Admissions & Support:
support@mockattempt.com
Phone / WhatsApp: +91 95400 07917
Course Code: PGSQ • **Updated:** 05 Nov 2026
https://mockattempt.com/courses/postgresql-training

PGSQ • RELATIONAL DATABASE INTERNALS & SQL ENGINEERING | **BEGINNER TO PRINCIPAL POSTGRESQL DBA & ARCHITECT**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

PostgreSQL Training

Master PostgreSQL 17 from MVCC Snapshot Isolation, WAL Durability, `pg_hba.conf` & Multi-Tenant `search_path` Schemas to Exact `NUMERIC`, `BIGINT GENERATED ALWAYS AS IDENTITY`, `TIMESTAMPTZ`, `INTERVAL`, `UUID`, `JSONB` GIN Indexing, TOAST `TEXT` & Transactional DDL — with runnable SQL scripts, 244 Topic Mock Tests (10,440 MCQs), and Spoken Voice + MockAttempt AI Interviews at ₹1,999 (Incl. 18% GST).

Duration: 15–17 Weeks • Live + Hands-On Projects (206 Hrs) • **Modules:** 16 • **Live Topics:** 42 • **Subtopics:** 336+ • **Labs & Tasks:** 84 • **Mocks / AI Vivas:** 244 / 244 • **Projects:** 16 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹1,999 (Net Tuition ₹1,694 + Included 18% GST ₹305) • **Split Option:** 100% Credit to Job Track • **Effective ROI:** ~₹10/Guided Hr (~₹48/Live Topic)

Faculty: 100% Active Working IT Professionals • **Live Batch Tracks (7:00 AM – 8:00 PM IST Flexible Slots):** Weekday Live Online Batch (Mon – Fri • 1 Hr/Day • 7:00 AM – 8:00 PM) (Monday to Friday (1 Hour Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 1-Hour Slot Based on Trainer Availability) • 17–19 Weeks Fast-Track) | Weekend Live Online Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Slot Based on Trainer Availability) • 19–21 Weeks Weekend Track)

✓ **Trainer LeetCode & HackerRank Mentorship (SQL & PL/pgSQL (LeetCode SQL 50 & HackerRank SQL)): Live trainer support to crack LeetCode & HackerRank problems, optimize complexity, and handle edge cases.**

Core Technologies & Engineering Stack: PostgreSQL 17 Engine & `psql` • MVCC, WAL & Autovacuum Internals • PgBouncer, `pg_hba.conf` & SCRAM-SHA-256 • Schemas, RBAC & `search_path` Hardening • `NUMERIC`, `BIGINT IDENTITY` & `TIMESTAMPTZ` • `UUID` & Binary `JSONB` + GIN Indexes • `TEXT` TOAST Storage & Transactional DDL

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Understand PostgreSQL's Multi-Version Concurrency Control (MVCC), Write-Ahead Logging (WAL), TOAST storage, and why readers never block writers
2. Harden PostgreSQL clusters with `scram-sha-256` in `pg_hba.conf`, PgBouncer pooling, PostgreSQL 15+ `public` schema lockdown, and explicit `search_path` rules
3. Choose optimal PostgreSQL data types (`NUMERIC` vs float, `TEXT` vs `CHAR(n)`, `BIGINT GENERATED ALWAYS AS IDENTITY` vs legacy `SERIAL`, `TIMESTAMPTZ`, `UUID`, and GIN-indexed `JSONB`)
4. Clear 244 Timed Mock Exams (10,000 PostgreSQL MCQs) and 200 Spoken Voice + MockAttempt AI PostgreSQL DBA & Backend Interviews
5. Rich PostgreSQL data modeling: strict constraints, JSONB hybrid documents, arrays, UUIDv7, and Row-Level Security (RLS).
6. Crack LeetCode and HackerRank problems in SQL & PL/pgSQL (LeetCode SQL 50 & HackerRank SQL) with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
7. Precision indexing: B-Tree, GIN (JSONB & Full-Text Search), GiST, BRIN, partial/expression indexes, and `pgvector` HNSW.
8. MVCC concurrency internals, WAL durability, autovacuum bloat prevention, declarative table partitioning, and CTE windowing.

