credativ

PostgreSQL 18

New Features & Improvements

Josef Machytka <josef.machytka@credativ.de>

2025-11-13 - PostgreSQL November Meetup Berlin

credativ GmbH

- Founded 1999 in Jülich, Germany
- Close ties to Open-Source Community
- More than 40 Open-Source experts
- Consulting, development, training, support (3rd-level / 24x7)
- Open-Source infrastructure with Linux, Kubernetes and Proxmox
- Open-Source databases with PostgreSQL
- DevSecOps with Ansible, Puppet, Terraform and others
- Since 2025 independent owner-managed company again

Josef Machytka

credativ

- Professional Service Consultant - PostgreSQL specialist at credativ GmbH
- 33+ years of experience with different databases
- PostgreSQL (13y), BigQuery (7y), Oracle (15y), MySQL (12y), Elasticsearch (5y), MS SQL (5y)
- 10+ years of experience with Data Ingestion pipelines, Data Analysis, Data Lake and Data Warehouse
- 3+ years of practical experience with different LLMs / AI including their architecture and principles
- From Czechia, living now 12 years in Berlin
- Often presenting at PostgreSQL Conferences and MeetUps in Europe

  - LinkedIn: linkedin.com/in/josef-machytka
  - ResearchGate: researchgate.net/profile/Josef-Machytka
  - Academia.edu: academia.edu/JosefMachytka
  - Medium: medium.com/@josef.machytka
  - sessionize: sessionize.com/josefmachytka

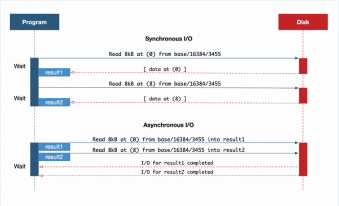
This Special Time of the Year

- We in PostgreSQL community have Christmas twice a year
- First when new major version is released in September/October
- Bringing new features, improvements, bug fixes
- And causing and avalanche of blog posts, articles, talks
- I summarized most important changes for you
- And I linked many useful resources for further reading

Image credit: ChatGPT

Asynchronous I/O

- New AIO subsystem allows multiple concurrent file reads
- Current implementation is only minimal
- Does not support writes - OLTP operations do not benefit
- Multi process model made implementation challenging
- PG17 introduced ”streaming I/O” - multiple pages in one request
- But still synchronous, one-at-a-time per backend process
- PG18 Async I/O significantly improves performance of:
  - Large sequential scans, GROUP BY, COPY (up to 2-4x faster)
  - Bitmap heap scans
  - Vacuum (some benchmarks up to 3-4x faster, some 0)
  - Read operations on cloud DBs with network-attached storage

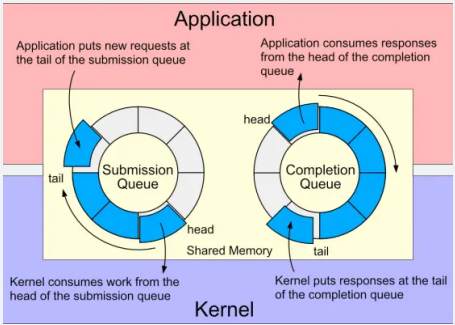


Images from the article
Accelerating Disk Reads with Asynchronous I/O

Async I/O: Configuration Options

- **io_method**:
  - worker - default, pool of I/O worker processes
  - io_uring - Linux-specific async I/O queues - needs Linux 5.1+
  - - requires a build with –with-liburing / -Dliburing
  - - some container runtimes disable it for security
  - sync - "fallback" synchronous I/O, but still uses AIO API

- **io_workers**:
  - 3 - default, too low for larger systems
  - Start with 1/4 of total CPU threads
  - Some benchmarks suggest up to 1/2 of threads



Images from the article
Why you should use io_uring

Async I/O: Other Configuration Options

credativ

- **io_combine_limit**:
  - 16 - default (i.e. 16 pages = 128kb) - dynamic
  - Without unit - number of pages, with kB/MB - size in kB/MB
  - How many pages can be combined in 1 request
  - Optimal value depends on underlying HW / OS capabilities
  - Hardware and protocols have segment and size limits
  - After some threshold, increasing value has no effect
  - Limited internally in PG by setting io_max_combine_limit

- **io_max_combine_limit**:
  - 16 - default (i.e. 16 pages = 128kb) - requires restart
  - Limits io_combine_limit - maximum pages per request
  - Cluster wide limit for all backends
  - Typical max: Unix 128 (1 MB), Windows 16 (128 kB)
  - Added to manage memory consumption & structure sizes

