Peter Geoghegan

@petervgeoghegan

https://speakerdeck.com/peterg/logical-database

Thinking About The Logical Database

PGCon 2022 — May 27, 2022

https://speakerdeck.com/peterg/logical-database

Overview

1. **Physical data independence**

   Defining the logical and physical database

2. **A cultural divide**

   2PL versus versioned storage

3. **Using high level semantic information to solve low-level problems**

   Bottom-up index deletion

4. **Seeing the wisdom in 2PL**

   “Transaction rollback” without UNDO

Physical data independence origins

- Term originally described a key advantage of relation systems over hierarchical systems

  - Tree-like organization — **one-to-many only**

  - **Child** nodes can have **exactly one parent**

  - Very inflexible — made schema migrations hard

- The **logical database** is *independent* of the **physical database**

- Term not in widespread use today, but must have been easily understood by users back in the 1970s

https://speakerdeck.com/peterg/logical-database

NEON

Physical data independence now

- Term seemed to take on a slightly different meaning once relational systems became dominant

  - Helps discussion of competing designs for access methods, concurrency control, etc

  - Relates useful work to costs, such as physical IOs

- It's easy to **overlook opportunities** to improve Postgres if you just take all this for granted

https://speakerdeck.com/peterg/logical-database

NEON

Table 1: Summary of Logical Database

Relation Name	Cardinality	Tuple Length	Tuples Per 4K Page
warehouse	W	89 bytes	46
district	W * 10	95 bytes	43
customer	W * 30K	655 bytes	6
stock	W * 100K	306 bytes	13
item	100K	82 bytes	49
order		24 bytes	170
new-order		8 bytes	512
order-line		54 bytes	75
history		46 bytes	89

TPC-C overview [7]. In this paper, we focus only on the access patterns and processing requirements of the benchmark. For concreteness, we will assume a relational database model, though most of the development is applicable to other data models. We first give an overview of all five transaction types in the benchmark and then give a more detailed account of each of the transactions in the following section.

Pictured: Diagram from A modeling study of the TPC-C Benchmark, Leutenegger & Dias 1993

Table 4: Throughput Model Summary : Single Node

resource	parameter	n	overhead	NewOrder V_1	Payment V_2	Status V_3	Delivery V_4	Stock V_5
CPU	select	1	10K	23	4.2	13.2	130	1
CPU	update	2	10K	11	3	0	120	0
CPU	insert	3	10K	12	1	0	0	0
CPU	delete	4	10K	0	0	0	10	0
CPU	commit	5	20K [1]	1	1	1	1	1
CPU	initIO	6	5K	$1 + mc$ $+10(mi + ms)$	1+2 2(mc)	2.2(mc) +mo+10(ml)	1+10(mc+mo+mn) +130(ml)	200(ms+ml)
CPU	application	7	0.1K	47	8	13	261	3
CPU	send/receive	8	15K	0	0	0	0	0
CPU	prepCommit	9	10K	0	0	0	0	0
CPU	initTransaction	10	20K	1	1	1	1	1
CPU	releaseLocks	11	35K	1	1	1	1	1
CPU	non-unique-select	12	25K	0	0.6	0.6	0	0
CPU	join	13	820K	0	0	0	0	1
disk	IO	14	25ms	$mc + 10(mi + ms)$	2.2(mc)	2.2(mc) +mo+10(ml)	10(mc+mo+mn) +130(ml)	200(ms+ml)

Pictured: Diagram from A modeling study of the TPC-C Benchmark, Leutenegger & Dias 1993

The logical database is independent of the physical database

- Logical database — abstract idea
  - Tables that consist of rows, that may have some known physical characteristics
  - May be grouped into “logical pages”, which are always transactionally consistent
  - Reductive, but potentially very useful
- Physical database — “what’s really going on” with storage and concurrency control
  - Could differ based on DBA choices about index or table AMs (per classic definition of “physical data independence”)
  - Interesting to me as a tool for discussing Postgres internals

https://speakerdeck.com/peterg/logical-database

NEON

Overview

1. **Physical data independence**

   Defining the logical and physical database

2. **A cultural divide**

   2PL versus versioned storage

3. **Using high level semantic information to solve low-level problems**

   Bottom-up index deletion

4. **Seeing the wisdom in 2PL**

   “Transaction rollback” without UNDO

https://speakerdeck.com/peterg/logical-database

NEON

“The truly monolithic piece of a DBMS is the transactional storage manager that typically encompasses four **deeply intertwined** components:

1. A **lock manager** for **concurrency control**.
2. A log manager for recovery.
3. A buffer pool for staging database I/Os.
4. Access methods for organizing data on disk.”

