>>

Multi-version Concurrency Control

- Multi-version Concurrency Control
- Concurrency Control in PostgreSQL

^ >>

❖ Multi-version Concurrency Control

Multi-version concurrency control (MVCC) aims to

- retain benefits of locking, while getting more concurrency
- by providing multiple (consistent) versions of data items

Achieves this by

- readers access an "appropriate" version of each data item
- writers make new versions of the data items they modify

Main difference between MVCC and standard locking:

- read locks do not conflict with write locks ⇒
- reading never blocks writing, writing never blocks reading

❖ Multi-version Concurrency Control (cont)

WTS = timestamp of tx that wrote this data item

Chained tuple versions: $tup_{oldest} \rightarrow tup_{older} \rightarrow tup_{newest}$

When a reader T_i is accessing the database

- ignore any data item D created after T_i started
    - checked by: WTS(D) > TS(T_i)
- use only newest version V accessible to T_i
    - determined by: max(WTS(V)) < TS(T_i)

❖ Multi-version Concurrency Control (cont)

When a writer T_i attempts to change a data item

- find newest version V satisfying WTS(V) < TS(T_i)
- if no later versions exist, create new version of data item
- if there are later versions, then abort T_i

Some MVCC versions also maintain RTS (TS of last reader)

- don't allow T_i to write D if RTS(D) > TS(T_i)

❖ Multi-version Concurrency Control (cont)

Advantage of MVCC

- locking needed for serializability considerably reduced

Disadvantages of MVCC

- visibility-check overhead (on every tuple read/write)
- reading an item *V* causes an update of *RTS(V)*
- storage overhead for extra versions of data items
- overhead in removing out-of-date versions of data items

Despite apparent disadvantages, MVCC is very effective.

❖ Multi-version Concurrency Control (cont)

Removing old versions:

- V_j and V_k are versions of same item
- $WTS(V_j)$ and $WTS(V_k)$ precede $TS(T_i)$ for all T_i
- remove version with smaller $WTS(V_x)$ value

When to make this check?

- every time a new version of a data item is added?
- periodically, with fast access to blocks of data

PostgreSQL uses the latter (vacuum).

❖ Concurrency Control in PostgreSQL

PostgreSQL uses two styles of concurrency control:

- multi-version concurrency control (MVCC)
  (used in implementing SQL DML statements (e.g. **`select`**))
- two-phase locking (2PL)
  (used in implementing SQL DDL statements (e.g. **`create table`**))

From the SQL (PLpgSQL) level:

- can let the lock/MVCC system handle concurrency
- can handle it explicitly via **`LOCK`** statements

❖ Concurrency Control in PostgreSQL (cont)

PostgreSQL provides read committed and serializable isolation levels.

Using the serializable isolation level, a **`select`**:

- sees only data committed before the transaction began
- never sees changes made by concurrent transactions

Using the serializable isolation level, an update fails:

- if it tries to modify an "active" data item
  (active = affected by some other tx, either committed or uncommitted)

The transaction containing the update must then rollback and re-start.

❖ Concurrency Control in PostgreSQL (cont)

Implementing MVCC in PostgreSQL requires:

- a log file to maintain current status of each T_i
- in every tuple:
  - **`xmin`** = ID of the tx that created the tuple
  - **`xmax`** = ID of the tx that replaced/deleted the tuple (if any)
  - **`xnew`** = link to newer versions of tuple (if any)
- for each transaction T_i:
  - a transaction ID (timestamp)
  - SnapshotData: list of active tx's when T_i started

<< ^ >>

❖ Concurrency Control in PostgreSQL (cont)

Rules for a tuple to be visible to T_i:

- the **`xmin`** (creation transaction) value must
  - be committed in the log file
  - have started before T_i's start time
  - not be active at T_i's start time
- the **`xmax`** (delete/replace transaction) value must
  - be blank or refer to an aborted tx, or
  - have started after T_i's start time, or
  - have been active at SnapshotData time

For details, see:
`backend/access/heap/heapam_visibility.c`

❖ Concurrency Control in PostgreSQL (cont)

Tx's always see a consistent version of the database.

But may not see the "current" version of the database.

E.g. T1 does select, then concurrent T2 deletes some of T1's selected tuples

This is OK unless tx's communicate outside the database system.

E.g. T1 counts tuples; T2 deletes then counts; then counts are compared

Use locks if application needs every tx to see same current version

- **`LOCK TABLE`** locks an entire table
- **`SELECT FOR UPDATE`** locks only the selected rows

Produced: 15 Apr 2021