DB 2

05 – Row Updates

Summer 2018

Torsten Grust
Universität Tübingen, Germany

1 Q_4 — Row Update

SQL probe Q_4 uses SQL DML statements (INSERT, UPDATE, DELETE) to alter the state of a table:

```
INSERT INTO ternary      UPDATE ternary       DELETE FROM ternary
  SELECT …                 SET    c = e1        WHERE a = e2
  FROM   …                 WHERE a = e2
```

INSERT: evaluate query to construct new rows.

UPDATE, DELETE: query the table and identify the affected rows.

Modify table storage to reflect the row updates.[1]

[1] We still assume that the table has *no* associated index structures.

(Re-)Create table ternary:

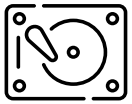
```
DROP TABLE IF EXISTS ternary;
CREATE TABLE ternary (a int NOT NULL, b text NOT NULL, c float);

INSERT INTO ternary(a, b, c)
  SELECT i,
         md5(i::text),
         log(i)
  FROM   generate_series(1, 1000, 1) AS i;

-- Q4: Perform row insertions/updates/deletions (⚠ no ANALYZE here, don't alter the table now):
EXPLAIN VERBOSE
  INSERT INTO ternary(a,b,c)
    SELECT t.a, 'Han Solo', t.c
    FROM   ternary AS t;

EXPLAIN VERBOSE
  UPDATE ternary AS t
    SET   c = -1
    WHERE t.a = 982;
```

Using EXPLAIN on Q_4: INSERT



```
EXPLAIN VERBOSE
  INSERT INTO ternary(a,b,c)
    SELECT t.a, 'Han Solo', t.c
    FROM   ternary AS t;
```

```
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Insert on public.ternary  (cost=0.00..20.00 rows=1000 width=44)
   ->  Seq Scan on public.ternary t  (cost=0.00..20.00 rows=1000 width=44)
         Output: t.a, 'Han Solo'::text, t.c
```

- **Seq Scan** scans table ternary to construct rows to be inserted, feeds 1000 rows into **Insert** for insertion.
- Width of inserted rows (over-)estimated to be 44 bytes = 4 (int) + 32 (text) + 8 (float) bytes.

Insert rows into table ternary (⚠️ uses EXPLAIN without ANALYZE to not actually modify the table at this point):

```
EXPLAIN VERBOSE
  INSERT INTO ternary(a,b,c)
    SELECT t.a, 'Han Solo', t.c
    FROM   ternary AS t;
```

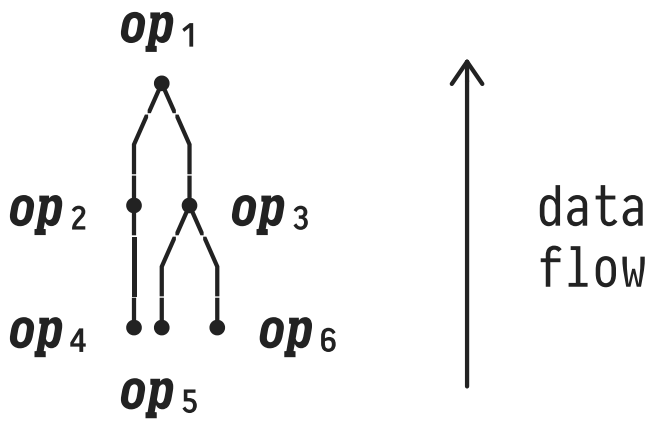
```
                                QUERY PLAN
-------------------------------------------------------------------------
Insert on public.ternary  (cost=0.00..20.00 rows=1000 width=44)
  ->  Seq Scan on public.ternary t  (cost=0.00..20.00 rows=1000 width=44)
        Output: t.a, 'Han Solo'::text, t.c
```

Any column of type text – regardless of length – appears to account for 32 bytes in the width output of EXPLAIN. Actual rows on page have the expected width.

Reading Complex EXPLAIN Outputs

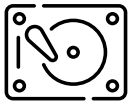
- EXPLAIN uses symbol -> and indentation to visualize larger, **tree-shaped** query evaluation plans:

```
QUERY PLAN
op1
  -> op2
       -> op4
  -> op3
       -> op5
       -> op6
```



- Read plans “inside out”, root op_1 delivers query result.

Using EXPLAIN on Q_4: UPDATE



```
EXPLAIN VERBOSE
  UPDATE ternary AS t
    SET   c = -1
    WHERE t.a = 982;
```

```
                                    QUERY PLAN
Update on public.ternary t  (cost=0.00..22.50 rows=1 width=51)
  ->  Seq Scan on public.ternary t  (cost=0.00..22.50 rows=1 width=51)
      ➡ Output: a, b, '-1'::double precision, ctid
        Filter: (t.a = 982)                        ⬆
```

- **Seq Scan** emits complete rows (only c was updated).
- Additionally feeds row ID (ctid) into **Update** to identify the affected row(s).