– Architecture of a Database System, Hellerstein et al [**emphasis** added]

The curse of knowledge — implicit knowledge, in particular

To a Postgres hackers, “physical database independence” might seem **absurdly obvious**

- But if you naturally “think in 2PL terms”, then it’s not obvious at all

- ARIES style transaction rollback implies tight coupling between physical and logical

- It’s well worth understanding *why* it isn’t obvious to engineers with a background in 2PL systems

https://speakerdeck.com/peterg/logical-database

NEON

2PL transactional storage versus versioned storage: per-system breakdown

- InnoDB, SQL Server, and Oracle considered 2PL systems here — use next-key locking, only limited use of multiversioning

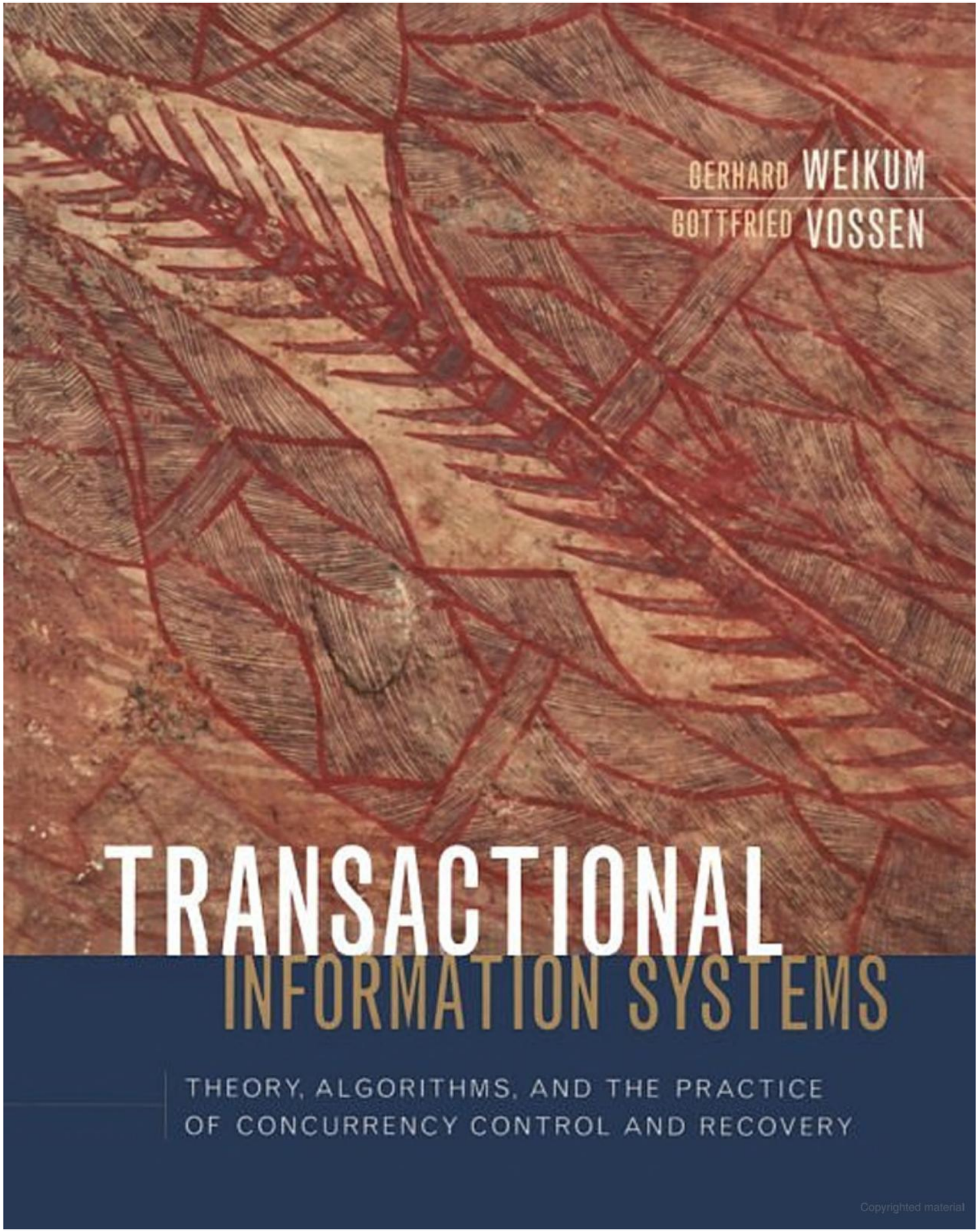
- RocksDB is a MySQL storage engine from Facebook that uses a log structured merge tree for tables and indexes