Eligibility & Prerequisites: Basic familiarity with tables, rows, and introductory SQL queries — covers PostgreSQL 17 architecture and SQL standards step-by-step from scratch • Any laptop/PC running PostgreSQL 16/17 (local installer or Docker `postgres:17`), `psql`, and DBeaver / pgAdmin 4 / DataGrip • No prior PostgreSQL industry experience required — Module 1 starts from environment setup, core mental models, and plain-English fundamentals. • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Commitment to attend live Weekday or Weekend classes, complete daily hands-on lab checklists, and practice topic-wise Mock Tests & AI Interviews.

2. DETAILED ZERO-TO-EXPERT MODULE, TOPIC & SUBTOPIC SYLLABUS (16 MODULES • 42 LIVE TOPICS • 336+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Stage 1: Day-1 Foundations & Core Mental Models (8 Topics / 64 Subtopics) • Modules 5–8: Stage 2: Applied Engineering, Data & API Workflows (8 Topics / 64 Subtopics) • Modules 9–12: Stage 3: Internals, Concurrency, Security & Performance (12 Topics / 96 Subtopics) • Modules 13–16: Stage 4: Production Architecture, Testing, CI/CD & Expert Mastery (14 Topics / 112 Subtopics)

MODULE 1 OF 16 • STAGE 1 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: PostgreSQL Object-Relational Architecture, MVCC Internals & Enterprise Advantages

Explore PostgreSQL's process/memory architecture (`shared_buffers`, WAL, 8 KB heap pages), MVCC non-blocking snapshot isolation, Transactional DDL, and rich extensibility (`pgvector`, PostGIS).

Objectives: ✓ PostgreSQL 17 Architecture — Object-Relational Engine, 8 KB Pages, WAL & MVCC Snapshot Isolation — How PostgreSQL achieves strict ACID compliance using Write-Ahead Logging (WAL), 8 KB slotted heap pages, `shared_buffers`, and `xmin`/`xmax` MVCC tuple snapshots. • ✓ Advantages of PostgreSQL — Transactional DDL, Partial/GIN Indexes, Rich Types & Extensions — Why PostgreSQL leads modern engineering stacks: atomic Transactional DDL (`BEGIN ... ALTER TABLE ... ROLLBACK`), Partial & Expression indexes, `JSONB`, `pgvector`, and BSD/PostgreSQL open licensing. • ✓ Build, verify, and ship Module Capstone: MVCC Tuple Visibility & Atomic `INSERT ... ON CONFLICT ... RETURNING` Ledger.

1.1 PostgreSQL 17 Architecture — Object-Relational Engine, 8 KB Pages, WAL & MVCC Snapshot Isolation

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: MVCC xmin / xmax tuple versioning & Readers never block writers invariant • Core Syntax, CLI & Production APIs: SELECT version(), current_setting('server_version_num'), pg_current_xact_id(); • Deep-Dive Mechanics: Write-Ahead Logging (WAL) crash durability & 8 KB slotted heap pages & shared_buffers • Advanced Patterns: Autovacuum dead-tuple cleanup & XID freeze & INSERT ... ON CONFLICT ... RETURNING • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: MVCC `xmin` Tuple Inspection & Atomic Upsert with `RETURNING` • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL 17 Architecture

Purpose: To understand and apply PostgreSQL 17 Architecture — Object-Relational Engine, 8 KB Pages, WAL & MVCC Snapshot Isolation in real PostgreSQL systems — specifically: How Po...

