PostgreSQL Tips & Tricks For App Devs

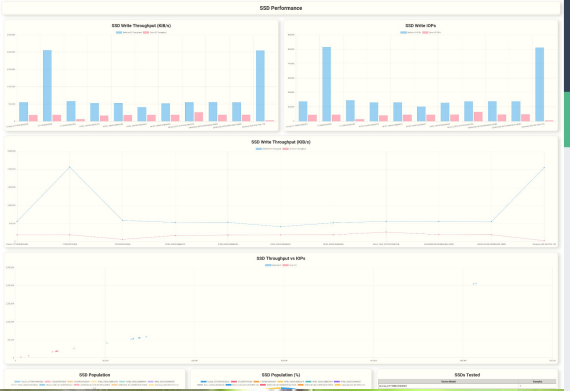
Work Smart, Not Hard!

PostgreSQL Conference Europe 2025 - Riga

Chris Ellis - @intrbiz@bergamot.social

Hello!

- I’m Chris
  - IT jack of all trades, studied Electronic Engineering
  - These days, mostly a technical architect
  - Spend most of my time building apps on top of PostgreSQL
- Been using PostgreSQL for about ~20 years
- Worked on various PostgreSQL and IoT projects



Classes

Classes

This Week | Next Week

Tuesday 05 September 2023

Intermediate
16:30 - 17:30
Kingston Dojo

Wednesday 06 September 2023

Advanced
17:00 - 18:00
Kingston Dojo

Tuesday 12 September 2023

Your Account

Your Membership

Pay As You Go
Come and train on an ad-hoc basis, pay per session, ideal for students starting out.

£25 per year - Membership Fee

£8 per class - Each class

Your next payment will be taken on

Pause Membership

Your Balance

Pay As You Go
Train as you need
Come and train on an ad-hoc basis, pay per session, ideal for students starting out.

2
Classes until
05 October 2023

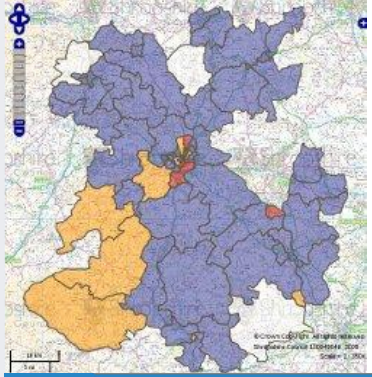
Your Details

Hello John Smith
8th Kyu
Email: info@mokuso.cloud
Mobile: 07848123456
Member since 05 September 2023

Change Details | Logout

Home | Classes | Account

Account



NEXTEAM

Shropshire Council

Home | Contact us | A to Z of services | Frequently asked questions

Shropshire Council | Family Information Directory | Community Directory

Search

returned 98 results in 0.13 Seconds

63 (Bridgnorth) Squadron Air Training Corps
Air Training Corps for young people between 13 and 20 years.

Albrighton Trust
Albrighton Trust provides recreation and education for people with disabilities.

Apprenticeships
Information about apprenticeships.

Archaeology Service
The Archaeology Service aims to provide community focused services which preserve and interpret the archaeological heritage of the county.

Ashley Music School
Ashley Music School offers tuition in piano and all keyboard, voice, woodwind, strings, drum kit, electric and acoustic guitar, recorders and theory.

Bishop's Castle IT Centre

Filter by category:
- [x] education and learning
- [ ] leisure and culture
- [ ] community and living
- [ ] health and social care
- [ ] environment and planning
- [ ] jobs and careers
- [ ] business
- [ ] transport and streets
- [ ] advice and benefits
- [ ] housing

Filter by location:
- [x] shropshire wide

<3 PostgreSQL

Right Tool For The Job?