  - Full-featured transactional database, comparable to InnoDB — compatibility important

  - Takes a similar approach to Postgres (replaces InnoDB's next-key locking with snapshot isolation)

https://speakerdeck.com/peterg/logical-database

NEON



2PL transactional storage versus versioned storage

- 2PL is a mutable state paradigm — though that's implicit

  - *Storage* is a transactional black box

  - **Minimal divergence** between logical and physical database is the design principle of interest — **pages themselves** are transactionally consistent

  - **Concurrency control** mostly belongs to the storage layer

- Postgres MVCC **versions logical rows** in storage — but *storage itself* is not transactional, and "objects" (relations including indexes) are *not* versioned

  - Less monolithic, more decoupled — enables flexible, extensible indexing

  - High degree of physical data independence comes *naturally*

https://speakerdeck.com/peterg/logical-database

NEON

“On the other hand, MyRocks adopted Snapshot Isolation model for Repeatable Read, which was **the same as found in PostgreSQL**. We considered the InnoDB style Gap Lock based implementation as well, but we concluded that Snapshot Isolation was **simpler to implement**.”

– MyRocks: LSM-Tree Database Storage Engine Serving Facebook's Social Graph,
Matsunobu et al 2020 [**emphasis** added]

Brief aside: MyRocks and long-running transactions

MyRocks handles long-running transactions gracefully compared to InnoDB

- Blog post from Mark Callaghan explains InnoDB issue
  - MySQL Bug #74919, “purge should remove intermediate rows between far snapshots”
- Speculation:
  - A greater degree of physical data independence likely enabled this optimization — it was simple enough with LSM storage, so why not do it?
  - More difficult in InnoDB because rows must be read from **transactionally consistent pages** — sometimes by reconstructing them using UNDO
- MyRocks primary **and** secondary indexes have xmin-like metadata, which seems like it might be a downside that naturally accompanies the flexibility

“InnoDB implemented UNDO logs as a linked list and needed to keep all changes in the list after creating a transaction snapshot. It also needed to rewind the list to find the row based on the consistent snapshot. This caused significant **slowdown** if there were **hot rows that changed a lot** and a **long running select** needed to read the row after creating a snapshot.

In MyRocks, a long running snapshot can maintain a reference to the **specific version** of the row needed.”

– MyRocks: LSM-Tree Database Storage Engine Serving Facebook's Social Graph,
Matsunobu et al 2020 [**emphasis** added]

“Locks versus latches” — confusingly unconfusing, or just confusing?

	Locks	Latches
Separate …	User transactions	Threads
Protect …	Database contents	In-memory data structures
During …	Entire transactions[2]	Critical sections
Modes …	Shared, exclusive, update, intention, escrow, schema, etc.	Read, writes, (perhaps) update
Deadlock …	Detection & resolution	Avoidance
… by …	Analysis of the waits-for graph, timeout, transaction abort, partial rollback, lock de-escalation	Coding discipline, instant-timeout requests, “lock leveling”[3]
Kept in …	Lock manager’s hash table	Protected data structure

Fig. 4.3 Locks and latches.

