Lecture 5

Type	Lecture

MVCC - Writes

MVCC creates row versions with metadata: `created_by` (Xmin) and `deleted_by` (Xmax). **Garbage Collection** (VACUUM in PostgreSQL) Needs to run regularly in order to remove old versions and deleted objects (rows) and to free up space for re-use.

Serializable Snapshot Isolation (SSI)

PostgreSQL uses special internal markers (called predicate locks) to track reads/writes.

They don’t block others, but silently record dependencies. If a bad combination of transactions is about to break the rules, PostgreSQL says "Stop!" and rolls one back.

under postgres SSI false positives can happen, but in in Precisely SSI (PSSI) there can be no false positives.

Predicate locking

predicate locking (which is a monitoring mechanism), which means that it keeps "locks" which allow it to determine when a write would have had an impact on the result of a previous read from a concurrent transaction, had it run first. these locks do not cause any blocking and therefore can not play any part in causing a deadlock.

False positives

false positives refer to situations where the system **detects a potential problem or conflict that actually wouldn't have caused any real issue** — but still takes action as if it did.

- PostgreSQL tries to **predict and prevent anomalies** by watching which data each transaction reads and writes.
- Sometimes, the system **thinks two transactions might conflict**, even though in reality they would not have broken any rules.
- To be safe, PostgreSQL might **roll back a transaction anyway**, just in case.

HOT updates (Heap-Only Tuple Update)

When data is updated, instead of creating a new record elsewhere, the system adds it in the same spot.

Advantages: faster, uses less space, indexes don’t need to be updated.

Solution: Don’t add indexes to fields that change often. Set a `fillfactor` so pages don’t get full too soon.

requirements for HOT updates: Updated column MUST NOT be an indexed column, There must be space on the page for new versions.

Fill factor management

`Fillfactor` tells PostgreSQL how much space to leave free on each page. So future updates (like HOT) can happen without needing to move data.

Tweak `fillfactor` to balance between space usage and update performance.

▼ Example

UNIVERSITY

Fill Factor

1. ALTER TABLE teacher SET (fillfactor = 60);
   Sets fillfactor for a table to a specific value

2. SELECT relname, reloptions FROM pg_class WHERE relname = 'teacher';
   Returns the filllfactor for table teacher

3. SELECT indexname, indexdef FROM pg_indexes WHERE tablename = teacher';
   Returns the indexes defined on table teacher

4. SELECT relname AS index_name, reloptions FROM pg_class WHERE relkind = 'i' AND relname = 'teacher_pkey';
   Returns the fillfactor of the index

5. ALTER INDEX teacher_pkey SET (fillfactor = 80);
   Sets fillfactor for an index to a specific value

HOT and COLD Updates

HOT update: fast, keeps things local. needs to access only the heap.

COLD update: slower, changes indexes and logs more info. needs to access different btrees.

must be on the same page.

Monitor updates using PostgreSQL tools to see which kind is happening. Adjust table/index design to favor HOT.

MVCC

MVCC allows multiple people to read and write data at the same time by making copies (versions).

KUTAISI INTERNATIONAL UNIVERSITY

MVCC

Databas

MVCC Concepts – but no standard

1. Readers do not block writers and writers do not block readers.
2. Conflicting writes: may lead to lost updates or need to be rolles back.
3. Serializability: needs additional protocol rules
4. Updates create new versions of a row.
5. Version Storage: Pointers are used to create a version chain.
   1. Append-Only: Postgres
   2. Delta: Oracle / MYSQL: old versions kept in the undo log
   3. Time-Travel: SAP Hana: old versions kept in Time-Travel Storage
6. Garbage Collection to remove old versions.
7. Indexes: needs to point to or find the current version of a tuple

Different Styles:

- PostgreSQL keeps all versions in one place.
- Oracle keeps old versions in a special log.
- SAP Hana stores history in a second table.

MVCC Time-Travel Version Storage

1. On every update, the old version of the row is moved to a second table.
2. The main table always holds the current version.
3. Pointers in Time-Travel table go from New-to-Old
4. Pointers from main table to time-travel table need to be updated on every update.

▼ Example

NAL

MVCC Time-Travel Version Storage

Main Table

row ID	Version	Key	Balance	created_by	Deleted_by	Pointer
1	A2	Harry	9	T8	0	TTT row3
2	B1	Rose	4	T7	0	TTT row2

T6: Harry buys 10 lessons.
T7: Rose takes a lesson.
T8: Harry takes a lesson

What does the main table look like after the updates?

What does the time-travel table look like after the updates?

Time-TravelTable

row ID	Version	Key	Balance	created_by	Deleted_by	Pointer
1	A0	Harry	0	T1	T6	-
2	B0	Rose	5	T5	T7	
3	A1	Harry	10	T6	0	TTT row1

Dynamic SQL Statements

Dynamic SQL is when your program builds an SQL command on-the-fly using user input or variables.

Instead of writing a fixed SQL command like `SELECT * FROM users`, you construct it dynamically, like `SELECT * FROM users WHERE name = '" + userInput + "'`.

❗ If someone types in malicious text, they could trick the system into doing bad stuff (like deleting all data).

Safe input:

```
$userInput = "Kutaisi";
$query = "SELECT * FROM teacher WHERE city = '" . $userInput . "'";
// Resulting SQL: SELECT * FROM teacher WHERE city = 'Kutaisi'
```

Malicious input:

```
$userInput = "Kutaisi'; DROP TABLE teacher; --";
$query = "SELECT * FROM teacher WHERE city = '" . $userInput . "'";
// Resulting SQL: SELECT * FROM teacher WHERE city = 'Kutaisi'; DROP TABLE teacher; --'
```

SQL Injection

SQL Injection is a type of attack that happens when someone enters **malicious text into a form or input field**, and that text is sent directly to the database and executed as SQL code.

Original code:

```
SELECT * FROM users WHERE username = 'foo' AND password = 'bar';
```

But a user enters: Username: `foo' --` **Password:** anything.

```
SELECT * FROM users WHERE username = 'foo' --' AND password = 'anything';
```

The
`--` makes the rest of the line a comment, so the password part is ignored! Now the attacker logs in just by entering a username.

Defenses against SQL Injection

1. **Prepared Statements (Best Option):** Separates data from code, so user input is always treated as a value.
2. **Input Validation:** Reject unexpected characters like `'` or `-`.
3. **Escaping Input:** Add special characters to make user input safe (but less reliable than prepared statements).