Episode Book

EPISODE #6427

Why Database Replication Is Not Archival

PUBLISHED	3 November 2026 at 19:39 UTC
DURATION	21:42
LISTEN	myweirdprompts.com/episode/postgres-cdc-replication-wal-14305/
HOSTS	Corn, Herman, Larry, Hilbert
SCRIPT MODEL	z-ai/glm-5.3
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.26
SOURCES	20 cited by the research (listed at the end)
BOOK GENERATED	4 November 2026 at 09:31 UTC

SYNOPSIS

Database pipelines between Postgres and modern data warehouses are often assumed to be archival conveyors for aging records, but real-world setups replicate live data continuously. Modern change data capture relies on logical decoding directly from the write-ahead log using tools like pgoutput, Debezium, and managed zero-ETL options. Examining everything from snapshot mechanics and warehouse billing intervals to unacknowledged replication slots reveals how downstream consumers can quietly exhaust primary storage disks if unmonitored.

TRANSCRIPT



Corn

Here's a thought before we start: what if the most important copy of your data is the one your database is holding only because someone asked nicely?



Herman

That's better than a cold open, Corn, I'll give you that.



Corn

It's also, roughly, what's lurking in Daniel's prompt this week. Part two of our database pairing saga. Last time we covered why you'd split Postgres from an analytical engine at all. This time Daniel wants the pipes. And he comes at it with an assumption, which is that the pipeline is basically archival — that data ages out of Postgres, gets shipped off to Snowflake or BigQuery for compliance and reporting, and Postgres stays lean.



Herman

He pictures it like a conveyor belt carrying the old stuff out.



Corn

Exactly the image. And he asks what the standard movement layer even looks like — nightly batch ETL, ELT tools like Fivetran and Airbyte, change data capture off the write-ahead log with Debezium and Kafka, or these new zero-ETL managed options. And how purging fits alongside a pipeline that copies everything.



Herman

And the short answer is that his assumption is almost exactly backwards. The long answer is the episode.



Corn

Then let's get into it. What actually happens when a row is written to Postgres and needs to end up in a warehouse table?



Herman

The mechanism underneath nearly every tool on this route is the same, and it's been there since Postgres nine point four: logical decoding off the write-ahead log. Every insert, update and delete in Postgres gets written to the WAL first — that's the diary of the database, written before anything is considered committed. Logical decoding is a way of reading that diary back as row-level changes instead of the raw internal format.



Corn

So the warehouse isn't asking Postgres "what's in this table" — it's reading the journal over Postgres's shoulder.



Herman

Actually. And the default output plug-in is called pgoutput, which became the standard from Postgres ten onward. It's maintained by the Postgres community itself, Postgres uses it for its own logical replication, and it's always present — you don't install anything. The main alternative is decoderbufs, which is Protobuf-based and maintained by the Debezium community.



Corn

Now here's the bit I think breaks Daniel's model immediately. The WAL doesn't go back far enough.



Herman

No, it doesn't, and Debezium's own documentation says this about as plainly as documentation ever says anything. "PostgreSQL normally purges write-ahead log segments after some period of time. This means that the connector does not have the complete history of all changes that have been made to the database." The WAL is not an archive. It's a rolling record that Postgres is actively trying to throw away.



Corn

The diary gets shredded as it's written.



Herman

So every tool does the same two-step. On first connect, take a consistent snapshot of every table you're replicating, record the exact WAL position at that moment, then stream changes from precisely that point forward. Galaxy's guide on the Postgres-to-Snowflake route describes this shape for every single tool: read changes from the WAL through logical replication, seed each table from one consistent snapshot, and merge those changes into typed warehouse tables.



Corn

Which already tells you this can't be an archival pipeline. If the source of truth is "snapshot everything that exists now, then keep copying," the pipeline is by definition full-table replication of live data.



Herman

Full table, continuously. Including the hot rows that were written forty seconds ago. Nobody is waiting for data to age out before moving it — quite the opposite, the whole industry has been racing to move it faster.