Async I/O: Other Configuration Options

credativ

- **io_max_concurrency**:
  - maximum number of concurrent I/O operations per process (capped to 64)
  - default -1 = selects number based on shared_buffers and max number of backends
  - Requires restart to change

- **effective_io_concurrency**:
  - number of concurrent I/O operations that can be executed simultaneously
  - new default 16, range 1-1000, value 0 disables async requests
  - Higher values for faster storage - 200 for NVMe SSDs
  - Very high values may increase I/O latency for all queries
  - Can be changed dynamically

Async I/O: Monitoring

- io_method = worker - uses dedicated background I/O worker processes
- Monitored via new pg_aios view:
  - Shows current I/O requests - status, result etc.
  - Status: DEFINED, STAGED, SUBMITTED, COMPLETED_IO, ...
  - Result: UNKNOWN, OK, PARTIAL, WARNING, ERROR
- Enhanced pg_stat_io view - new columns: read_bytes, write_bytes, extend_bytes
- Parameter track_io_timing
  - Enables additional I/O timing statistics
  - Default is off - if on, adds some overhead
- On Linux level using iotop command

```
-- output from iotop command:

   TID PRIO USER      DISK READ   DISK WRITE>    COMMAND
617931 be/4 postgres  32.17 M/s     0.00 B/s postgres: io worker 0
617932 be/4 postgres  72.11 M/s     0.00 B/s postgres: io worker 1
617933 be/4 postgres  41.50 M/s     0.00 B/s postgres: io worker 2
617934 be/4 postgres  58.00 M/s     0.00 B/s postgres: io worker 4
```

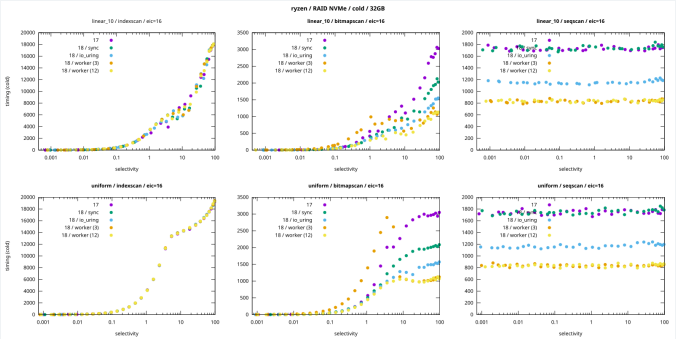
How it works under the hood

credativ

- `src/backend/storage/aio/README.md`
- `src/backend/storage/aio/method_worker.c`
- Postmaster creates on startup a pool of background worker processes
- Each worker is a separate OS process
- Workers still perform synchronous IO, they just read in parallel
- Request for I/O sent to shared memory submission queue
- Worker wakes up, reads data, puts it into shared buffers, notifies backend
- Each woken IO worker can woke 2 more - note about future improvement to N
- `src/include/storage/aio.h`
- Main file defining AIO interface
- Details shown in Doxygen PostgreSQL source code - aio.h File Reference

Async I/O Benchmarks

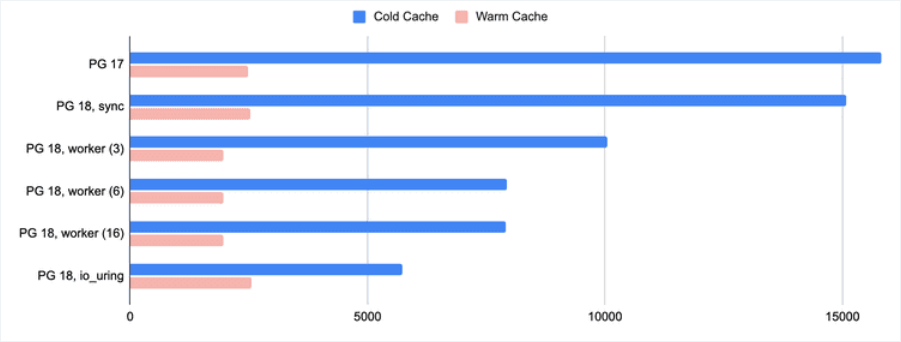
- Tomas Vondra: Tuning AIO in PostgreSQL 18:
- Benchmarks Ryzen 9900X - 12 cores/ 24 threads, 32 GB RAM, 4 NVMe SSD RAID0
- Tests for PG 17, PG 18 with sync, PG 18 with io_uring, PG 18 with AIO worker (3 and 12)
- (uniform = entirely random distribution, linear_10 = sequential with 10% randomness)
- index scan - no effect, index scans do not use AIO yet, only sync I/O
- bitmap scan - 3 workers slower than sync for low selectivity queries, 12 workers 2x faster
- sequential scan - workers 2x faster than sync, io_uring in between