Text Search

AS A: customer

I Want: to be easily able to find an applicable fault code for my appliance when raising a repair

So That: to get a better chance of my appliance being fixed first time

Text Search - Simple

```
SELECT *
FROM reference.fault_code
WHERE
 to_tsvector('english',
  title || ' ' || coalesce(description, '')
 )
 @@ to_tsquery('english', 'leak');
```

Text Search - Simple Yet Fast

```
CREATE INDEX fc_text_idx
ON reference.fault_code
USING GIN
(to_tsvector('english',
   title || ' ' || coalesce(description, '')
));
```

Text Search - Simple Yet Fast

```
Seq Scan on fault_code  (cost=0.00..870.51 rows=15
width=170) (actual time=0.084..24.966 rows=37
loops=1)
  Rows Removed by Filter: 2978
Planning Time: 0.172 ms
Execution Time: 25.069 ms
```

Text Search - Simple Yet Fast

```
Bitmap Heap Scan on fault_code  (cost=3.03..22.53
rows=15 width=170) (actual time=0.044..0.167 rows=37
loops=1)
  Heap Blocks: exact=20
  ->  Bitmap Index Scan on fc_text_idx
(cost=0.00..3.03 rows=15 width=0) (actual
time=0.027..0.028 rows=37 loops=1)
Planning Time: 0.308 ms
Execution Time: 0.271 ms
```

Text Search - Realistic

```
ALTER TABLE reference.fault_code
 ADD COLUMN vector TSVECTOR;

CREATE INDEX fc_vector_idx
ON reference.fault_code
USING GIN (vector);
```

Text Search - Realistic

```
UPDATE reference.fault_code
SET vector =
  setweight(
    to_tsvector(coalesce(category,'')), 'A'
  ) ||
  setweight(
    to_tsvector(coalesce(description,'')), 'B'
  );
```

Text Search - Realistic

```
SELECT
  ts_rank_cd(vector,
    websearch_to_tsquery(...)), *
FROM reference.fault_code
WHERE vector @@ websearch_to_tsquery(
  'english', 'leaking door')
ORDER BY 1;
```

AS A: complaints analyst

I Want: to be able to filter call recordings by matched keywords / topics

So That: to prioritize which calls to proactively investigate

Tags / Topics / Keywords

```
CREATE TABLE comms.call (
  id                UUID          NOT NULL,
  phone             TEXT          NOT NULL,
  transcript        JSON          NOT NULL,
  …
  topics            TEXT[]                 ,
);
```

Tags / Topics / Keywords

```
SELECT *
FROM comms.call
WHERE topics @> ARRAY['breakdown'];

SELECT *
FROM comms.call
WHERE topics @> ARRAY['breakdown', 'boiler'];
```

Tags / Topics / Keywords

```
CREATE TABLE comms.call (
   id                UUID          NOT NULL,
   phone             TEXT          NOT NULL,
   transcript        JSON          NOT NULL,
   …
   keywords          JSONB                 ,
);
```

Tags / Topics / Keywords

```
SELECT *
FROM comms.call
WHERE keywords @>
        '{"make": "bosch"}'::JSONB;
```

Tags / Topics / Keywords

```
CREATE INDEX topics_idx
ON comms.call USING GIN (topics);

CREATE INDEX keywords_idx
ON comms.call USING GIN (keywords);
```

GIS

AS A: customer

I Want: to find classes at venues near to me

So That: I can book classes that I can easily get to

Location Search

```
CREATE TABLE club.venue (
   id              UUID        NOT NULL,
   name            TEXT        NOT NULL,
   description     TEXT        NOT NULL,
   address         TEXT        NOT NULL,
   location        POINT
);
```

Location Search

```
SELECT *
FROM club.venue
WHERE st_dwithin(location, $1, 2000);
```

AS A: repair provider

I Want: to allocate visits to different engineers nearest to their operating areas

So That: we can optimally allocate which engineers attend which appointments

Location Matching

```
CREATE TABLE provider.engineer (
   id     UUID          NOT NULL,
   name   TEXT          NOT NULL,
   area   Geometry(MultiPolygon, 4326)
);
```

Location Matching

```
SELECT *
FROM provider.engineer
WHERE st_contains(area, $1);
```

Location Matching

```
SELECT *
FROM provider.engineer
WHERE st_intersects(area,
   st_buffer(
      st_point(-71.104, 42.315, 4326),
      0.025
   )
);
```

Location Search / Matching - Faster

```
CREATE INDEX venue_location_idx
ON club.venue GIST (location);
```

All Together Now

```
SELECT *
FROM search.content
WHERE vector @@ to_tsquery('library')
AND st_dwithin(location, my_location, 2000)
AND tags @> ARRAY['service_catalogue'];
```