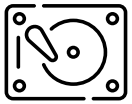
Update and delete rows in table ternary (⚠️ uses EXPLAIN without ANALYZE to not actually modify the table at this point):

```
EXPLAIN VERBOSE
  UPDATE ternary AS t
    SET   c = -1
    WHERE t.a = 982;
```

```
                                QUERY PLAN
-----------------------------------------------------------------------------
 Update on public.ternary t  (cost=0.00..22.50 rows=1 width=51)
   ->  Seq Scan on public.ternary t  (cost=0.00..22.50 rows=1 width=51)
         Output: a, b, '-1'::double precision, ctid
         Filter: (t.a = 982)
```

Width of 51 > 45 accounts for the additional 6 bytes of the emitted ctid (row ID) field.

Using EXPLAIN on Q_4: DELETE

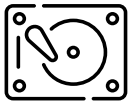


```
EXPLAIN VERBOSE
  DELETE FROM ternary AS t
  WHERE  t.a = 982;
```

```
                                  QUERY PLAN
Delete on public.ternary t  (cost=0.00..22.50 rows=1 width=6)
  ->  Seq Scan on public.ternary t  (cost=0.00..22.50 rows=1 width=6)
        Output: ctid
        Filter: (t.a = 982)
```

- **Seq Scan** returns affected row IDs (of 6 bytes each) only.
- We turn to **Filter** (makes scan skip non-qualifying rows) later in this course.

2 How do Row Updates Alter the Table Storage?

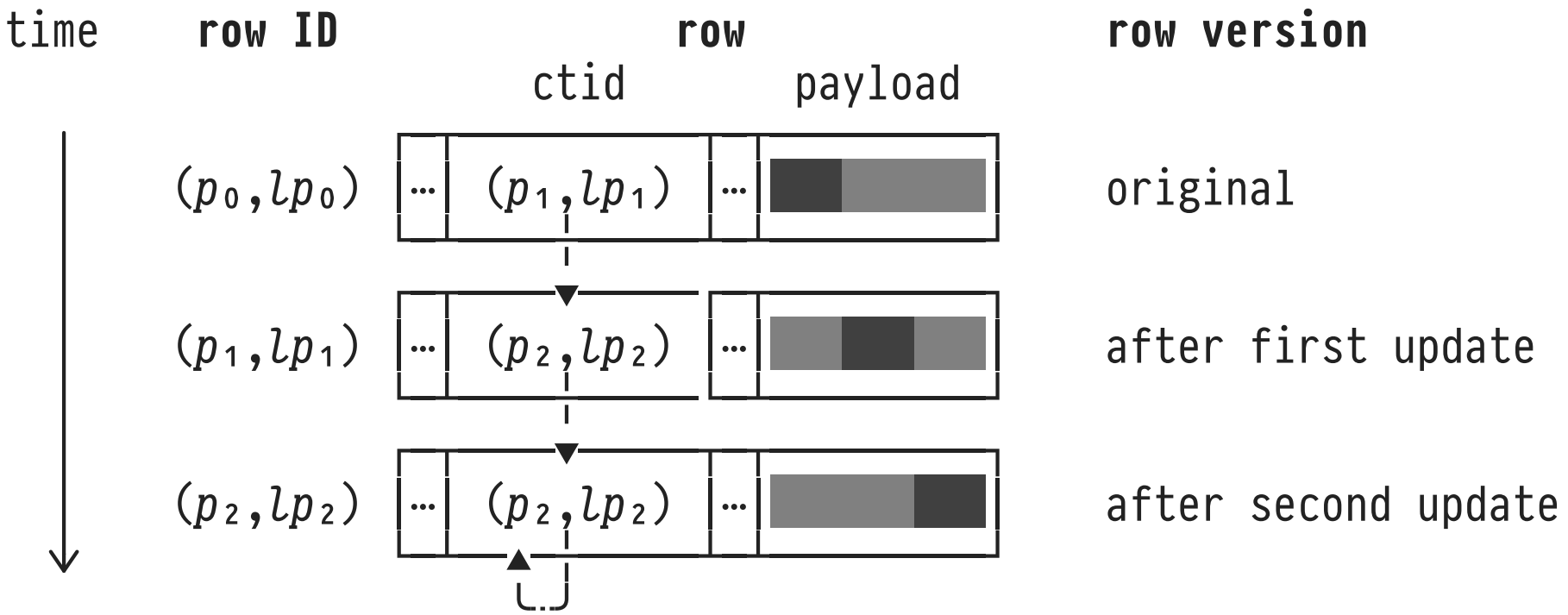


Let us take a closer look at how plan operator Update alters the target table's heap file pages. We find:[2]

- Rows are *not* updated in-place. A **new version of the row is created** — original and updated row co-exist.
- Any database user (query, application) sees exactly one version of any row at any time. Different users may see different row versions.
- A separate VACUUM ("garbage collection") step collects and removes old versions that cannot be seen by any user.

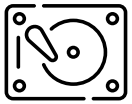
[2] This implementation of Update is typical for all DBMS that implement **Multi-Version Concurrency Control (MVCC)**. We discuss MVCC later in this course.