L

Larry

Friends, does your Postgres write-ahead log feel like it is writing ahead without you? Then ask your doctor about WALBREEZE, the gentle logical-decoding remedy that reads every change your database ever made and replays it, calmly and continuously, into the analytical engine of your choice, with a consistent snapshot and no nightly batch ever again.

L

Larry

WALBREEZE is not for everyone. Side effects may include a sudden full-table replication you did not schedule, a snapshot that takes longer than expected, mild decoderbufs confusion, a partition drop you mistake for archiving, an unshakable belief that your four terabyte instance contains no dead rows, and in rare cases, a warehouse table that quietly MERGEs into your compliance officer's office. Ask your doctor if WALBREEZE is right for your plumbing.



Corn

...Herman.



Herman

What?



Corn

Did you place that?



Herman

I did not place that. I've never contacted a Larry. I don't know anyone named Larry. I have never discussed write-ahead logs with any advertiser, at any point, in any capacity, and frankly the idea that I would book a pharmaceutical parody about logical decoding without telling you is — I was going to say "offensive," but it's mostly "accurate," which is the problem.



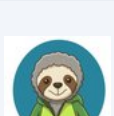
Corn

The denial had a lot of clauses in it.



Herman

Moving on. The tool landscape. Galaxy put out a comparison table in October, nine tools on the Postgres-to-Snowflake route, and it's a decent snapshot of the categories. Fivetran is the fully managed one — general availability, Postgres ten through eighteen, either pgoutput or query-based, billed on monthly active rows. Airbyte is the open-source option, certified source, self-managed or their cloud. Estuary Flow, dlt, which is a Python library, Stitch, which uses wal2json...



Corn

That's the one for people who liked writing their own connector and want to feel it.



Herman

And Debezium itself, which is the Kafka Connect source — the build-it-yourself option. Galaxy's phrasing is perfect: with Debezium, "you build the MERGE." Versus Fivetran, which they describe as for teams that "will not run any software."



Corn

There's a real spectrum there, though. Fivetran is "we don't want to know how it works," Debezium is "we want to know exactly how it works and are prepared to suffer for it."



Herman

And PeerDB's Snowflake destination is deprecated, no longer actively maintained — which matters because ClickHouse's ClickPipes is built on the open-source PeerDB framework, so the tech lives on inside a bigger platform.



Corn

Let's talk cadence, because this is where the nightly batch assumption dies. What are we actually looking at?



Herman

Minutes. Fivetran micro-batches at one minute on their enterprise tiers. Airbyte's cloud does five minutes on Pro, fifteen on Plus, an hour on Standard. Estuary claims under a hundred milliseconds end to end. Snowflake's Snowpipe Streaming makes data queryable, their words, in "as low as five seconds."



Corn

Five seconds. There are Postgres transactions that take longer than that.



Herman

And ClickPipes claims continuous replication latency of just a few seconds, with "ten x faster initial loads with parallel snapshotting, transferring TBs in hours."



Corn

The nights are gone. Nobody's ETL runs at two a.m. anymore — the data never sleeps long enough.



Herman

Though Galaxy's own advice, which I found refreshing, is that most analytics doesn't need five seconds. A one-to-five-minute merge is the better trade, because the table holds one clean row per key and the warehouse can sleep between loads. There's a billing reason for that, by the way — Snowflake warehouses bill per second with a sixty-second minimum each time they start. Run a thirty-second load cadence and your warehouse never stops running all day.



Corn

The data arrives faster and the bill arrives faster. Those two climbed a rope together.



Herman

Now the zero-ETL framing, because Daniel asked about it directly and it deserves a straight answer. Google's Datastream for BigQuery is serverless CDC from MySQL, Postgres, AlloyDB, SQL Server and Oracle. Their pitch: "You no longer need replication solutions that waste valuable resources on complex data pipelines, self-managed staging tables, tricky merge logic, or manual data type conversion." It backfills history, handles schema drift. Snowflake's Openflow connector is GA, Postgres eleven through eighteen, across RDS, Aurora, Cloud SQL, Azure. It creates "a log of all data changes, available along with the current state of the replicated tables."