What Topic Is Next → Next Topic 1.2: Advantages of PostgreSQL — Transactional DDL, Partial/GIN Indexes, Rich Types & Extensions

1.2 Advantages of PostgreSQL — Transactional DDL, Partial/GIN Indexes, Rich Types & Extensions

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Atomic Transactional DDL migrations & Partial Indexes (CREATE INDEX ... WHERE) • Core Syntax, CLI & Production APIs: BEGIN; ALTER TABLE ...; CREATE INDEX ... WHERE active = true; COMMIT; • Deep-Dive Mechanics: Expression & Covering (INCLUDE) indexes & Extensible ecosystem (PostGIS, pgvector, pg_stat_statements) • Advanced Patterns: SQL standard compliance (Window Functions, CTEs, MERGE) & Permissive PostgreSQL open-source license • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Transactional DDL Rollback Proof & Partial Index Definition • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for Advantages of PostgreSQL

Purpose: To understand and apply Advantages of PostgreSQL — Transactional DDL, Partial/GIN Indexes, Rich Types & Extensions in real PostgreSQL systems — specifically: Why PostgreS...

What Topic Is Next → Module 1 Capstone (MVCC Tuple Visibility & Atomic `INSERT ... ON CONFLICT ... RETURNING` Ledger) → Next Topic 2.1: Installing & Tuning PostgreSQL 17 — `initdb --data-checksums`, `$PGDATA` & `postgresql.conf`

Module 1 Hands-On Capstone Project (100 Marks): MVCC Tuple Visibility & Atomic `INSERT ... ON CONFLICT ... RETURNING` Ledger — Write a PostgreSQL 17 SQL script inspecting live vs dead tuple counters (`pg_stat_user_tables`), transaction IDs (`pg_current_xact_id()`), and atomic idempotent upserts via `ON CONFLICT DO U...

MODULE 2 OF 16 • STAGE 1 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Cluster Initialization (`initdb`), `postgresql.conf` Tuning & Secure Client Connectivity

Initialize `$PGDATA` clusters cleanly, tune `shared_buffers`/`work_mem`, lock down `pg_hba.conf` with `scram-sha-256` TLS, and scale connections via PgBouncer.

Objectives: ✓ Installing & Tuning PostgreSQL 17 — `initdb --data-checksums`, `$PGDATA` & `postgresql.conf` — Provisioning PostgreSQL 17 with hardware page checksums (`--data-checksums`) and sizing `shared_buffers` (~25% RAM), `work_mem`, and `wal_level = replica`. • ✓ Connecting to PostgreSQL — Port 5432, `pg_hba.conf` (`scram-sha-256`), TLS & PgBouncer Pooling — Securing TCP/Unix-socket connections via `pg_hba.conf` (`hostssl ... scram-sha-256`), `psql` connection URIs, `sslmode=verify-full`, and PgBouncer process-per-connection pooling. • ✓ Build, verify, and ship Module Capstone: Cluster Memory Parameter Auditor & Active Session Diagnostics (`pg_stat_activity`).

2.1 Installing & Tuning PostgreSQL 17 — `initdb --data-checksums`, `$PGDATA` & `postgresql.conf`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Initdb --data-checksums silent corruption guard & $PGDATA directory layout (base/, pg_wal/) • Core Syntax, CLI & Production APIs: initdb -D /var/lib/postgresql/data --data-checksums -E UTF8 + ALTER SYSTEM SET ... • Deep-Dive Mechanics: Shared_buffers vs OS page cache (~25% RAM) & Work_mem per-operation multiplication hazard • Advanced Patterns: ALTER SYSTEM + pg_reload_conf() live reload & UTF8 server encoding baseline • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Runtime `pg_settings` Encryption & Session Timeout Audit • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for Installing

Purpose: To understand and apply Installing & Tuning PostgreSQL 17 — `initdb --data-checksums`, `$PGDATA` & `postgresql.conf` in real PostgreSQL systems — specifically: Provisioni...