Row Version Chains

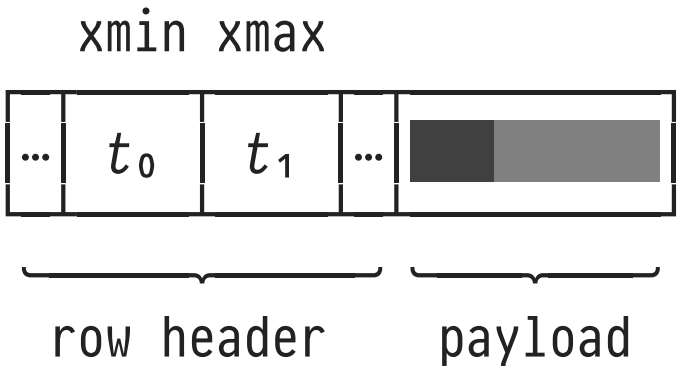


- Original and updated versions of a row form a **chain**, linked by the rows' IDs (held in field ctid).

Row Visibility and Timestamps



1. Each row carries two **timestamps** — xmin and xmax — that mark its first and last time of existence.
2. Each query/update is executed at some timestamp T which defines the rows that are **visible** to the operation:



row is visible for any operation with timestamp $t_0 \leqslant T < t_1$

($t_1 = \infty$: row has not been updated yet)

- DBMS uses system-wide virtual timestamps (*transaction IDs*), see PostgreSQL built-in function txid_current().

Explore row version chaining and row visibility for table ternary. Table has 9 pages, where the last page has space to spare. Update a row on page 9:

```
-- recreate a clean ternary table
 DROP TABLE IF EXISTS ternary;
 CREATE TABLE ternary (a int NOT NULL, b text NOT NULL, c float);
 INSERT INTO ternary(a, b, c)
   SELECT i,
          md5(i::text),
          log(i)
   FROM   generate_series(1, 1000, 1) AS i;

-- 1 list rows on page 9
 SELECT t.ctid, t.* FROM ternary AS t WHERE t.ctid >= '(9,1)';
```

ctid	a	b	c	
(9,1)	964	8065d07da4a77621450aa84fee5656d9	2.98407703390283	
(9,2)	965	eeb69a3cb92300456b6a5f4162093851	2.98452731334379	
[...]				
(9,18)	981	287e03db1d99e0ec2edb90d079e142f3	2.99166900737995	
(9,19)	982	fec8d47d412bcbeece3d9128ae855a7a	2.99211148778695	⬅ will be updated
(9,20)	983	6aab1270668d8cac7cef2566a1c5f569	2.99255351783214	
[...]				
(9,36)	999	b706835de79a2b4e80506f582af3676a	2.99956548822598	
(9,37)	1000	a9b7ba70783b617e9998dc4dd82eb3c5	3	

```
-- 2 check row header contents before update
 SELECT t_ctid, lp, lp_off, lp_len, t_xmin, t_xmax,
        (t_infomask::bit(16) & b'0010000000000000')::int::bool  AS "updated row?",
        (t_infomask2::bit(16) & b'0100000000000000')::int::bool AS "has been HOT updated?"
        FROM heap_page_items(get_raw_page('ternary', 9));
```

t_ctid	lp	lp_off	lp_len	t_xmin	t_xmax	updated row?	has been HOT updated?	
(9,1)	1	8120	72	12708	0	f	f	
(9,2)	2	8048	72	12708	0	f	f	
[...]								
(9,18)	18	6896	72	12708	0	f	f	
(9,19)	19	6824	72	12708	0	f	f	⬅ will be updated
(9,20)	20	6752	72	12708	0	f	f	
[...]								
(9,36)	36	5600	72	12708	0	f	f	
(9,37)	37	5528	72	12708	0	f	f	


```
-- 3 check current transaction ID (≡ virtual timestamp)
 SELECT txid_current();
```

txid_current	
12710	⬅ current operation timestamp

```
 SELECT txid_current();
```

