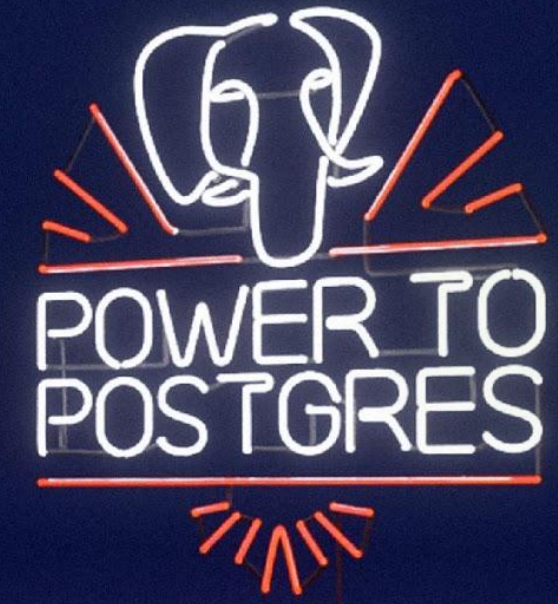
MVCCの素顔



Koichi Suzuki

Nov. 10, 2021

EDB

謝辞

この Webinar は、PostgreSQL Evangelist である Bruce Momjian氏の全面的な協力で、氏の持つプレゼンテーション資料を利用して行うものです

オリジナルの資料は https://momjian.us/main/writings/pgsql/mvcc.pdf で公開されています

内容

多版型同時実行制御 (MVCC, Multi-Version Concurrency Control) の Postgres での実装と、その弱点の克服の方法を説明します

- MVCC 入門
- MVCC の実装の詳細
- MVCC のクリーンナップ要件とその振る舞い
- SERIALIZABLE 隔離レベルの概要

自己紹介

- NTT 研究所, NTT DATA, NTT DATA 先端技術にて
    - Unix の日本語化
    - Oracle 移植と日本語化
    - UniSQL オブジェクトリレーショナルデータベース開発
    - PostgreSQL (NTT OSS センター)
    - Postgres-XC, Postgres-XL 水平分散型データベースリーダ
- 2ndQuadrant
- 現在 EDB メンバ

MVCC の素顔を見てみる理由

- クエリが並列実行される様子が予測できるようになる
- MVCC がもたらす性能上の効果を管理する
- ストレージ空間の再利用を理解する

MVCC とは何か？

多版型同時実行制御 (MVCC, Multi-Version Concurrency Control) によって、データベースに対する読み書き負荷が大きな場合であっても、Postgres は高い並列度でデータベースの実行ができるようになっています。MVCC が提供する重要な機能は、「読み手は書き手の邪魔をせず (ブロックせず)、書き手は読み手の邪魔をしない (ブロックしない)」というものです。

このWeb では、Postgres で MVCC がどのように実装されているか、そして、MVCC の弱点を克服するための最適化を説明します。

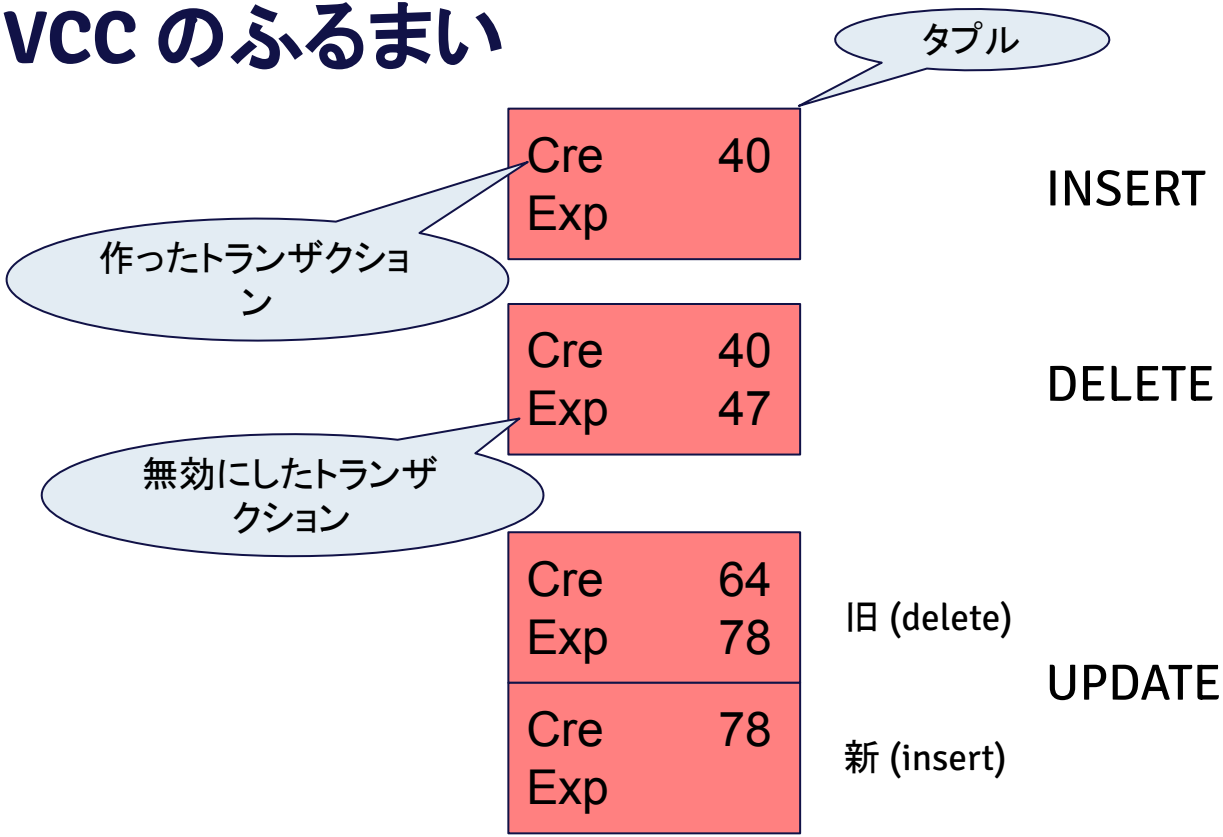
これには、Postgres内部での行更新のしかた、データベースの内部のお掃除が含まれます。

また、SERIALIZABLE 分離レベルについても説明します。

MVCC をサポートしている他のデータベース

- Oracle
- DB2 (一部)
- MySQL with InnoDB
- Informix
- Firebird
- MSSQL (オプション, デフォルトでは無効)

MVCC のふるまい

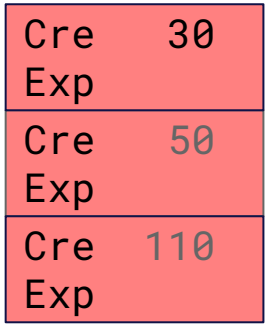


MVCC のスナップショット

- MVCC のスナップショットを使うと、SQL 文から見えるタプルがどれかを制御できます
- スナップショットは、READ COMMITTED 隔離モードの場合は各SQL文の開始時に、REPEATABLE READ 及び SERIALIZABLE 隔離モードの場合はトランザクションの開始時に取得します
- このように、スナップショットの取得頻度で各隔離モードのふるまいを制御しているのです (SERIALIZABLE には追加の機能が必要です)
- スナップショットで集める情報:
  - コミットしたトランザクションの最大の番号
  - 実行中のトランザクションの番号
- スナップショットの情報を使って、どのトランザクションの結果を実行中のどのSQL文に見せるかを Postgres は決めているのです

MVCC スナップショットを使った行可視性の決定

生成のみ

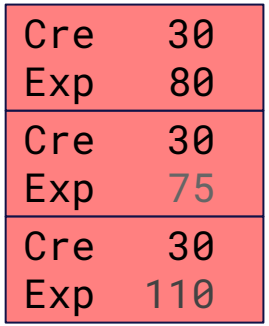


見える

見えない

見えない

生成と無効化



見えない

見える

見える

シーケンシャルスキャン

スナップショット:

コミットしたトランザクションの最大の番号: 99

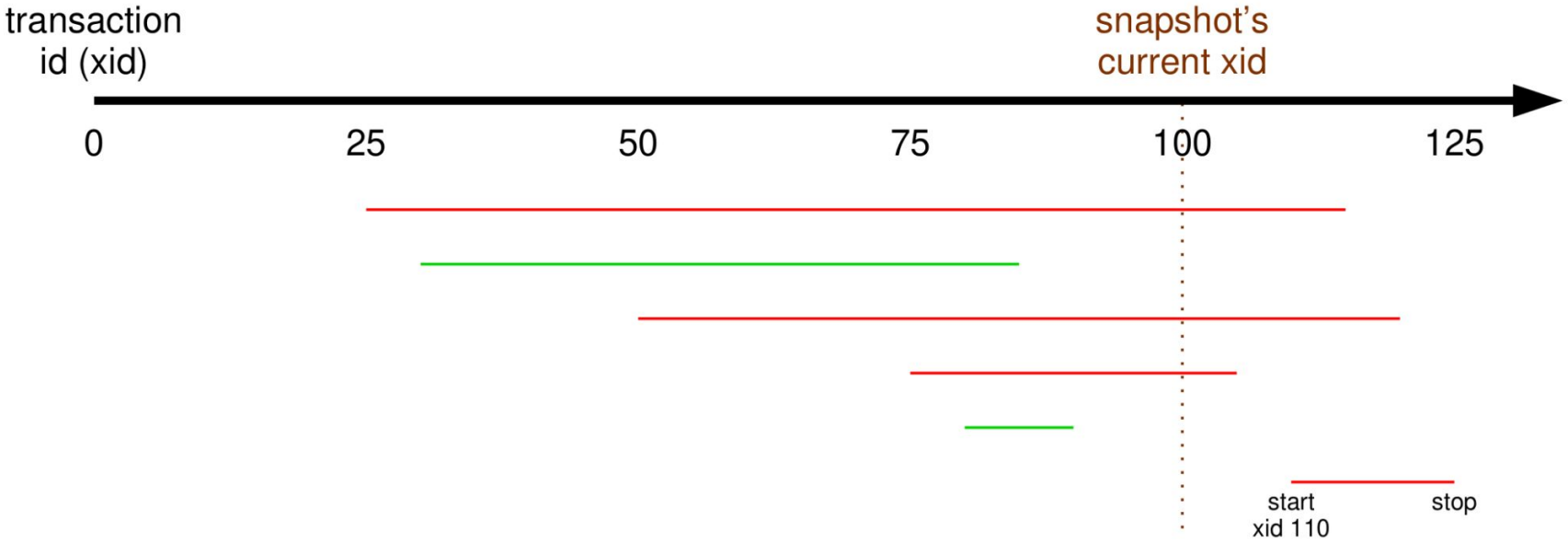
実行中のトランザクション: 25, 50, 75

簡単のため、他の全てのトランザクションはコミットしたものとする

自分のトランザクションの番号は 100

内部的には、Cre xid は ‘xmin’, Exp xid は ‘xmax’ のシステムカラムに格納されています

MVCC スナップショットを時系列で書いてみる



Green is visible. Red is invisible.

