GREENPLUM DATABASE

深入浅出Greenplum内核系列直播

揭秘Greenplum MVCC并发控制

钉钉直播

8月28日 16：00–17：00

Greenplum中文社区

https://cn.greenplum.org

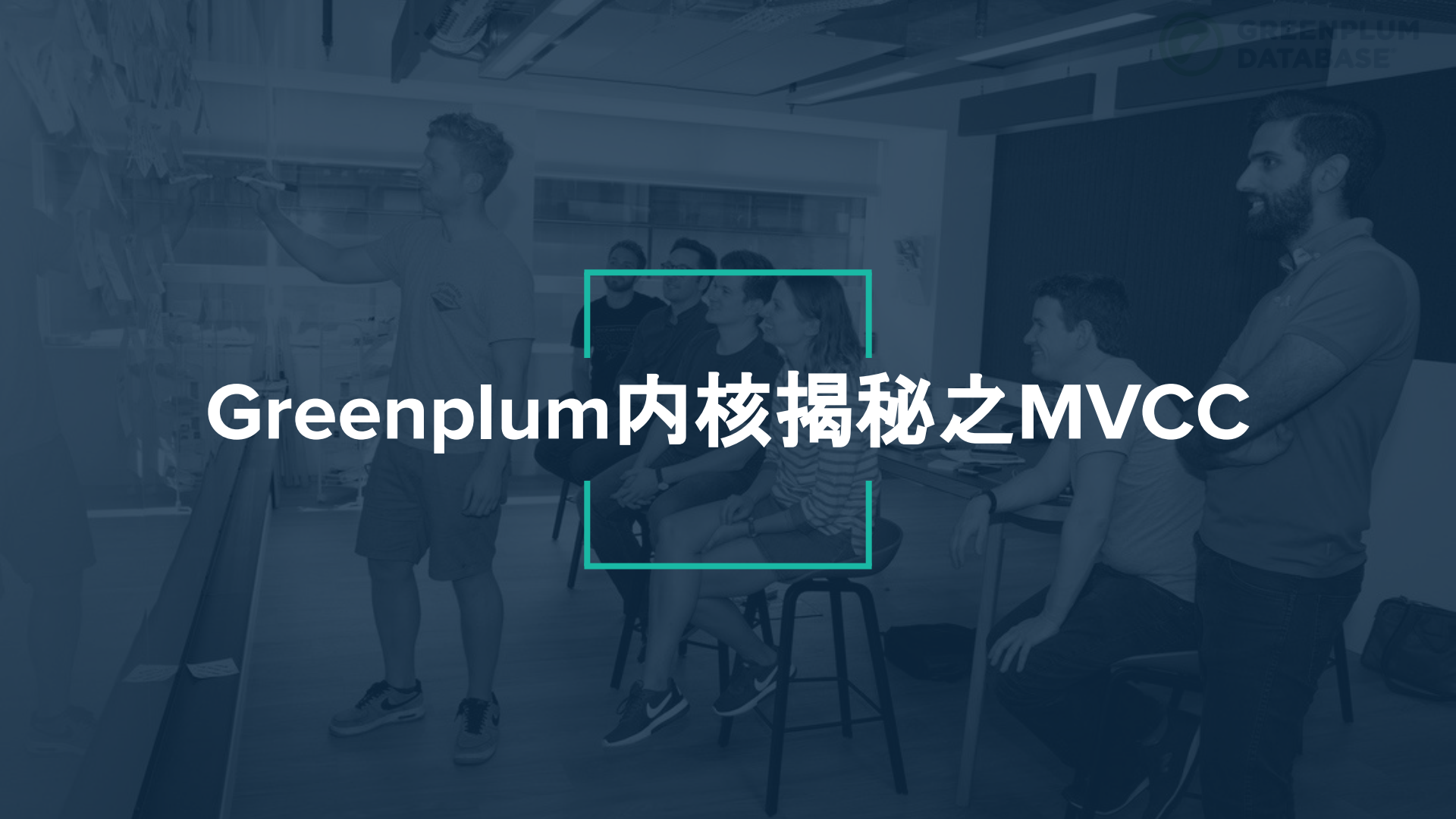
博文 · 资料 · 文档 · 项目

GREENPLUM DATABASE®

微信技术讨论群
添加入群小助手：gp_assistant

微信公众号
技术干货、行业热点、活动预告

欢迎访问Greenplum中文社区：cn.greenplum.org



Greenplum内核揭秘之MVCC

GREENPLUM DATABASE®

为什么我们需要了解MVCC

了解并发查询的行为

管理MVCC对于性能的影响

了解存储空间的重用

GREENPLUM DATABASE®

常用的并发控制技术

Multi-version Concurrency Control (MVCC)

Strict Two-Phase Locking (S2PL)

Optimistic Concurrency Control (OCC)

GREENPLUM DATABASE®

事务

事务的本质是将多个步骤捆绑为一个“全有或全无”操作

步骤之间的中间状态对于其他并发事务是不可见的

如果某些故障导致事务无法完成，则所有步骤都不会影响数据库

GREENPLUM DATABASE

隔离级别

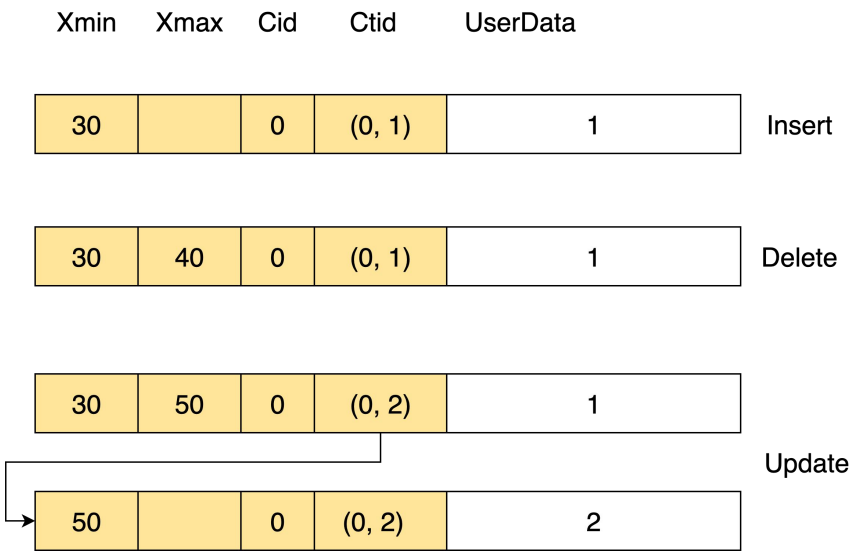
Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read	Serialization Anomaly
Read uncommitted	Allowed, but not in PG	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible	Possible
Repeatable read	Not possible	Not possible	Allowed, but not in PG	Possible
Serializable	Not possible	Not possible	Not possible	Not possible

