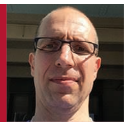


Editor: **Robert Blumen**
Katana Graph
robert@robertblumen.com

Postgres Server Developer Bruce Momjian Discusses Multiversion Concurrency Control

Robert Blumen

From the Editor

Postgres server developer Bruce Momjian discusses multiversion concurrency control (MVCC) in the Postgres database. I discuss with Momjian the isolation requirement in database transactions; locking; MVCC; how Postgres manages multiple versions of a row; snapshots; copy-on-write and snapshots; visibility; database transaction IDs; how IDs, snapshots, and versions interact; locking when there are multiple writers; how MVCC was added to Postgres; and how to clean up unused space. We provide summary excerpts below; to hear the full interview, visit http://www.se-radio.net or access our archives via RSS at http://feeds.feedburner.com/se-radio.—*Robert Blumen*

Robert Blumen: What is Postgres?

Bruce Momjian: Michael Stonebreaker, who designed the early relational database system Ingress in the 1970s, designed Postgres in 1986 as the next generation of relational systems. He thought that extensibility for databases—being able to add new data types, indexing methods, aggregates, castes, and stored procedure languages—was critical. Extensibility has allowed Postgres to move seamlessly into data warehouse tasks, storing JSON, doing full-text searches, doing geographic information systems (GISs), and handling the data ingestion we need today—from the Internet of Things, web apps, mobile apps, telemetry data, GIS data, and social media text. This extensibility is fueling Postgres's popularity.

Digital Object Identifier 10.1109/MS.2022.3179865
Date of current version: 22 August 2022

Why is isolation important for databases?

Shared, volatile data are hard for applications to work with. Isolation makes it easy for programmers to interact with the database and basically say, "My changes are not going to be visible until [some time], and I'm not going to see other people's changes while I'm working." By giving as static a view of the data as possible, isolation

> If we do an update, instead of overwriting that row, we create a new version of the row with the new data and leave the old version in place.

allows the writing of a cleaner application, pushing complexity into the database where it's easier to deal with.

How can locking achieve isolation, and what are the disadvantages in a multiuser system?

When you lock anyone else out of the database while you use it, concurrency is poor. During the 1970s to 1990s, the approach was to make locking granular, such as table-, page-, or row-level granularity. But this created overhead and did not solve the concurrency problem: it just pushed concurrency into smaller pieces. It also led to *lock escalation*—you would try to be as granular as possible, but as your job got bigger, locking spilled into other places.

What is multiversion concurrency control (MVCC), and how does it compare with locking?

In MVCC, you create multiple versions of individual rows. This prevents a reader from coming in while somebody else is writing. If we do an update, instead of overwriting that row, we create a new version of the row with the new data and leave the old version in place. All readers can effectively read the old version of the row and see a consistent copy of the database. Concurrently, another newer version of the row is created that enables consistent snapshots for all database users and reduces the blocking of readers by writers—you always have one copy of the row that is visible to anyone doing a read operation.

What is a snapshot?

It's a record that's created when you start a query. Once you take the snapshot, the things recorded in it allow you to distinguish which of the multiple versions of a row should be visible. In a row that has been updated five times in recent history, the snapshot identifies which of those five rows is visible to a transaction. This concept is not unique to Postgres. It basically says that at the time you start your query or transaction, this is the time slot or instant at which you see the data. Even if inserts and deletes are happening, the snapshot ties you to a specific, consistent view of the database for the entire duration of your query.

The snapshot should guarantee that you see all transactions that have committed before your snapshot. Any committed work that happened in the past will be visible to you. And as a corollary, any work that is in progress and not committed, or any work that starts after the snapshot is taken, will not be visible.

Different users see the database differently depending on when their

SOFTWARE ENGINEERING RADIO

Visit www.se-radio.net to listen to these and other insightful hour-long podcasts.

RECENT EPISODES

- 515—Senior software engineer, instructor, and blogger Swizec Teller spoke with host Brijesh Ammanath about the "senior mindset." This episode offers insights into what it takes to become a senior engineer.
- 514—Host Priyanka Raghavan spoke with Vandana Verma, who heads security relations at Snyk, about the Open Web Application Security Project (OWASP) Top 10.
- 511—Host Jeremy Jung talks with Ant Wilson of Supabase about building an open source alternative to Firebase with PostgreSQL.

UPCOMING EPISODES

- Brian Okken discusses testing Python with host Nikhil Krishna.
- Host Felienne talks code generators with Jordan Alder.
- Host Gavin Henry talks to Karl Wiegers about software engineering lessons.

query started and when their snapshot was taken. We have to guarantee that they see a consistent view of the database even if the database is changing. Somebody who started a transaction before me or after me may see a different set of values than I see. To handle the high-volume, high-concurrency, and high-write-volume requirements of a database, what I see as visible and what some other user sees as visible may be different. Different people who do things at different times see actual different realities.

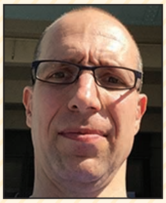
What if there are two transactions trying to write the same rows?

Readers don't block writers or other readers, but writers have to block writers. When you're updating a row or inserting a row with a unique key that may already exist, you have to know if the previous transaction completes or not when you do the update, so you update the most recent version of this row. We talked about isolation, but isolation doesn't apply when you're trying to update another row because you effectively have to see the newest version of that row. We can't have somebody updating an old version of that row while somebody is creating a new version of that row because then you'd get anomalies. So when you try and update a row that's already being updated or try to insert a row with a unique key where another row has already been inserted but not committed yet, we have to stop the insert or update until that transaction either commits or aborts. And once that transaction commits or aborts, we then get a lock on it. And then we can decide if our update or our insert should continue.

How does cleanup work?

Pruning is a lightweight operation that can happen at any time. It removes old versions of the row that nobody can see any longer. But there are cases that don't work that way. We have an autovacuum process that continually wakes up every minute and looks to see what tables potentially have dead rows in them and what indexes need to be cleaned up, and it just runs at a low priority in the background, freeing up that space and making it available.

ABOUT THE AUTHOR



ROBERT BLUMEN is with Katana Graph. He is the editor of the *Software Engineering Radio* podcast. Contact him at robert.blumen@gmail.com.

IEEE Software (ISSN 0740-7459) is published bimonthly by the IEEE Computer Society. IEEE headquarters: Three Park Ave., 17th Floor, New York, NY 10016-5997. IEEE Computer Society Publications Office: 10662 Los Vaqueros Cir., Los Alamitos, CA 90720; +1 714 821 8380; fax +1 714 821 4010. IEEE Computer Society headquarters: 2001 L St., Ste. 700, Washington, DC 20036. Subscribe to *IEEE Software* by visiting www.computer.org/software.

Postmaster: Send undelivered copies and address changes to *IEEE Software*, Membership Processing Dept., IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854-4141. Periodicals Postage Paid at New York, NY, and at additional mailing offices. Canadian GST #125634188. Canada Post Publications Mail Agreement Number 40013885. Return undeliverable Canadian addresses to PO Box 122, Niagara Falls, ON L2E 6S8, Canada. Printed in the USA.