見えるトランザクションは、トランザクション id 100が開始される前に完了したもののみ

実装の詳細

- 全てのクエリはコミュニティ版の Postgres で動作しています
- ヒープの内部情報を見るため、contrib モジュールである `pageinspect` をインストールしています
- フリースペースマップの情報を表示するため、`pg_freespacemap` もインストールしています。

設定

```
CREATE TABLE mvcc_demo (val INTEGER);

DROP VIEW IF EXISTS mvcc_demo_page0;

CREATE EXTENSION pageinspect;

CREATE EXTENSION pg_freespacemap;

CREATE VIEW mvcc_demo_page0 AS
     SELECT '(0,' || lp || ')' AS ctid,
          CASE lp_flags
               WHEN 0 THEN 'Unused'
               WHEN 1 THEN 'Normal'
               WHEN 2 THEN 'Redirect to ' || lp_off
               WHEN 3 THEN 'Dead'
          END,
          t_xmin::text::int8 AS xmin,
          t_xmax::text::int8 AS xmax,
          t_ctid
     FROM heap_page_items(get_raw_page('mvcc_demo', 0))
     ORDER BY lp;
```

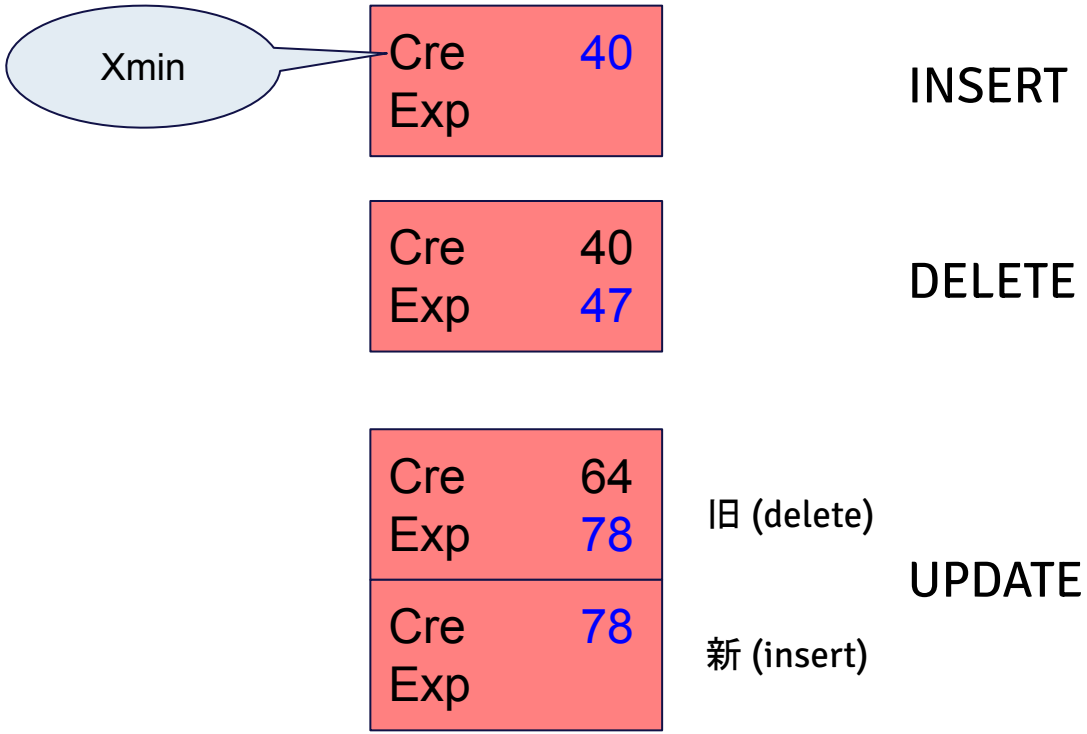
pageinspect:
https://www.postgresql.jp/document/13/html/pageinspect.html
pg_freespacemap:
https://www.postgresql.jp/document/13/html/pgfreespacemap.html

INSERT で xmin を使う

```
DELETE FROM mvcc_demo;

INSERT INTO mvcc_demo VALUES (1);

SELECT xmin, xmax, * FROM mvcc_demo;
 xmin | xmax | val
------+------+-----
 5409 |    0 |   1
```

ここで使われているSQL文は、https://momjian.us/main/writings/pgsql/mvcc.sql で入手できます

INSERT で mxin を使う



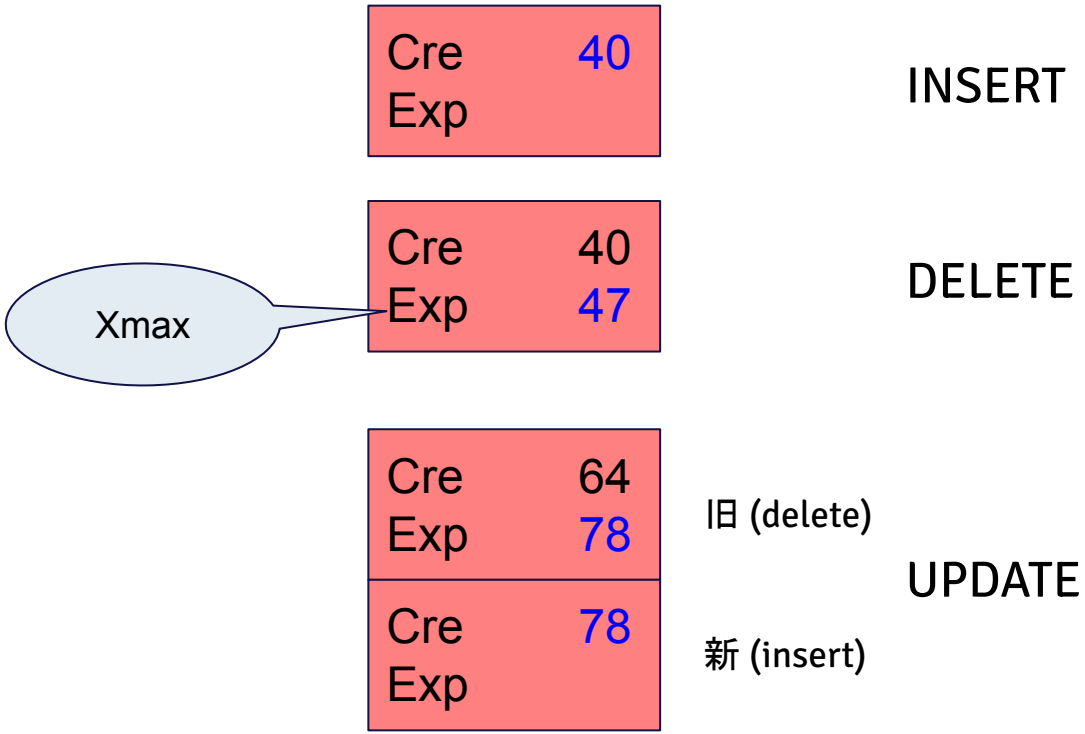
DELETE で xmax を使う

```
DELETE FROM mvcc_demo;

INSERT INTO mvcc_demo VALUES (1);

SELECT xmin, xmax, * FROM mvcc_demo;
 xmin | xmax | val
------+------+-----
 5411 |    0 |   1

BEGIN WORK;

DELETE FROM mvcc_demo;
```

DELETE で xmax を使う

```
SELECT xmin, xmax, * FROM mvcc_demo;
 xmin | xmax | val
------+------+-----

           SELECT xmin, xmax, * FROM mvcc_demo;
            xmin | xmax | val
           ------+------+-----
            5411 | 5412 |    1

SELECT txid_current();
Txid_current
--------------
        5412

COMMIT WORK;
```

DELETE で xmax を使う



UPDATE でXmin と Xmax を使う

```
DELETE FROM mvcc_demo;

INSERT INTO mvcc_demo VALUES (1);

SELECT xmin, xmax, * FROM mvcc_demo;
 xmin | xmax | val
------+------+-----
 5413 |    0 |   1

BEGIN WORK;

UPDATE mvcc_demo SET val = 2;
```

UPDATE でXmin と Xmax を使う

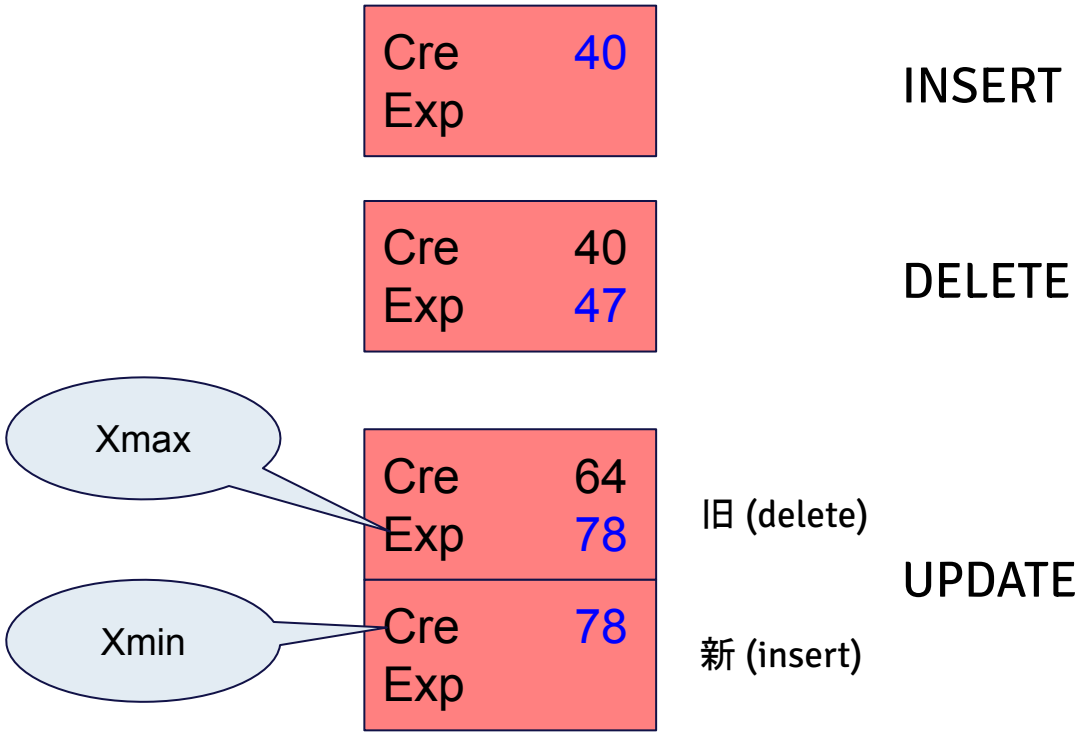
```
SELECT xmin, xmax, * FROM mvcc_demo;
 xmin | xmax | val
------+------+-----
 5414 |    0 |   2
```

UPDATE したトランザクションからは更新後のタプルが見える

```
          SELECT xmin, xmax, * FROM mvcc_demo;
           xmin | xmax | val
          ------+------+-----
           5413 | 5414 |   1

COMMIT WORK;
```

他のトランザクションからは更新前のタプルが見える

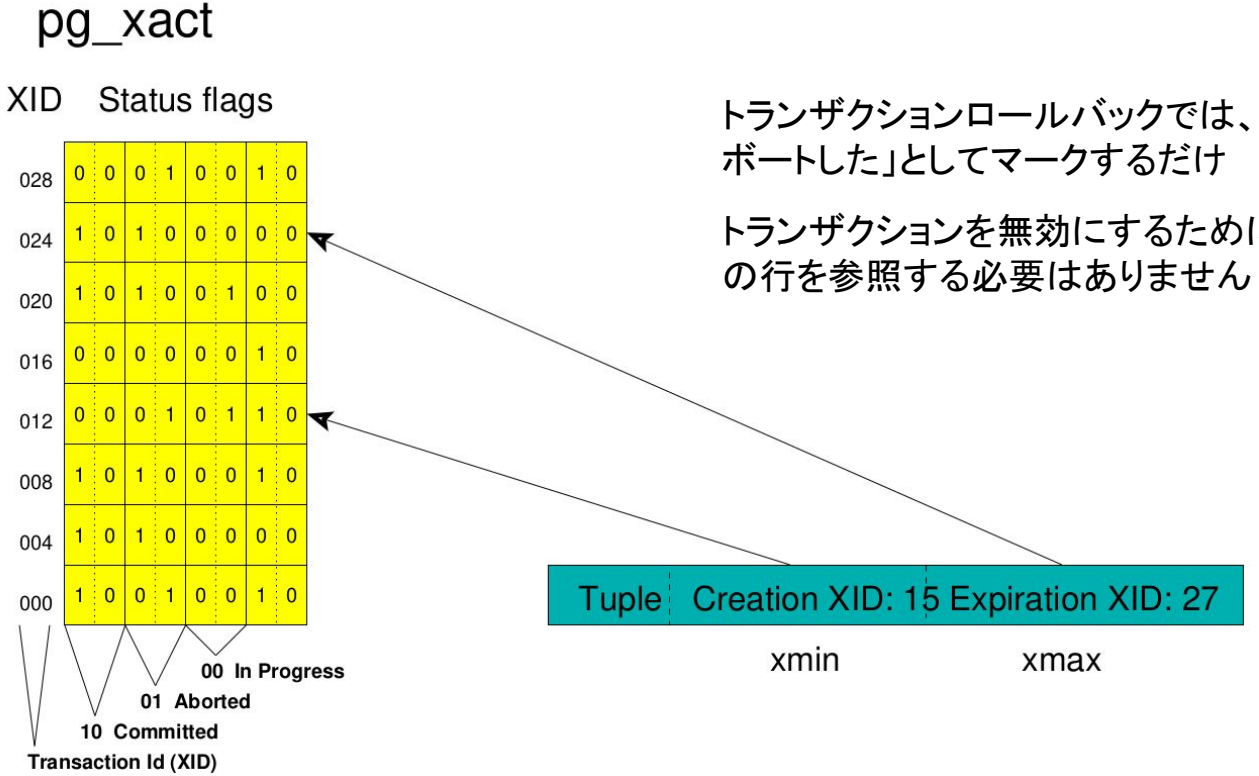
UPDATE でXmin と Xmax を使う



Abort したトランザクションの ID もデータベースには残る

```
DELETE FROM mvcc_demo;

INSERT INTO mvcc_demo VALUES (1);

BEGIN WORK;

DELETE FROM mvcc_demo;

ROLLBACK WORK;

SELECT xmin, xmax, * FROM mvcc_demo;
 xmin | xmax | val
------+------+-----
 5415 | 5416 | 1
```

