMMIT をしていないので、ユーザーB はロールバックセグメントのデータを参照していることになります。これが読み取り一貫性です。

- ④ ユーザーB に DEPTNO が 30 の DNAME 列を DBA に変更して下さい。

UPDATE DEPT SET DNAME = 'DBA' WHERE DEPTNO = 30;



※上記の画面のまま操作ができなくなることを確認して下さい。

現在、ユーザーA が、DEPTNO が 30 の行の DNAME に対し変更をしたまま COMMIT していないので、ロックがかかっています。そこに、ユーザーB もロックがかかったデータを変更しようとしたため、操作ができないまま待機状態になっています。

⑤ ユーザーAで COMMIT を実行して下さい。ロックが解除され、ユーザーBの UPDATE 文が実行されます。

COMMIT;

⑥ ユーザーBで DEPT 表を表示し、DEPTNO が 30 の DNAME が変更されていることを確認して下さい。

SELECT * FROM DEPT;

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESERCH	DALLAS
30	DBA	CHICAGO
40	OPERATIONS	BOSTON