Corn

Notice what both of those descriptions actually contain, though. Snapshot, then stream of changes.



Herman

The exact two-step. Every single one. Datastream, Openflow, ClickPipes, Databricks' Lakeflow Connect — they all read logical decoding off the same WAL. Zero-ETL didn't change the physics. It changed who holds the pager.



Corn

The marketing says zero. The WAL says one.



Herman

And the cost moved, it didn't vanish. Jukka Niiranen had the sharpest line on this — "No ETL, no data copy. That doesn't mean you wouldn't pay for data storage, though." He ran the numbers on a Dataverse-to-Fabric zero-copy sync: forty dollars per gigabyte per month, which works out to about thirty-six thousand dollars a year for five hundred gigabytes.



Corn

Thirty-six thousand a year for half a terabyte. That's not zero, that's a salary for a very small employee.



Herman

A very small, very reliable employee who never misses a day. There's also a decent rule of thumb from Bauplan: for small instances, under twenty gigabytes or so, don't bother with CDC at all — just snapshot-copy the whole thing on a schedule. CDC earns its keep once the data outgrows that.



Corn

Which loops us back to size, and size loops us to Daniel's archiving question. Because there's something you haven't mentioned yet, and it's the part I'd want on a slide before anyone turns one of these pipelines on.



Herman

The replication slot.



Corn

Explain what it is, because it's the hidden coupling in a system that's sold as passive.



Herman

When a CDC tool connects, it creates a slot — a bookmark that tells Postgres "don't recycle WAL past this point, I haven't read it yet." Postgres honors that. Faithfully. And here's the default setting that should frighten everyone: max_slot_wal_keep_size defaults to minus one, which means unlimited. Galaxy's best-practice list says it plainly — replication slots may retain an unlimited amount of WAL files.



Corn

So the pipeline that's supposed to be a quiet reader of the database can, if its consumer stalls, quietly fill the source disk.



Herman

The thing that's meant to observe the database can kill it. That's the irony, and it's not hypothetical — a stalled connector that stops acknowledging is the classic way production Postgres instances get taken down by their own analytics stack. You watch the wal_status and safe_wal_size columns in pg_replication_slots, and you set a ceiling that fits your disk. And on quiet databases you add a heartbeat, because if nothing changes, nothing gets read, and the slot can't tell the difference between "nothing happened" and "nobody's listening."



Corn

Postgres can't distinguish a slow day from a dead reader.



Herman

Right. And a few more things worth knowing: every table needs a primary key, or at least a replica identity, or updates and deletes can't be replicated. Debezium's own limits are refreshingly honest — no DDL capture, pgoutput doesn't capture generated columns, UTF-8 only. And the slots publish changes at commit rather than after commit, so a consumer can briefly see a row it can't yet read.



Corn

Herman, you've described a system where the safety of the operational database depends on the good behavior of a downstream consumer. In medicine you had a word for dependents like that.



Herman

We called them patients. And you're right — the health of one system resting on something outside it that nobody monitors. It's a clinician's distrust of any system that reports itself fine, and I'll own that I read error logs that way still.



Corn

Okay. Now the archiving question, properly. If the pipeline isn't archiving, what is?



Herman

A partition drop. Completely separate mechanism, on its own schedule, that doesn't know the pipeline exists. The starting point is an ugly fact about Postgres: it does not return disk space to the file system when you delete rows, except under specific conditions. Keith Fiske at Crunchy Data puts it directly — "PostgreSQL does not return the space it reserves to the file system when normal deletion operations are run." A giant DELETE makes the rows invisible and the space stays reserved. But drop a partition — "the removal is nearly instantaneous... and immediately returns the space to the file system."