Unknown Unknowns

AS A: product owner

I Want: to be able to analyse how the questions we ask customers effect sales

So That: we can optimise the get a quote user flow

Unknown Unknowns

```
CREATE TABLE insurance.quote (
  id              UUID          NOT NULL,
  customer_id     UUID          NOT NULL,
  status          STATUS        NOT NULL,
  price           NUMERIC       NOT NULL,
  answers         JSONB
);
```

Unknown Unknowns

```
SELECT count(*),
       count(*) FILTER (WHERE (answers ->> 'locks')
                         IS NULL),
       count(*) FILTER (WHERE (answers ->> 'locks')
                         IS NOT NULL),
       count(*) FILTER (WHERE (answers ->> 'locks')
                         = '3-lever'),
       count(*) FILTER (WHERE (answers ->> 'locks')
                         = 'unknown')
FROM insurance.quotes;
```

AS A: tech-lead

I Want: to prevent my developers inserting invalid data

So That: we find problems, before they really become problems

Check Constraints

```
ALTER TABLE insurance.quote

ADD CONSTRAINT answers_chk

CHECK (

  jsonb_typeof( answers ) = 'object'

);
```

Stopping Things Going Wrong

AS A: customer

I Want: I don’t want to get billed twice for my subscription

So That: should be obvious really…

Subscriptions

```
CREATE TABLE club.subscription (
   id              UUID        NOT NULL,
   member_id       UUID        NOT NULL,
   plan_id         UUID        NOT NULL,
   status          STATUS      NOT NULL,
   …
);
```

Subscriptions

```
CREATE UNIQUE INDEX active_subs
ON club.subscription
   (member_id)
WHERE status = 'active';
```

Invoicing With SQL

AS A: app developer

I Want: to get paid by the users of my app

So That: all is good in the world

Generate Invoices - Writable CTEs

```
WITH invoice_commission AS (
    UPDATE billing.commission_record
    SET invoice_id = 123
    WHERE invoice_id IS NULL
    RETURNING *
) INSERT INTO billing.invoice
SELECT 123, current_date, sum(value) AS total
FROM invoice_commission;
```

Get Latest Invoice - Lateral Joins

```
SELECT t.*, q.*
FROM platform.tenant t
LEFT JOIN LATERAL (
    SELECT invoice_date, total
    FROM billing.invoice i
    WHERE i.tenant_id = t.id
    ORDER BY invoice_date DESC
    LIMIT 1
) q ON (true);
```

Tasks & Queues

AS A: platform

I Want: ensure that we process subscription payments and payment events, and can replay them if needed

So That: our payments handling does not require manual intervention

Queues - A Simple Queue / Task

```
CREATE TABLE queue.event (
  created     TIMESTAMP  NOT NULL,
  updated     TIMESTAMP             ,
  status      INTEGER    NOT NULL,
  payload     TEXT
);
```

Queues - Fetch A Batch

```
SELECT ctid, * FROM queue.event
WHERE status < 5 AND (status = 0 OR
 updated < (now() - '1 hour'::INTERVAL))
ORDER BY created DESC
LIMIT 1 /* Or more */
FOR UPDATE SKIP LOCKED;
```

Queues - Index Time

```
CREATE INDEX queue_event_idx
ON queue.event (created)
WHERE status < 5;
```

Queues - Fetch A Batch

```
 Limit
 (cost=0.29..0.86 rows=10 width=54)
 (actual time=0.060..0.114 rows=10 loops=1)
  ->  LockRows
      (cost=0.29..4920.33 rows=86401 width=54)
      (actual time=0.057..0.109 rows=10 loops=1)
        ->  Index Scan Backward using queue_event_idx on event
            (cost=0.29..4056.32 rows=86401 width=54)
            (actual time=0.037..0.060 rows=10 loops=1)
              Filter: ((status < 5) AND ((status = 0) OR
                       (updated < (now() - '1 hour'::interval))))
Planning Time: 0.260 ms
Execution Time: 0.179 ms
```

Queues - Retry An Event

```
UPDATE queue.event
SET updated = now(),
    status = status + 1
WHERE ctid = '(719,117)';
```

Queues - Processed An Event

```
UPDATE queue.event
SET updated = now(),
    status = 2147483647
WHERE ctid = '(720,2)';
```

Mind The Gap

AS A: DBA

I Want: efficiently store energy meter data in PostgreSQL

So That: we don’t waste too much storage space

Roll Ups

```
CREATE TABLE iot.daily_reading (
   meter_id        UUID        NOT NULL,
   read_range      DATERANGE   NOT NULL,
   energy          BIGINT,
   energy_profile BIGINT[],
   PRIMARY KEY (device_id, read_range)
);
```

