



What you should know about constraints in PostgreSQL (and what's new in 18)

Gülçin Yıldırım Jelínek

PGConf.EU, 22 October 2025

Agenda

-
- 1 **Insights from pg_constraint**
 - 2 **Temporal keys: WITHOUT OVERLAPS, PERIOD**
 - 3 **NOT NULL as a first-class constraint**
 - 4 **NOT ENFORCED**
 - 5 **Partitioned tables improved**
-

Select * from me;

Current:

- Product Manager at Xata
- Postgres Contributor
- Co-founder of Prague PostgreSQL Meetup
- Co-founder & General Coordinator of Kadin Yazilimci (Women Devs Turkey)
- Co-founder & Chair of Diva: Dive into AI Conference

Past:

- Board Member at Postgres Europe
- Staff Engineer at EDB, 2ndQuadrant



What is a constraint?

Constraints are rules enforced by the database to ensure data integrity, they keep your data valid and consistent.

pg_constraint

The catalog `pg_constraint` stores check, not-null, primary key, unique, foreign key, and exclusion constraints on tables. (Column constraints are not treated specially. Every column constraint is equivalent to some table constraint.)

User-defined constraint triggers (created with `CREATE CONSTRAINT TRIGGER`) also give rise to an entry in this table.

Check constraints on domains are stored here, too.

Rows in the pg_constraint catalog

contype	
c	CHECK
f	FOREIGN KEY
n	NOT NULL
p	PRIMARY KEY
u	UNIQUE
x	EXCLUDE
t	constraint trigger

pg_constraint

The catalog pg_constraint stores check, not-null, primary key, unique, foreign key, and exclusion constraints on tables. **(Column constraints are not treated specially. Every column constraint is equivalent to some table constraint.)**

User-defined constraint triggers (created with CREATE CONSTRAINT TRIGGER) also give rise to an entry in this table.

Check constraints on domains are stored here, too.

Column constraints vs table constraints

```
CREATE TABLE products_oct (  
    price numeric CHECK (price > 0)  
);
```

→ Column constraint

```
CREATE TABLE products_nov (  
    price numeric,  
    CHECK (price > 0)  
);
```

→ Table constraint

Column constraints vs table constraints

```
SELECT
    rel.relname AS table_name,
    c.conname,
    c.contype,
    c.conrelid::regclass AS table_ref,
    c.conkey,
    pg_get_constraintdef(c.oid, true) AS constraint_def
FROM pg_constraint c
JOIN pg_class rel ON rel.oid = c.conrelid
WHERE rel.relname IN ('products_oct', 'products_nov');
```

Column constraints vs table constraints

```
-[ RECORD 1 ]--+-+-----  
table_name    | products_nov  
conname       | products_nov_price_check  
contype       | c  
table_ref     | products_nov  
conkey        | {1}  
constraint_def | CHECK (price > 0::numeric)  
-[ RECORD 2 ]--+-+-----  
table_name    | products_oct  
conname       | products_oct_price_check  
contype       | c  
table_ref     | products_oct  
conkey        | {1}  
constraint_def | CHECK (price > 0::numeric)
```

pg_constraint

The catalog `pg_constraint` stores check, not-null, primary key, unique, foreign key, and exclusion constraints on tables. (Column constraints are not treated specially. Every column constraint is equivalent to some table constraint.)

User-defined constraint triggers (created with `CREATE CONSTRAINT TRIGGER`) also give rise to an entry in this table.

Check constraints on domains are stored here, too.

Constraint trigger

CREATE CONSTRAINT TRIGGER

- Can be DEFERRABLE and controllable by SET CONSTRAINTS
- Must be an AFTER ROW trigger
 - Cannot be BEFORE or INSTEAD OF
- WHEN conditions are evaluated immediately
 - Not deferred even if the trigger is
- Always row-level (FOR EACH ROW)
- Apply only to plain tables
 - Not to foreign tables
- Stored in pg_constraint with contype = 't'

Constraint trigger

User-defined triggers that behave like constraints (*sort of*)

- MVCC behaviour differs between built-in constraints vs triggers
- Laurenz Albe: “[Deferrable trigger](#) would be a better description.”

pg_constraint

The catalog `pg_constraint` stores check, not-null, primary key, unique, foreign key, and exclusion constraints on tables. (Column constraints are not treated specially. Every column constraint is equivalent to some table constraint.)

User-defined constraint triggers (created with `CREATE CONSTRAINT TRIGGER`) also give rise to an entry in this table.

Check constraints on domains are stored here, too.

Domain

```
CREATE DOMAIN email_address AS text  
    CHECK (VALUE ~* '^[^@]+@[^@]+\.[^@]+$');
```

```
CREATE TABLE users (  
    id serial PRIMARY KEY,  
    email email_address NOT NULL  
);
```

```
INSERT INTO users(email) VALUES ('not-an-email');
```



```
INSERT INTO users(email) VALUES ('ok@example.com');
```



Domain

```
SELECT c.conname,  
       pg_get_constraintdef(c.oid, true) AS definition,  
       t.typname AS domain_name  
FROM pg_constraint c  
JOIN pg_type t ON t.oid = c.contypid  
WHERE c.contype = 'c'  
      AND c.contypid <> 0;
```









conname	definition	domain_name
email_address_check	CHECK (VALUE ~* '^[^@]+@[^@]+\.[^@]+\$')	email_address

PostgreSQL 18

What's new?