アボートしたIDも含め、トランザクション状態は集中記録する



トランザクションロールバックでは、その IDを「アボートした」としてマークするだけ

トランザクションを無効にするためにそれぞれの行を参照する必要はありません

行ロックでXmax を使う

```
DELETE FROM mvcc_demo;

INSERT INTO mvcc_demo VALUES (1);

BEGIN WORK;

SELECT xmin, xmax, * FROM mvcc_demo;
 xmin | xmax | val
------+------+-----
 5416 |    0 |   1

SELECT xmin, xmax, * FROM mvcc_demo FOR UPDATE;
 xmin | xmax | val
------+------+-----
 5416 |    0 |   1
```

行ロックで xmax を使う

```
SELECT xmin, xmax, * FROM mvcc_demo;
 xmin | xmax | val
------+------+-----
 5416 | 5417 |   1

COMMIT WORK;
```

タプルのビット HEAP_XMAX_EXCL_LOCK を使って、xmax が行削除した xid ではなく、ロックを保持している xid であることを示している

複数のSQL文からなるトランザクション

トランザクションが複数のSQL文で構成される場合、追加の行追跡の機構が必要です。それぞれの文が個別の可視性ルールを持っているからです。

たとえば、カーソルの内容は、同じトランザクションで後のSQL文が行を更新しても、その内容に変更があってはなりません。

このような行の追跡は、cmin/cmax とよばれるシステムコマンド id カラムを使って実装されています。これは、内部的には1つのカラムなのです。

INSERT でCmin を使う

```
DELETE FROM mvcc_demo;

BEGIN WORK;

INSERT INTO mvcc_demo VALUES (1);
INSERT INTO mvcc_demo VALUES (2);
INSERT INTO mvcc_demo VALUES (3);
```

INSERT でCmin を使う

```
SELECT xmin, cmin, xmax, * FROM mvcc_demo;
 xmin | cmin | xmax | val
------+------+------+-----
 5419 |    0 |    0 |   1
 5419 |    1 |    0 |   2
 5419 |    2 |    0 |   3

COMMIT WORK;
```

DELETE でCmin を使う

```
DELETE FROM mvcc_demo;

BEGIN WORK;

INSERT INTO mvcc_demo VALUES (1);
INSERT INTO mvcc_demo VALUES (2);
INSERT INTO mvcc_demo VALUES (3);
```

DELETE でCmin を使う

```
SELECT xmin, cmin, xmax, * FROM mvcc_demo;
 xmin | cmin | xmax | val
------+------+------+-----
 5421 |    0 |    0 |   1
 5421 |    1 |    0 |   2
 5421 |    2 |    0 |   3

DECLARE c_mvcc_demo CURSOR FOR
   SELECT xmin, xmax, cmax, * FROM mvcc_demo;
```

DELETE でCmin を使う

```
DELETE FROM mvcc_demo;

SELECT xmin, cmin, xmax, * FROM mvcc_demo;
 xmin | cmin | xmax | val
------+------+------+-----

FETCH ALL FROM c_mvcc_demo;
 xmin | xmax | cmax | val
------+------+------+-----
 5421 | 5421 |    0 |   1
 5421 | 5421 |    1 |   2
 5421 | 5421 |    2 |   3

COMMIT WORK;
```

これらの行は、このトランザクションで生成削除しているので、他のトランザクションでは見えません。そのため、行の状態を見るためにはカーソルを使う必要があります。

UPDATE でCmin を使う

```
DELETE FROM mvcc_demo;

BEGIN WORK;

INSERT INTO mvcc_demo VALUES (1);
INSERT INTO mvcc_demo VALUES (2);
INSERT INTO mvcc_demo VALUES (3);

SELECT xmin, cmin, xmax, * FROM mvcc_demo;
 xmin | cmin | xmax | val
------+------+------+-----
 5422 |    0 |    0 |   1
 5422 |    1 |    0 |   2
 5422 |    2 |    0 |   3

DECLARE c_mvcc_demo CURSOR FOR
    SELECT xmin, xmax, cmax, * FROM mvcc_demo;
```

UPDATE でCmin を使う

```
UPDATE mvcc_demo SET val = val * 10;

SELECT xmin, cmin, xmax, * FROM mvcc_demo;
 xmin | cmin | xmax | val
------+------+------+-----
 5422 |    3 |    0 |  10
 5422 |    3 |    0 |  20
 5422 |    3 |    0 |  30

FETCH ALL FROM c_mvcc_demo;
 xmin | xmax | cmax | val
------+------+------+-----
 5422 | 5422 |    0 |   1
 5422 | 5422 |    1 |   2
 5422 | 5422 |    2 |   3

COMMIT WORK;
```

異なるトランザクションで行を変更する

```
DELETE FROM mvcc_demo;

INSERT INTO mvcc_demo VALUES (1);

SELECT xmin, xmax, * FROM mvcc_demo;
 xmin | xmax | val
------+------+-----
 5424 |    0 |   1

BEGIN WORK;

INSERT INTO mvcc_demo VALUES (2);
INSERT INTO mvcc_demo VALUES (3);
INSERT INTO mvcc_demo VALUES (4);
```

異なるトランザクションで行を変更する

```
SELECT xmin, cmin, xmax, * FROM mvcc_demo;
 xmin | cmin | xmax | val
------+------+------+-----
 5424 |    0 |    0 |   1
 5425 |    0 |    0 |   2
 5425 |    1 |    0 |   3
 5425 |    2 |    0 |   4

UPDATE mvcc_demo SET val = val * 10;
```

異なるトランザクションで行を変更する

```
SELECT xmin, cmin, xmax, * FROM mvcc_demo;
 xmin | cmin | xmax | val
------+------+------+-----
 5425 |    3 |    0 |  10
 5425 |    3 |    0 |  20
 5425 |    3 |    0 |  30
 5425 |    3 |    0 |  40
```

```
SELECT xmin, xmax, cmax, * FROM mvcc_demo;
 xmin | xmax | cmax | val
------+------+------+-----
 5424 | 5425 |    3 |   1
```

```
COMMIT WORK;
```

Combo Command id

- cmin と cmax は内部的には単一のシステムカラムなので、同一の複数SQL文のトランザクションで生成し、かつ、削除した行の状態を直接記録することはできません。
- このため、特別の combo command id を作り、実際の cmin と cmax の値を格納しているローカルメモリハッシュを参照するようにしています。

UPDATE で combo command id を使う

```
-- use TRUNCATE to remove even invisible rows

TRUNCATE mvcc_demo;

BEGIN WORK;

DELETE FROM mvcc_demo;
DELETE FROM mvcc_demo;
DELETE FROM mvcc_demo;

INSERT INTO mvcc_demo VALUES (1);
INSERT INTO mvcc_demo VALUES (2);
INSERT INTO mvcc_demo VALUES (3);
```

UPDATE で combo command id を使う

```
SELECT xmin, cmin, xmax, * FROM mvcc_demo;
 xmin | cmin | xmax | val
------+------+------+-----
 5427 |    3 |    0 |   1
 5427 |    4 |    0 |   2
 5427 |    5 |    0 |   3

DECLARE c_mvcc_demo CURSOR FOR
    SELECT xmin, xmax, cmax, * FROM mvcc_demo;

UPDATE mvcc_demo SET val = val * 10;
```

UPDATE で combo command id を使う

```
SELECT xmin, cmin, xmax, * FROM mvcc_demo;
 xmin | cmin | xmax | val
------+------+------+-----
 5427 |    6 |    0 |  10
 5427 |    6 |    0 |  20
 5427 |    6 |    0 |  30

FETCH ALL FROM c_mvcc_demo;
 xmin | xmax | cmax | val
------+------+------+-----
 5427 | 5427 |    0 |   1
 5427 | 5427 |    1 |   2
 5427 | 5427 |    2 |   3
```

UPDATE で combo command id を使う

```
SELECT   t_xmin AS xmin,
         t_xmax::text::int8 AS xmax,
         t_field3::text::int8 AS cmin_cmax,
         (t_infomask::integer & X'0020'::integer)::bool AS is_combocid
FROM heap_page_items(get_raw_page('mvcc_demo', 0))
ORDER BY 2 DESC, 3;
 xmin | xmax | cmin_cmax | is_combocid
------+------+-----------+-------------
 5427 | 5427 |         0 | t
 5427 | 5427 |         1 | t
 5427 | 5427 |         2 | t
 5427 |    0 |         6 | f
 5427 |    0 |         6 | f
 5427 |    0 |         6 | f

COMMIT WORK;
```

最後のクエリでは /contrib/pageinspect を使っています。これで、内部ヒープページの構造が見えるようになり、現在のスナップショットでは見れないような行データも見れるようになります。

(ビット 0x0020 は、内部的には HEAP_COMBOCID とよばれています)。

MVCC 実装のまとめ

xmin: 生成したトランザクションの番号。INSERT と UPDATE で設定される

xmax: 無効にしたトランザクションの番号。UPDATE と DELETE で設定される。明示的な行ロックでも使用される。

cmin/cmax: タプルの生成あるいは無効化したコマンド番号を識別するのに使う。同一トランザクションでタプルを生成無効化した場合、combo command id の格納にも用いる。明示的な行ロックでも使われる。

旧来のストレージのお掃除の要件

旧来の単一行バージョン (non-MVCC) データベースシステムでは、ストレージスペースを掃除する必要があります。

- 削除された行
- アボートしたトランザクションが生成した行

MVCC のストレージのお掃除の要件

MVCC では、更に次の要件があります :

- UPDATE で新しい行を生成します (既存の行が置き換えられるわけではありません) 。そのため、古い行が占めていたストレージ空間を最終的に再利用しなければなりません。
- 削除した行を**後で**掃除する必要があります (掃除は、このような行が見えるトランザクションがなくなった後でなければ実行できません)。

Postgres では、旧来の要件もMVCC特有の要件もどちらも処理しています

お掃除の振る舞い

Postgres では、お掃除は自動的に行われます:

- 単一ページのお掃除は、必要の都度、行へのアクセスの間に実行されます。個別には、SELECT, UPDATE 及び DELETE でページにアクセスした時です。
- バックグラウンドで実行される autovacuum プロセスでは、まとめてお掃除が実行されます。