What Topic Is Next → Next Topic 2.2: Connecting to PostgreSQL — Port 5432, `pg_hba.conf` (`scram-sha-256`), TLS & PgBouncer Pooling

2.2 Connecting to PostgreSQL — Port 5432, `pg_hba.conf` (`scram-sha-256`), TLS & PgBouncer Pooling [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: One OS backend process per client connection & Pg_hba.conf first-match host-based auth • Core Syntax, CLI & Production APIs: postgresql://app_user@db.internal:5432/academy?sslmode=verify-full&application_name=api • Deep-Dive Mechanics: Scram-sha-256 vs deprecated md5 / trust & Sslmode=verify-full MITM protection • Advanced Patterns: PgBouncer transaction-mode connection pooling & Application_name observability in pg_stat_activity • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Session Context & Connection Metadata Inspector • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for Connecting to PostgreSQL

Purpose: To understand and apply Connecting to PostgreSQL — Port 5432, `pg_hba.conf` (`scram-sha-256`), TLS & PgBouncer Pooling in real PostgreSQL systems — specifically: Securing...

What Topic Is Next → Module 2 Capstone (Cluster Memory Parameter Auditor & Active Session Diagnostics (`pg_stat_activity`)) → Next Topic 3.1: PostgreSQL SQL Syntax — Unquoted Lowercase Folding, Single vs Double Quotes, `$$` & Clause Order

Module 2 Hands-On Capstone Project (100 Marks): Cluster Memory Parameter Auditor & Active Session Diagnostics (`pg_stat_activity`) — Write a PostgreSQL DBA diagnostic script auditing key runtime settings (`pg_settings`), idle-in-transaction timeouts, and active client backend states (`pg_stat_activity`).

MODULE 3 OF 16 • STAGE 1 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: PostgreSQL Lexical Syntax, Identifier Folding & `CREATE DATABASE` Internals

Master ANSI string literals vs double-quoted case-sensitive identifiers, dollar-quoting (`$$`), logical `SELECT` evaluation order, and template-based `CREATE DATABASE` isolation.

Objectives: ✓ PostgreSQL SQL Syntax — Unquoted Lowercase Folding, Single vs Double Quotes, `$$` & Clause Order — Mastering PostgreSQL's strict ANSI lexical rules: unquoted identifiers fold to lowercase (`UserID` -> `userid`), `'single quotes'` denote values, `"double quotes"` denote identifiers, and `$$` avoids backslash/quote escaping. • ✓ PostgreSQL `CREATE DATABASE` — `template0` vs `template1`, `UTF8` ICU Collations & Cross-DB Isolation — Creating physically isolated databases inside a PostgreSQL cluster (`OWNER`, `ENCODING 'UTF8'`, `TEMPLATE template0`, `LOCALE_PROVIDER icu`, `CONNECTION LIMIT`, and PG 13+ `DROP DATABASE ... WITH (FORCE)`). • ✓ Build, verify, and ship Module Capstone: Snake-Case Identifier Standardizer & Database Collation / Limit Hardener.

3.1 PostgreSQL SQL Syntax — Unquoted Lowercase Folding, Single vs Double Quotes, `$$` & Clause Order [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Unquoted identifier lowercase folding rule & Single quotes (values) vs double quotes (identifiers) • Core Syntax, CLI & Production APIs: WITH ranked AS (...) SELECT col_name, tagraw 'text'tag FROM ... WHERE ... ORDER BY ... L... • Deep-Dive Mechanics: Dollar-quoted string constants (tag...tag) & Logical SQL evaluation order (FROM before SELECT) • Advanced Patterns: $1, $2 parameterized query placeholders & CTE WITH inline optimization in PG 12+ • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Lowercase Identifier Folding & Dollar-Quoted String Verification • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL SQL Syntax

Purpose: To understand and apply PostgreSQL SQL Syntax — Unquoted Lowercase Folding, Single vs Double Quotes, `$$` & Clause Order in real PostgreSQL systems — specifically: Master...