Async I/O Benchmarks

credativ

- Waiting for Postgres 18: Accelerating Disk Reads with Asynchronous I/O:
- AWS c7i.8xlarge instance 32 vCPUs, 64 GB RAM, 100GB io2 EBS volume 20,000 IOPS



Async I/O: Resources

credativ

- Asynchronous & Direct IO (PostgreSQL Source Code git master)
- Tomas Vondra: Tuning AIO in PostgreSQL 18
- Waiting for Postgres 18: Accelerating Disk Reads with Asynchronous I/O
- Boosting Typical Query Patterns - PostgreSQL 18's Performance Enhancements (PGConf.EU 2025)
- Get Excited About Postgres 18
- What went wrong with AIO (PGconf.DEV 2025)
- PostgreSQL 18 Asynchronous I/O (Neon blog)
- PostgreSQL 18: A Comprehensive Guide to New Features for DBAs and Developers

Database Checksums enabled by default

- initdb enables data checksums by default
- Can be disabled with –no-data-checksums
- Slight performance overhead for writes (up to 2%)
- Checksums calculated on writes, stored in page header
- Recalculates checksum on reads, compares with stored value
- Detects silent data corruption immediately
- Important parameters:
  - data_checksums (on/off) - read-only, show if checksums are enabled
  - ignore_checksum_failure (on/off) - ignore checksum failure on read
  - zero_damaged_pages (on/off) - zero out pages with checksum failure

- PostgreSQL 18 enables data checksums by default (credativ blog)
- Database in Distress: Testing and Repairing Different Types of Database Corruption (PGConf.EU 2025)

Detection of Corruption with Checksums

- On read, PostgreSQL recalculates and verifies the checksum for each block
- Mismatched checksums indicate corruption and trigger an error
- Errors can be skipped setting ”ignore_checksum_failure=ON”
- Corrupted pages remain unchanged; only error handling behavior is modified

```
-- ignore_checksum_failure="off";  --- Query stops on the first checksum error
test=# select * from pg_toast.pg_toast_17453;
WARNING: page verification failed, calculated checksum 19601 but expected 152
ERROR: invalid page in block 0 of relation base/16384/16402


-- ignore_checksum_failure="on";  --- Query skips checksum errors and continues
test=# select * from pg_toast.pg_toast_17453;
WARNING:  page verification failed, calculated checksum 29668 but expected 57724
WARNING:  page verification failed, calculated checksum 63113 but expected 3172
WARNING:  page verification failed, calculated checksum 59128 but expected 3155
```

Detection of Corruption with Checksums

credativ

- Damaged blocks can be ”zeroed out” by setting ”zero_damaged_pages=ON”
- Corrupted pages are replaced in memory with zeroed pages during query execution
- Zeroed pages are automatically written to disk, permanently overwriting data
- Setting ”zero_damaged_pages=OFF” does not restore lost data or undo changes
- Only scanned blocks are zeroed; unscanned corrupted pages may still cause errors

```
-- zero_damaged_pages=OFF --> Query stops on the first checksum error
test=# select * from test_table_bytea;
WARNING: page verification failed, calculated checksum 19601 but expected 152
ERROR: invalid page in block 0 of relation base/16384/16402


-- zero_damaged_pages=ON --> Query skips damaged blocks and continues
test=# select * from pg_toast.pg_toast_17453;
WARNING: page verification failed, calculated checksum 29668 but expected 57724
WARNING: invalid page in block 204 of relation base/16384/17464; zeroing out page
WARNING: page verification failed, calculated checksum 63113 but expected 3172
WARNING: invalid page in block 222 of relation base/16384/17464; zeroing out page
WARNING: page verification failed, calculated checksum 59128 but expected 3155
```

What about zero_damaged_pages and Autovacuum?

credativ

- Autovacuum removes already zeroed pages
- What if we set globally zero_damaged_pages=ON
- And autovacuum finds corrupted pages?
- Answer is in `src/backend/postmaster/autovacuum.c`

```
/*
 * Force zero_damaged_pages OFF in the autovac process, even if it is set
 * in postgresql.conf.  We don't really want such a dangerous option being
 * applied non-interactively.
 */
SetConfigOption("zero_damaged_pages", "false", PGC_SUSET, PGC_S_OVERRIDE);
```