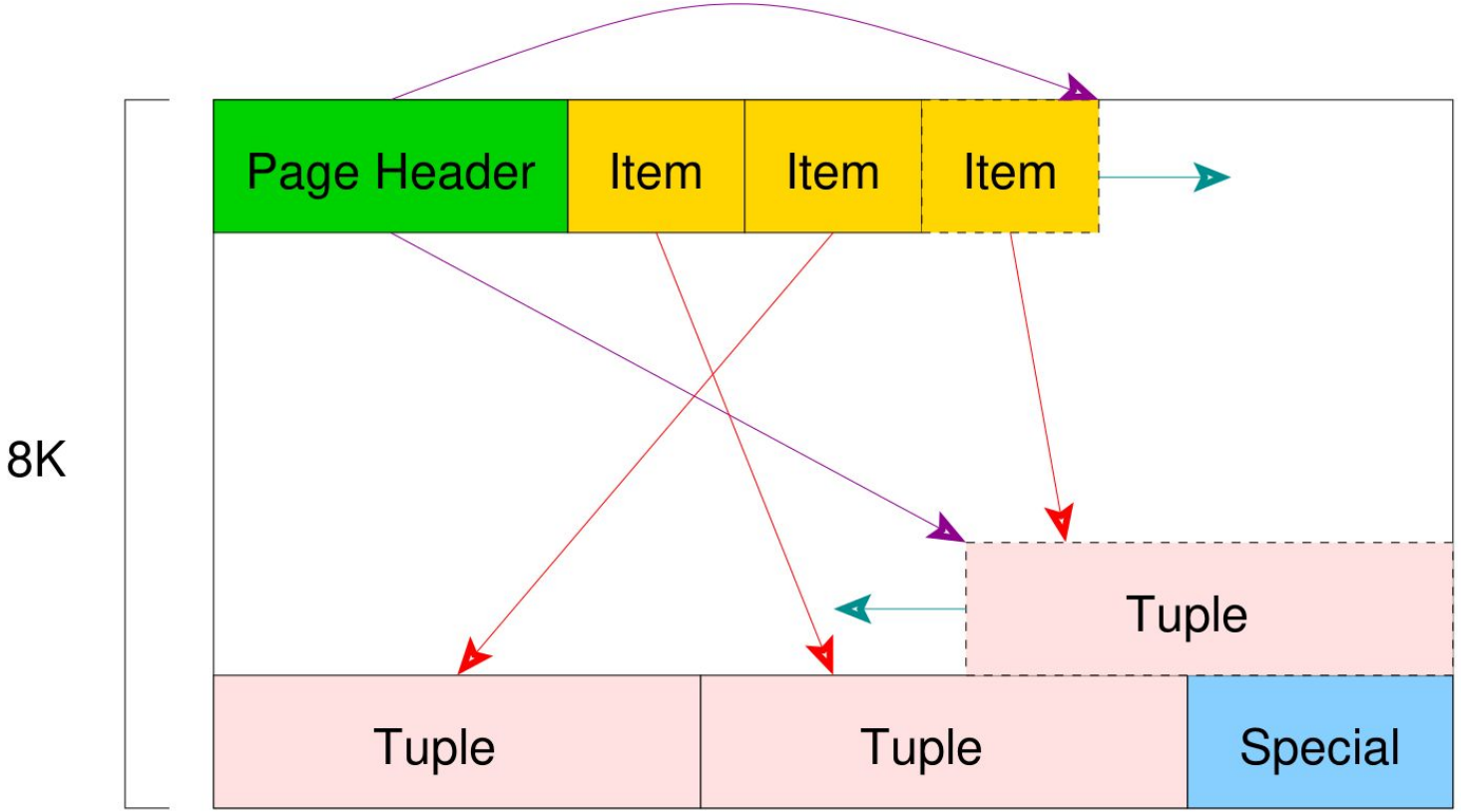
手動で VACUUM を起動してお掃除をすることもできます。

種々のお掃除の局面

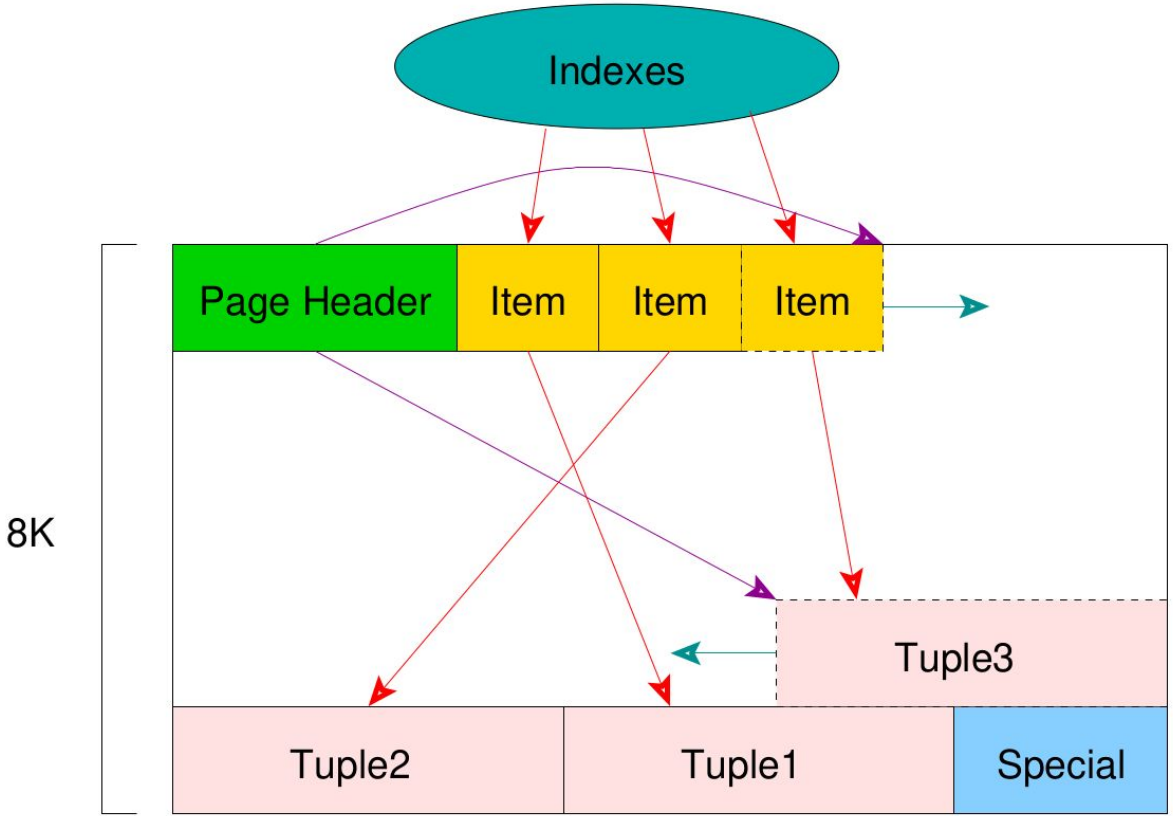
お掃除では、いくつかの異なるエンティティで使われている空間の再利用が行われます:

- ヒープタプル/行 (最も大きい)
- ヒープアイテムポインタ (最も小さい)
- インデックスエントリ

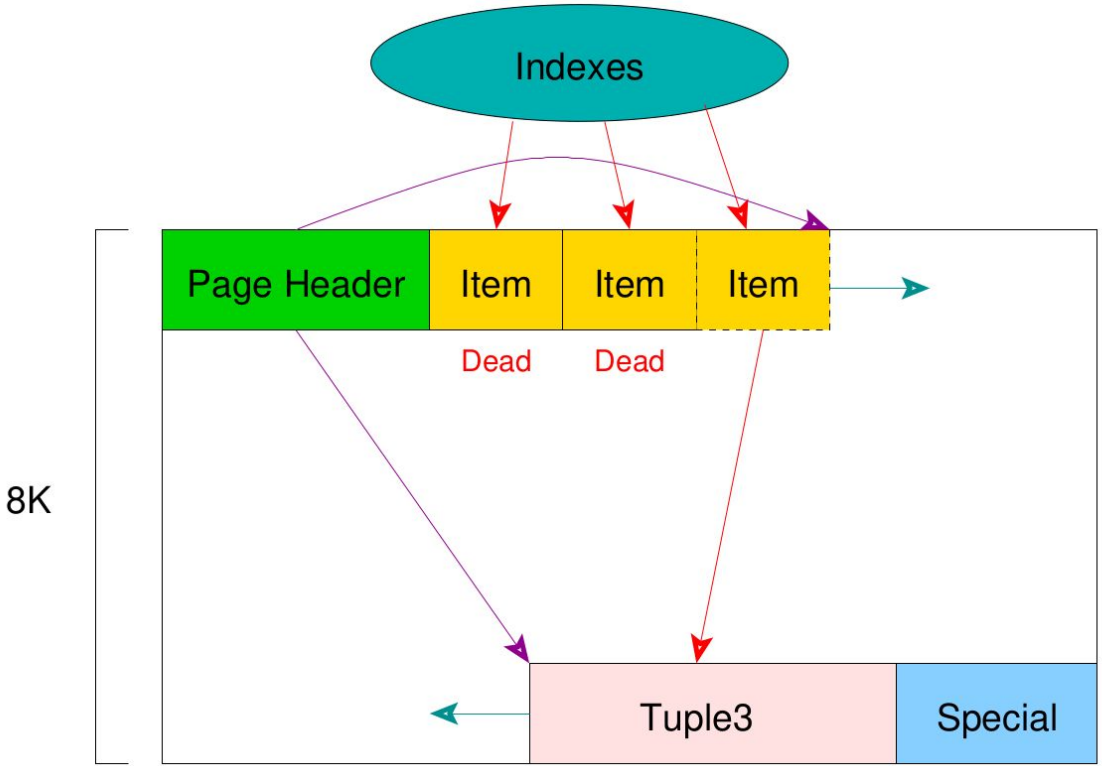
ヒープページの内部



インデックスが指すのはアイテムであり、タプルではありません

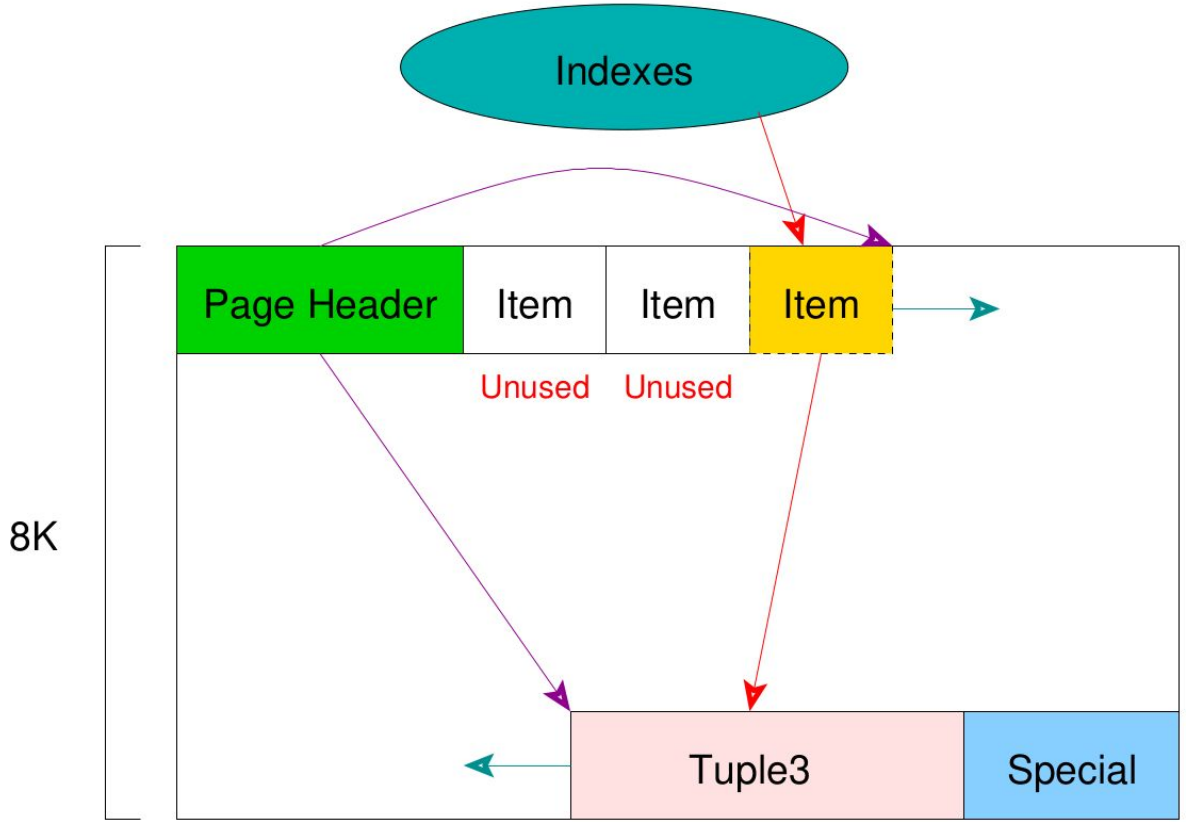


ヒープタプル空間の再利用



インデックスがまだ指しているので、アイテムポインタの再利用はここではできません

後で VACUUM がアイテムの再利用を行います



VACUUM がインデックスのお掃除をすると、「無効」だったアイテムは「未使用」になります

削除した行のお掃除

```
TRUNCATE mvcc_demo;

-- それぞれのページの空きを 10% 未満にする
INSERT INTO mvcc_demo SELECT 0 FROM generate_series(1, 240);

-- 空きスペースの割合 (%) を計算する
SELECT (100 * (upper - lower) /
          pagesize::float8)::integer AS free_pct
FROM page_header(get_raw_page('mvcc_demo', 0));
 free_pct
----------
         6

INSERT INTO mvcc_demo VALUES (1);
```