What Topic Is Next → Next Topic 3.2: PostgreSQL `CREATE DATABASE` — `template0` vs `template1`, `UTF8` ICU Collations & Cross-DB Isolation

3.2 PostgreSQL `CREATE DATABASE` — `template0` vs `template1`, `UTF8` ICU Collations & Cross-DB Isolation [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Template1 default vs pristine template0 & Strict physical cross-database query isolation • Core Syntax, CLI & Production APIs: CREATE DATABASE academy_prod WITH OWNER = app_owner ENCODING = 'UTF8' TEMPLATE = template0 C... • Deep-Dive Mechanics: Cannot run CREATE DATABASE inside transaction block & ENCODING UTF8 + ICU deterministic collations • Advanced Patterns: CONNECTION LIMIT per-database cap & DROP DATABASE ... WITH (FORCE) session termination • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Database Catalog Encoding & Connection Limit Inspection • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL `CREATE DATABASE`

Purpose: To understand and apply PostgreSQL `CREATE DATABASE` — `template0` vs `template1`, `UTF8` ICU Collations & Cross-DB Isolation in real PostgreSQL systems — specifically: C...

What Topic Is Next → Module 3 Capstone (Snake-Case Identifier Standardizer & Database Collation / Limit Hardener) → Next Topic 4.1: PostgreSQL Schemas & `search_path` — Namespace Resolution, PG 15+ `public` Hardening & Multi-Tenancy

Module 3 Hands-On Capstone Project (100 Marks): Snake-Case Identifier Standardizer & Database Collation / Limit Hardener — Write a clean snake_case PostgreSQL SQL query using CTEs (`WITH`), dollar-quoted strings, and `COALESCE` alongside role/database `CONNECTION LIMIT` and `STATEMENT_TIMEOUT` governance.

MODULE 4 OF 16 • STAGE 1 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Namespaces (`Schemas`), `search_path` Security & PostgreSQL 15+ `public` Lockdown

Organize multi-domain or multi-tenant databases cleanly using PostgreSQL Schemas, understand `search_path` object resolution, and enforce `CREATE SCHEMA ... AUTHORIZATION` least-privilege rules.

Objectives: ✓ PostgreSQL Schemas & `search_path` — Namespace Resolution, PG 15+ `public` Hardening & Multi-Tenancy — How `Cluster -> Database -> Schema -> Table` hierarchy allows cross-schema Foreign Keys and joins while `search_path` controls unqualified table/function resolution. • ✓ PostgreSQL `CREATE SCHEMA` — `AUTHORIZATION`, Inline Atomic DDL & Default Privileges — Provisioning role-owned schemas (`CREATE SCHEMA IF NOT EXISTS analytics AUTHORIZATION analytics_owner`) and configuring `ALTER DEFAULT PRIVILEGES` for read-only reporting roles. • ✓ Build, verify, and ship Module Capstone: Multi-Domain Schema Isolation (`billing` + `catalog`) with Explicit Cross-Schema Foreign Keys.

4.1 PostgreSQL Schemas & `search_path` — Namespace Resolution, PG 15+ `public` Hardening & Multi-Tenancy [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Database -> Schema -> Table namespace hierarchy & Search_path ordered resolution ($user, public) • Core Syntax, CLI & Production APIs: SHOW search_path; REVOKE CREATE ON SCHEMA public FROM PUBLIC; ALTER ROLE app SET search_path... • Deep-Dive Mechanics: PostgreSQL 15+ REVOKE CREATE ON SCHEMA public & USAGE vs CREATE schema privileges • Advanced Patterns: SECURITY DEFINER search_path pinning & Zero-overhead cross-schema Foreign Keys & JOINs • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Cross-Schema Foreign Key & Explicit `search_path` Resolution • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL Schemas

Purpose: To understand and apply PostgreSQL Schemas & `search_path` — Namespace Resolution, PG 15+ `public` Hardening & Multi-Tenancy in real PostgreSQL systems — specifically: Ho...