- Allow the specification of **non-overlapping PRIMARY KEY, UNIQUE, and foreign key constraints**. This is specified by **WITHOUT OVERLAPS** for PRIMARY KEY and UNIQUE, and by **PERIOD** for foreign keys, all applied to the last specified column.
- Allow **CHECK and foreign key constraints** to be specified as **NOT ENFORCED**. This also adds column **pg_constraint.conenforced**.
- Require primary/foreign key relationships to use either **deterministic collations** or the **the same nondeterministic collations**. The restore of a pg_dump, also used by pg_upgrade, will fail if these requirements are not met; schema changes must be made for these upgrade methods to succeed.
- Store column **NOT NULL** specifications in **pg_constraint**. This allows names to be specified for NOT NULL constraint. This also adds NOT NULL constraints to foreign tables and NOT NULL inheritance control to local tables.
- Allow ALTER TABLE to set the **NOT VALID** attribute of **NOT NULL** constraints
- Allow modification of the inheritability of **NOT NULL** constraints. The syntax is ALTER TABLE ... ALTER CONSTRAINT ... **[NO] INHERIT**.
- Allow **NOT VALID foreign key** constraints on partitioned tables.
- Allow **dropping of constraints ONLY** on **partitioned tables**. This was previously erroneously prohibited.

Temporal keys, temporal database?

A time period data type, including the ability to represent time periods with no end (infinity or forever)	
The ability to define valid and transaction time period attributes and bitemporal relations	
System-maintained transaction time	
Temporal primary keys, including non-overlapping period constraints	
Temporal constraints, including non-overlapping uniqueness and referential integrity	
Update and deletion of temporal records with automatic splitting and coalescing of time periods	
Temporal queries at current time, time points in the past or future, or over durations	
Predicates for querying time periods, often based on Allen's interval relations	

“ Queries to a temporal database return facts that were, are, or will be actual at a time that may differ from current time.”

Boris Novikov

Querying Temporal Data

Add temporal PK and UNIQUE constraints

- Add WITHOUT OVERLAPS clause to PRIMARY KEY and UNIQUE constraints
- PRIMARY KEY (id, valid_at WITHOUT OVERLAPS)
- CREATE EXTENSION IF NOT EXISTS btree_gist;

```
postgres=# CREATE TABLE rooms (  
    room_id    int PRIMARY KEY,  
    name       text NOT NULL  
);
```

```
CREATE TABLE bookings (  
    room_id    int          NOT NULL REFERENCES rooms(room_id),  
    during     tstzrange    NOT NULL,  
    -- Temporal PK: last column uses WITHOUT OVERLAPS  
    PRIMARY KEY (room_id, during WITHOUT OVERLAPS)  
);  
CREATE TABLE  
CREATE TABLE
```

Add temporal PK and UNIQUE constraints

```
postgres=# INSERT INTO rooms VALUES (101, 'Blue'), (102, 'Green');
INSERT 0 2
postgres=# INSERT INTO bookings VALUES
    (101, tstzrange('2025-09-20 10:00+02', '2025-09-20 11:00+02', '[]')),
    (101, tstzrange('2025-09-20 11:00+02', '2025-09-20 12:00+02', '[]'));
INSERT 0 2
```

```
postgres=# INSERT INTO bookings VALUES (101, tstzrange('2025-09-20
10:30+02', '2025-09-20 10:45+02', '[]'));
ERROR:  conflicting key value violates exclusion constraint
"bookings_pkey"
DETAIL:  Key (room_id, during)=(101, ["2025-09-20
08:30:00+00", "2025-09-20 08:45:00+00"]) conflicts with existing key
(room_id, during)=(101, ["2025-09-20 08:00:00+00", "2025-09-20
09:00:00+00"]).
```

Add temporal PK and UNIQUE constraints

```
postgres=# INSERT INTO bookings VALUES  
  (102, tstzrange('2025-09-20 10:30+02', '2025-09-20 11:30+02', '[]'));  
INSERT 0 1
```

```
postgres=# INSERT INTO bookings VALUES  
  (101, tstzrange('2025-09-20 12:00+02', '2025-09-20 12:00+02', '[]'));  
ERROR:  empty WITHOUT OVERLAPS value found in column "during" in  
relation "bookings"
```


Summary: Temporal PK and UNIQUE constraints

- Add WITHOUT OVERLAPS clause to PRIMARY KEY and UNIQUE constraints
- Backed by GIST indexes instead of B-tree indexes
- Essentially exclusion constraints with
 - = for the scalar parts of the key
 - && for the temporal part

```
CONSTRAINT bookings_no_overlap
    EXCLUDE USING gist (
        room_id WITH =,      -- same business key
        during WITH &&       -- ranges overlap
    )
```