Pictured: Diagram from “Modern B-Tree Techniques“ by Goetz Graefe — Transactional Techniques chapter

"While the number of **choices** for physical database design is **confusing**, the most significant source of complexity in physical database design is that **many decisions are interdependent**"

– Options in physical database design, Graefe 93 [**emphasis** added]

Free space management for a 2PL based heap table access method

Free space management might be the single best example of this cultural divide

- Designs use UNDO, participate in concurrency control/rollback

  - Very **subtle bugs** possible due to transaction abort with concurrent access

  - What if a transaction aborts, free space related UNDO needs to restore original larger row in place, but finds that there isn't quite enough space in the page?

- These requirements must make "transactional storage" seem natural

  - Limits physical data independence

https://speakerdeck.com/peterg/logical-database

NEON

“The space reserved by one terminated transaction might get carried over as the reservation of another transaction! This is possible since the RID released by one transaction (through the delete of the corresponding record) might be reused to identify a newly inserted record of another transaction.

If the latter transaction did not use the space released by the former while inserting that record, then **that space will not be available to anyone** until the latter **transaction** terminates.”

— Algorithms for Flexible Space Management in Transaction Systems Supporting Fine-Granularity Locking, Mohan & Haderle 1994 [**emphasis** added]

Overview

1. **Physical data independence**

   Defining the logical and physical database

2. **A cultural divide**

   2PL versus versioned storage

3. **Using high level semantic information to solve low-level problems**

   Bottom-up index deletion

4. **Seeing the wisdom in 2PL**

   “Transaction rollback” without UNDO

Bottom-up index deletion in B-Trees

Applies “logical database” information during opportunistic cleanup

- Feature added to Postgres 14
- Greatly helps workloads with **many non-HOT updates** — at least for any “logically unchanged” indexes (often the majority of indexes on the table)
- Targets “version duplicates” from “logically unchanged” indexes, through incremental passes

  - Triggers when a non-HOT updated is about to split the page

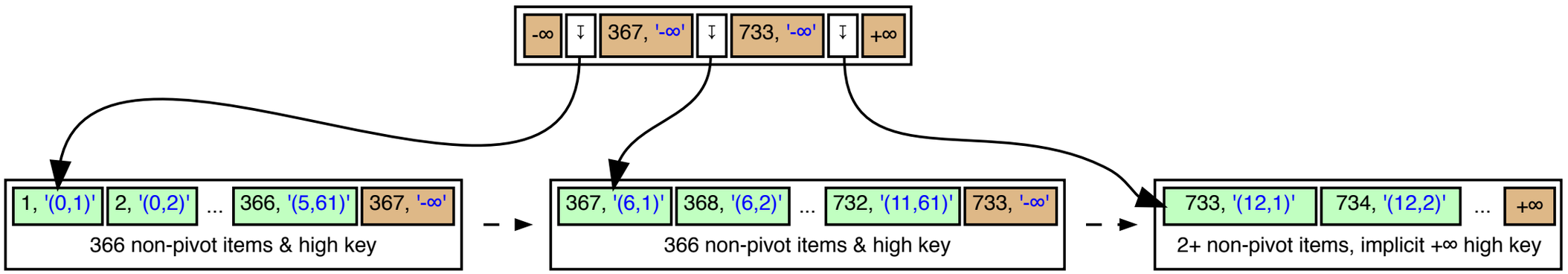
  - Makes non-HOT updaters “clean up their own mess” proactively

  - Often manages to prevent **all** “version churn page splits”

https://speakerdeck.com/peterg/logical-database

NEON

Primary key (B-Tree index) on an identity column



https://speakerdeck.com/peterg/logical-database

NEON

Thinking about time and space

- The cost of failing to delete *any* index tuples is kept low (this is essential)

  - Table AM/heapam side **gives up** when any one heap page yields no deletable index tuples

  - May need to “learn what works” for a given workload through **trial and error**

- Most pages have plenty of “extra space” due to generic B-Tree space overhead — 30%+ free space on each page is common

- Buys us more time by freeing space — but the **relationship** between **time and space** is *surprisingly* loose

  - Deleting only a few index tuples often buys us a considerable amount of time (until VACUUM reaches the same leaf page, say)