削除した行のお掃除

```
SELECT * FROM mvcc_demo_page0

OFFSET 240;
  ctid   | case   | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,241) | Normal | 5430 |    0 | (0,241)

DELETE FROM mvcc_demo WHERE val > 0;

INSERT INTO mvcc_demo VALUES (2);

SELECT * FROM mvcc_demo_page0
OFFSET 240;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,241) | Normal | 5430 | 5431 | (0,241)
 (0,242) | Normal | 5432 |    0 | (0,242)
```

削除した行のお掃除

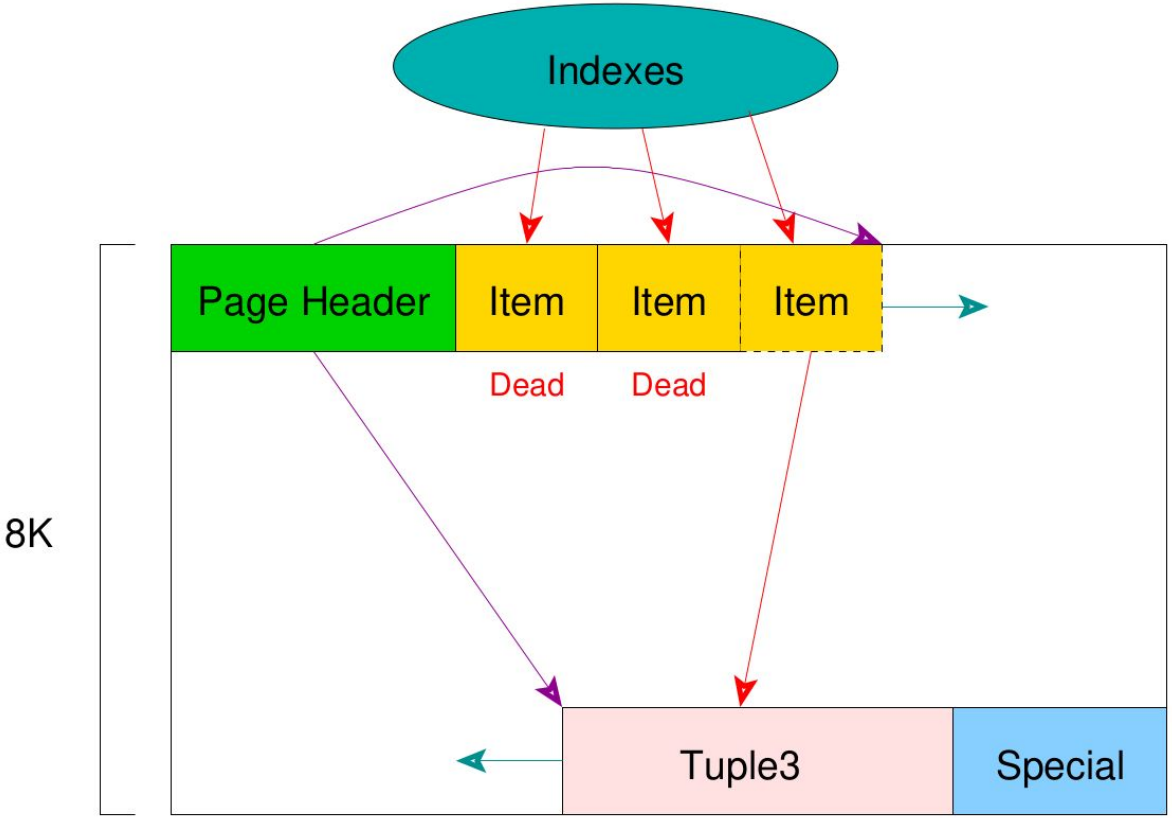
```
DELETE FROM mvcc_demo WHERE val > 0;

INSERT INTO mvcc_demo VALUES (3);

SELECT * FROM mvcc_demo_page0
OFFSET 240;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,241) | Dead   |      |      |
 (0,242) | Normal | 5432 | 5433 | (0,242)
 (0,243) | Normal | 5434 |    0 | (0,243)
```

通常のマルチユーザで使う場合、同じデータベースで実行中の他のトランザクションが無効になった行を参照する可能性があるため、お掃除は後になって実行されます。しかしながら、これは単に遅れて行われるだけで、振る舞いは同じです。

削除した行のお掃除

```
-- SELECT を使って強制的に単一ページのお掃除を行う
SELECT * FROM mvcc_demo
OFFSET 1000;
 val
-----

SELECT * FROM mvcc_demo_page0
OFFSET 240;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,241) | Dead   |      |      |
 (0,242) | Dead   |      |      |
 (0,243) | Normal | 5434 |    0 | (0,243)
```

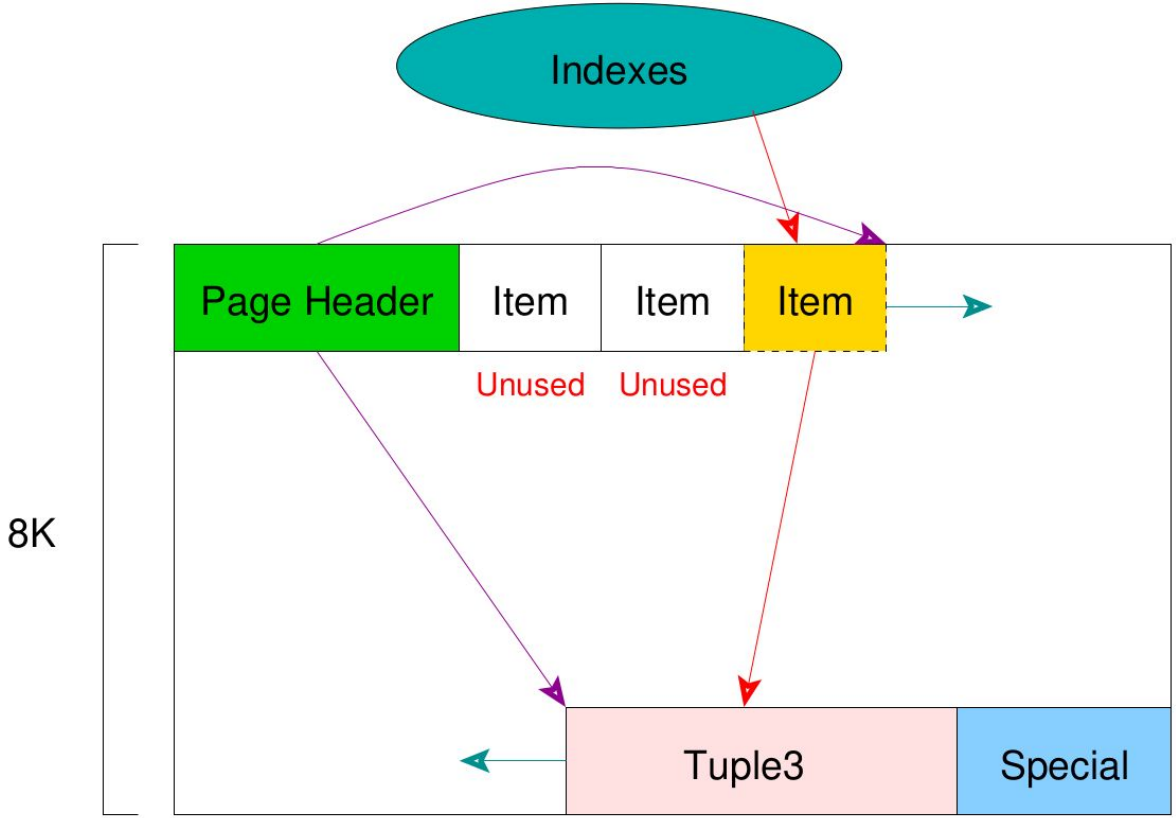
こんな風になります



削除した行のお掃除

```
SELECT pg_freespace('mvcc_demo');
 pg_freespace
--------------
(0,0)

VACUUM mvcc_demo;

SELECT * FROM mvcc_demo_page0
OFFSET 240;
 ctid    |  case   | xmin | xmax | t_ctid
---------+---------+------+------+---------
 (0,241) | Unused |      |      |
 (0,242) | Unused |      |      |
 (0,243) | Normal | 5434 |    0 | (0,243)
```

こんな風になります



フリースペースマップ

```
SELECT pg_freespace('mvcc_demo');
 pg_freespace
--------------
 (0,416)
```

VACUUM は、フリースペースマップの更新も行います。フリースペースマップでは、大量の空きスペース　があるページを記録しています。この情報を使って、 INSERT や一部の UPDATE (ページ境界をまたぐような場合) をどのページに行うかを決めます。単一ページのお掃除ではフリースペースマップの更新は行いません。

他のフリースペースマップの例

```
TRUNCATE mvcc_demo;

VACUUM mvcc_demo;

SELECT pg_freespace('mvcc_demo');
 pg_freespace
--------------
```

他のフリースペースマップの例

```
INSERT INTO mvcc_demo VALUES (1);

VACUUM mvcc_demo;

SELECT pg_freespace('mvcc_demo');
 pg_freespace
--------------
 (0,8128)

INSERT INTO mvcc_demo VALUES (2);

VACUUM mvcc_demo;

SELECT pg_freespace('mvcc_demo');
 pg_freespace
--------------
 (0,8096)
```

他のフリースペースマップの例

```
DELETE FROM mvcc_demo WHERE val = 2;

VACUUM mvcc_demo;

SELECT pg_freespace('mvcc_demo');
 pg_freespace
--------------
 (0,8128)
```

VACUUM では、ファイルの最後のページも除去します

```
DELETE FROM mvcc_demo WHERE val = 1;

VACUUM mvcc_demo;

SELECT pg_freespace('mvcc_demo');
 pg_freespace
--------------

SELECT pg_relation_size('mvcc_demo');
 pg_relation_size
------------------
                0
```

VACUUM FULL を使えば、テーブルファイルのサイズを最小限度まで小さくできますが、テーブル全体の排他ロックが必要になります。

単一ページで更新された古い行の最適なお掃除

UPDATEでの古いタプルが占めるストレージ空間は、削除した行と同様に再利用することができます。しかしながら、ある種の行更新ではアイテムの再利用もできるようになっています。すなわち、場合によっては古いUPDATE のアイテムの再利用ができるのです。

この場合、VACUUM が必要な「無効」アイテムは使いません。

具体的には、このようなアイテムの再利用はHOT 更新 (heap-only-tuple) チェインを使って行います。ここでは、チェインは同じヒープページ上にあり、チェインの中のインデックスの値はすべて同じになります。