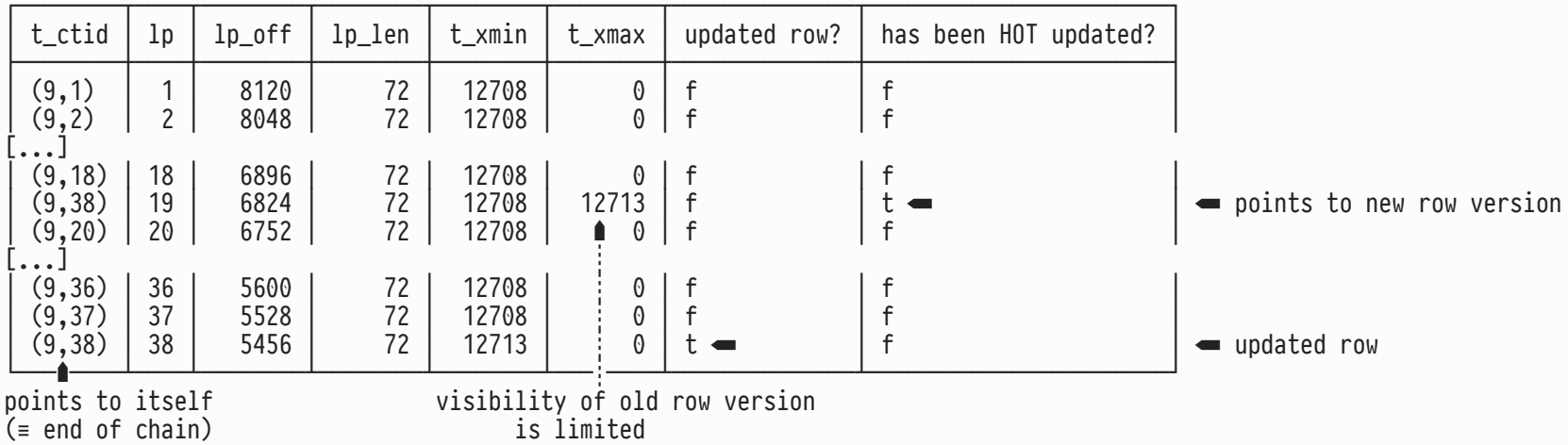
txid_current	
12711	⬅ current operation timestamp

```
-- 4 update one row
 UPDATE ternary AS t SET c = -1 WHERE t.a = 982;

-- 5 check visible contents of page 9 after update
 SELECT t.ctid, t.* FROM ternary AS t WHERE t.ctid >= '(9,1)';
```

ctid	a	b	c	
(9,1)	964	8065d07da4a77621450aa84fee5656d9	2.98407703390283	
(9,2)	965	eeb69a3cb92300456b6a5f4162093851	2.98452731334379	
[...]				
(9,18)	981	287e03db1d99e0ec2edb90d079e142f3	2.99166900737995	
(9,20)	983	6aab1270668d8cac7cef2566a1c5f569	2.99255351783214	⬅ row (9,19) is gone (≡ invisible)
[...]				
(9,36)	999	b706835de79a2b4e80506f582af3676a	2.99956548822598	
(9,37)	1000	a9b7ba70783b617e9998dc4dd82eb3c5	3	
(9,38)	982	fec8d47d412bcbeece3d9128ae855a7a	-1	⬅ updated row

```
-- 6 check row header contents after update
 SELECT t_ctid, lp, lp_off, lp_len, t_xmin, t_xmax,
        (t_infomask::bit(16) & b'0010000000000000')::int::bool  AS "updated row?",
        (t_infomask2::bit(16) & b'0100000000000000')::int::bool AS "has been HOT updated?"
        FROM heap_page_items(get_raw_page('ternary', 9));
```

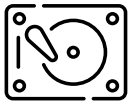



```
db2=# SELECT txid_current();
```

txid_current
12714

◀ 12713 < 12714 ⇒ old row version is invisible to us, as expected

Impact of Updates Beyond the Row's Page



- Updates on full pages may lead to row relocation across pages: versions then have row IDs (p_i, lp_i), (p_{i+1}, lp_{i+1}) where $p_i \neq p_{i+1}$.
  - Traversal of longer update chains may lead to I/O-costly “page hopping.”
  - $\Longrightarrow$ Perform VACUUM to collect inaccessible old versions. From outside page, point to most recent row directly.

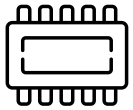
- PostgreSQL optimizes for the good-natured case where $p_i = p_{i+1}$ and indexed row fields have *not* been changed.
  - Such **heap-only tuple (HOT) updates** have page-internal impact only, no maintenance outside page required.

INSERT INTO ternary VALUES (…,…,…)	UPDATE ternary SET c = e_1 WHERE a = e_2	DELETE FROM ternary WHERE a = e_2

UPDATE: affects updated column(s) only.
INSERT, DELETE: operate on full rows, all column BATs of table ternary will be affected.

- MonetDB uses user-specific **Δ tables ("delta tables")** to represent changes. Column BATs are *not* modified immediately. Global visibility of changes is delayed.

Using EXPLAIN on Q_4: DELETE



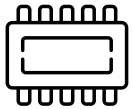
```
sql> EXPLAIN DELETE FROM ternary WHERE a = 981;

 ⋮
ternary     :bat[:oid] := sql.tid(sql, "sys", "ternary");
a0          :bat[:int] := sql.bind(sql, "sys", "ternary", "a", 0:int);
deleted_rows:bat[:oid] := algebra.thetaselect(a0, ternary, 981:int, "==");
sql_delete             := sql.delete(sql, "sys", "ternary", deleted_rows);
 ⋮
```

- algebra.thetaselect(b_1,b_2,v,θ) returns the oids of those rows r in algebra.projection(b_2,b_1), for which predicate tail(r) θ v holds.

- sql.delete(…) modifies the BAT of currently visible rows (obtainable via sql.tid(…)) for table ternary.
  ⚠ However, no column BAT is changed yet.