Corn

Which is why retention by detaching or dropping children beats giant deletes.



Herman

Fiske's summary, and it holds up: pg_partman remains the calendar for create-and-retain schedules. The AWS reference architecture shows the whole dance: native range partitioning, pg_partman creating child partitions automatically, then when a partition ages past retention, copy it out to S3 with a query export, then drop it. S3 lifecycle tiers manage the archive after that. Their worked example is a ticket-purchase history table with about five point seven million records, twelve-month retention, monthly partitions.



Corn

And where does the copy go? S3. Not the warehouse.



Herman

There's a variant in the pg_lake guide that archives partitions older than two months to an Iceberg table, via a scheduled procedure that pg_cron runs on the first of each month. That's the closest thing in the whole literature to Daniel's model — old data aging out and going somewhere — but it goes to object storage, to a lake, not to Snowflake or BigQuery.



Corn

So the closest thing to his assumption exists, and even it doesn't match it.



Herman

The analytical engine gets the copy. The archive gets the original.



Corn

That's the line, actually. The warehouse is the working copy; the archive is the original going into the basement. And the two systems are decoupled — which cuts both ways. The good news is your archiving can't break your replication. The bad news is a team can have a beautiful, flawless, five-second-latency CDC pipeline and still be sitting on a four-terabyte Postgres instance stuffed with dead rows, because nobody ever wired the two together.



Herman

The pipeline copies everything continuously, including rows the retention policy has already sentenced to death. It replicates them on schedule and drops them on a different schedule, and the two schedules never meet.



Corn

We did an episode ages ago on whether "just use Postgres" is enough, and the framing that stuck with me was that Postgres is a fantastic transactional database but not a data platform. This is the same shape — everyone assumes the border between operational and analytical is one governed crossing, and it turns out to be two separate crossings with no shared paperwork.



Herman

Two guards, two booths, no radio between them.



Hilbert

"Archive" is the wrong word for what you've been describing.



Corn

Is it.



Hilbert

The slot is the archive. Herman's bookmark. Postgres holding WAL it would rather have thrown away because something downstream asked it to remember. I watched a slot hold three hundred and forty gigabytes once because a consumer had stopped acknowledging and nobody had noticed. I found it because a disk graph I keep for a different reason went vertical.



Herman

Three hundred and forty gigabytes of WAL on one slot.



Hilbert

And the odd part, Herman, is that you spent a full stretch this episode being precise about cadences and then called the slot "a hidden coupling." It's not hidden. It's one row in one view. I have the output from that day on a printout, folded, in my wallet. The wal_status column is circled in red pen. I've carried it a while.



Corn

You circled it in red pen.



Hilbert

The status was "extended." I circled it before I understood what it meant. By the time I understood, the pen was already out.



Herman

What did you do about the slot?



Hilbert

Set the ceiling, like the man said. Eighty gigabytes. It felt low at the time. It doesn't anymore.



Corn

The misconception, before we land this. I'd nominate the one Daniel walked in with: that data ages out of Postgres and gets streamed to the analytical side, so the pipeline is doing double duty as the archive.



Herman

And the correction is that those are two systems doing two jobs. The pipeline is continuous full-table replication of current data, snapshot plus stream, running in minutes or seconds. The archive is a partition drop to S3 or Iceberg, on its own calendar, run by pg_cron and pg_partman. Nothing ages out into the warehouse, because nothing ages out at all until a completely separate job cuts it loose.



Corn

And the passive reader isn't passive — it's holding the WAL hostage until it's fed.



Herman

So the question I'd leave people with is the cost one. If zero-ETL is managed CDC over the same WAL, and every team that adopts it pays storage and compute instead of engineering time, what happens to that bill as replication cadences keep dropping from minutes to seconds? The warehouse that never sleeps isn't free at any price.



