DevSheets

PostgreSQL Advanced

Types, arrays, JSONB and windows.

01 Types worth knowing

SERIAL	Auto-increment via a sequence
NUMERIC(10,2)	Exact decimal, for money
REAL	4-byte float; FLOAT is 8-byte
TEXT	Any length; VARCHAR(n) caps it
TIMESTAMPTZ	UTC, shown in the session zone
INTERVAL	Duration: '1 hour 30 minutes'

Cast with '123'::INTEGER or CAST(x AS NUMERIC); pg_typeof(x) names a type. Identity columns are the SQL-standard alternative to SERIAL.

02 UUID, CITEXT & INET

```
CREATE EXTENSION IF NOT EXISTS citext;
CREATE TABLE users (
  id UUID DEFAULT uuidv7() PRIMARY KEY,
  email CITEXT UNIQUE,  -- any case
  ip_address INET       -- validated
);
```

uuidv7() (new in 18) is time-ordered; gen_random_uuid() is random v4. Both are built in. Prefer NUMERIC to MONEY.

03 Range types

```
CREATE TABLE bookings (
  room_id INTEGER, during TSTZRANGE);
INSERT INTO bookings VALUES (101,
  '[2024-01-01 14:00, 2024-01-01 16:00)');
SELECT * FROM bookings WHERE during &&
  '[2024-01-01 15:00, 2024-01-01 17:00)';
```

[includes a bound and) excludes it. && is true when two ranges overlap.

04 Arrays

```
CREATE TABLE products (tags TEXT[]);
INSERT INTO products VALUES
  (ARRAY['electronics', 'computers']),
  ('{education, programming}');
SELECT * FROM products
WHERE 'computers' = ANY(tags)
  AND tags @> ARRAY['electronics'];
```

ARRAY[] and a quoted '{}' literal both build arrays. ANY() matches one element; @> needs every listed element.

05 Array functions

```
SELECT tags[1], array_length(tags, 1),
       array_append(tags, 'new'),
       array_remove(tags, 'old'),
       unnest(tags)          -- row each
FROM products;
SELECT category, array_agg(DISTINCT tag)
FROM product_tags GROUP BY category;
```

Arrays are 1-based. array_length returns NULL, not 0, for an empty array. array_agg gathers rows into an array.

06 Query JSONB

```
CREATE TABLE events (data JSONB NOT NULL);
INSERT INTO events VALUES
  ('{"user": "jane", "items": ["book"]}');
SELECT data->>'user' FROM events  -- text
WHERE data->>'action' = 'login';
SELECT data->'items'->0           -- jsonb
FROM events WHERE data ? 'items';
```

-> returns jsonb and ->> returns text; ? tests for a top-level key. JSON arrays are 0-based. Prefer JSONB to JSON.

07 Update & index JSONB

```
UPDATE events SET data =
  jsonb_set(data, '{status}', '"done"')
WHERE data @> '{"user": "john"}';
CREATE INDEX idx_data ON events
  USING gin (data);
CREATE INDEX idx_user ON events
  ((data->>'user'));
```

jsonb_set adds a missing key by default. The GIN index serves @> and ?; the expression index serves data->>'user' lookups.

08 Ranking functions

```
SELECT name, salary,
  ROW_NUMBER() OVER (ORDER BY salary DESC),
  RANK()       OVER (ORDER BY salary DESC),
  DENSE_RANK() OVER (ORDER BY salary DESC),
  ROW_NUMBER() OVER (PARTITION BY dept
    ORDER BY salary DESC) AS dept_rank
FROM employees;
```

For 90, 90, 80: ROW_NUMBER gives 1, 2, 3; RANK 1, 1, 3; DENSE_RANK 1, 1, 2. PARTITION BY restarts per group.

09 Running totals & LAG

```
SELECT day, amount,
  SUM(amount) OVER (ORDER BY day) AS total,
  AVG(amount) OVER (ORDER BY day ROWS
    BETWEEN 2 PRECEDING AND CURRENT ROW),
  amount - LAG(amount) OVER (ORDER BY day)
    AS growth
FROM daily_sales;
```

Rows tied on day share one running total (the default RANGE frame); ROWS counts physical rows.