Using EXPLAIN on Q_4: INSERT



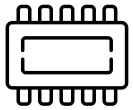
```
sql> EXPLAIN INSERT INTO ternary(a,b,c) VALUES (1001, 'Han Solo', -2);

 ⋮
sql_append := sql.append(sql, "sys", "ternary", "a", 1001:int);
sql_append := sql.append(sql, "sys", "ternary", "b", "Han Solo");
sql_append := sql.append(sql, "sys", "ternary", "c", -2:dbl);
 ⋮
```

- For ternary(a,b,c), a row insert translates into three individual operations on the column BATs.
- sql.append(…) saves the inserted value in the **Δ^i table** associated with each column BAT.

  ⚠ The column BATs do not change yet.

Using EXPLAIN on Q_4: UPDATE



```
sql> EXPLAIN UPDATE ternary SET c = -1 WHERE a = 982;

 ⋮
ternary       :bat[:oid] := sql.tid(sql, "sys", "ternary");
a0            :bat[:int] := sql.bind(sql, "sys", "ternary", "a", 0:int);
updated_rows  :bat[:oid] := algebra.thetaselect(a0, ternary, 982:int, "==");
updated_rows_a:bat[:int] := algebra.projection(updated_rows, a0);
updated_c     :bat[:dbl] := algebra.project(updated_rows_a, -1:dbl);
sql_update               := sql.update(sql, "sys", "ternary", "c",
                                       updated_rows, updated_c);
 ⋮
```

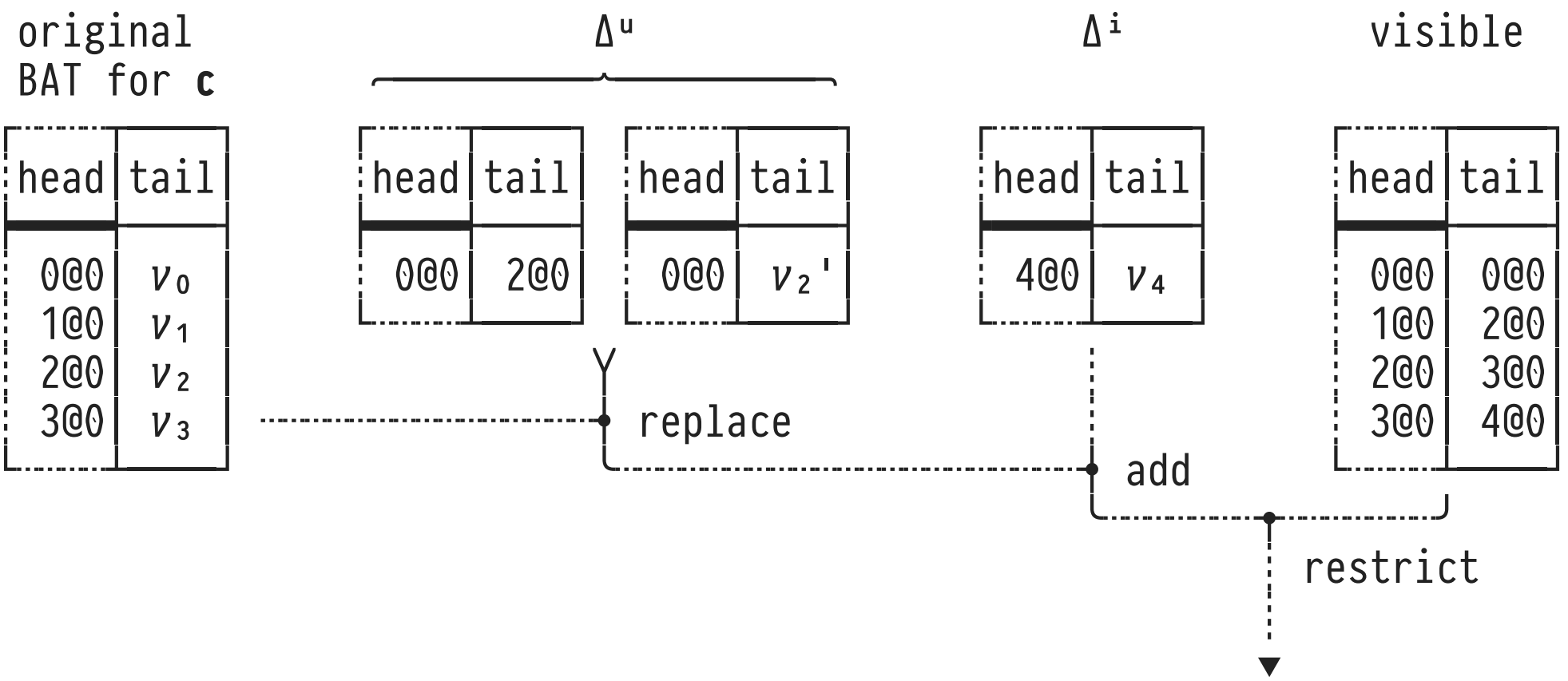
- BATs updated_rows and updated_c contain oids and c values of the changed rows.[3]
- sql.update(…) saves these changes in the **Δ^u table** for the BAT of column c. ⚠️ The column BAT is not changed yet.

[3] algebra.project(*b*,*v*) returns *b* with all tail values set to *v*.