MVCC是什么

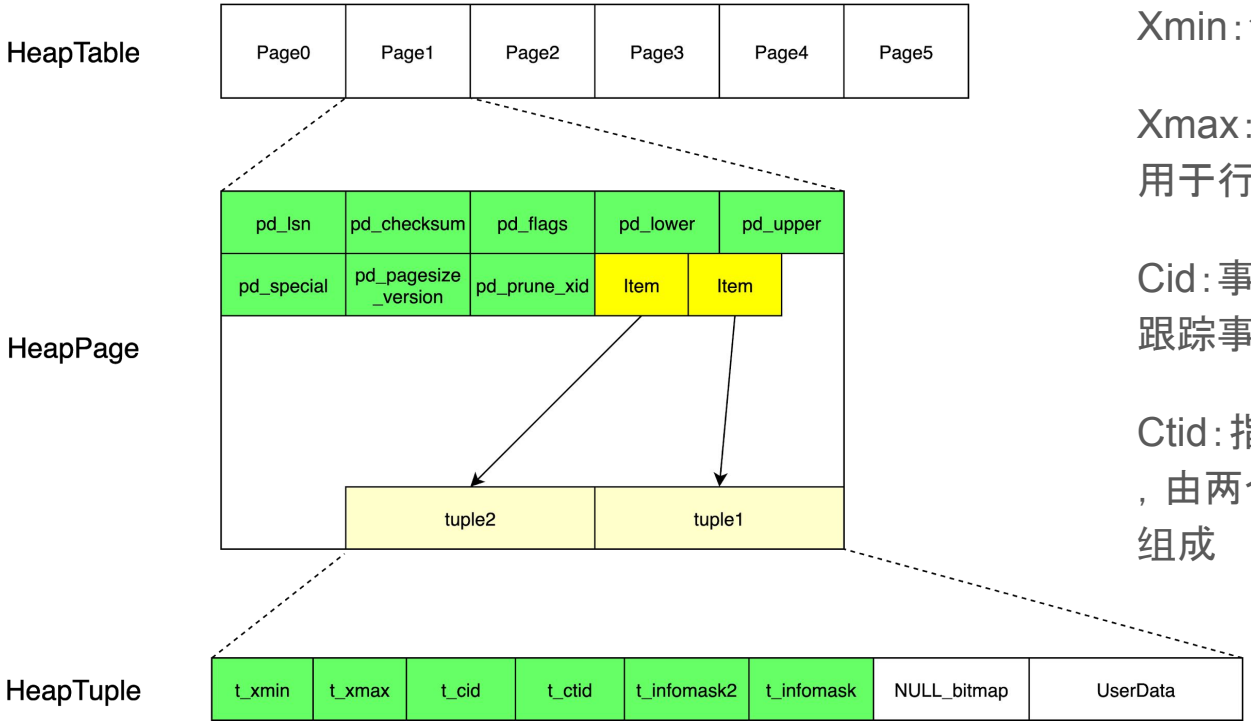
每个写操作都会创建数据项目的新版本，同时保留旧版本

这种方法允许Greenplum在读写同时进行的情况下仍然能提供高并发特性

MVCC最大的特点是读操作不会阻塞写操作，写操作也不会阻塞读操作



Heap表页面布局



Xmin：创建Tuple的事务ID

Xmax：删除Tuple的事务ID，有时用于行锁

Cid：事务内的查询命令编号，用户跟踪事务内部的可见性

Ctid：指向下一个版本tuple的指针，由两个成员blocknumber：offset组成

GREENPLUM DATABASE

快照

MVCC的快照用来控制哪个元组对于当前查询可见

在READ COMMITTED隔离级别，每个查询开始时生成快照。在REPEATABLE READ隔离级别，在每个事务开始时生成快照

快照理论上是一个正在运行的事务列表

Greenplum使用快照判断一个事务是否已提交

快照

Xmin：所有小于Xmin的事务都已经提交

Running：正在执行的事务列表

Xmax：所有大于等于Xmax的事务都未提交

快照和可见性

Xmin	Xmax	Data	
30		1	可见
50		2	不可见
110		3	不可见
30	80	1	不可见
30	75	2	可见
30	120	3	可见

顺序扫描

快照

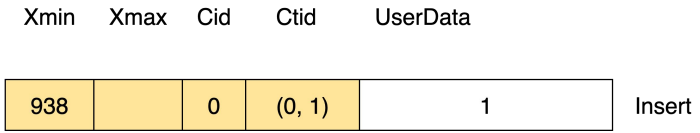
XMAX：100

XMIN：40

RUNNING：50 75 90

Insert

```
demo=# create table mvcc(val int);
CREATE TABLE

demo=# insert into mvcc values(1);
INSERT 0 1

demo=# select xmin, xmax, * from mvcc;
 xmin | xmax |   val
------+------+---------
  938 |    0     |    1
(1 row)
```



Delete

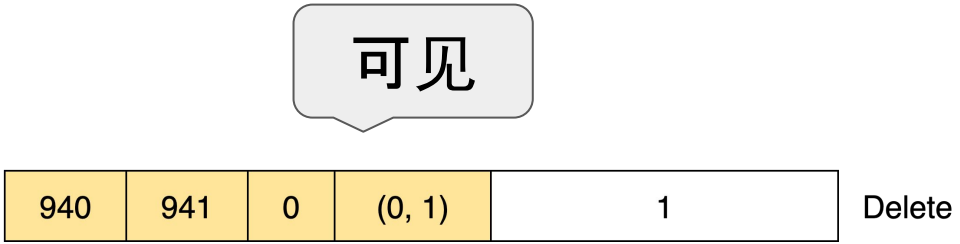
Session 1

```
demo=# insert into mvcc values(1);
INSERT 0 1

demo=# select xmin, xmax, * from mvcc;
 xmin | xmax |   val
------+------+---------
  940 |    0     |    1
(1 row)

demo=# begin;
BEGIN

demo=# delete from mvcc;
DELETE 1
```

Session 2

```
demo=# select xmin, xmax, * from mvcc;
 xmin | xmax |   val
------+------+---------
  940 |  941   |      1
(1 row)
```

Delete



快照

XMIN：900

XMAX：950

RUNNING：941

Update

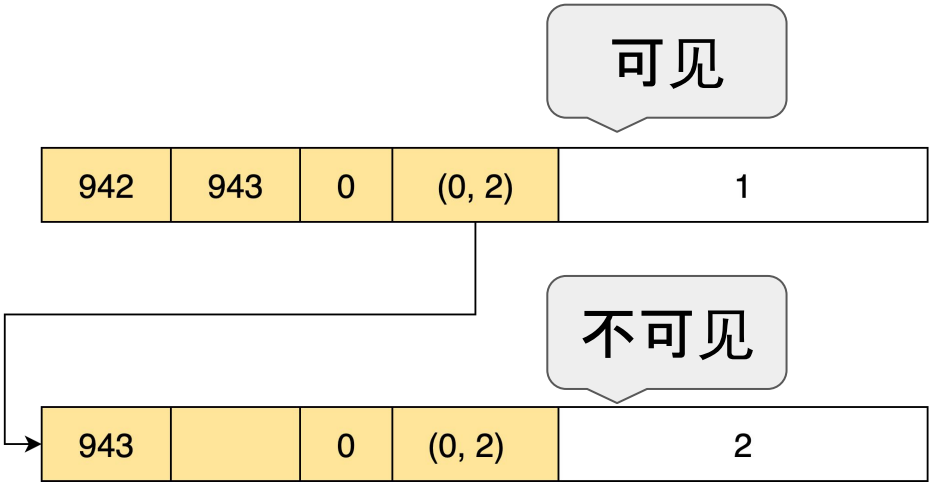
Session 1

```
demo=# insert into mvcc values(1);
INSERT 0 1

demo=# select xmin, xmax, * from mvcc;
 xmin | xmax |   val
------+------+---------
  942 |    0     |     1
(1 row)

demo=# begin;
BEGIN

demo=# update mvcc set val = 2;
UPDATE 1

demo=# select xmin, xmax, * from mvcc;
 xmin | xmax |   val
------+------+---------
  943 |    0     |     2
(1 row)
```

Session 2

```
demo=# select xmin, xmax, * from mvcc;
 xmin | xmax |   val
------+------+---------
  942 |  943   |     1
(1 row)
```

Update



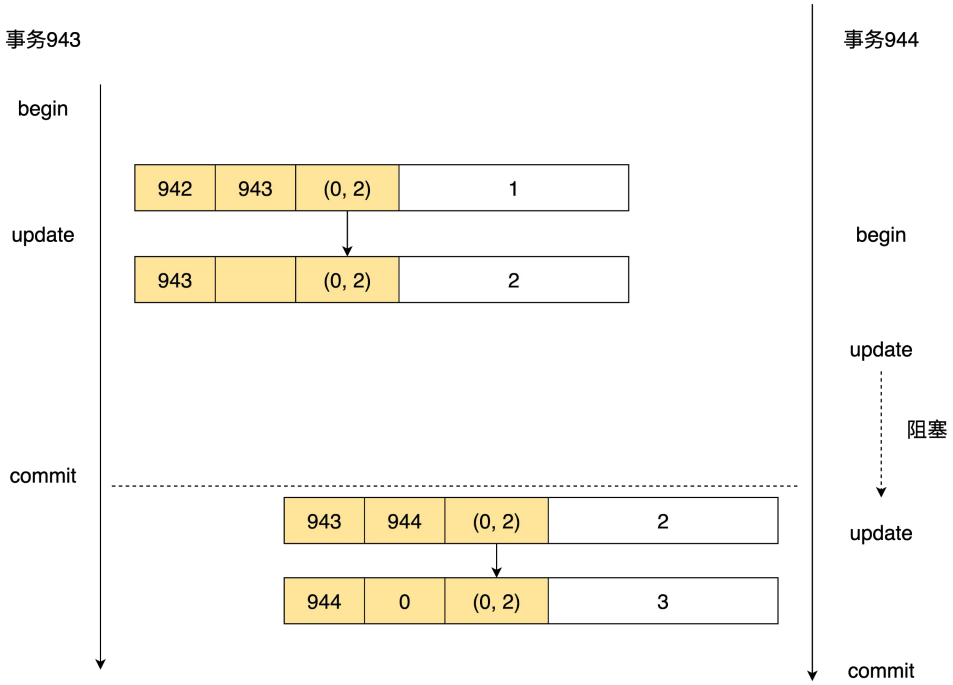
快照

XMIN：900

XMAX：950

RUNNING：943

并发Update


事务943
begin
942
943
(0, 2)
1
update
943
(0, 2)
2
commit
943
944
(0, 2)
2
944
0
(0, 2)
3
事务944
begin
update
阻塞
update
commit