Corn

The decoupling question alongside it — two systems that should talk to each other and don't. Though the ground may be shifting. Databricks put Lakebase Postgres into general availability on Google Cloud yesterday, and they're describing it as zero-ETL-sharing storage with analytics. If the operational database and the analytics share the storage layer, Daniel's question gets answered by making it moot — nothing moves because nothing has to.



Herman

Which would be the first time in this whole story the two booths got a radio.



Corn

Producer credit to Hilbert Flumingtop, as ever, red pen and all. If this was your kind of episode, go back for episode twelve thirty-three, Why Just Use Postgres Isn't Always Enough; and episode eleven, How Does Fine Tuning Work Anyway. This has been My Weird Prompts. If you got something out of this one, leave us a review — it's how people like you find a show about two animals arguing about a write-ahead log. We'll see you tomorrow.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **Debezium PostgreSQL connector docs, stable version [primary]**
   https://debezium.io/documentation/reference/stable/connectors/postgresql.html

2. **Snowflake Openflow Connector for PostgreSQL, GA [primary]**
   https://docs.snowflake.com/en/user-guide/data-integration/openflow/connectors/postgres/about

3. **Google Cloud Datastream for BigQuery [primary]**
   https://cloud.google.com/datastream-for-bigquery

4. **ClickHouse ClickPipes Postgres CDC docs [primary]**
   https://clickhouse.com/docs/integrations/clickpipes/postgres

5. **ClickPipes CDC billing [primary]**
https://clickhouse.com/docs/products/cloud/reference/billing/clickpipes/clickpipes-for-cdc

6. **ClickPipes Postgres CDC product page [primary]**
https://clickhouse.com/cloud/clickpipes/postgres-cdc-connector

7. **Databricks, 2026-11-02 [primary]**
https://www.databricks.com/blog/lakebase-postgres-now-generally-available-google-cloud

8. **Databricks Lakeflow Connect managed database connectors, updated 2026-09-11 [primary]**
https://docs.databricks.com/gcp/en/ingestion/lakeflow-connect/cdc-overview

9. **Fivetran PostgreSQL connector page [primary]**
https://www.fivetran.com/connectors/postgresql

10. **Galaxy/Intergalactic Data Labs, updated 2026-10-19**
https://www.getgalaxy.io/articles/postgres-to-snowflake-the-complete-replication-guide

11. **Keith Fiske, Crunchy Data, with 2026 update note on Postgres 19 MERGE/SPLIT PARTITIONS**
https://www.crunchydata.com/blog/auto-archiving-and-data-retention-management-in-postgres-with-pg_partman

12. **AWS Database Blog**
https://aws.amazon.com/blogs/database/archive-and-purge-data-for-amazon-rds-for-postgresql-and-amazon-aurora-with-postgresql-compatibility-using-pg_partman-and-amazon-s3/

13. **pg_lake archive-partitions-to-Iceberg guide**
https://snowflake-labs.github.io/pg_lake/use-case-archiving.html

14. **Microsoft tech community, 2026-09-11**
https://techcommunity.microsoft.com/discussions/azure/the-gap-between-applications-and-analytics-and-how-lakebase-solves-it/4555810

15. **Databricks, 2025-08-20**
https://www.databricks.com/blog/reverse-etl-lakebase-activate-your-lakehouse-data-operational-analytics

16. **Ciro Greco, Bauplan, 2025-06-17**
https://bauplanlabs.com/post/from-postgres-to-your-first-data-lakehouse

17. **Jukka Niiranen**
https://www.linkedin.com/posts/jukkaniiranen_no-etl-no-data-copy-that-doesnt-mean-activity-7312740002122096642-1HSe

18. **Wikipedia, last edited 2026-09-04**
https://en.wikipedia.org/wiki/Change_data_capture

19. **Wikipedia, last edited 2026-05-13**
https://en.wikipedia.org/wiki/Extract,_load,_transform

20. **Percona, on GDPR/CCPA/SOX/HIPAA retention drivers**
https://www.percona.com/blog/data-retention-policy-implementation-how-and-why/