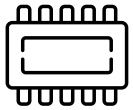
⚠️ The use of BAT updated_rows_a above appears unnecessary in this case (could be replaced by updated_rows).

Δ and Visibility Tables

In column c, update v_2 to v_2', insert v_4, delete v_1:



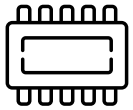
Applying Changes on Demand



- When a query needs to see the changes made to column `c`, apply all changes accumulated in the `c`'s Δ tables:
  1. Load — yet unmodified — column BAT for `c`.
  2. Read `c`'s Δ^{u} table and perform value replacements (via `bat.replace(…)`).
  3. Read `c`'s Δ^{i} table and perform value inserts (via `bat.append(…)`).
  4. Restrict `c` to currently visible rows (via `algebra.projection(…)`).

- Make changes permanent only once we want them to be seen globally by all users (*COMMIT* → transaction management).

Delay Change Propagation: Disable *AUTO COMMIT*



- To experiment with Δ-based change management, disable MonetDB's default *auto commit* behavior (use `mclient` option `--autocommit` to turn *auto commit* off):

```
DELETE FROM ternary WHERE a = 981;
INSERT INTO ternary(a,b,c) VALUES (1001, 'Han Solo', -2);
UPDATE ternary SET c = -1 WHERE a = 982;
```

- Without *auto commit*, changes are still pending at this point. Thus:

```
SELECT t.c              ⎫  reads Δᵘ, Δⁱ (+ visibility table)
FROM   ternary AS t;    ⎭  to reflect changes on t.c
```

Demonstrate Δ tables and delayed change propagation on table ternary.

1️⃣ (Re-)Create and populate table ternary:

```
$ mclient -d scratch -l sql

 DROP TABLE IF EXISTS ternary;
 CREATE TABLE ternary (a int NOT NULL, b text NOT NULL, c double);

 INSERT INTO ternary(a, b, c)
   SELECT value, md5(value), log(value)
   FROM   generate_series(1, 1001);

 SELECT t.* FROM ternary AS t LIMIT 5;
+------+----------------------------------+-------------------------+
| a    | b                                | c                       |
+======+==================================+=========================+
|    1 | c4ca4238a0b923820dcc509a6f75849b |                       0 |
|    2 | c81e728d9d4c2f636f067f89cc14862c |       0.6931471805599453 |
|    3 | eccbc87e4b5ce2fe28308fd9f2a7baf3 |       1.0986122886681098 |
|    4 | a87ff679a2f3e71d9181a67b7542122c |       1.3862943611198906 |
|    5 | e4da3b7fbbce2345d7772b0674a318d5 |       1.6094379124341003 |
+------+----------------------------------+-------------------------+
```