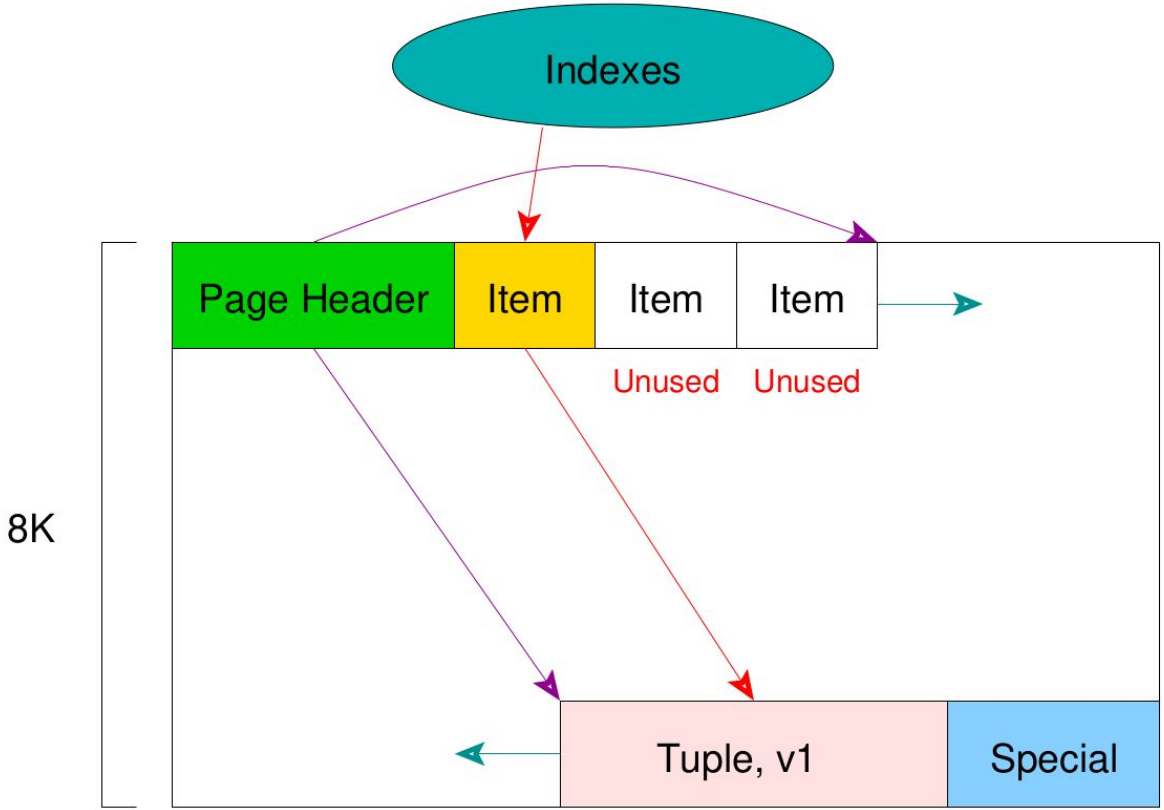
HOT 更新行の単一ページのお掃除

HOT 更新のアイテムは、それがチェインの途中にある場合は解放することができます(「未使用」になります)。チェインの先頭や最後の場合にはこれはできません。チェインの先頭にあるのは特別な「リダイレクト」アイテムポインタで、インデックスはこれを参照しています。全てのインデックスの値が HOT/リダイレクトチェインでは同じであるため、これが可能になります。

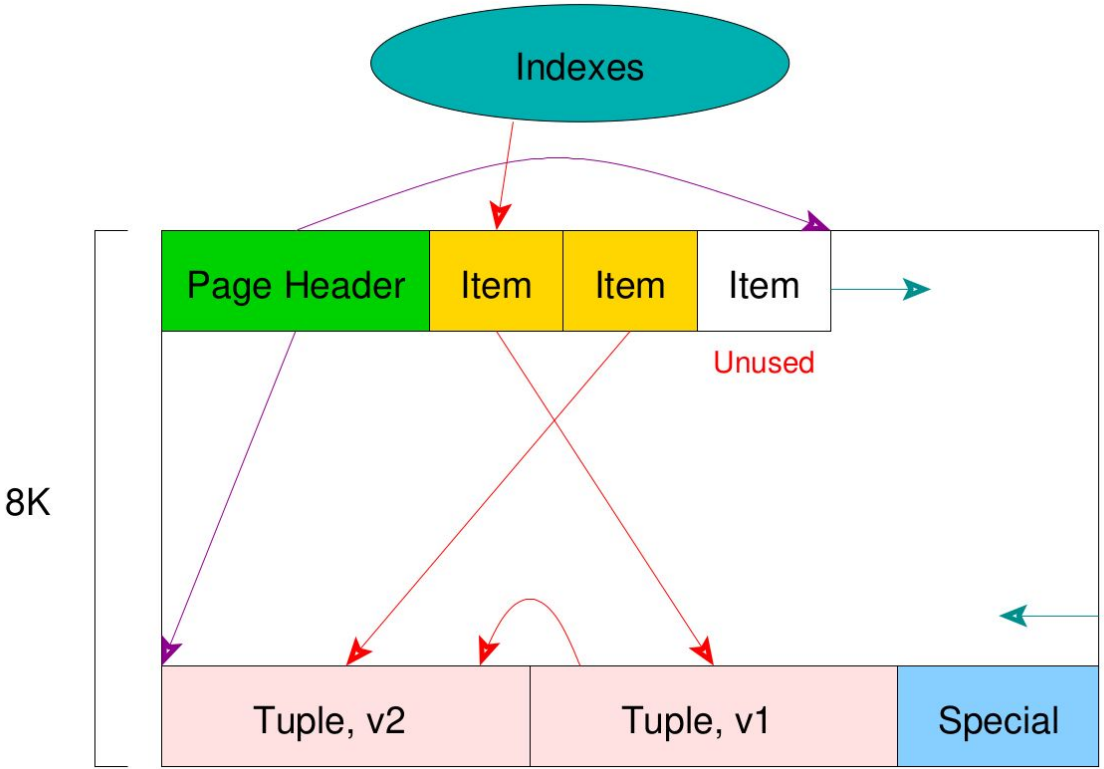
HOT チェインでのインデックス生成は複雑です。これらのチェインでは新たにインデックスが張られたカラムの値に矛盾がある場合があるからです。これは、インデックスをHOTチェインの最後に張り、インデックスを作ったトランザクションの後で開始したトランザクションでのみ使われるようにすることで実行することができます。

具体的には、インデックスを生成した後のトランザクションは、MVCC の可視性ルールによって、矛盾する HOT チェインの値が見えないようになっているのです。これらのトランザクションからはチェインの最後しか見えません。