  - This is even true for most **individual** leaf pages in extreme cases

https://speakerdeck.com/peterg/logical-database

NEON

Keeping the physical database “in sync” with the logical database

- The logical database is (at best) an **imaginary ideal** for Postgres B-Tree indexes
  - Unlike a 2PL system, where physical pages “directly embody the logical database”
  - In Postgres, the state of any individual leaf page can “diverge from that ideal” to some degree without it really being a problem
  - Bottom-up deletion **limits** the **accumulation** of versions at the level of each **logical row** (or “logical page”, perhaps)
    - “How many dead tuples will my query have to access per row” might be very different to “what’s the ratio of dead to live tuples in the index as a whole”. **Concentration** matters.
- Many complicated details at every layer make it easy to **miss simple things**
  - Postgres keeps old versions in indexes to support concurrency control (avoids need for next key locking) — which is **intrinsically** related to application semantics
  - The B-Tree code and the heapam code cooperate, so we can apply information from both when driving the process

https://speakerdeck.com/peterg/logical-database

NEON

Overview

1. **Physical data independence**

   Defining the logical and physical database

2. **A cultural divide**

   2PL versus versioned storage

3. **Using high level semantic information to solve low-level problems**

   Bottom-up index deletion

4. **Seeing the wisdom in 2PL**

   “Transaction rollback” without UNDO

Avoiding the false dichotomy

The Postgres approach to versioned storage has real merit, and on balance has worked out well

- But lots of smart people have worked on 2PL systems, and we surely have more to learn from them
- Solutions that **span multiple levels of abstraction** seem necessary with 2PL — some of these may have real merit, even within Postgres

  - An idealized logical version of physical storage is a good **mental model** in some cases

- Where can this be taken next?

https://speakerdeck.com/peterg/logical-database

NEON

Idea (1/3): free space management

- Free space management in a 2PL + heap table based system practically demands “ownership” of space by transactions

  - As we’ve seen, this is complicated and messy with 2PL

  - But organizing heap tuples into physical pages along similar principles still makes sense — we still ought to **capture** naturally occurring **locality** *implicitly*

- Having some notion of per transaction/connection **heap page affinity** seems valuable

  - In many cases transactions that are inserted together will later be read, deleted, or frozen together too

  - Chaotic free space management has been tied to problems with the industry standard TPC-C benchmark, when run by Postgres

https://speakerdeck.com/peterg/logical-database

NEON

Idea (2/3): transaction rollback

- Obviously 2PL style transaction rollback implemented by occurs *when the transaction aborts*

  - *"*Second phase" of 2PL releases locks, which cannot happen until abort completes — so abort/undo processing is **tied to concurrency control**

  - Can't just hand the work off to something akin to an autovacuum daemon, as an optimization. Risk is that the application will rerun the transaction again and again, and **deadlock** with the asynchronous daemon process — again and again.

- But there may be more subtle reasons why this "optimization" might well harm performance

  - When a transaction **aborts** and is **retried**, it is likely that the **same free space** it releases during abort will get **reused** for the **same purpose**

  - It's almost certain that all of the pages that undo modifies are **still dirty**

  - Furthermore, the contents of pages will **start out** in a pristine state, when rows are first inserted on pages. We avoid **mixing** a group of related logical rows with some **random** and **unrelated** logical row later on.

Idea (3/3): XID freezing issues

- Any transaction can use any heap page for new tuples currently — which is *excessively* flexible

  - Organizing principle of free space management should have some sense of logical/transaction time — avoid mixing old with the new, make pages "settle" naturally through **hysteresis**.

  - Might make sense to have **strict invariants** about what XIDs are permissible in individual pages ranges

- Conjecture: might be easier to address problems in this area by **expanding scope** to include smgr and free space management code

  - In other words, take a few limited steps in the direction of adding "transactional storage"

  - "Logical freezing" seems much harder without invariants that tie transactions to physical heap pages

https://speakerdeck.com/peterg/logical-database

NEON

Conclusions

- Many design decisions in this area are interdependent

  - Makes discussion and comparison very confusing

- The logical/physical database is useful as a broad conceptual framework — albeit an imperfect one.

  - The precise definition is likely to vary over time

- Particularly helpful as a way of understanding 2PL designs, where concurrency control is baked in to access methods and storage, by necessity

https://speakerdeck.com/peterg/logical-database