GREENPLUM DATABASE

Abort

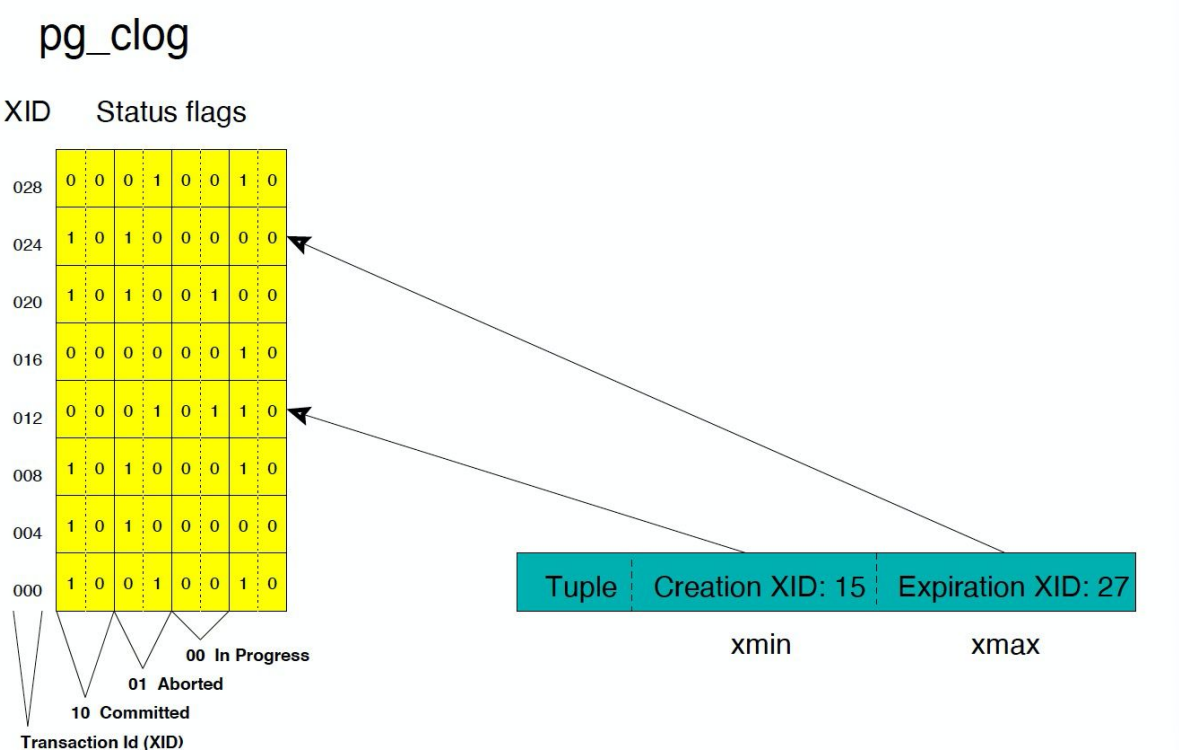
Session 1

```
demo=# insert into mvcc values(1);
INSERT 0 1

demo=# begin;
BEGIN

demo=# delete from mvcc;
DELETE 2

demo=# abort ;
ROLLBACK
```

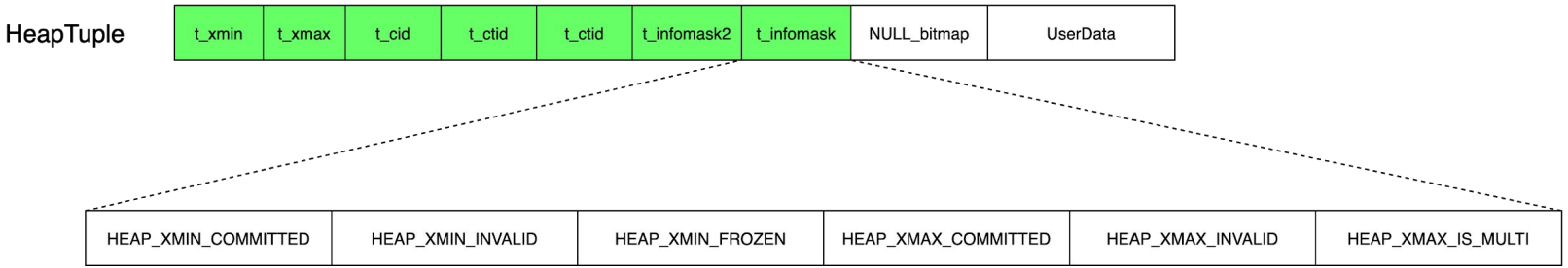
Session 2

```
demo=# select xmin, xmax, * from mvcc;
 xmin | xmax |   val
------+------+---------
  944 |  945 |     1
(1 row)
```

为什么不需要在Abort时修改Xmax



标记committed or INVALID



行锁

```
demo=# insert into mvcc values(1);
INSERT 0 1
demo=# begin;
BEGIN
demo=# select xmin, xmax, * from mvcc;
 xmin | xmax |   val
------+------+---------
  948 |    0     |   1
(1 row)

demo=# select xmin, xmax, * from mvcc for update;
 xmin | xmax |   val
------+------+---------
  948 |    0     |    1
(1 row)

demo=# select xmin, xmax, * from mvcc;
 xmin | xmax |   val
------+------+---------
  948 |  949   |    1
(1 row)
```