DevSheets

PostgreSQL Advanced

CTEs, indexes, tuning and locks in 18.

10 Writable & materialized CTEs

```
WITH moved AS (
  DELETE FROM orders
  WHERE created < now() - INTERVAL '1 year'
  RETURNING *
)
INSERT INTO archive SELECT * FROM moved;
```

A plain SELECT CTE used once is inlined. AS MATERIALIZED computes it separately; NOT MATERIALIZED forces inlining.

11 Recursive CTEs

```
WITH RECURSIVE org AS (
  SELECT id, name, 0 AS level
  FROM employees WHERE manager_id IS NULL
  UNION ALL
  SELECT e.id, e.name, org.level + 1
  FROM employees e
  JOIN org ON e.manager_id = org.id
) SELECT * FROM org ORDER BY level, name;
```

Repeats until the second SELECT adds no rows. For data that can loop, add CYCLE id SET is_cycle USING path.

12 Full-text search

```
ALTER TABLE articles ADD COLUMN doc
  tsvector GENERATED ALWAYS AS
  (to_tsvector('english', body)) STORED;
CREATE INDEX ON articles USING gin (doc);
SELECT title FROM articles
WHERE doc @@ to_tsquery('english',
  'postgres & (index | tuning) & !slow');
```

Generated columns are virtual by default in 18: write STORED to index one. ts_rank(doc, query) scores matches.

13 Index types

btree	Default; fits most queries
gin	Arrays, JSONB, full-text
gist	Geometric data, ranges
brin	Huge tables in key order
hash	Equality (=) only

```
CREATE INDEX idx_created ON logs
  USING brin (created_at);
```

14 Partial, expression & concurrent

```
CREATE INDEX idx_active ON orders (user_id)
WHERE status = 'active';
CREATE INDEX idx_lower_email
  ON users (lower(email));
CREATE INDEX CONCURRENTLY idx_email
  ON users (email);
```

Queries must match the WHERE or expression to use them. CONCURRENTLY allows writes but can't run in a transaction; a failure leaves an INVALID index.

15 EXPLAIN & slow queries

```
EXPLAIN (ANALYZE, VERBOSE)
SELECT * FROM orders WHERE user_id = 123;
SELECT query, calls, mean_exec_time
FROM pg_stat_statements
ORDER BY mean_exec_time DESC LIMIT 10;
SELECT relname, indexrelname, idx_scan
FROM pg_stat_user_indexes ORDER BY idx_scan;
```

ANALYZE runs the statement; 18 adds BUFFERS by default. pg_stat_statements needs shared_preload_libraries.

16 VACUUM & settings

```
VACUUM (ANALYZE) users;   -- reclaim, stats
VACUUM FULL users;        -- rewrite, locks
ALTER SYSTEM SET work_mem = '64MB';
SELECT pg_reload_conf();  -- applies it
ALTER SYSTEM SET shared_buffers = '4GB';
-- applies only after a server restart
```

Autovacuum handles routine VACUUM; FULL blocks all access. work_mem applies per sort or hash step.

17 Materialized views

```
CREATE MATERIALIZED VIEW monthly AS
SELECT date_trunc('month', created_at)
         AS month, SUM(amount) AS total
FROM orders GROUP BY 1;
CREATE UNIQUE INDEX ON monthly (month);
REFRESH MATERIALIZED VIEW CONCURRENTLY
  monthly;
```

CONCURRENTLY keeps the view readable during a refresh but needs a unique index on plain columns.

18 Transactions & locks

```
BEGIN ISOLATION LEVEL REPEATABLE READ;
SAVEPOINT before_risky;
DELETE FROM logs;
ROLLBACK TO before_risky;  -- undo DELETE
SELECT * FROM orders WHERE id = 123
FOR UPDATE;                -- row lock
COMMIT;
```

READ COMMITTED is the default. pg_advisory_lock(key) holds until pg_advisory_unlock(key) or session end, even past ROLLBACK.