Roll Ups

t_xmin	t_xmax	t_cid	t_xvac	t_ctid	t_infomask2	t_infomask	t_hoff
4	4	4	4	6	2	2	1

24 bytes

device_id	read_at	temperature	light
16	8	4	4

32 bytes

AS A: customer

I Want: to be able to visualise my energy consumption

So That: I can better understand how I consume my energy and can reduce my usage

Generate Series - Presenting Data

```
SELECT r.device_id, t.time, array_agg(r.read_at),
       avg(r.temperature), avg(r.light)
FROM generate_series(
   '2022-10-06 00:00:00'::TIMESTAMP,
   '2022-10-07 00:00:00'::TIMESTAMP, '10 minutes') t(time)
JOIN iot.alhex_reading r
   ON (r.device_id = '26170b53-ae8f-464e-8ca6-2faeff8a4d01'::UUID
       AND r.read_at >= t.time
       AND r.read_at < (t.time + '10 minutes'))
GROUP BY 1, 2
ORDER BY t.time;
```

Window Functions - Roll Up

```
SELECT
   commission AS daily_total,
   sum(commission) OVER
   (PARTITION BY date_trunc('week', day))
    AS weekly_total
FROM billing.daily;
```

Window Functions - Counters

```
SELECT
 day,
 energy,
 energy - coalesce(lag(energy)
    OVER (ORDER BY day), 0) AS consumed
FROM iot.meter_reading
ORDER BY day;
```

Custom Aggregates - Mind The Gap

```
WITH days AS (
  SELECT t.day::DATE
  FROM generate_series('2017-01-01'::DATE,
'2017-01-15'::DATE, '1 day') t(day)
), data AS (
   SELECT *
   FROM iot.meter_reading
   WHERE day >= '2017-01-01'::DATE
   AND   day <= '2017-01-15'::DATE
)
```

Custom Aggregates - Mind The Gap

```
SELECT day,
       coalesce(energy,
         (((next_read - last_read)
            / (next_read_time - last_read_time))
            * (day - last_read_time))
            + last_read) AS energy_interpolated
FROM (
    ... from next slide ...
) q
ORDER BY day
```

Custom Aggregates - Mind The Gap

```
SELECT t.day, d.energy,
  last(d.day)    OVER lookback    AS last_read_time,
  last(d.day)    OVER lookforward AS next_read_time,
  last(d.energy) OVER lookback    AS last_read,
  last(d.energy) OVER lookforward AS next_read
FROM days t
LEFT JOIN data d ON (t.day = d.day)
WINDOW
  lookback AS (ORDER BY t.day),
  lookforward AS (ORDER BY t.day DESC)
```

Custom Aggregates - Mind The Gap

```
CREATE FUNCTION last_agg(anyelement, anyelement)
RETURNS anyelement LANGUAGE SQL IMMUTABLE STRICT AS $$
        SELECT $2;
$$;

CREATE AGGREGATE last (
        sfunc = last_agg,
        basetype = anyelement,
        stype = anyelement
);
```

Any Questions?

Appendix - Mind The Gap

```
WITH days AS (
  SELECT t.day::DATE
  FROM generate_series('2017-01-01'::DATE, '2017-01-15'::DATE, '1 day') t(day)
), data AS (
       SELECT *
       FROM iot.meter_reading
       WHERE day >= '2017-01-01'::DATE AND day <= '2017-01-15'::DATE
)
SELECT day, coalesce(energy_import_wh, (((next_read - last_read) / (next_read_time - last_read_time)) * (day -
last_read_time)) + last_read) AS energy_import_wh_interpolated
FROM (
  SELECT t.day, d.energy_import_wh,
       last(d.day) OVER lookback AS last_read_time,
       last(d.day) OVER lookforward AS next_read_time,
       last(d.energy_import_wh) OVER lookback AS last_read,
       last(d.energy_import_wh) OVER lookforward AS next_read
  FROM days t
  LEFT JOIN data d ON (t.day = d.day)
  WINDOW
       lookback AS (ORDER BY t.day),
       lookforward AS (ORDER BY t.day DESC)
) q  ORDER BY q.day
```