2️⃣ Perform updates on ternary, then query column c. ⚠️ Turn *auto commit* off such that changes accumulate in the Δ tables:

```
$ mclient -d scratch -l sql --autocommit ⬅

sql>EXPLAIN DELETE FROM ternary WHERE a = 981;
[...]
  X_9:bat[:int] := sql.bind(X_5, "sys", "ternary", "a", 0:int);
  C_6:bat[:oid] := sql.tid(X_5, "sys", "ternary");
  C_21 := algebra.thetaselect(X_9, C_6, 981:int, "==");
  X_26 := sql.delete(X_5, "sys", "ternary", C_21);
[...]
sql>DELETE FROM ternary WHERE a = 981;

sql>EXPLAIN UPDATE ternary SET c = -1 WHERE a = 982;
[...]
  X_10:bat[:int] := sql.bind(X_6, "sys", "ternary", "a", 0:int);
  C_7:bat[:oid] := sql.tid(X_6, "sys", "ternary");
  C_22 := algebra.thetaselect(X_10, C_7, 982:int, "==");
  X_24 := algebra.projection(C_22, X_10);
  X_28 := algebra.project(X_24, -1:dbl);
  X_30 := sql.update(X_6, "sys", "ternary", "c", C_22, X_28);
[...]
```

```
sql>UPDATE ternary SET c = -1 WHERE a = 982;

sql>EXPLAIN INSERT INTO ternary(a,b,c) VALUES (1001, 'Han Solo', -2);
[...]
  X_20 := sql.append(X_7, "sys", "ternary", "a", 1001:int);
  X_25 := sql.append(X_20, "sys", "ternary", "b", "Han Solo");
  X_28 := sql.append(X_25, "sys", "ternary", "c", -2:dbl);
[...]
sql>INSERT INTO ternary(a,b,c) VALUES (1001, 'Han Solo', -2);

sql>EXPLAIN SELECT t.c FROM ternary AS t;
[...]
  C_5:bat[:oid] := sql.tid(X_4, "sys", "ternary");
  X_8:bat[:dbl] := sql.bind(X_4, "sys", "ternary", "c", 0:int);
  (C_13:bat[:oid], X_14:bat[:dbl]) := sql.bind(X_4, "sys", "ternary", "c", 2:int);  ⬅ access Δᵘ
  X_11:bat[:dbl] := sql.bind(X_4, "sys", "ternary", "c", 1:int);,                   ⬅ access Δⁱ
  X_16 := sql.delta(X_8, C_13, X_14, X_11);                                         ⬅ apply Δᵘ, Δⁱ
  X_17 := algebra.projection(C_5, X_16);                                            ⬅ restrict to visible
[...]
sql> SELECT t.* FROM ternary AS t;
+------+----------------------------------+-------------------------+
| a    | b                                | c                       |
+======+==================================+=========================+
|    1 | c4ca4238a0b923820dcc509a6f75849b |                       0 |
[...]
|  980 | d79aac075930c83c2f1e369a511148fe |       6.887552571664617 | ⬅ row with a = 981 deleted
|  982 | fec8d47d412bcbeece3d9128ae855a7a |                      -1 | ⬅ updated
|  983 | 6aab1270668d8cac7cef2566a1c5f569 |       6.890609120147166 |
|  984 | d93ed5b6db83be78efb0d05ae420158e |       6.891625897052253 |
|  985 | 54a367d629152b720749e187b3eaa11b |       6.892641641172089 |
|  986 | fe7ee8fc1959cc7214fa21c4840dff0a |       6.893656354602635 |
|  987 | df6d2338b2b8fce1ec2f6dda0a630eb0 |       6.894670039433482 |
|  988 | 9908279ebbf1f9b250ba689db6a0222b |       6.895682697747868 |
|  989 | a1140a3d0df1c81e24ae954d935e8926 |      6.8966943316227125 |
|  990 | 4fac9ba115140ac4f1c22da82aa0bc7f |       6.897704943128636 |
|  991 | 692f93be8c7a41525c0baf2076aecfb4 |       6.898714534329988 |
|  992 | 860320be12a1c050cd7731794e231bd3 |       6.899723107284872 |
|  993 | 7b13b2203029ed80337f27127a9f1d28 |       6.900730664045173 |
|  994 | 934815ad542a4a7c5e8a2dfa04fea9f5 |       6.901737206656574 |
|  995 | 2bcab9d935d219641434683dd9d18a03 |       6.902742737158593 |
|  996 | 0b8aff0438617c055eb55f0ba5d226fa |       6.903747257584598 |
|  997 | ec5aa0b7846082a2415f0902f0da88f2 |       6.904750769961838 |
|  998 | 9ab0d88431732957a618d4a469a0d4c3 |       6.905753276311464 |
|  999 | b706835de79a2b4e80506f582af3676a |       6.906754778648554 |
| 1000 | a9b7ba70783b617e9998dc4dd82eb3c5 |       6.907755278982137 |
| 1001 | Han Solo                         |                      -2 | ⬅ inserted
+------+----------------------------------+-------------------------+
```

3️⃣ Replay MAL commands on mclient console:

Re-create and populate table ternary:

```
$ mclient -d scratch -l sql

 DROP TABLE IF EXISTS ternary;
 CREATE TABLE ternary (a int NOT NULL, b text NOT NULL, c double);

 INSERT INTO ternary(a, b, c)
   SELECT value, md5(value), log(value)
   FROM   generate_series(1, 1001);
```