标志位HEAP_XMAX_EXCL_LOCK用来表示xmax是用做行锁而不是表示删除

GREENPLUM DATABASE

MVCC总结

xmin：创建元组的事务ID

xmax：使元组过期的事务ID，由Update或Delete语句设置。在某些情况下，用作行锁

cmin/cmax：用于标志事务内的可见性。如果在一个事务内先创建再过期，则使用combo command id

GREENPLUM DATABASE

MVCC的清理需求

在更新元组时会创建一个新的元组，所以旧的元组需要清理

在删除元组时，只会标记xmax，不会立即删除元组

失败的事务所创建的元组

GREENPLUM DATABASE®

何时执行清理操作

在查询操作访问到某个页面时，会清理这个页面

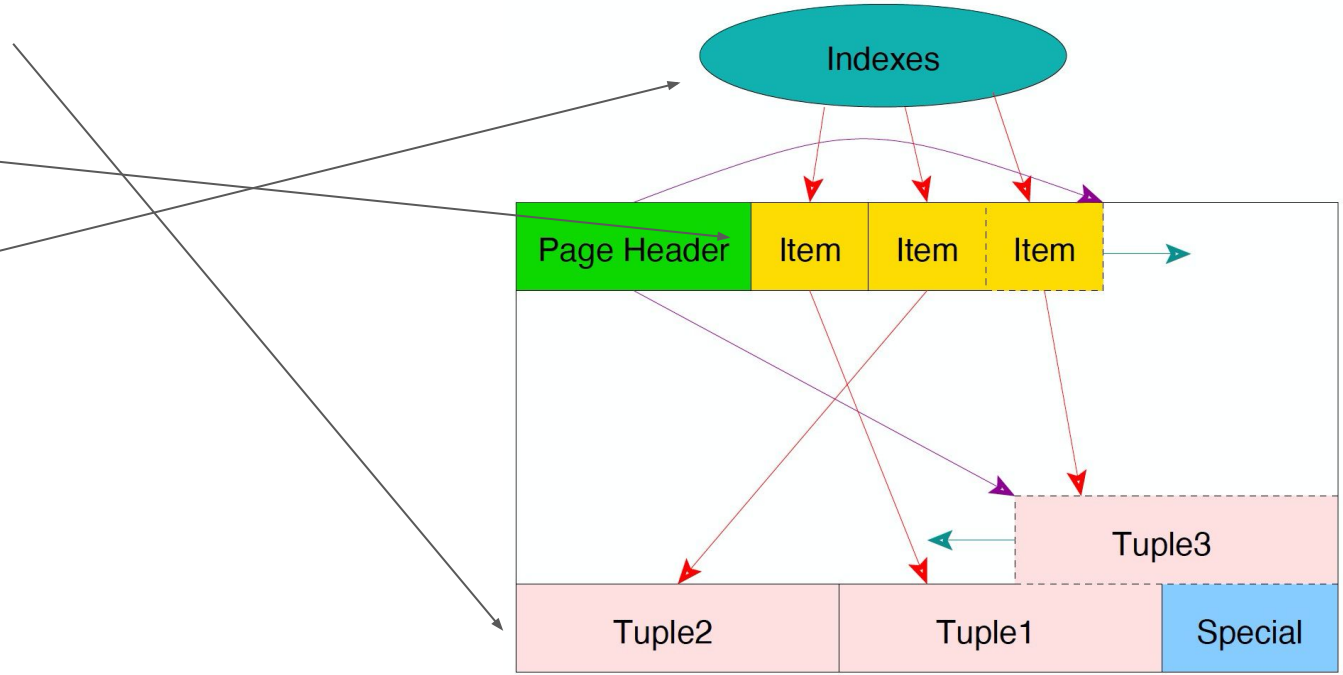
通过vacuum操作来清理

清理哪些内容

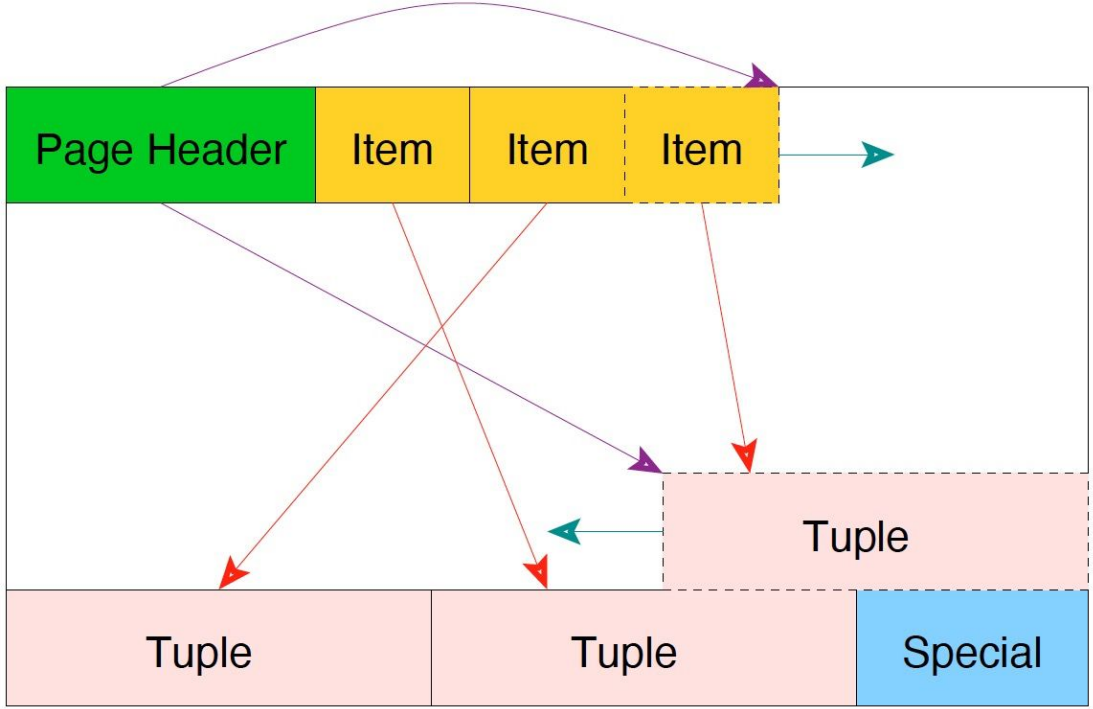
元组

元组指针

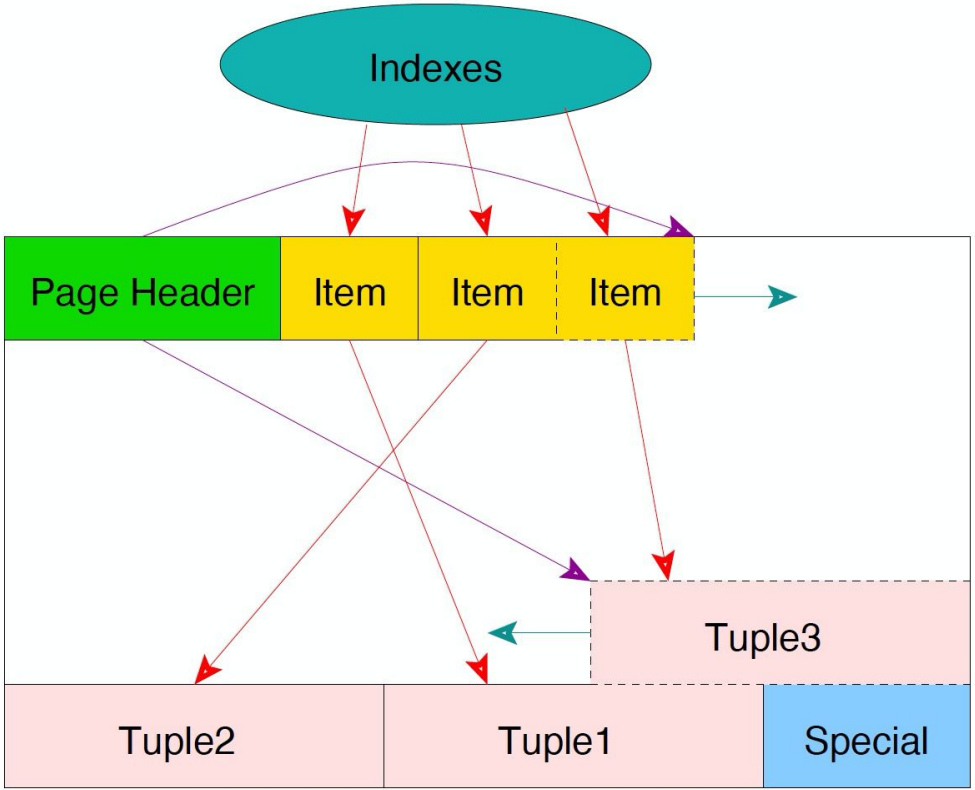
索引项



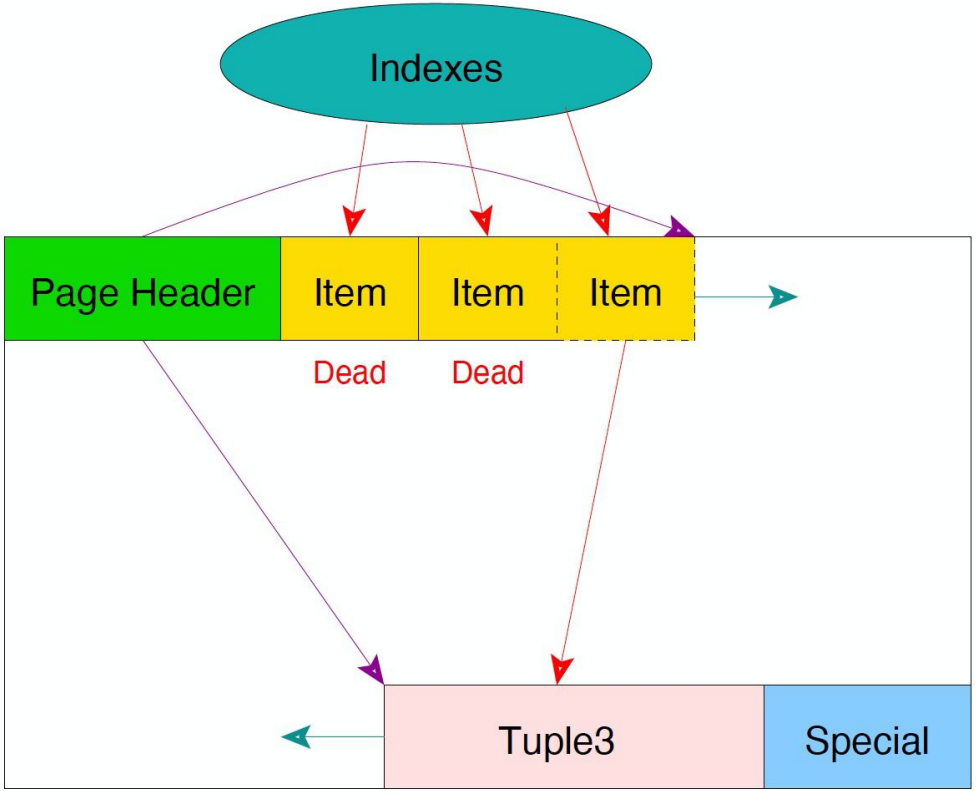
清理已删除元组



清理已删除元组

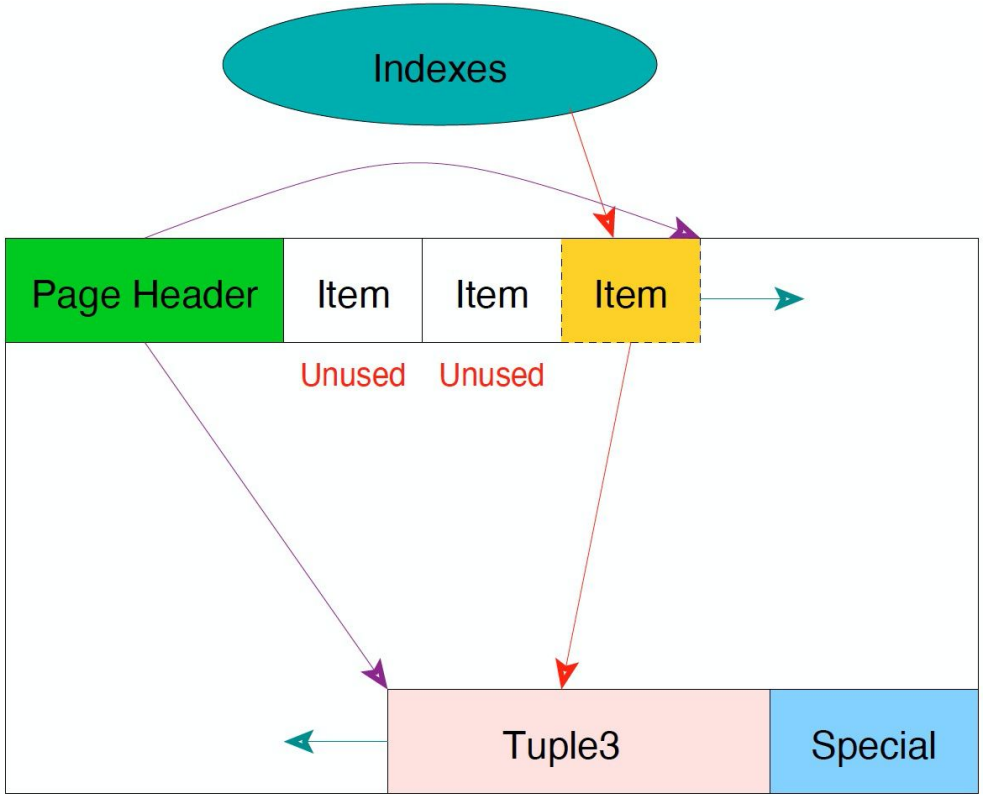

Indexes
Page Header
Item
Item
Item
Tuple3
Tuple2
Tuple1
Special

清理已删除元组



在单页清理中，不会清理索引项。所以在Heap中，只会清理tuple，而其Item会被标记为Dead。

清理已删除元组



在Vacuum操作中会清理索引项，并把对应的Item标记为Unused，这样Item就能够被继续复用。

清理已删除元组

```
demo=# insert into mvcc select 0 from generate_series(1, 210);
INSERT 0 210

demo=# SELECT (100 * (upper - lower) / pagesize::float8)::integer AS free_pct
FROM page_header(get_raw_page('mvcc', 0));
 free_pct
----------
        7
(1 row)

demo=# INSERT INTO mvcc VALUES (1);
INSERT 0 1
```