Upgrade strategy for pg_upgrade

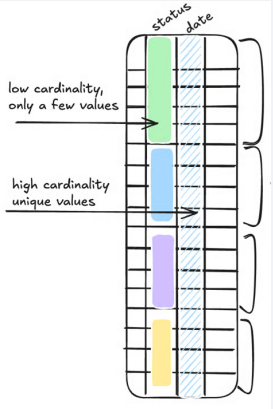
credativ

- pg_upgrade requires both old and new clusters with same checksum setting
- Either both have checksums enabled, or both disabled
- Otherwise, pg_upgrade will fail with a checksum mismatch error
- Adding checksums ”on the fly” during upgrade is not supported yet
- For upgrading a pre-18 cluster without checksums, we have two options
  - 1. Disable checksums on the new cluster running initdb with –no-data-checksums
  - (After upgrade, checksums can be enabled on the new cluster using pg_checksums)
  - 2. Enable checksums on the old cluster before upgrade using pg_checksums
- Dry run check with pg_upgrade –check will verify checksum compatibility

B-tree skip scans

- Multi-column indexes for queries filtering on non-leading columns
- Useful for low-cardinality leading columns
- Reduces need for additional indexes, speeds up queries
- Slower than regular index scan, but faster than sequential scan
- In older versions:
  - Only leading column could be used
  - Additional indexes often needed
  - Without skip scan query performed sequential scan

- PostgreSQL 18 B-tree Skip Scan for Better Queries
- PostgreSQL documentation: Multicolumn Indexes

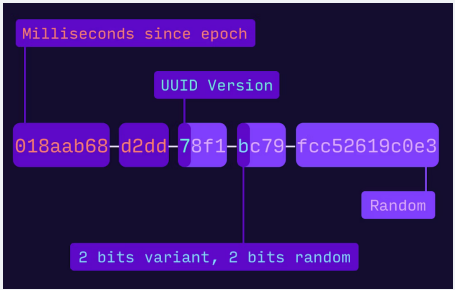


Images from the article
Get Excited About Postgres 18

Native UUIDv7 support

- UUID is generally 16 bytes (128 bits) integer
- Just presented as hexadecimal string
- Function uuidv7() generates timestamp-ordered UUIDs
- Include 12-bit millisecond precision timestamp
- Replacement for random UUIDs and BIGINT IDs
- Improves index locality, reduces B-tree page splits
- But it also creates some security leakage
- Exposes timestamp of creating record
- Exposes time difference between records

- Fun With UUIDs (PGDay Lowlands 2025)

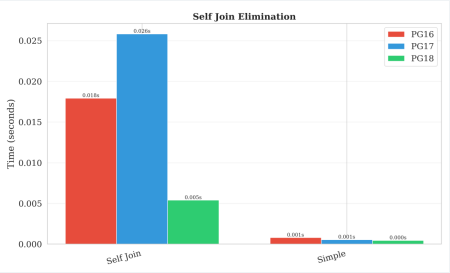


Images from the article
Goodbye integers. Hello UUIDv7!

Query Planner Optimizations

- Redundant Self-joins automatically removed
- Detects joins where both sides are the same table
- SELECT * FROM t1 WHERE x IN (SELECT t3.x FROM t1 t3);
- SELECT * FROM t1,t2, t1 t3 WHERE t1.x = t2.x AND t2.x = t3.x;
- Sometimes generated by ORMs libraries/query builders
- Feature was discussed and developed since 2018

- PostgreSQL Lands Self-Join Elimination Optimization



Graph from slides

Robert Mello: Boosting Typical Query Patterns: PostgreSQL 18's Performance Enhancements (PGConf.EU 2025)

Query Planner Optimizations

credativ

- Multiple OR conditions on the same column
  - Converted to ANY an array:
  - SELECT * FROM t1 WHERE x = 1 OR x = 2 OR x = 3;
  - becomes: SELECT * FROM t1 WHERE x = ANY(ARRAY[1,2,3]);
  - Uses bitmap index scans more efficiently
  - Only reads index leaf pages with matching values -> reports up to 100x faster

- Col IN (values...) converted to ANY(array)
  - Similar optimization for IN lists:
  - SELECT * FROM t1 WHERE x IN (1,2,3);
  - becomes: SELECT * FROM t1 WHERE x = ANY(ARRAY[1,2,3]);

