データベースシステム
第11回

理工学部情報科学科
松澤　智史

本日の内容

- トランザクション
  - ACID特性
  - コミットとロールバック

トランザクション

- 分けることのできない実行単位
- INSERT, UPDATE, DELETE等の命令のまとまりを指す
- データの障害復旧に強い

障害発生時に困ること

銀行口座AからBに10万円移動したい

処理1
口座Aから
10万円引く

処理2
口座Bに
10万円足す

口座A
残高100万円

口座B
残高200万円

障害発生時に困ること

銀行口座AからBに10万円移動したい

処理1
口座Aから
10万円引く

処理2
口座Bに
10万円足す

口座A
残高90万円

口座B
残高200万円

処理1が完了した段階でシステムダウンが発生！

障害発生時に困ること

システム復旧後

口座A
残高90万円

口座B
残高200万円

トランザクションとは

- 一連の処理の実行を途中の段階で確定させない
- 何かトラブルがあった場合はすべての途中処理を無効にする

トランザクションの特性

トランザクションは以下の特性を満たす必要がある

- 原子性(ATOMICITY)
- 一貫性 (CONSISTENCY)
- 隔離性または独立性 (ISOLATION)
- 持続性 または永続性(DURABILITY)

これら 4つの特性は、それぞれの名前の頭文字をとって
ACID 特性と呼ぶ

原子性(ATOMICITY)

- トランザクションを構成する処理の結果がすべて有効になるか、またはすべて無効になるかのいずれかであること

- 例
  - 処理Aと処理Bがある
  - 処理Aと処理Bの両方の処理が完了した場合に,
    双方の結果は有効になる
  - 処理Aまたは処理Bのいずれかの処理のみ完了した場合は,
    双方の結果は無効となる

一貫性 (CONSISTENCY)

- トランザクション開始と終了時にあらかじめ与えられた整合性を満たすことを保証する性質

- 例
  - 負の数を取らない条件が設定されている項目がある
  - トランザクション実行中（複数の処理を実行）する間に，その項目が負の数となるような処理は行えない

隔離性 (ISOLATION)

- トランザクション中に行われる操作の過程が他の操作から隠蔽される性質
  ※設定やDBの種類によっては厳密には満たさない（後述）

- 例

A	B
100	200

項目AからBへ数値を50移動する処理をトランザクションとする

A	B
50	200

項目Aから50引く

A	B
50	250

項目Bに50足す

この状態のテーブルは
他から参照されてはいけない

持続性 (DURABILITY)

- トランザクション操作の完了通知をユーザが受けた時点で
  その操作は永続的となり，結果が失われない性質

- 例
  - トランザクション操作を永続性記憶装置上にログとして記録し，
    システムに異常が発生したらそのログを用いて
    異常発生前の状態まで復旧する

トランザクションの実行

- データベース操作を間違ってしまった時や
  正常に行えなかった場合は元に戻す
  - 元に戻す操作＝ロールバックという
- トランザクションの最後まで処理が行えた場合には確定する
  - 処理を確定する操作＝コミットという

ロールバック

- トランザクションで指定した処理で，
  現在まで行った処理をすべて無効化する
  ※更新前の状態に戻す

- 後進復帰ともいう

コミット

- トランザクション内処理を実際のデータベースに反映すること

- コミットを行った時点で持続性が発生し，
  ロールバック等は行えない

MySQLでのトランザクション利用

- AUTOCOMMIT変数の確認

```
mysql> select @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            1 |
+--------------+
1 row in set (0.01 sec)

mysql>
```

多くの場合，クライアント実行時に
自動で1に設定される

- 1であればSQL文実行時に自動コミットされている

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

mysql> select @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            0 |
+--------------+
1 row in set (0.00 sec)

mysql>
```

MySQLでのトランザクション利用

- トランザクション開始

```
myql> begin;
SQL文
SQL文
  .
  .
mysql> commit;
or
mysql> rollback;
```

MySQLでのトランザクション利用

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

mysql> update student set name = 'aaa';
Query OK, 4 rows affected (0.00 sec)
Rows matched: 4  Changed: 4  Warnings: 0

mysql> select * from student;
+----+------+----------+-------------+
| id | name | dep_code | email       |
+----+------+----------+-------------+
|  1 | aaa  |       63 | alice@is.jp |
|  2 | aaa  |       64 | bob@bs.jp   |
|  3 | aaa  |       63 | char@is.jp  |
|  4 | aaa  |       64 | dave@bs.jp  |
+----+------+----------+-------------+
4 rows in set (0.00 sec)

mysql>
```

トランザクション開始したクライアントからの表示

MySQLでのトランザクション利用

```
mysql> use test_db;
Database changed
mysql> select * from student;
+----+---------+----------+-------------+
| id | name    | dep_code | email       |
+----+---------+----------+-------------+
|  1 | Alice   |       63 | alice@is.jp |
|  2 | Bob     |       64 | bob@bs.jp   |
|  3 | Charlie |       63 | char@is.jp  |
|  4 | Dave    |       64 | dave@bs.jp  |
+----+---------+----------+-------------+
4 rows in set (0.00 sec)

mysql>
```

別のクライアントからの表示すると変更されていない

MySQLでのトランザクション利用

```
mysql> select * from student;
+----+------+----------+-------------+
| id | name | dep_code | email       |
+----+------+----------+-------------+
|  1 | aaa  |       63 | alice@is.jp |
|  2 | aaa  |       64 | bob@bs.jp   |
|  3 | aaa  |       63 | char@is.jp  |
|  4 | aaa  |       64 | dave@bs.jp  |
+----+------+----------+-------------+
4 rows in set (0.00 sec)

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

mysql> select * from student;
+----+---------+----------+-------------+
| id | name    | dep_code | email       |
+----+---------+----------+-------------+
|  1 | Alice   |       63 | alice@is.jp |
|  2 | Bob     |       64 | bob@bs.jp   |
|  3 | Charlie |       63 | char@is.jp  |
|  4 | Dave    |       64 | dave@bs.jp  |
+----+---------+----------+-------------+
4 rows in set (0.00 sec)

mysql>
```

