JUST USE POSTGRESQL

WAAROM HET JE EERSTE KEUZE IS VOOR EEN [RELATIONELE] DATABASE

Wilbert van Dolleweerd <wilbert@arentheym.com>

OPEN SOURCE, VOLWASSEN, SAAI

> ⚠ Deze talk heeft een mening!

- Bestaat in open source vorm vanaf 1996.
- Correcte ACID[1] naleving.
- Correctheid belangrijker dan snelheid.
- Voldoet in hoge mate aan de SQL standaard.

---

1. Atomicity, Consistency, Isolation, Durability

VEEL UITBREIDINGEN

- PostGIS - werken met ruimtelijke (GIS) data
- TimeScaleDB - time-series database ideaal voor sensoren, metrics en events
- pg_cron - cron-jobs in de database
- pgvector - opslag van vectoren, vaak gebruikt in AI
- postgres_fdw - Foreign Data Wrappers, leg een verbinding naar externe bronnen
- PostgREST - zet een REST API laag op je database

MULTI-PARADIGMA:

- OLTP[1]
- OLAP[2]
- DocumentDB
- Event store
- Time-series database
- GIS database
- Vector database

---

1. Online Transaction Processing
2. Online Analytical Processing

TYPES

- Echte enums

```
CREATE TYPE nimmacode_location AS ENUM ('Nijmegen', 'Arnhem');

CREATE TABLE lightningtalk (
    id               serial PRIMARY KEY,
    name             text,
    current_location nimmacode_location
);
```

TYPES

- Samengestelde types

```
CREATE TYPE address AS (street text, city text, zipcode text);

CREATE TABLE person (
    id      serial PRIMARY KEY,
    name    text,
    home    address
);

INSERT INTO person (name, home)
VALUES (
    'Wilbert',
    ('Beethovenlaan 80', 'Doorwerth', '6865EC')
);

SELECT name, (home).city AS city FROM person;
```

TYPES

- Domein types met constraints

```
CREATE DOMAIN gigabecquerel AS numeric
CHECK (VALUE >= 0);

CREATE TABLE radioactive_sources (
    id              serial PRIMARY KEY,
    name            text NOT NULL,
    activity_gbq    gigabecquerel NOT NULL
);

INSERT INTO radioactive_sources (name, activity_gbq)
VALUES ('Cesium 137', 15.7);
```

TYPES

- Reeksen

```
CREATE TABLE reservations (
    id          serial PRIMARY KEY,
    guest_name  text NOT NULL,
    period      daterange NOT NULL
);

-- [ = inclusief, ( = exclusief
INSERT INTO reservations (guest_name, period)
VALUES
    ('Alice', '[2025-05-01, 2025-05-05]'),
    ('Bob',   '(2025-05-10, 2025-05-15]');

-- Retourneer reservation waar onderstaande datum in valt
SELECT * FROM reservations
WHERE period @> DATE '2025-05-02';
```

TYPES

- Bits

```
CREATE TABLE sensors (
    id      serial PRIMARY KEY,
    flags   BIT(4)   -- precies 4 bits
    -- flags BIT varying(4) -- tot 4 bits
);

INSERT INTO sensors (flags) VALUES (B'1010'); -- geldig
INSERT INTO sensors (flags) VALUES (B'101'); -- niet geldig

-- Haal sensors op waarvan het derde bit hoog staat
SELECT * FROM sensors WHERE flags & B'0010' = B'0010';

-- Alternatief
SELECT * FROM sensors WHERE substring(flags FROM 3 FOR 1) = B'1';
```

TYPES

- Arrays

```
CREATE TABLE products (
    id      serial PRIMARY KEY,
    name    text NOT NULL,
    tags    text[]
);

INSERT INTO products (name, tags)
VALUES
    ('Laptop', ARRAY['electronics', 'portable']),
    ('T-shirt', '{"clothing","cotton"}');  -- alternatieve notatie

-- Haal alle producten op met de tag 'electronics'
SELECT * FROM products
WHERE 'electronics' = ANY (tags);
```

TYPES

- JSON(B) data

```
CREATE TABLE products (
    id      serial PRIMARY KEY,
    name    text NOT NULL,
    details jsonb
);

INSERT INTO products (name, details)
VALUES (
    'Laptop',
    '{"brand": "Dell", "specs": {"ram": "16GB", "ssd": "512GB"}}');

SELECT name, details->>'brand' AS brand FROM products;

SELECT name, details->'specs'->>'ram' AS ram_memory FROM products;
```

TABLE PARTITIONING

```sql
CREATE TABLE orders (
    id          serial,
    order_date  date NOT NULL,
    amount      numeric,
    customer_id int
) PARTITION BY RANGE (order_date);

CREATE TABLE orders_2024 PARTITION OF orders
    FOR VALUES FROM ('2024-01-01') TO ('2024-12-31');
CREATE TABLE orders_2025 PARTITION OF orders
    FOR VALUES FROM ('2025-01-01') TO ('2025-12-31')

INSERT INTO orders (order_date, amount, customer_id)
VALUES ('2024-03-10', 199.99, 42),
       ('2025-07-01', 299.00, 77);

SELECT * FROM orders
WHERE order_date BETWEEN '2025-01-01' AND '2025-12-31';
```

VOLLEDIGE ONDERSTEUNING VOOR TRANSACTIONAL DDL[1]

Database schema wijzigingen kunnen in een transactie draaien.

```
BEGIN;

CREATE TABLE demo (id int);
ALTER TABLE demo ADD COLUMN name text;

ROLLBACK; -- Draai alles terug
```

---

1. Data Definition Language

LISTEN/NOTIFY

Verstuur events via pub/sub

```
-- Luister naar een kanaal
LISTEN new_order;

-- Stuur een event
NOTIFY new_order, 'Order #123 toegevoegd';

-- Resulteert in:
Asynchronous notification "new_order"
  with payload "Order #123 toegevoegd"
```

- 
- Cache invalidatie
- Verversen van een materialized view

UNLOGGED TABLES

Alle wijzigingen worden eerst weggeschreven naar een log (WAL[1]) voordat ze toegepast worden op de data pages. Het idee hierachter is:

> *Schrijf nooit data pages naar disk totdat de wijziging eerst in het log staat.*

Dit zorgt ervoor dat we altijd kunnen herstellen naar een consistente state met behulp van de WAL als er iets misgaat.

---

1. Write Ahead Log

UNLOGGED TABLES

```
CREATE UNLOGGED TABLE events_buffer (
    id        serial PRIMARY KEY,
    payload   jsonb
);

INSERT INTO events_buffer (payload)
VALUES ('{"type": "click", "user": 123}')
```

Met unlogged sla je de WAL over. Ideaal voor:

- Importeren van grote datasets (ETL)
- Cache achtige constructies
- Aggregraties die je opnieuw kan bouwen

⚠ Doe dit alleen met data die verloren mag gaan bij een crash.

VRAGEN?