外部キーと参照整合性

# 参照整合性の崩壊

外部キーとは「他のテーブルの関連(リレーション)を結ぶ役割を担う列」つまり、他のテーブルの主キー列などの値が格納されています。外部キー値が指し示す先にきちんとデータ(行)が存在していてリレーションシップが成立していることを**参照整合性(referential integrity)**といいます。

・従業員表 ・店舗表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMP\_NO | ENAME | WORK\_STORE |  | STORE\_NO | SNAME |
| 00001  00002  00004  00005  00006 | 小林友治  小林圭輔  稲田直樹  河井ゆずる  村上信五 | 003  001  001  参照整合性が  成立している  001  002 |  | 001  002  003  005  006 | 中崎町店  森ノ宮店  生駒店  阿倍野店  尼崎店 |

この時、店舗表にある生駒店のデータを削除すると従業員表の小林友治さんの勤務店舗が店舗表に管理されていない状態になります。このことを「参照整合性の崩壊」といいデータベースにおいて避けなければならない現象です。

# 崩壊の原因

参照整合性の崩壊を引き起こすデータ操作には、次にあげる４つのパターンがあります。

１．「他の行から参照されている主キー」の行を削除してしまう。

２．「他の行から参照されている主キー」を更新してしまう。

３．「存在しない行を参照する外部キー」を追加してしまう。

４．「存在しない行を参照する外部キー」に更新してしまう

ハンズオン　参照整合性の崩壊を経験してみる

１．店舗表で管理されている中崎町の店舗番号を変更する。

UPDATE STORE SET STORE\_NO = '999'

WHERE STORE\_NO = '001' ;

２．従業員の名前と勤務している店舗名の表示を行う。

SELECT E.EMP\_NO, E.ENAME, E.WORK\_STORE, S.SNAME

FROM EMPLOYEE AS E

JOIN STORE AS S ON ( E.WORK\_STORE = S.STORE\_NO ) ;

３．トランザクションの取消しを行う。

ROLLBACK ;

# 外部キー制約

ハンズオンの通り、正しいSQL文を実行すると簡単に参照整合性の崩壊が発生します。

このような人によるミスを防ぐために、DBMSには**外部キー制約(FOREIGN KEY制約)**が備わっています。また、参照整合性制約とも呼ばれます。**この制約は参照元テーブルの外部キーに対して定義を行います**。参照元と参照先のデータ型は同じである必要があります。

・外部キー制約の基本構文

CREATE TABLE 表名 (

列名 データ型 REFERENCEES 参照先表名 ( 参照先列名 ) ,

…

) ;

CREATE TABLE 表名 (

列名 データ型 ,

…

FOREIGN KEY ( 参照元列名 ) REFERENCES 参照先表名 ( 参照先列名 )

) ;

※参照元列名は自分の表の列名を指定して、

参照先表名と参照先列名は自分以外の表の指定を行う。

・従業員表 ・店舗表(参照先表名)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMP\_NO | ENAME | WORK\_STORE |  | STORE\_NO | SNAME |
| 00001  00002  00004 | 小林友治  小林圭輔  稲田直樹 | 003  001  001 |  | 001  002  003 | 中崎町店  森ノ宮店  生駒店 |

**└参照元列名(外部キー)** **└参照先列名**

この場合は、従業員表に対して外部キー制約を設定する。

すでに表を作成している場合は、ALTER TABLE文で制約の追加及び削除が可能。

・外部キーの追加

ALTER TABLE 表名 ADD FOREIGN KEY (参照元列名)

REFERENCES 参照先表名 (参照先列名) ;

・外部キーの削除

ALTER TABLE 表名 DROP FOREIGN KEY **制約名** ;

※制約名は、SHOW CREATE TABLE 表名で確認することができます。

ハンズオン　外部キー制約を定義する **※必須**

１．従業員表の勤務店舗列に店舗表の店舗コードを参照する外部キー制約を定義する。

ALTER TABLE EMPLOYEE ADD FOREIGN KEY ( WORK\_STORE )

REFERENCES STORE ( STORE\_NO ) ;

２．店舗表にある中崎町のデータを削除してみる。

DELETE FROM STORE

WHERE STORE\_NO = '001' ;

３．店舗表にある森ノ宮の店舗コードを変更してみる。

UPDATE STORE

SET STORE\_NO = '999'

WHERE STORE\_NO = '002' ;

４．従業員表に存在しない店舗コードで従業員の追加をしてみる。

INSERT INTO EMPLOYEE

VALUES( '10001', '陣内智則', '1974-02-22', '0660030002',

'兵庫県加古川市加古川町北在家２０００',

DATE(NOW()), 2, 100, 950, '**999**', '00001' ) ;

５．従業員表に存在しない店舗コードで従業員の変更をしてみる。

UPDATE EMPLOYEE

SET WORK\_STORE = '999'

WHERE EMP\_NO = '00001' ;

トランザクションの分離

# 同時実行の副作用

以前、トランザクションについて学習しました。銀行の振込を例に上げると自分のお金を減らす処理と相手のお金を増やす処理は両方が正しく処理されないとデータが不正な値になってしまいます。この**セットで処理をしないといけない範囲をトランザクション**、**不正な更新を防ぐための仕組みとしてコミット、ロールバック**を学習しました。また、この「トランザクションが一部だけ実行されることはあってはならない」性質のことを**原子性**といいます。