GREENPLUM DATABASE

清理已删除元组

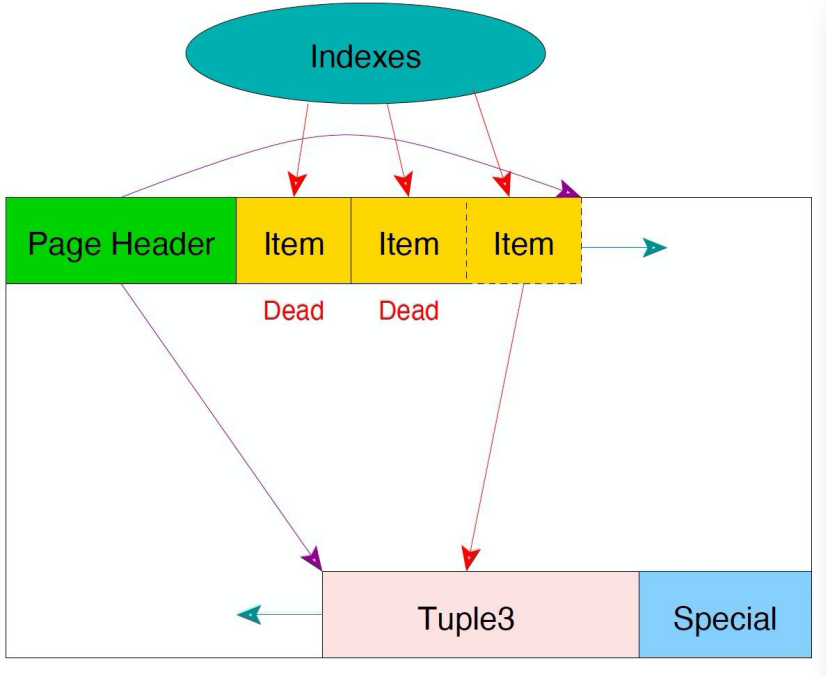
```
demo=# SELECT * FROM mvcc_demo_page0 OFFSET 210;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,211) | Normal |  978 |    0 | (0,211)
(1 row)

demo=# DELETE FROM mvcc WHERE val > 0;
DELETE 1

demo=# INSERT INTO mvcc VALUES (2);
INSERT 0 1

demo=# SELECT * FROM mvcc_demo_page0 OFFSET 210;
ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,211) | Normal |  978 |  980 | (0,211)
 (0,212) | Normal |  981 |    0 | (0,212)
(2 rows)
```

清理已删除元组

```
demo=# DELETE FROM mvcc WHERE val > 0;
DELETE 1

demo=# INSERT INTO mvcc VALUES (3);
INSERT 0 1

demo=# SELECT * FROM mvcc_demo_page0 OFFSET 210;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,211) | Dead   |      |      |
 (0,212) | Normal |  981 |  982 | (0,212)
 (0,213) | Normal |  983 |    0 | (0,213)
(3 rows)
```

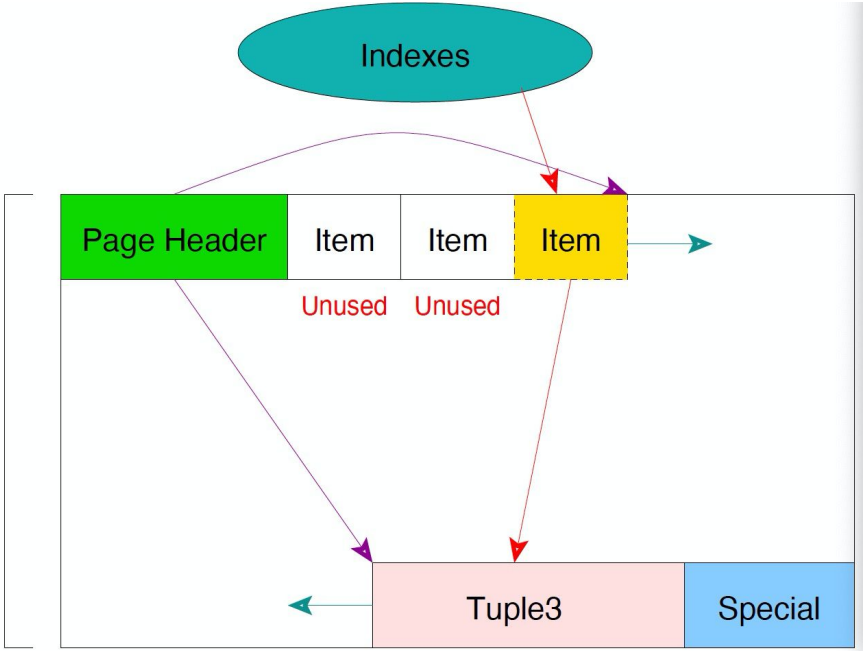
清理已删除元组

```
demo=# SELECT * FROM mvcc OFFSET 1000;
 val
-----
(0 rows)

demo=# SELECT * FROM mvcc_demo_page0 OFFSET
210;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,211) | Dead   |      |      |
 (0,212) | Dead   |      |      |
 (0,213) | Normal |  983 |    0 | (0,213)
(3 rows)
```



清理已删除元组

```
demo=# SELECT pg_freespace('mvcc');
 pg_freespace
--------------
 (0,0)
(1 row)

demo=# VACUUM mvcc;
VACUUM

demo=# SELECT * FROM mvcc_demo_page0 OFFSET
210;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,211) | Unused |      |      |
 (0,212) | Unused |      |      |
 (0,213) | Normal |  983 |    0 | (0,213)
(3 rows)
```



清理已删除元组

```
demo=# SELECT pg_freespace('mvcc');
 pg_freespace
--------------
 (0,544)
(1 row)
```