What Topic Is Next → Next Topic 4.2: PostgreSQL `CREATE SCHEMA` — `AUTHORIZATION`, Inline Atomic DDL & Default Privileges

4.2 PostgreSQL `CREATE SCHEMA` — `AUTHORIZATION`, Inline Atomic DDL & Default Privileges [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: CREATE SCHEMA IF NOT EXISTS idempotency & AUTHORIZATION owner assignment • Core Syntax, CLI & Production APIs: CREATE SCHEMA IF NOT EXISTS analytics AUTHORIZATION app_owner; ALTER DEFAULT PRIVILEGES IN S... • Deep-Dive Mechanics: Inline CREATE SCHEMA name CREATE TABLE ... syntax & ALTER DEFAULT PRIVILEGES for future tables • Advanced Patterns: DROP SCHEMA ... RESTRICT vs CASCADE safety & Pg_namespace system catalog inspection • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Idempotent `CREATE SCHEMA` & `pg_namespace` Verification • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL `CREATE SCHEMA`

Purpose: To understand and apply PostgreSQL `CREATE SCHEMA` — `AUTHORIZATION`, Inline Atomic DDL & Default Privileges in real PostgreSQL systems — specifically: Provisioning role-...

What Topic Is Next → Module 4 Capstone (Multi-Domain Schema Isolation (`billing` + `catalog`) with Explicit Cross-Schema Foreign Keys) → Next Topic 5.1: PostgreSQL Data Types Overview — Rich Native Types (`inet`, Ranges, Arrays), Casting (`::`) & Column Alignment

Module 4 Hands-On Capstone Project (100 Marks): Multi-Domain Schema Isolation (`billing` + `catalog`) with Explicit Cross-Schema Foreign Keys — Create isolated `catalog` and `billing` schemas within a single database, link `billing.invoices` to `catalog.courses` via schema-qualified Foreign Keys, and lock down `search_path`.

MODULE 5 OF 16 • STAGE 2 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Type System Architecture & Exact Financial Arithmetic (`NUMERIC`)

Master PostgreSQL's rich native type catalog (column alignment padding, domains, arrays, ranges) and prevent IEEE-754 floating-point rounding bugs in financial ledgers using exact `NUMERIC(p, s)`.

Objectives: ✓ PostgreSQL Data Types Overview — Rich Native Types (`inet`, Ranges, Arrays), Casting (`::`) & Column Alignment — Choosing expressive self-validating PostgreSQL types (`inet` IP subnets, `tstzrange` exclusion constraints, `pg_typeof()`, `CAST` / `::`) and minimizing 8-byte struct alignment padding. • ✓ PostgreSQL `NUMERIC` / `DECIMAL` — Exact Base-10 Arithmetic vs IEEE-754 Float Drift, Scale & `NaN` — Why financial ledgers, tax engines, and token balances require exact `NUMERIC(precision, scale)` (synonym `DECIMAL`) over inexact binary `DOUBLE PRECISION` (`0.1::float8 + 0.2::float8 <> 0.3`). • ✓ Build, verify, and ship Module Capstone: IEEE-754 Float Drift Proof & Exact `NUMERIC(14, 2)` Double-Entry Wallet Ledger.

5.1 PostgreSQL Data Types Overview — Rich Native Types (`inet`, Ranges, Arrays), Casting (`::`) & Column Alignment [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Rich native types (inet, cidr, macaddr, tstzrange) & :: PostgreSQL cast vs ANSI CAST(x AS type) • Core Syntax, CLI & Production APIs: SELECT pg_typeof(val), '10.20.1.5/24'::inet << '10.20.0.0/16'::cidr, pg_column_size(row(...)... • Deep-Dive Mechanics: 8-byte / 4-byte / 1-byte struct alignment padding & Pg_typeof() and pg_column_size() inspection • Advanced Patterns: CREATE DOMAIN reusable constrained scalar types & Exclusion constraints with range types (EXCLUDE USING gist) • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Native `inet`/`cidr` Subnet Containment & Tuple Padding Comparison • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL Data Types Overview

Purpose: To understand and apply PostgreSQL Data Types Overview — Rich Native Types (`inet`, Ranges, Arrays), Casting (`::`) & Column Alignment in real PostgreSQL systems — specif...