- Forbid empties
- Only support ranges and multiranges for PK/UQs

Add temporal FK constraints

- Add PERIOD clause to FK definitions

```
CONSTRAINT bookings_fk_availability  
  FOREIGN KEY (room_id, PERIOD during)  
  REFERENCES room_availability (room_id, PERIOD available)
```

- Supported for range and multirange types
- Temporal FKs check for range containment instead of equality
- Matches the behavior of SQL standard
- ON {UPDATE,DELETE} {CASCADE,SET NULL,SET DEFAULT} are not supported yet

NOT ENFORCED in CHECK constraints

- Add support for the NOT ENFORCED/ENFORCED flag for constraints, with support for check constraints

```
ALTER TABLE users
```

```
  ADD CONSTRAINT email_format CHECK (email ~ '^[^@]+@[^@]+\.[^@]+$') NOT ENFORCED;
```

- New column: `pg_constraint.conenforced`
- CHECK constraints do not currently support ALTER operations
 - To change enforceability drop and recreate the constraint

NOT ENFORCED in FK constraints

- Expand NOT ENFORCED constraint flag to foreign key constraints
- **NOT ENFORCED**
 - Integrity checks are no longer required
 - Triggers will not be created and constraint will be marked as NOT VALID
- **ENFORCED → NOT ENFORCED**
 - Triggers will be dropped
 - The constraint will be marked as NOT VALID
- **NOT ENFORCED → ENFORCED**
 - Triggers will be created
 - Constraints will be changed to VALID

Fix collation handling for FKs

- Require primary/foreign key relationships to `use either deterministic collations or the the same nondeterministic collations`
- The restore of a `pg_dump`, also used by `pg_upgrade`, will fail if these requirements are not met; schema changes must be made for these upgrade methods to succeed.

Add pg_constraint rows for NOT NULL constraints

- `contype='n'`
- Propagated to other tables:
 - Inheritance
 - Creating/attaching partitions
 - `CREATE TABLE .. LIKE`
- `conislocal` and `coninhcount` (used for CHECK constraints before)
 - Not by constraint name but by name of the column they apply to
 - So constraints names can be different across a hierarchy
- The inheritance status can be controlled
 - If a parent has one, then all children will have it
 - They can be marked `NO INHERIT`, then children will not inherit
- They show up in `\d+`

Add pg_constraint rows for NOT NULL constraints

- This also opens the door for allowing UNIQUE+NOT NULL to be used for functional dependency determination, as envisioned by commit e49ae8d3bc58. It's likely possible to allow DEFERRABLE constraints as follow up work, as well.
- <https://git.postgresql.org/gitweb/?p=postgresql.git;a=object;h=e49ae8d3bc58>

NOT NULL constraints as NOT VALID

- Add NOT NULL without scanning the table
 - `ALTER TABLE .. ADD CONSTRAINT .. NOT NULL NOT VALID`
- Validate later to avoid ACCESS EXCLUSIVE LOCK
 - `ALTER TABLE .. VALIDATE CONSTRAINT ..` or
 - `ALTER TABLE .. ALTER COLUMN .. SET NOT NULL`
- Inheritance
 - Parent VALID → child can't stay INVALID
 - Matches ENFORCED / NOT ENFORCED constraint logic
- Catalog behavior
 - Validity stored in `pg_constraint.convalidated`
- `pg_dump/restore`
 - Dumps invalid NOT NULLs separately
 - Adds & validates them after data load (like CHECK constraints)

NOT NULL inheritance

- Allow modification of the inheritability of NOT NULL constraints
- `ALTER TABLE .. ALTER CONSTRAINT .. SET [NO] INHERIT`
- **NO INHERIT → INHERIT**
 - Meta-data only change
 - Adds (if missing) a pg_constraint entry
- **INHERIT → NO INHERIT**
 - Starting point
 - Parent: `conislocal = true, connoinherit = false`
 - Child: `conislocal = false, coninhcount = 1`
 - Postgres does not drop the child constraints
 - The parent constraint stops being their ancestor
 - Each child constraint becomes independent - effectively a local constraint
 - `conparentid = NULL, conislocal = true, coninhcount = 0`

NOT VALID FKs on partitioned tables

- Allow NOT VALID foreign key constraints on partitioned tables
- Per-partition validation
 - Validate each partition independently
 - Minimizes lock scope and duration
- Hierarchy-level validation
 - Run once on the parent table
 - Validates all unvalidated child FKs in one go
- Foundation for NOT ENFORCED constraints

DROP ONLY on partitioned tables

- Allow dropping of constraints ONLY on partitioned tables
- ALTER TABLE ONLY parent DROP CONSTRAINT some_constraint
 - Before this was not allowed
 - Now, removes the constraint from the parent only
 - Leaves each partition's constraint intact
 - Stops future partitions from inheriting that constraint
- ADD ONLY restriction remains
 - You can't add a constraint only on the partitioned parent with children present



Postgres at scale

Thank you!

 gulcin@xata.io

 xata.io