ROLLBACK;でトランザクション開始時に戻る

MySQLでのトランザクション利用

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

mysql> update student set name='Alice2' where id = 1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from student;
+----+---------+----------+-------------+
| id | name    | dep_code | email       |
+----+---------+----------+-------------+
|  1 | Alice2  |       63 | alice@is.jp |
|  2 | Bob     |       64 | bob@bs.jp   |
|  3 | Charlie |       63 | char@is.jp  |
|  4 | Dave    |       64 | dave@bs.jp  |
+----+---------+----------+-------------+
4 rows in set (0.00 sec)

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

mysql> select * from student;
+----+---------+----------+-------------+
| id | name    | dep_code | email       |
+----+---------+----------+-------------+
|  1 | Alice2  |       63 | alice@is.jp |
|  2 | Bob     |       64 | bob@bs.jp   |
|  3 | Charlie |       63 | char@is.jp  |
|  4 | Dave    |       64 | dave@bs.jp  |
+----+---------+----------+-------------+
4 rows in set (0.00 sec)

mysql>
```

MySQLでの注意点

- AUTOCOMMITが0の時はbegin不要
  - 勝手にトランザクションが始まる
  - SELECT等変更のない命令でもトランザクション開始となる
    - 場合によっては非効率
  - 明示的にcommitする必要がある

- AUTOCOMMITはクライアント再起動すると1になる（注意）

- AUTOCOMMITが1の場合でも
  begin;以降はトランザクションを開始する

トランザクション複数同時実行時の注意

- 隔離性 (ISOLATION)を満たさなくなるケースがある
  - ロストアップデート
  - ダーティリード
  - ノンリピータブルリード
  - ファントムリード

ロストアップデート

更新されたはずのデータが更新前に戻ってしまい消失する現象
以下は①～④の順番に実行されたと想定する

トランザクション1

①
```
SELECT 値 INTO :S
FROM T
WHERE ID = 2
```

③
```
UPDATE T SET 値 = :S*2
WHERE ID = 2
```

テーブルT

ID	値
1	50
2	30

トランザクション2

②
```
SELECT 値 INTO :S
FROM T
WHERE ID = 2
```

④
```
UPDATE T SET 値 = :S*2
WHERE ID = 2
```

この場合トランザクション2が行われた後もテーブルの2行目の値列は60となる

ダーティリード

ロールバックされるコミット前の更新データを読み取り誤読する現象
以下は①～④の順番に実行されたと想定する

トランザクション1

①
```
SELECT 値 INTO :S
FROM T
WHERE ID = 2
```

②
```
UPDATE T SET 値 = :S*2
WHERE ID = 2
```

④
```
ROLLBACK
```

テーブルT

ID	値
1	50
2	30

トランザクション2

③
```
SELECT 値 INTO :S
FROM T
WHERE ID = 2
```

この場合トランザクション2の変数:Sには60が格納されてしまう

ノンリピータブルリード

あるトランザクション処理が2回同一のデータを読んだ際に異なる値を読む現象
以下は①～④の順番に実行されたと想定する

トランザクション1

①
```
SELECT 値 INTO :S
FROM T
WHERE ID = 2
```

③
```
UPDATE T SET 値 = :S*2
WHERE ID = 2
```

テーブルT

ID	値
1	50
2	30

トランザクション2

②
```
SELECT 値 INTO :S
FROM T
WHERE ID = 2
```

④
```
SELECT 値 INTO :S
FROM T
WHERE ID = 2
```

この場合トランザクション2の②と④で格納される変数：Sには
それぞれ30と60が格納されてしまう

ファントムリード

トランザクション処理が2回，同一範囲のデータを読みだしたときに1回目にはなかったデータを2回目で読み取ってしまう現象
以下は①～④の順番に実行されたと想定する

トランザクション1

②
```
INSERT INTO T
VALUES(3,20)
```

テーブルT

ID	値
1	50
2	30

トランザクション2

①
```
SELECT SUM(値)
FROM T
```

③
```
SELECT SUM(値)
FROM T
```

①の結果は80だが，③の結果は100となる

隔離性水準

- 隔離性は厳密に満たすとパフォーマンスが低下するため，
  設定やDB（エンジン）の種類によって許容する範囲に種類がある

隔離性水準	ダーティリード	ノンリピータブルリード	ファントムリード
Read Uncommitted	あり	あり	あり
Read Committed	なし	あり	あり
Repeatable Read	なし	なし	あり
Serializable	なし	なし	なし

MySQLデフォルトのDB(InnoDB)ではRepeartable Readが初期設定になっている

隔離性水準の確認

```
mysql> SELECT @@GLOBAL.transaction_isolation;
+--------------------------------+
| @@GLOBAL.transaction_isolation |
+--------------------------------+
| REPEATABLE-READ                |
+--------------------------------+
1 row in set (0.00 sec)

mysql>
```

変数 GLOBAL.transaction_isolationを参照する

隔離性水準の設定方法

```
mysql> SET TRANSACTION ISOLATION LEVEL
       Read Uncommitted
または  Read Committed
または  Repeatable Read
または  Serializable
```

まとめ

- トランザクション
- ACID特性
- コミットとロールバック
- 隔離性水準

質問あればどうぞ