Query Planner Optimizations

- Hash joins & GROUP BY use less memory
- Elimination of redundant columns in GROUP BY
- SELECT DISTINCT reorders columns for better index use
- Improved planning for partitioned tables
- Speed Up of INTERSECT/EXCEPT, window functions

Image credit: ChatGPT

Improvements in Generated Columns

credativ

- STORED generated column can now be included into logical replication
- Added support for VIRTUAL generated columns
- Columns computed on-the-fly from other columns
- VIRTUAL generated columns are new default
- STORED must be specified explicitly
- Virtual columns:
  - Reduce storage needs, always return up-to-date values
  - Improve INSERT/UPDATE performance
  - Cannot be indexed or constrained, complex expressions may be slower

```
CREATE TABLE orders (
    id SERIAL PRIMARY KEY,
    subtotal DECIMAL(10,2),
    tax_rate DECIMAL(5,4) DEFAULT 0.0875,
    total DECIMAL(10,2) GENERATED ALWAYS AS (subtotal * (1 + tax_rate)) );
```

Enhanced RETURNING clause

credativ

- INSERT/UPDATE/DELETE/MERGE can access OLD and NEW values
- RETURNING old.*, new.* syntax
- Simplifies auditing, logging, change tracking
- PostgreSQL 18 Enhanced Returning

```
UPDATE employees SET salary = $1 WHERE employee_id = $2
RETURNING old.salary AS previous_salary,
          new.salary AS updated_salary;


INSERT INTO inventory (product_name, quantity) VALUES ('Laptop Computer', 25)
ON CONFLICT (product_name)
DO UPDATE SET quantity = inventory.quantity + EXCLUDED.quantity,
              last_updated = CURRENT_TIMESTAMP
RETURNING old.quantity AS original_quantity,
        new.quantity AS updated_quantity;
```

Native OAuth 2.0 authentication

credativ

- OAuth = Open Authorization protocol
- Used for token-based authentication on internet without sharing passwords
- OAuth support must be enabled at build time
- Supports different types of authentication:
  - Single Sign-On (SSO) for database access
  - Centralized identity management using OAuth providers like Google
  - Integration with enterprise identity systems (Okta)
- Configured via pg_hba.conf using method ”oauth”

- PostgreSQL 18 Meets OAuth2: how Native Support Works with Keycloak (credativ blog)
- PostgreSQL 18 OAuth Support

Security and Authentication Improvements

credativ

- **MD5 deprecated & SCRAM improvements**
  - MD5 authentication officially deprecated
  - MD5 considered unsuitable as a cryptographic hash algorithm
  - MD5 password hashes are vulnerable to pass-the-hash attacks
  - I.e. attacker stealing hash can authenticate without knowing password

- **SCRAM-SHA-256** is the recommended replacement
  - SCRAM = Salted Challenge Response Authentication Mechanism
  - Protects against password interception and replay attacks
  - Uses stronger hashing (SHA-256) and salting

- **SCRAM pass-through for postgres_fdw and dblink**
  - No plaintext passwords needed anymore in FDW connections
  - Simplifies configuration and enhances security

Performance Improvements

credativ

- **Parallel creation of GIN indexes**
  - Significantly speeds up creation of large GIN indexes
  - Uses multiple workers to build index in parallel
  - max_parallel_maintenance_workers controls workers
  - Benchmarks suggest value up to 1/2 CPU cores is best
  - In PG17 creation of GIN index could take hours on large tables
  - Important for JSONB and full-text search GIN indexes

- **Enhanced EXPLAIN output**
  - Buffer usage shown by default, includes CPU time, avg read time
  - WAL writes and index lookup counts
  - Per connection I/O stats

Image credit: ChatGPT

Improved pg_upgrade

credativ

- Planner statistics migrated automatically
- ANALYZE after upgrade no longer needed

- New `--swap` method to swap data directories
- I.e. moves directories from old cluster to new
- Instead of copying, cloning or linking files
- It replaces system catalogs with new cluster files
- Destroys old cluster, so backup is needed

- Parameter `--jobs` for parallel checks
- Speeds up upgrade process significantly