Vacuum还会更新free space map

free space map用来帮助insert 和update操作寻找目标页

单页清理不会更新free space map

GREENPLUM DATABASE

在单页清理中处理跟新操作中的旧元组

Update中产生的旧元组可以像被删除元组一样处理

有些元组的Item也可以被回收

使用Heap only tuple的话会甚至可以直接复用被删除的tuple

GREENPLUM DATABASE

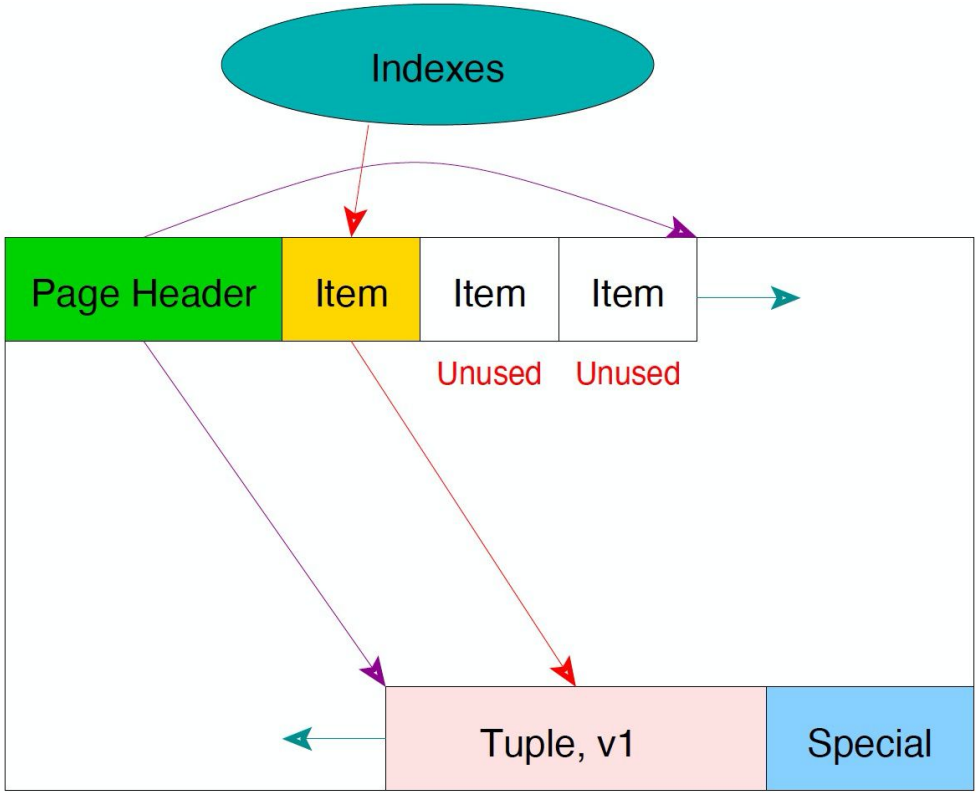
Heap Only Tuple（HOT）

如果每次更新都需要插入索引项，会导致索引的膨胀

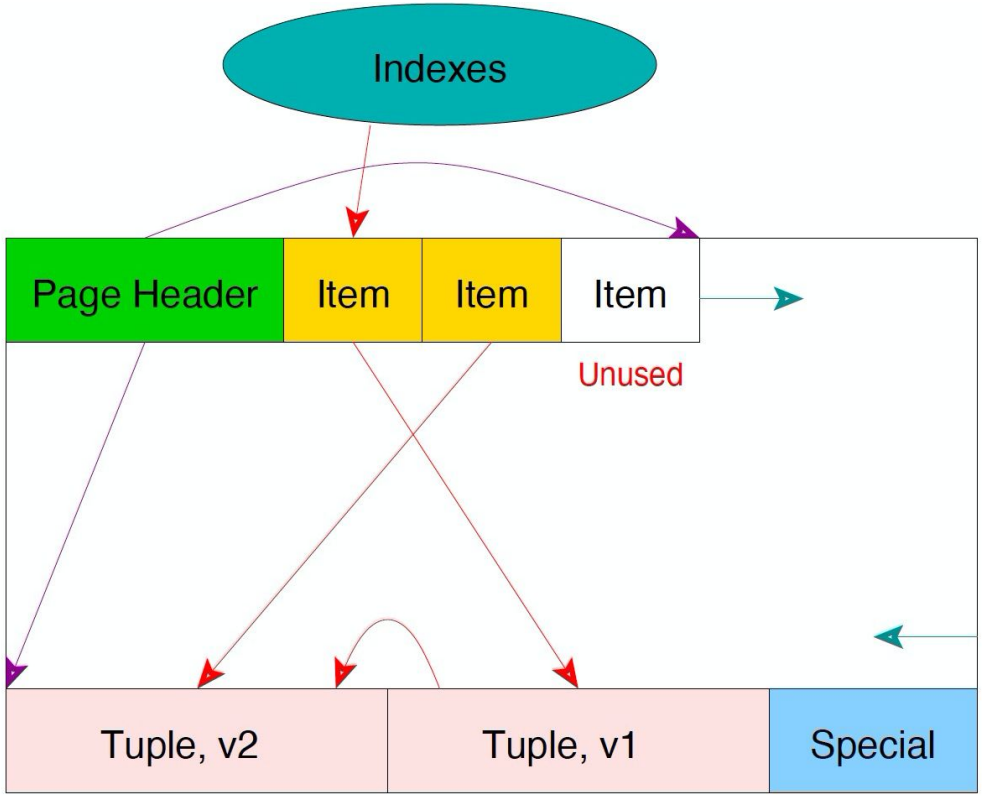
我们使用链式更新机制（HOT）来避免插入新的索引项

使用HOT机制的条件：更新不涉及索引列，新旧元组在同一个页面内

Heap only tuple



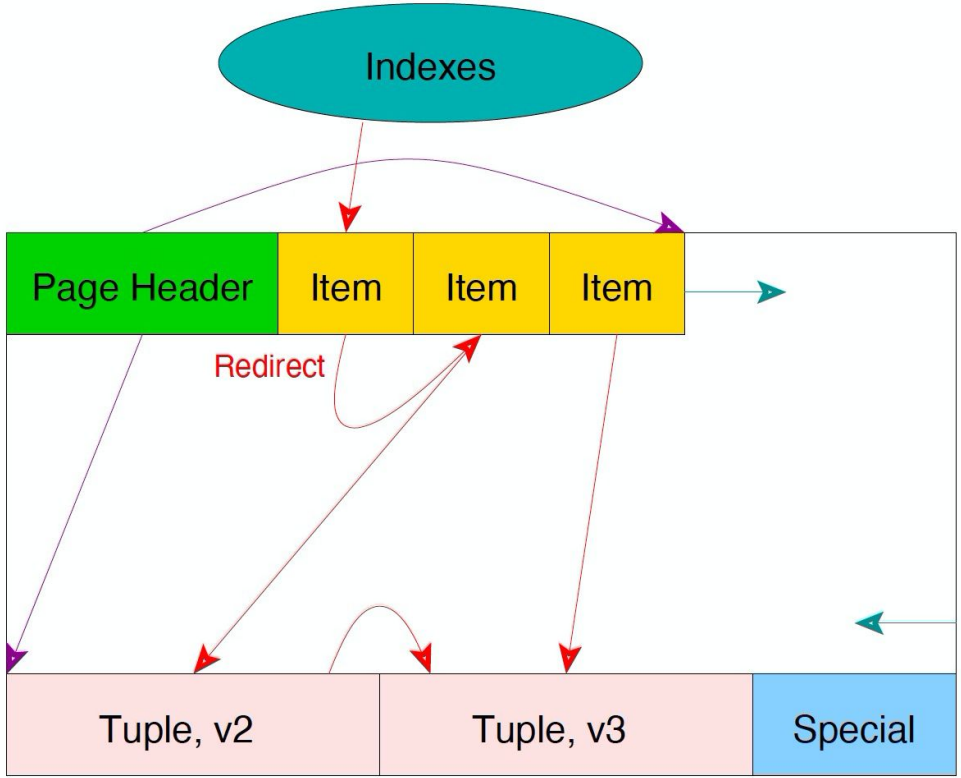
Heap only tuple



不会新增index项，而是通过旧tuple的c_tid指向新的tuple

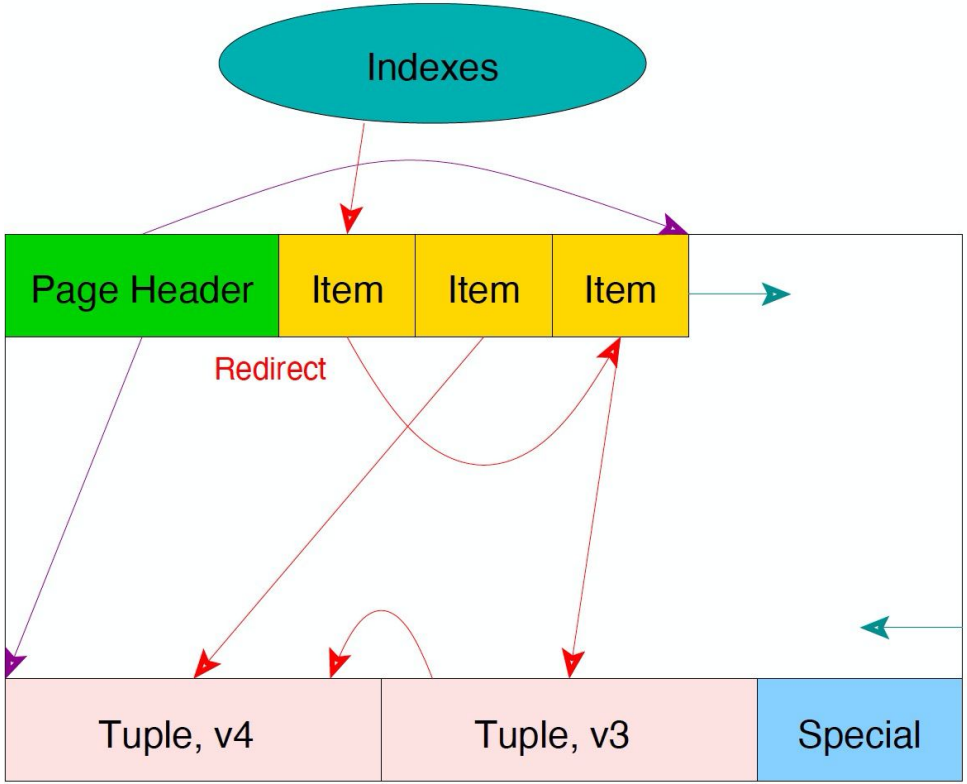
GREENPLUM DATABASE

Heap only tuple

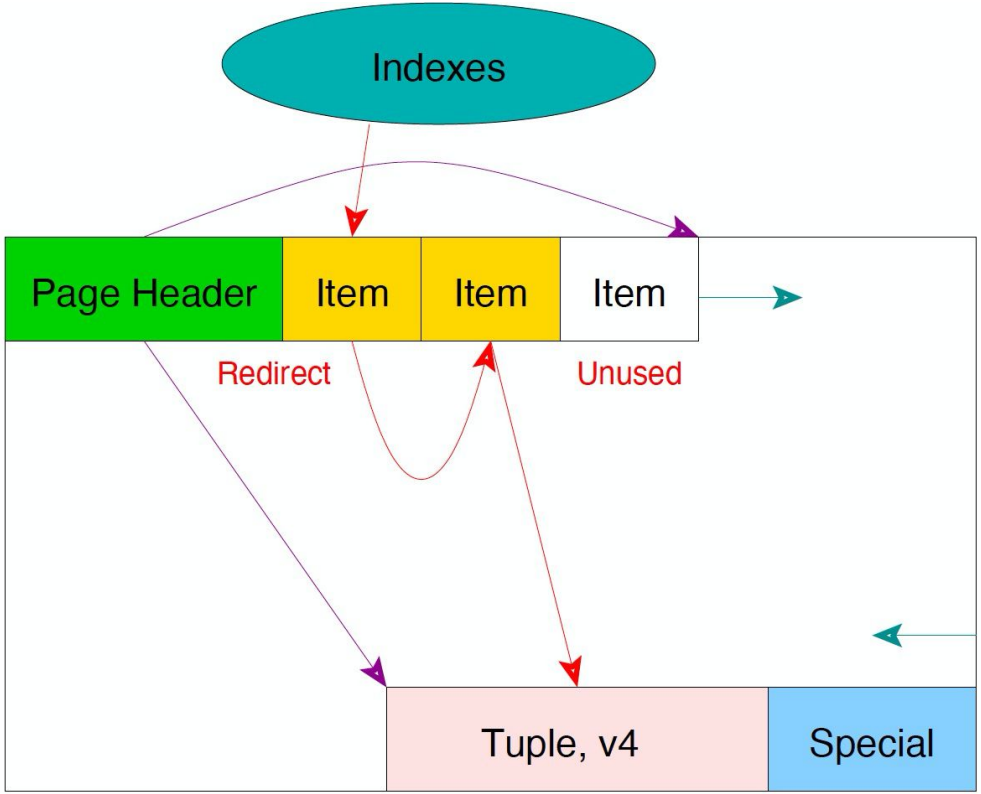


GREENPLUM DATABASE

Heap only tuple



Heap only tuple



HOT链的首尾节点保留，中间的所有节点都是可以清理的。

清理旧的被更新的tuple

```
demo=# INSERT INTO mvcc SELECT 0 FROM
generate_series(1, 210);
INSERT 0 210

demo=# INSERT INTO mvcc VALUES (1);
INSERT 0 1

demo=# SELECT * FROM mvcc_demo_page0 OFFSET
210;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,211) | Normal |  986 |    0 | (0,211)
(1 row)
```

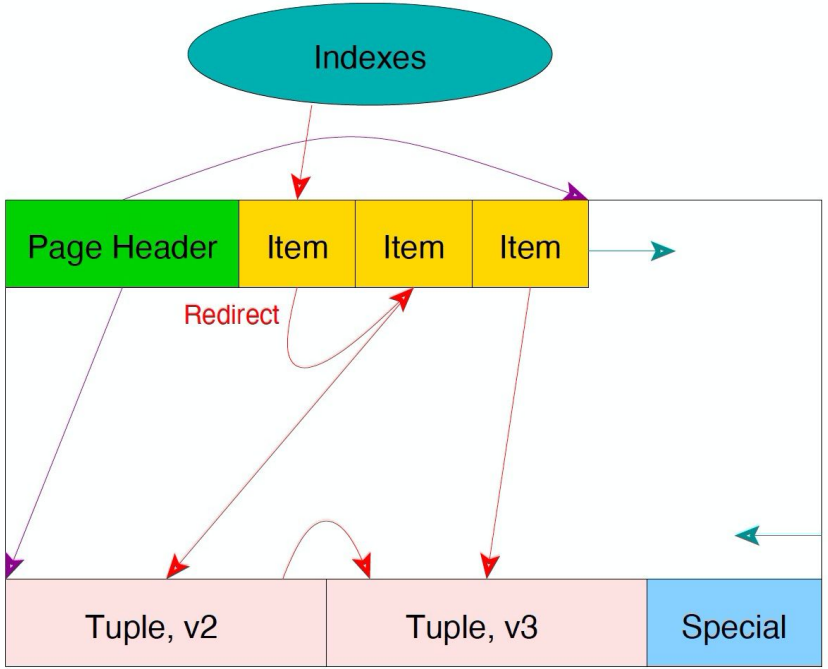
GREENPLUM DATABASE

清理旧的被更新的tuple

```