単一行の初期状態

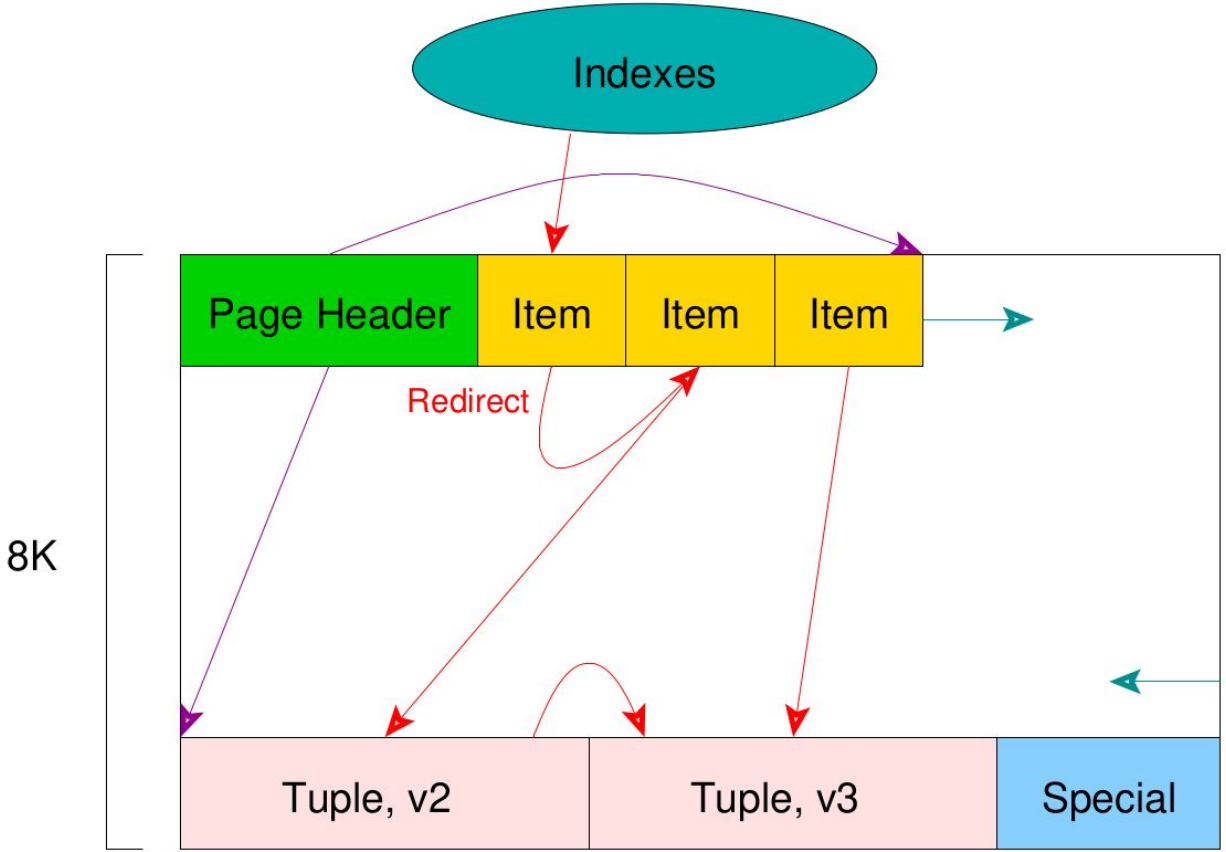


UPDATE で更新後の新しい行を追加

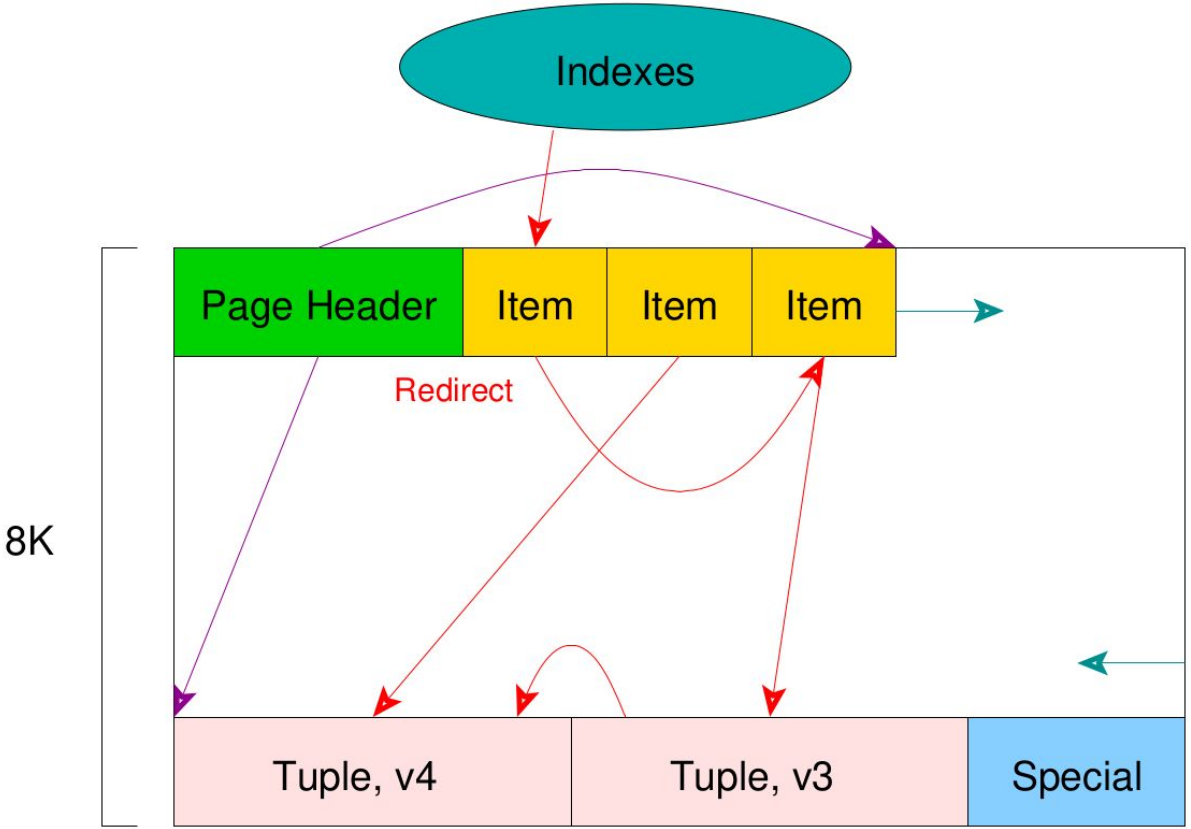


インデックスエントリは追加されません。インデックスは　HOT チェインの先頭を指しているだけだからです。

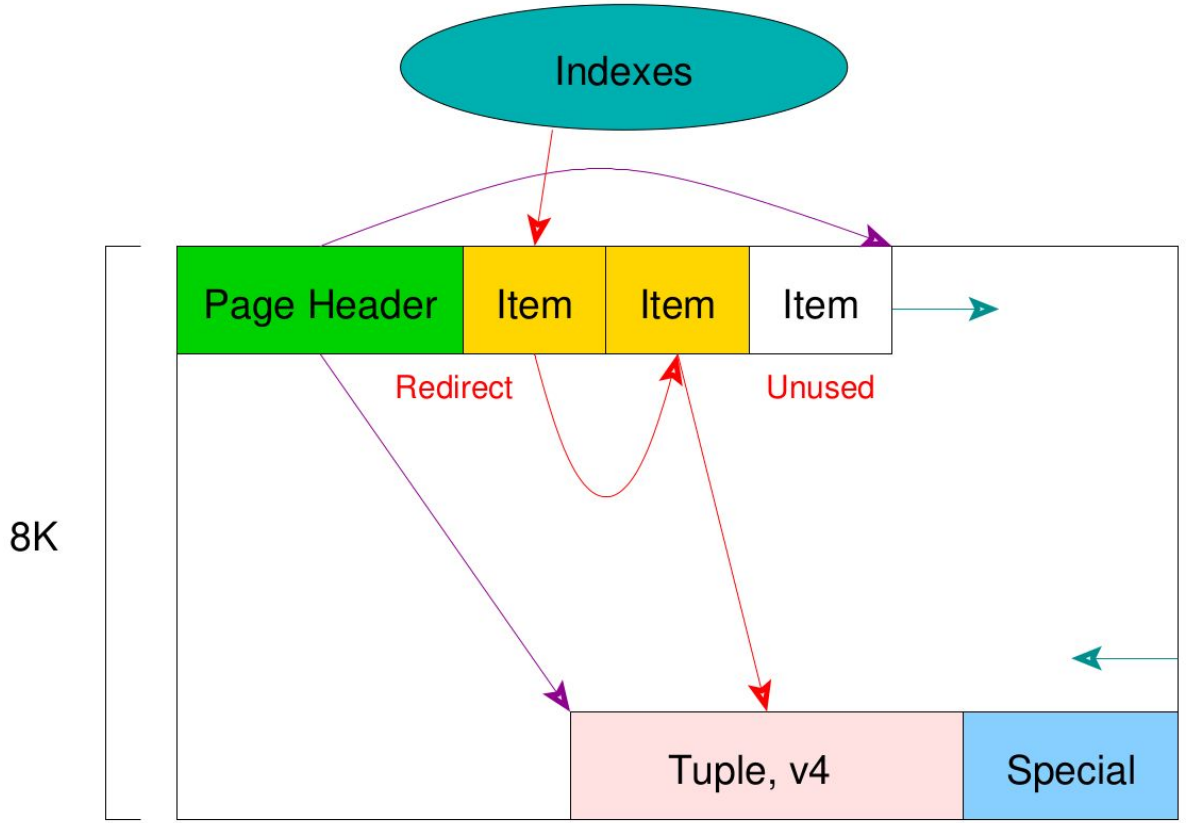
リダイレクトを使ってインデックスを正しく保ちます



UPDATE で他の古い行が置き換えられます



後で全ての古い UPDATE 行が除去されます



更新前の古い行のお掃除

```
TRUNCATE mvcc_demo;

INSERT INTO mvcc_demo SELECT 0 FROM generate_series(1, 240);

INSERT INTO mvcc_demo VALUES (1);

SELECT * FROM mvcc_demo_page0
OFFSET 240;
  ctid   | case   | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,241) | Normal | 5437 |    0 | (0,241)
```

更新前の古い行のお掃除

```
UPDATE mvcc_demo SET val = val + 1 WHERE val > 0;

SELECT * FROM mvcc_demo_page0
OFFSET 240;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,241) | Normal | 5437 | 5438 | (0,242)
 (0,242) | Normal | 5438 |    0 | (0,242)
```

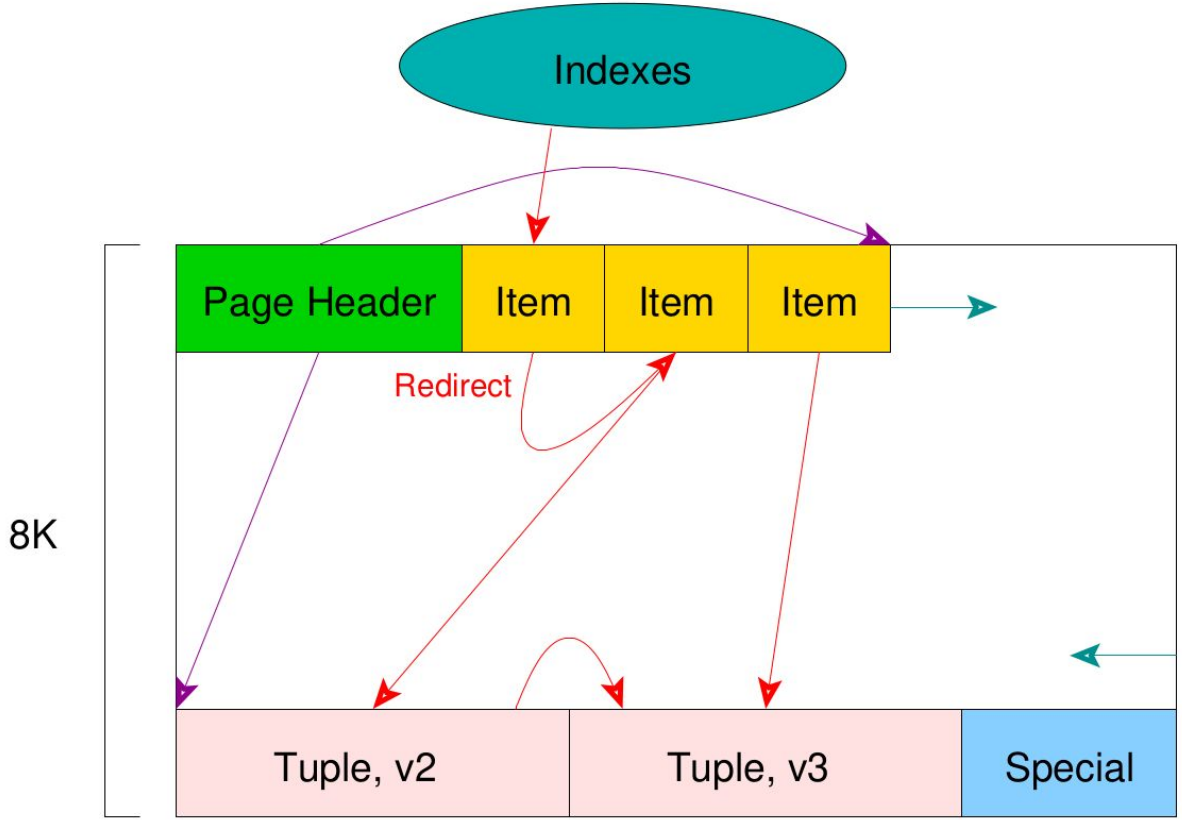
更新前の古い行のお掃除

```
UPDATE mvcc_demo SET val = val + 1 WHERE val > 0;

SELECT * FROM mvcc_demo_page0
OFFSET 240;
```

```
  ctid   |    case          | xmin | xmax | t_ctid
---------+------------------+------+------+---------
 (0,241) | Redirect to 242 |      |      |
 (0,242) | Normal           | 5438 | 5439 | (0,243)
 (0,243) | Normal           | 5439 |    0 | (0,243)
```

インデックスエントリは追加されません。インデックスは HOT チェインの先頭を指しているだけだからです。

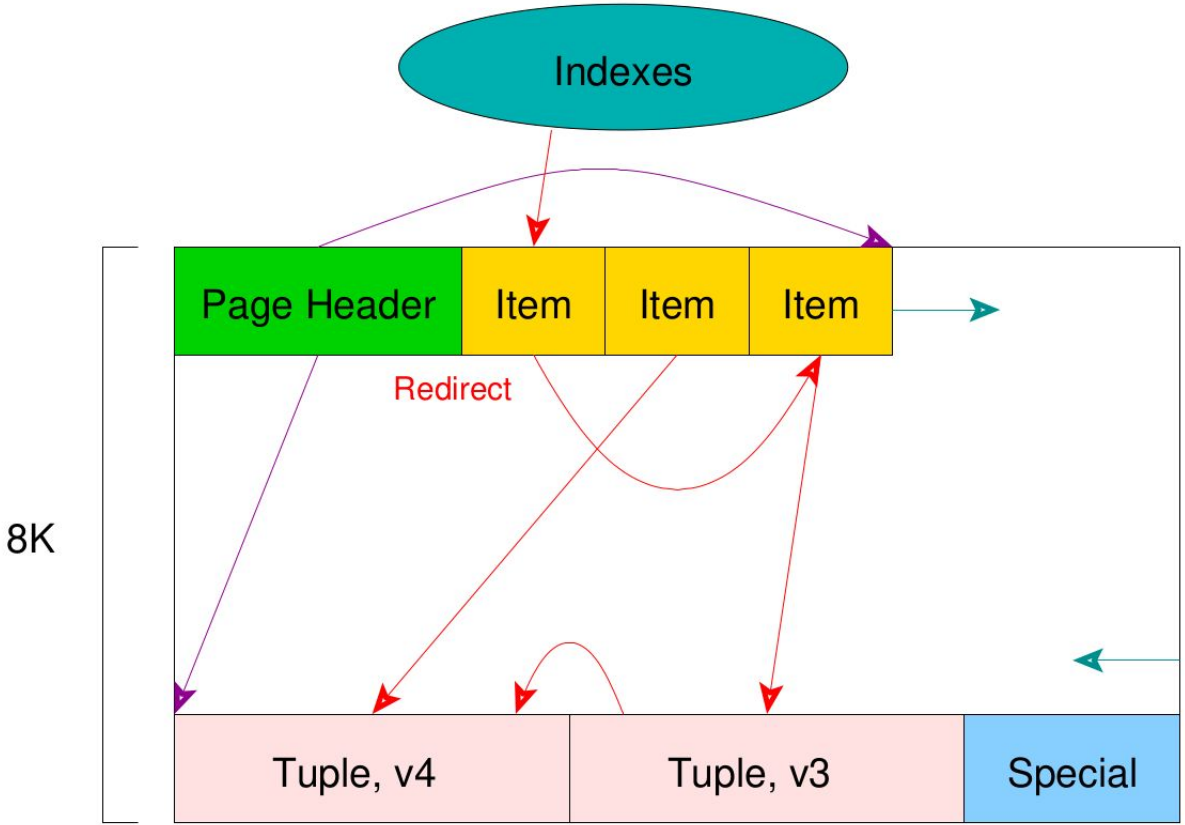
このようになります



更新前の古い行のお掃除

```
UPDATE mvcc_demo SET val = val + 1 WHERE val > 0;

SELECT * FROM mvcc_demo_page0
OFFSET 240;

  ctid   |    case          | xmin | xmax | t_ctid
---------+------------------+------+------+---------
 (0,241) | Redirect to 243 |      |      |
 (0,242) | Normal           | 5438 | 5439 | (0,243)
 (0,243) | Normal           | 5439 |    0 | (0,243)
```