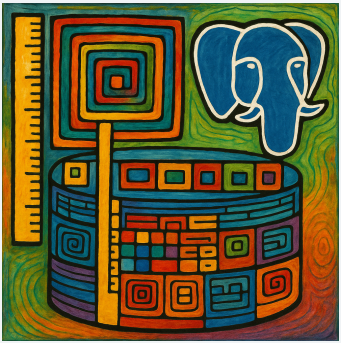


Image credit: ChatGPT

Backup Improvements

- **pg_dumpall improvements**
  - Can produce custom and directory format dumps
  - Speeds up full-cluster dumps significantly
- **fine-grained pg_dump options**
  - `--with-data`, `--with-schema`
  - `--with-statistics`
  - `--no-data`, `--no-schema`
  - `--no-statistics`, `--statistics-only`
  - `--sequence-data`

```
pg_dump --statistics-only mydb > stats.sql

SELECT * FROM pg_catalog.pg_restore_relation_stats(
    'version', '180000'::integer,
    'schemaname', 'public',
    'relname', 'async_io_test',
    'relpages', '80777'::integer,
    'reltuples', '1.004389e+06'::real,
    'relallvisible', '80777'::integer,
    'relallfrozen', '3768'::integer );
```

Example from slides
Robert Mello: Boosting Typical Query Patterns: PostgreSQL 18's Performance Enhancements (PGConf.EU 2025)

Constraints Improvements

credativ

- **NOT VALID constraints**
- Constraints can be added as NOT VALID
- Existing data not checked initially
- Validated later using VALIDATE CONSTRAINT
- adding NOT NULL without downtime impact

```
-- Add a NOT NULL constraint without full table scan:
ALTER TABLE orders
    ADD CONSTRAINT orders_customer_not_null
    NOT NULL (customer_id) NOT VALID;

-- Later, validate the constraint when convenient:
ALTER TABLE orders
    VALIDATE CONSTRAINT orders_customer_not_null;
```

Temporal Constraints

credativ

- **Temporal (range-aware) constraints**
- WITHOUT OVERLAPS for temporal data integrity
- Only the last column in PK/UNIQUE can be marked WITHOUT OVERLAPS
- PostgreSQL has several range types:
  - daterange - range of dates
  - tsrange - range of timestamps without time zone
  - tstzrange - range of timestamps with time zone
  - int4range - range of 4-byte integers
  - numrange - range of numeric values
- Internally it behaves like exclude constraint on range types
- Feature requires btree_gist extension - new index is GiST

- PostgreSQL 18 Temporal Constraints

Creating Temporal Data

```
-- Table with a temporal primary key (id + valid period must be unique and non-
overlapping per id):
CREATE TABLE subscriptions (
    sub_id INT,
    valid_daterange TSTZRANGE,
    PRIMARY KEY (sub_id, valid_daterange WITHOUT OVERLAPS));


-- Insert some data - '[)' means inclusive lower bound, exclusive upper bound
INSERT INTO subscriptions (sub_id, valid_daterange) VALUES
    (1, tstzrange('2023-01-01', '2023-06-30', '[)')),
    (1, tstzrange('2023-07-01', 'infinity', '[)'));


-- This insert would violate the temporal constraint and fail:
INSERT INTO subscriptions (sub_id, valid_daterange) VALUES
    (1, tstzrange('2023-06-15', '2023-12-31', '[)'));


-- Example SELECT query - with containment operator @> - uses GiST index:
SELECT * FROM subscriptions WHERE sub_id = 1 AND valid_daterange @> '2023-05-01'::timestamptz;
```

Temporal Constraints

- Enforces non-overlapping periods per id
- Useful for subscriptions, bookings, contracts, etc.
- Can be referenced by foreign keys with PERIOD syntax
- Ensures child record periods fit within parent validity

```
-- Another table referencing the temporal key:
CREATE TABLE usage_logs (
    sub_id INT,
    use_period TSRANGE,
    FOREIGN KEY (sub_id, PERIOD use_period)
    REFERENCES subscriptions(sub_id, PERIOD valid_daterange));
```

Unique index with NULLS DISTINCT

- Allows unique indexes to treat NULLs as distinct or not distinct
- By default, NULLs are treated as distinct (multiple NULLs allowed)
- Improves data integrity for columns that can contain NULLs

```
-- Before: multiple NULLs were allowed in a UNIQUE column
CREATE TABLE users (
    email TEXT UNIQUE -- (could have multiple NULL emails)
);


-- Now: disallow multiple NULLs by marking NULLs "not distinct"
CREATE TABLE accounts (
    username TEXT UNIQUE,
    email TEXT UNIQUE NULLS NOT DISTINCT -- only one NULL allowed
);


-- Create a unique index with NULLS NOT DISTINCT
CREATE UNIQUE INDEX users_email_idx ON users (email)
    NULLS NOT DISTINCT;
```

Thank you for your attention!

All my slides