demo=# UPDATE mvcc SET val = val + 1 WHERE val > 0;
UPDATE 1
demo=# SELECT * FROM mvcc_demo_page0 OFFSET
210;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,211) | Normal |  986 |  988 | (0,212)
 (0,212) | Normal |  988 |    0 | (0,212)
(2 rows)
```

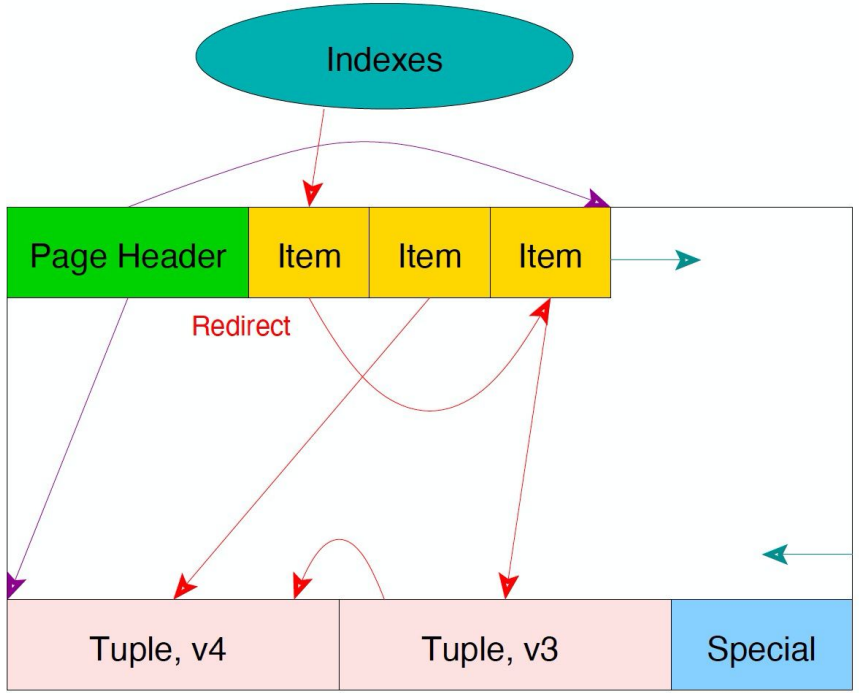
清理旧的被更新的tuple

```
demo=# UPDATE mvcc SET val = val + 1 WHERE val > 0;
UPDATE 1
demo=# SELECT * FROM mvcc_demo_page0 OFFSET
210;
  ctid   |      case       | xmin | xmax | t_ctid
---------+-----------------+------+------+---------
 (0,211) | Redirect to 212 |      |      |
 (0,212) | Normal          |  988 |  989 | (0,213)
 (0,213) | Normal          |  989 |    0 | (0,213)
(3 rows)
```



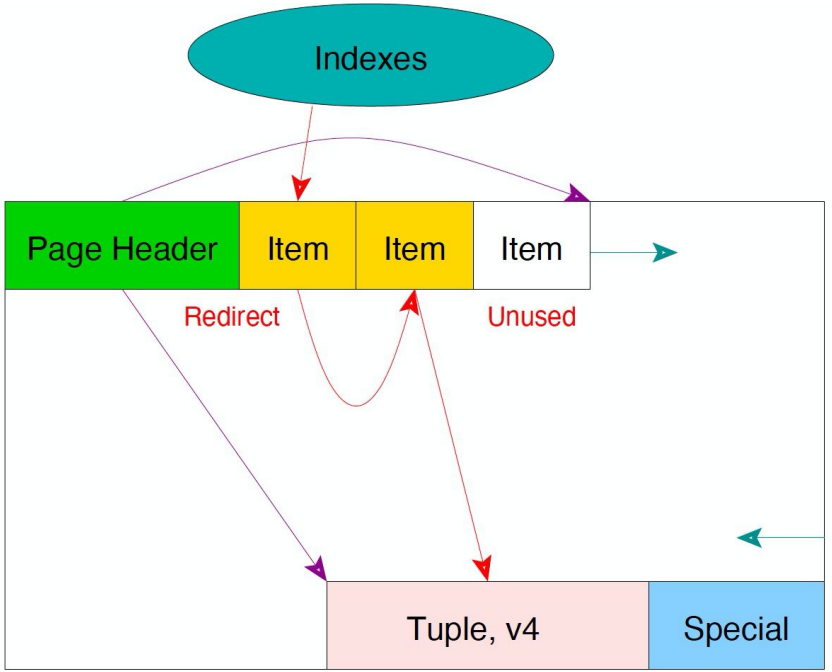
清理旧的被更新的tuple

```
demo=# UPDATE mvcc SET val = val + 1 WHERE val > 0;
UPDATE 1

demo=# SELECT * FROM mvcc_demo_page0 OFFSET
210;
  ctid   |      case       | xmin | xmax | t_ctid
---------+-----------------+------+------+---------
 (0,211) | Redirect to 213 |      |      |
 (0,212) | Normal          |  990 |    0 | (0,212)
 (0,213) | Normal          |  989 |  990 | (0,212)
(3 rows)
```