What Topic Is Next → Next Topic 5.2: PostgreSQL `NUMERIC` / `DECIMAL` — Exact Base-10 Arithmetic vs IEEE-754 Float Drift, Scale & `NaN`

5.2 PostgreSQL `NUMERIC` / `DECIMAL` — Exact Base-10 Arithmetic vs IEEE-754 Float Drift, Scale & `NaN` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: NUMERIC vs DECIMAL exact equivalence & IEEE-754 0.1 + 0.2 binary float rounding bug • Core Syntax, CLI & Production APIs: amount NUMERIC(14, 2) NOT NULL CHECK (amount >= 0), ROUND(amount * 0.18, 2) • Deep-Dive Mechanics: Precision (total digits) vs scale (fractional digits) & PostgreSQL NUMERIC NaN > 0 quirk (CHECK val <> 'NaN') • Advanced Patterns: Why NUMERIC beats locale-bound MONEY type & Unconstrained NUMERIC vs NUMERIC(14, 2) • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Float8 Rounding Failure vs Exact `NUMERIC(12, 2)` Ledger Proof • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL `NUMERIC` / `DECIMAL`

Purpose: To understand and apply PostgreSQL `NUMERIC` / `DECIMAL` — Exact Base-10 Arithmetic vs IEEE-754 Float Drift, Scale & `NaN` in real PostgreSQL systems — specifically: Why...

What Topic Is Next → Module 5 Capstone (IEEE-754 Float Drift Proof & Exact `NUMERIC(14, 2)` Double-Entry Wallet Ledger) → Next Topic 6.1: PostgreSQL `VARCHAR(n)` vs `CHAR(n)` vs `TEXT` — `varlena` Internals, Character Count & Zero-Lock Limits

Module 5 Hands-On Capstone Project (100 Marks): IEEE-754 Float Drift Proof & Exact `NUMERIC(14, 2)` Double-Entry Wallet Ledger — Demonstrate binary `FLOAT8` (`0.1 + 0.2`) inaccuracy versus exact `NUMERIC` arithmetic and build a `CHECK (balance >= 0)` wallet ledger calculating inclusive/exclusive

GST taxes.

MODULE 6 OF 16 • STAGE 2 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Character Storage Internals (`VARCHAR` vs `CHAR` vs `TEXT`) & Three-Valued `BOOLEAN` Logic

Understand why PostgreSQL `VARCHAR(n)` and `TEXT` share identical `varlena` performance (while `CHAR(n)` wastes space!), and avoid catastrophic `NOT IN (NULL)` bugs with 3-Valued Boolean Logic (`IS DISTINCT FROM`).

Objectives: ✓ PostgreSQL `VARCHAR(n)` vs `CHAR(n)` vs `TEXT` — `varlena` Internals, Character Count & Zero-Lock Limits — Why PostgreSQL stores `VARCHAR(n)`, `VARCHAR`, and `TEXT` using the exact same `varlena` C structure (counting Unicode characters, not bytes!), and why `CHAR(n)` wastes space with trailing spaces. • ✓ PostgreSQL `BOOLEAN` — 1-Byte Storage, Three-Valued Logic (`TRUE`/`FALSE`/`NULL`) & `IS DISTINCT FROM` — Mastering 1-byte SQL `BOOLEAN` columns, `FILTER (WHERE ...)` aggregates, and avoiding silent data-loss bugs from `NULL` in `<>`, `=`, and `NOT IN` clauses. • ✓ Build, verify, and ship Module Capstone: Zero-Lock `TEXT` Constraint Validator & Null-Safe Boolean Filter Engine.