Replay MAL commands:

```
$ mclient -d scratch -l mal

 sql.init();
 sql := sql.mvc();

# DELETE FROM ternary WHERE a = 981;
 ternary     :bat[:oid] := sql.tid(sql, "sys", "ternary");
 a0          :bat[:int] := sql.bind(sql, "sys", "ternary", "a", 0:int);
 deleted_rows:bat[:oid] := algebra.thetaselect(a0, ternary, 981:int, "==");
 sql_delete             := sql.delete(sql, "sys", "ternary", deleted_rows);

 io.print(deleted_rows);
#-------------------------#
# h t  # name
# void  void  # type
#-------------------------#
[ 0@0,  980@0  ]                        ⬅ oid of the deleted row


# UPDATE ternary SET c = -1 WHERE a = 982;
 ternary:bat[:oid]  := sql.tid(sql, "sys", "ternary");
 io.print(ternary);
#-------------------------#
# h t  # name
# void  oid  # type
#-------------------------#
[...]
[ 978@0,  978@0  ]
[ 979@0,  979@0  ]                      ⬅ visibility: row with oid 980@0 missing
[ 980@0,  981@0  ]
[ 981@0,  982@0  ]
```

```
[...]

 updated_rows  :bat[:oid] := algebra.thetaselect(a0, ternary, 982:int, "==");
 updated_rows_a:bat[:int] := algebra.projection(updated_rows, a0);
 updated_c     :bat[:dbl] := algebra.project(updated_rows_a, -1:dbl);
 sql_update               := sql.update(sql, "sys", "ternary", "c", updated_rows, updated_c);

 io.print(updated_rows, updated_c);
#-------------------------#
# t t t  # name
# void  void  dbl  # type
#-------------------------#
[ 0@0,  981@0,  -1  ]                    ◀ Δᵘ: oid of updated row and new c value


# INSERT INTO ternary(a,b,c) VALUES (1001, 'Han Solo', -2);
 sql_append := sql.append(sql, "sys", "ternary", "a", 1001:int);
 sql_append := sql.append(sql, "sys", "ternary", "b", "Han Solo");
 sql_append := sql.append(sql, "sys", "ternary", "c", -2:dbl);


# SELECT t.c FROM ternary AS t;
 ternary:bat[:oid] := sql.tid(sql, "sys", "ternary");
 c0     :bat[:dbl] := sql.bind(sql, "sys", "ternary", "c", 0:int);

 io.print(ternary);
#-------------------------#
# h t  # name
# void  oid  # type
#-------------------------#
[ 0@0,  0@0  ]
[...]
[ 978@0,  978@0  ]
[ 979@0,  979@0  ]           ◀ visibility: row with oid 980@0 still missing
[ 980@0,  981@0  ]
[ 981@0,  982@0  ]
[ 982@0,  983@0  ]
[ 983@0,  984@0  ]
[ 984@0,  985@0  ]
[ 985@0,  986@0  ]
[ 986@0,  987@0  ]
[ 987@0,  988@0  ]
[ 988@0,  989@0  ]
[ 989@0,  990@0  ]
[ 990@0,  991@0  ]
[ 991@0,  992@0  ]
[ 992@0,  993@0  ]
```

```
[ 993@0,  994@0  ]
[ 994@0,  995@0  ]
[ 995@0,  996@0  ]
[ 996@0,  997@0  ]
[ 997@0,  998@0  ]
[ 998@0,  999@0  ]
[ 999@0,  1000@0  ]          ⬅ visibility: new row with oid 1000@0 now visible

 io.print(c0);
#--------------------------#
# h t  # name
# void  dbl  # type
#--------------------------#
[ 0@0,  0  ]
[...]
[ 979@0,  6.887552571664617  ]
[ 980@0,  6.8885724595653635  ]
[ 981@0,  6.889591308354466  ]      ⬅ column BAT for c not updated yet: row 981@0 still visible
[ 982@0,  6.890609120147166  ]
[ 983@0,  6.891625897052253  ]
[ 984@0,  6.892641641172089  ]
[ 985@0,  6.893656354602635  ]
[ 986@0,  6.894670039433482  ]
[ 987@0,  6.895682697747868  ]
[ 988@0,  6.8966943316227125  ]
[ 989@0,  6.897704943128636  ]
[ 990@0,  6.898714534329988  ]
[ 991@0,  6.899723107284872  ]
[ 992@0,  6.900730664045173  ]
[ 993@0,  6.901737206656574  ]
[ 994@0,  6.902742737158593  ]
[ 995@0,  6.903747257584598  ]
[ 996@0,  6.904750769961838  ]
[ 997@0,  6.905753276311464  ]
[ 998@0,  6.906754778648554  ]
[ 999@0,  6.907755278982137  ]     ⬅ column BAT for c not updated yet: row 1000@0 not present yet

# access the Δ tables using sql.bind(..., 1) and sql.bind(..., 2)
 inserted_c:bat[:dbl]                          := sql.bind(sql, "sys", "ternary", "c", 1:int);     ⬅ access Δⁱ
 (updated_rows:bat[:oid], updated_c:bat[:dbl]) := sql.bind(sql, "sys", "ternary", "c", 2:int);     ⬅ access Δᵘ

 io.print(inserted_c);
#--------------------------#
# h t  # name
# void  dbl  # type
#--------------------------#
[ 1000@0, -2  ]
```

```
 io.print(updated_rows, updated_c);
#-------------------------#
# t t t  # name
# void  oid dbl  # type
#-------------------------#
[ 0@0,  981@0,  -1  ]

# implement changes (this is equivalent to sql.delta(sql, c0, updated_rows, updated_c, inserted_c))
 bat.replace(c0, updated_rows, updated_c, true);    ⬅ apply Δᵘ
 bat.append(c0, inserted_c);                        ⬅ apply Δⁱ
 c:bat[:dbl] := algebra.projection(ternary, c0);    ⬅ restrict visibility

 io.print(c);
[...]
[ 978@0,  6.88653164253051  ]
[ 979@0,  6.887552571664617  ]
[ 980@0,  -1  ]                    ⬅ updated
[ 981@0,  6.890609120147166  ]
[ 982@0,  6.891625897052253  ]
[ 983@0,  6.892641641172089  ]
[ 984@0,  6.893656354602635  ]
[ 985@0,  6.894670039433482  ]
[ 986@0,  6.895682697747868  ]
[ 987@0,  6.8966943316227125  ]
[ 988@0,  6.897704943128636  ]
[ 989@0,  6.898714534329988  ]
[ 990@0,  6.899723107284872  ]
[ 991@0,  6.900730664045173  ]
[ 992@0,  6.901737206656574  ]
[ 993@0,  6.902742737158593  ]
[ 994@0,  6.903747257584598  ]
[ 995@0,  6.904750769961838  ]
[ 996@0,  6.905753276311464  ]
[ 997@0,  6.906754778648554  ]
[ 998@0,  6.907755278982137  ]
[ 999@0,  -2  ]                   ⬅ inserted (also: 1000 rows only since 1 row deleted)
```