清理旧的被更新的tuple

```
demo=# SELECT * FROM mvcc OFFSET 1000;
 val
-----
(0 rows)

demo=# SELECT * FROM mvcc_demo_page0 OFFSET
210;
  ctid   |      case       | xmin | xmax | t_ctid
---------+-----------------+------+------+---------
 (0,211) | Redirect to 212 |      |      |
 (0,212) | Normal          |  990 |    0 | (0,212)
 (0,213) | Unused          |      |      |
(3 rows)
```



手动清理

```
demo=# VACUUM mvcc;
VACUUM
demo=# SELECT * FROM mvcc_demo_page0 OFFSET
210;
  ctid   |      case       | xmin | xmax | t_ctid
---------+-----------------+------+------+---------
 (0,211) | Redirect to 212 |      |      |
 (0,212) | Normal          |  990 |    0 | (0,212)
 (0,213) | Unused          |      |      |
(3 rows)
```

Vacuum不会移除Redirect

手动清理

```
demo=# INSERT INTO mvcc VALUES (1);
INSERT 0 1
demo=# INSERT INTO mvcc VALUES (2);
INSERT 0 1
demo=# INSERT INTO mvcc VALUES (3);
INSERT 0 1

demo=# SELECT ctid, xmin, xmax FROM
mvcc_demo_page0;
 ctid  | xmin | xmax
-------+------+------
 (0,1) |  996 |    0
 (0,2) |  997 |    0
 (0,3) |  998 |    0
(3 rows)

demo=# DELETE FROM mvcc;
DELETE 3
```

手动清理

```
demo=# SELECT ctid, xmin, xmax FROM
mvcc_demo_page0;
 ctid  | xmin | xmax
-------+------+------
 (0,1) |  996 |  999
 (0,2) |  997 |  999
 (0,3) |  998 |  999
(3 rows)

demo=# VACUUM mvcc;
VACUUM
demo=# SELECT pg_relation_size('mvcc');
 pg_relation_size
------------------
                0
(1 row)
```

GREENPLUM DATABASE®

Index update的问题

如果发生了索引列的更新则不能使用redirect

仍然会引起索引的膨胀

Update 索引列

```
emo=# CREATE INDEX i_mvcc_val on mvcc (val);
CREATE INDEX
demo=# INSERT INTO mvcc SELECT 0 FROM
generate_series(1, 210);
INSERT 0 210
demo=# INSERT INTO mvcc VALUES (1);
INSERT 0 1

demo=# SELECT * FROM mvcc_demo_page0 OFFSET
210;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,211) | Normal | 1004 |    0 | (0,211)
(1 row)
```

Update 索引列

```
demo=# UPDATE mvcc SET val = val + 1 WHERE val > 0;
UPDATE 1
demo=# SELECT * FROM mvcc_demo_page0 OFFSET
210;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,211) | Normal | 1004 | 1005 | (0,212)
 (0,212) | Normal | 1005 |    0 | (0,212)
(2 rows)

demo=# UPDATE mvcc SET val = val + 1 WHERE val > 0;
UPDATE 1
demo=# SELECT * FROM mvcc_demo_page0 OFFSET
210;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,211) | Dead   |      |      |
 (0,212) | Normal | 1005 | 1006 | (0,213)
 (0,213) | Normal | 1006 |    0 | (0,213)
(3 rows)
```

Update 索引列

```
demo=# UPDATE mvcc SET val = val + 1 WHERE val > 0;
UPDATE 1
demo=# SELECT * FROM mvcc_demo_page0 OFFSET
210;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,211) | Dead   |      |      |
 (0,212) | Dead   |      |      |
 (0,213) | Normal | 1006 | 1007 | (0,214)
 (0,214) | Normal | 1007 |    0 | (0,214)
(4 rows)
```

Update 索引列

```
demo=# SELECT * FROM mvcc OFFSET 1000;
 val
-----
(0 rows)

demo=# SELECT * FROM mvcc_demo_page0 OFFSET
210;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,211) | Dead   |      |      |
 (0,212) | Dead   |      |      |
 (0,213) | Dead   |      |      |
 (0,214) | Normal | 1007 |    0 | (0,214)
(4 rows)
```

Update 索引列

```
demo=# VACUUM mvcc;
VACUUM
demo=# SELECT * FROM mvcc_demo_page0 OFFSET
210;
  ctid   |  case  | xmin | xmax | t_ctid
---------+--------+------+------+---------
 (0,211) | Unused |      |      |
 (0,212) | Unused |      |      |
 (0,213) | Unused |      |      |
 (0,214) | Normal | 1007 |    0 | (0,214)
(4 rows)
```

GREENPLUM DATABASE

总结

清理只发生在对于任何执行中的事务都不可见的tuple上

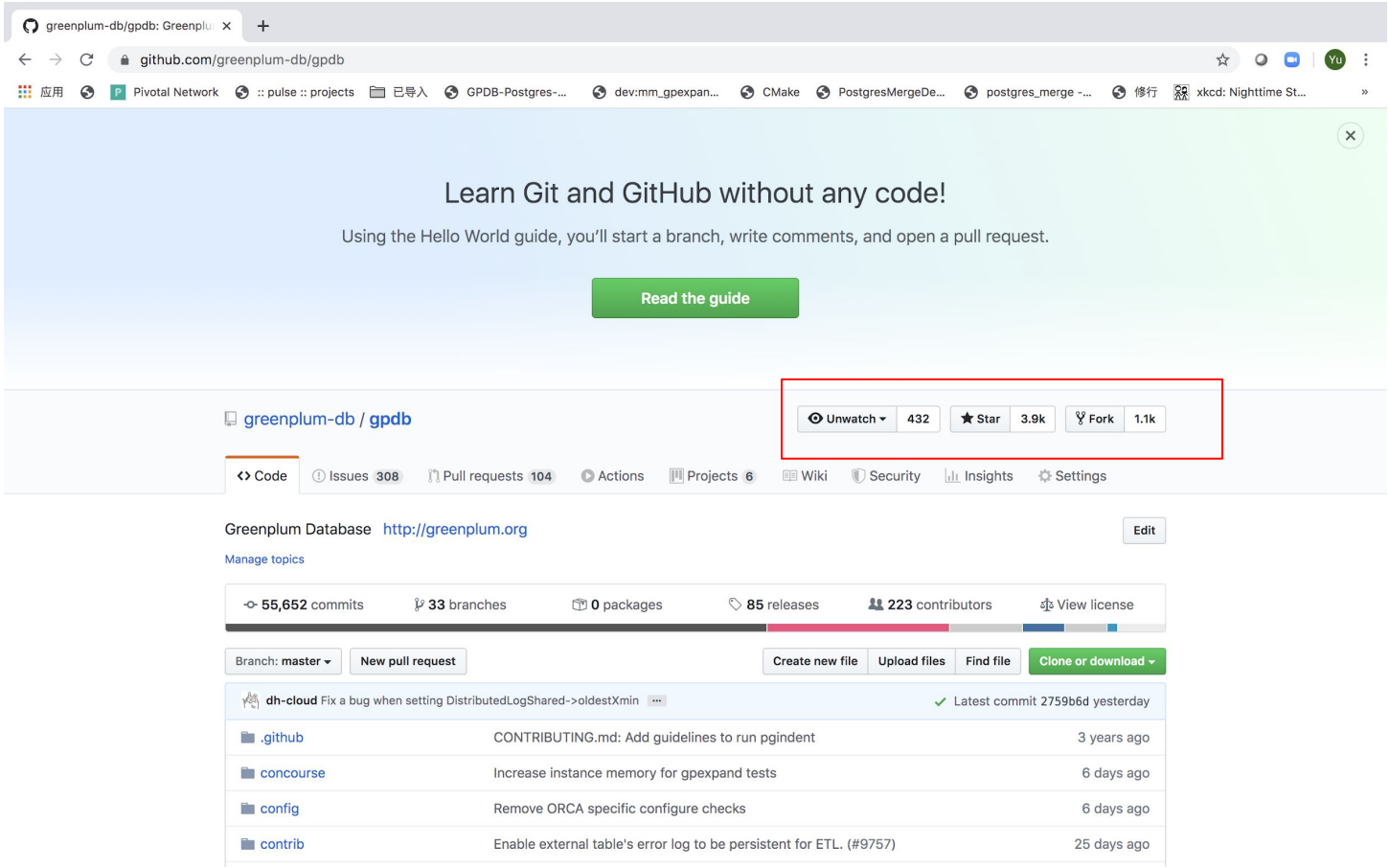
HOT发生在处于单个页面，并有相同的索引值的tuple上

多数的清理工作由单页清理操作完成，但是单页清理只涉及Heap Page

Vacuum则会进行更加彻底的清理，包括tuple item index

准备工作

从源代码开始：下载编译Greenplum源代码



GREENPLUM DATABASE®

微信技术讨论群
添加入群小助手：gp_assistant

微信公众号
技术干货、行业热点、活动预告

欢迎访问Greenplum中文社区：cn.greenplum.org