しかし、これだけだとデータの整合性を保つのは難しいです。なぜなら、学習環境ではデータベースに対してSQL文を送るのは皆さん自身だけです。しかし、世の中で利用されている情報システムにおいては、**多くの利用者が一つのDBMSに対してたくさんのSQL文が実行されます**。DBMSはそれらの要求を同時に処理しようとするので、トランザクションで編集中のデータが他の利用者によって変更されてしまうと正しいデータ管理ができなくなってしまいます。

中崎町店

森ノ宮店

生駒店

秋元真夏の給与を+100

COMMIT

秋元真夏の給与を+250

COMMIT

結局、秋元真夏の給与は

何を設定すればいいの？

秋元真夏の給与を+70

ROLLBACK

# トランザクションの分離

このような状態にならないようにDBMSは個々のトランザクションが他のトランザクションから影響を受けないように分離させて処理を実行します。この分離するための制御を実現するためにDBMSには、**ロック(lock)**という仕組みが備わっています。

ロックとは、あるトランザクションが現在読み書きしている行に鍵をかけ、他の人のトランザクションから読み書き出来ないようにしています。ロックはトランザクションがコミットまたは、ロールバックされると解除され、他の人のトランザクションがその行を読み書き出来るようになります。

・ロックの動作イメージ

ロック

トランザクションA

秋元真夏の給与を+100

トランザクションB

秋元真夏の給与を+250

|  |  |  |
| --- | --- | --- |
| EMP\_NO | ENAME | SALARY |
| 00011 | 秋元真夏 | ~~950~~ 1050 |

待機

トランザクションAがコミットされた（**ロックの解除**）

ロック

トランザクションB

秋元真夏の給与を+250

|  |  |  |
| --- | --- | --- |
| EMP\_NO | ENAME | SALARY |
| 00011 | 秋元真夏 | ~~1050~~ 1300 |

ハンズオン　ロックが掛かっている確認する

１．コマンドプロンプトを２つ開いて、ともにdbuserでmysqlにログインする。

２．秋元真夏の給与を+100にする更新を行う。（ロックの発生）

UPDATE EMPLOYEE SET SALARY = SALARY + 100

WHERE EMP\_NO = '00011' ;

SELECT ENAME, SALARY FROM EMPLOYEE WHERE EMP\_NO = '00011' ;

３．別のコマンドプロンプトから秋元真夏の給与を+250にする更新を行う。(待機の発生)

UPDATE EMPLOYEE SET SALARY = SALARY + 250

WHERE EMP\_NO = '00011' ;

４．初めのトランザクションの確定処理を行う。(ロックの解除)

COMMIT;

５．待機状態のトランザクションが実行されたことを確認する。

SELECT ENAME, SALARY FROM EMPLOYEE WHERE EMP\_NO = '00011' ;

６．秋元真夏の給与を元に戻してトランザクションを確定させる。

UPDATE EMPLOYEE SET SALARY = 950

WHERE EMP\_NO = '00011' ;

COMMIT;

# デッドロック

データベースで同時にたくさんのトランザクションが実行されると、**デッドロック(dead lock)**が発生する可能性があります。デッドロックは次のような状態の時に発生します。

従業員表

店舗表

①中崎町の情報を更新

③河井ゆずるの情報を更新

②河井ゆずるの情報を更新

④中崎町の情報を更新

トランザクションB

トランザクションA

デッドロック発生

待機

待機

デッドロックとは、お互いがデータのロックを行い相手のロック解除を待ち続けている状態になります。こうなると処理が停止してしまうので、多くのDBMSにはデッドロックを検知して自動的に解決する仕組みが備わっています。

ハンズオン　デットロックを発生させてみる

１．コマンドプロンプトを２つ開いて、ともにdbuserでmysqlにログインする。

２．店舗表にある中崎町の情報を更新する。（ロック）

UPDATE STORE SET HOLIDAY = '金'

WHERE STORE\_NO = '001' ;

３．別のコマンドプロンプトで従業員表にある河井ゆずるの情報を更新する。（ロック）

UPDATE EMPLOYEE SET JOB\_NO = 2

WHERE EMP\_NO = '00005' ;

４．初めのコマンドプロンプトで従業員表にある河井ゆずるの情報を更新する。（待機）

UPDATE EMPLOYEE SET JOB\_NO = 3

WHERE EMP\_NO = '00005' ;

５．２つ目のコマンドプロンプトで店舗表にある中崎町の情報を更新する。（待機）

UPDATE STORE SET HOLIDAY = '月'

WHERE STORE\_NO = '001' ;

６．両トランザクションの店舗表と従業員表のデータを確認する。

SELECT \* FROM STORE WHERE STORE\_NO = '001' ;

SELECT \* FROM EMPLOYEE WHERE EMP\_NO = '00005' ;

７．デッドロックが発生しなかった方のトランザクションを取消しする。

ROLLBACK;

このように、トランザクションを強制的に失敗させることでデッドロック状態は解決されます。しかし、少なくとも１つのトランザクションは失敗してしまうので**可能な限りデッドロックは避けるべき**です。そもそも**デッドロックは相手と違う順番でロックを行おうとするために発生する**現象です。以下の内容を意識することでデッドロックを未然に防ぐことが可能になります。

・デッドロックの予防する方法

対策１：トランザクションの時間を短くする

対策２：同じ順番でロックするようにする。