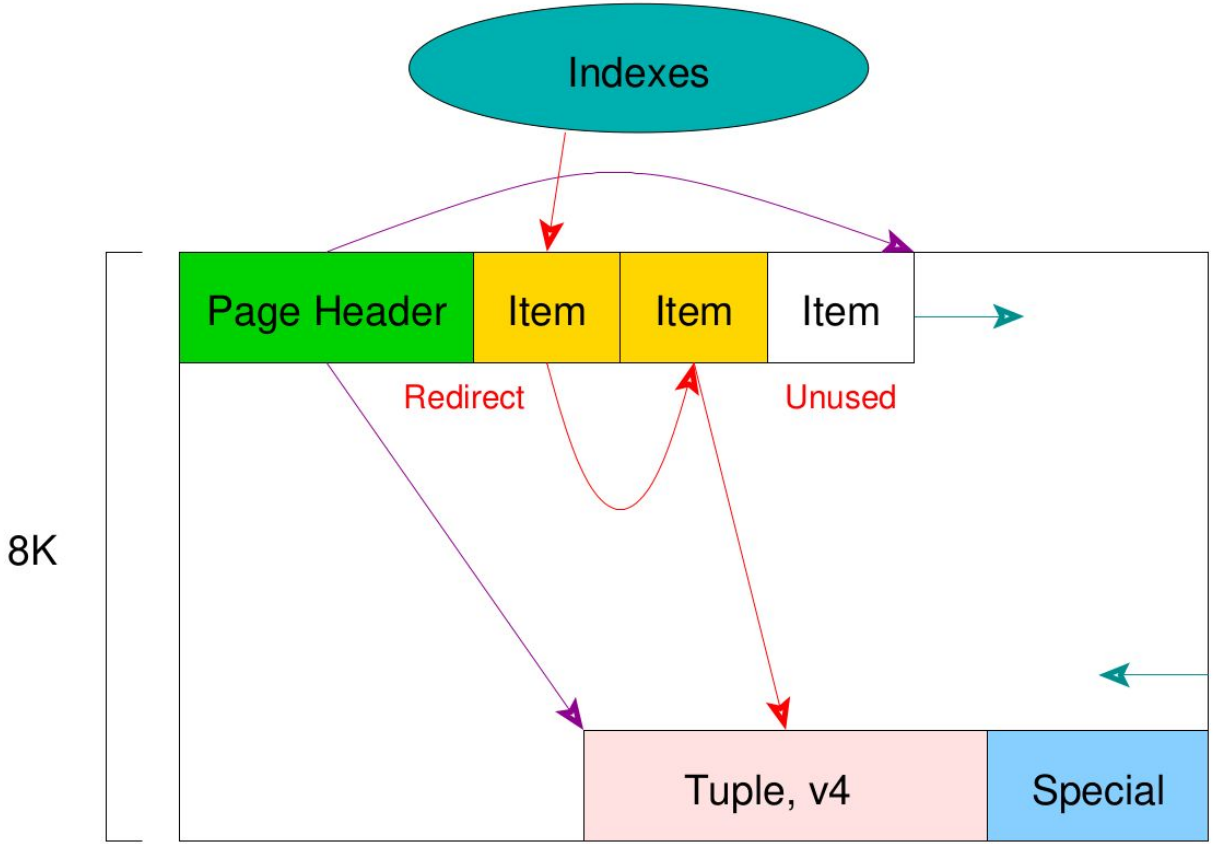
このようになります



更新前の古い行のお掃除

```
-- トランザクションがコミットされたので、HOTチェインではtid を
-- 「未使用」とすることができます
SELECT * FROM mvcc_demo
OFFSET 1000;
val
-----

SELECT * FROM mvcc_demo_page0
OFFSET 240;

  ctid   |    case          | xmin | xmax | t_ctid
---------+------------------+------+------+---------
 (0,241) | Redirect to 242 |      |      |
 (0,242) | Normal           | 5438 | 5439 | (0,243)
 (0,243) | Unused           |      |      |
```

このようになります



VACUUM ではリダイレクトは除去されません

```
VACUUM mvcc_demo;

SELECT * FROM mvcc_demo_page0
OFFSET 240;
  ctid   |       case        | xmin | xmax | t_ctid
---------+-------------------+------+------+---------
 (0,241) | Redirect to 242 |      |      |
 (0,242) | Normal            | 5440 |    0 | (0,242)
 (0,243) | Unused            |      |      |
```

手動 VACUUM によるお掃除

```
TRUNCATE mvcc_demo;

INSERT INTO mvcc_demo VALUES (1);
INSERT INTO mvcc_demo VALUES (2);
INSERT INTO mvcc_demo VALUES (3);

SELECT ctid, xmin, xmax
FROM mvcc_demo_page0;
 ctid  | xmin | xmax
-------+------+------
 (0,1) | 5442 |    0
 (0,2) | 5443 |    0
 (0,3) | 5444 |    0

DELETE FROM mvcc_demo;
```

手動 VACUUM によるお掃除

```
SELECT ctid, xmin, xmax
FROM mvcc_demo_page0;
 ctid  | xmin | xmax
-------+------+------
 (0,1) | 5442 | 5445
 (0,2) | 5443 | 5445
 (0,3) | 5444 | 5445

-- autovacuum を動かすには更新量が足りない

VACUUM mvcc_demo;

SELECT pg_relation_size('mvcc_demo');
 pg_relation_size
------------------
                0
```

インデックスがある場合の UPDATE の問題

インデックスのあるカラムを更新した場合はアイテムの「リダイレクト」を使うことができません。チェインが全てのインデックスで使えるものでなければならず、そのためリダイレクト /HOT 更新ではインデックス値が変わったことで追加のインデックスエントリを作ることができないからです。

この場合には、アイテムポインタは DELETE と同様に「無効」にすることしかできません。

これまで示した UPDATE の例では、インデックスのあるカラムを更新していないことにご注意ください。

mvcc_demo カラムにインデックスを作る

```
CREATE INDEX i_mvcc_demo_val on mvcc_demo (val);
```

インデックスのあるカラムの更新

```
TRUNCATE mvcc_demo;

INSERT INTO mvcc_demo SELECT 0 FROM generate_series(1, 240);
INSERT INTO mvcc_demo VALUES (1);

SELECT * FROM mvcc_demo_page0
OFFSET 240;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,241) | Normal | 5449 |    0 | (0,241)
```

インデックスのあるカラムの更新

```
UPDATE mvcc_demo SET val = val + 1 WHERE val > 0;

SELECT * FROM mvcc_demo_page0
OFFSET 240;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,241) | Normal | 5449 | 5450 | (0,242)
 (0,242) | Normal | 5450 |    0 | (0,242)

UPDATE mvcc_demo SET val = val + 1 WHERE val > 0;

SELECT * FROM mvcc_demo_page0

OFFSET 240;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,241) | Dead   |      |      |
 (0,242) | Normal | 5450 | 5451 | (0,243)
 (0,243) | Normal | 5451 |    0 | (0,243)
```

インデックスのあるカラムの更新

```
UPDATE mvcc_demo SET val = val + 1 WHERE val > 0;

SELECT * FROM mvcc_demo_page0
OFFSET 240;
  Ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,241) | Dead   |      |      |
 (0,242) | Dead   |      |      |
 (0,243) | Normal | 5451 | 5452 | (0,244)
 (0,244) | Normal | 5452 |    0 | (0,244)
```

インデックスのあるカラムの更新

```
SELECT * FROM mvcc_demo
OFFSET 1000;
 val
-----

SELECT * FROM mvcc_demo_page0
OFFSET 240;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,241) | Dead   |      |      |
 (0,242) | Dead   |      |      |
 (0,243) | Dead   |      |      |
 (0,244) | Normal | 5452 |    0 | (0,244)
```

インデックスのあるカラムの更新

```
VACUUM mvcc_demo;

SELECT * FROM mvcc_demo_page0
OFFSET 240;

  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,241) | Unused |      |      |
 (0,242) | Unused |      |      |
 (0,243) | Unused |      |      |
 (0,244) | Normal | 5452 |    0 | (0,244)
```

お掃除のまとめ

お掃除の方法	きっかけ	範囲	ヒープタプル再利用	非 HOT アイテムの状態	HOTアイテムの状態	インデックスのお掃除	FSMの更新
単一ページ	SELECT, UPDATE, DELETE	単一ヒープページ	Yes	無効	未使用	No	No
VACUUM	autovacuum または手動	可能な全てのヒープページ	Yes	未使用	未使用	Yes	Yes

お掃除ができるのは、タプルが見える実行中のトランザクションがない場合のみです。

HOTアイテムはUPDATEのチェインで、単一ページに閉じており、同一のインデックスカラムの値を持っているものです。

通常の利用形態では、単一ページのお掃除で主要なお掃除が実行されますが、 VACUUM では「無効」なアイテムポインタを再利用でき、不要なインデックスエントリを除去でき、フリースペースマップ (FSM) を更新することができます。

SERIALIZABLE 分離レベル

- Postgres は、厳密な SERIALIZABLE 分離レベルを実装した唯一の RDBMS です。
- 他の RDBMS で SERIALIZABLE と呼んでいる分離レベルは、Postgres では REPEATABLE READ 分離レベルに相当します
- REPEATABLE READ が分離レベルとして不十分な例、Postgres での実装、代替案についてお話します

不都合な例

次の送金トランザクションを考えます

1. 送金元の残金を読む
2. 送金先の残金を読む
3. 送金元から送金額を引く
4. 送金先に送金額を足す

この送金の例で、A → B、B → A の送金を逐次及び並列に実行してみます。

A の残高が 100万円、B の残高が200万円、送金額は A→B が 20万円、B → A が 30万円とします。

不都合な例

逐次実行 Tx1, Tx2 の順

Tx1:

1. BEGIN
2. R(A) → 100万, R(B) → 200万
3. W(A) → 80万, W(B)→ 220万
4. COMMIT

Tx2:

1. BEGIN
2. R(B)→220万, R(A)→80万
3. W(B)→190万, W(A)→110万
4. COMMIT

不都合な例

並列実行

Tx1:

1. BEGIN
2. R(A) → 100万, R(B) → 200万
3. W(A) → 80万, W(B)→ 220万
4. COMMIT

Tx2:

1. BEGIN
2. R(A) → 100万, R(B) → 200万
3. W(A) → 130万, W(B)→ 170万
4. COMMIT

Tx1 の結果が消えてしまった！

不都合の原因

誤った読み出しデータに基づいて書き込みを行ったためで、以下の並列シーケンスに矛盾がある

R(A) → W(B)
R(B) → W(A)

これを検出して、片方のトランザクションをアボートさせる必要がある。

MVCC のスナップショットではこの検出はできない

検出方法

Predicate Lock とよばれる手法:

- どのトランザクションがどのタプルを読んだかを記録する
- 書き込みを行う時は、この書き込みに矛盾する読み込みを行っているかどうかをチェックする
- 読み込みを行うときは、この読み込みと矛盾する書き込みを行っているかどうかをチェックする

これは、MVCC とは独立した実装になっている。(src/backend/storage/lmgr/predicate.c)

- ロックの単位は、行・ページ・テーブル
- ロック数が多くなると、行 → ページ → テーブルとロック単位をエスカレーションする

開発者によれば、上記のオーバヘッドは 5% 位とのこと (PGCon での話)。

SERIALIZABLE vs. SELECT FOR {UPDATE|SHARE}

- SELECT FOR {UPDATE|SHARE} を使っても同様のことが可能
  - タプルをロックして一部逐次処理に切り替えている
- アプリケーションに手が入る
  - 更新無矛盾性の補償はアプリケーションの責任
- オーバヘッドは SERIALIZABLE の方が低いと言われている

ご清聴ありがとうございます

Koichi Suzuki

koichi.suzuki@enterprisedb.com