6.1 PostgreSQL `VARCHAR(n)` vs `CHAR(n)` vs `TEXT` — `varlena` Internals, Character Count & Zero-Lock Limits

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Shared varlena storage across VARCHAR and TEXT & Why CHAR(n) blank-padding is an anti-pattern in PG • Core Syntax, CLI & Production APIs: email VARCHAR(254) NOT NULL /* or TEXT CHECK (char_length(email) <= 254) */ • Deep-Dive Mechanics: Unicode character limit vs UTF-8 byte length & Zero-rewrite ALTER TABLE widening VARCHAR(50)->VARCHAR(100) • Advanced Patterns: TEXT + CHECK (...) NOT VALID online length cap & 1-byte short varlena header for strings <= 126 bytes • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Unicode Character Length vs UTF-8 Byte Size & `CHAR(n)` Waste Proof • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL `VARCHAR(n)` vs `CHAR(n)` vs `TEXT`

Purpose: To understand and apply PostgreSQL `VARCHAR(n)` vs `CHAR(n)` vs `TEXT` — `varlena` Internals, Character Count & Zero-Lock Limits in real PostgreSQL systems — specifically...

What Topic Is Next → Next Topic 6.2: PostgreSQL `BOOLEAN` — 1-Byte Storage, Three-Valued Logic (`TRUE`/`FALSE`/`NULL`) & `IS DISTINCT FROM`

6.2 PostgreSQL `BOOLEAN` — 1-Byte Storage, Three-Valued Logic (`TRUE`/`FALSE`/`NULL`) & `IS DISTINCT FROM`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: 1-byte native BOOLEAN (NOT NULL DEFAULT false) & Three-Valued Logic (TRUE, FALSE, UNKNOWN/NULL) • Core Syntax, CLI & Production APIs: is_verified BOOLEAN NOT NULL DEFAULT false, WHERE flag IS NOT TRUE, COUNT(*) FILTER (WHERE a... • Deep-Dive Mechanics: IS TRUE / IS NOT TRUE null-safe predicates & IS DISTINCT FROM null-safe inequality operator • Advanced Patterns: COUNT(*) FILTER (WHERE cond) clean aggregation & Partial Index over full B-Tree on boolean flags • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Three-Valued `BOOLEAN` Trap (`<> true` vs `IS NOT TRUE`) & `FILTER` • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL `BOOLEAN`

Purpose: To understand and apply PostgreSQL `BOOLEAN` — 1-Byte Storage, Three-Valued Logic (`TRUE`/`FALSE`/`NULL`) & `IS DISTINCT FROM` in real PostgreSQL systems — specifically:...

What Topic Is Next → Module 6 Capstone (Zero-Lock `TEXT` Constraint Validator & Null-Safe Boolean Filter Engine) → Next Topic 7.1: PostgreSQL `BIGINT` (`int8`) — 64-Bit Range, Preventing `2.14B` `INT4` Exhaustion & FK Type Matching

Module 6 Hands-On Capstone Project (100 Marks): Zero-Lock `TEXT` Constraint Validator & Null-Safe Boolean Filter Engine — Demonstrate PostgreSQL's 3-valued boolean trap where `WHERE is_banned <> true` silently drops `NULL` rows (fixed via `IS NOT TRUE`) and `NOT IN (SELECT nullable_col ...)` returns 0 rows (fix...

MODULE 7 OF 16 • STAGE 2 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: 64-Bit Primary Keys (`BIGINT`) & SQL-Standard `GENERATED AS IDENTITY` vs Legacy `SERIAL`

Prevent 2.14-billion `INT4` primary key exhaustion outages by defaulting to `BIGINT` (`int8`), and replace legacy `SERIAL` macros with SQL-standard `GENERATED ALWAYS AS IDENTITY`.

Objectives: ✓ PostgreSQL `BIGINT` (`int8`) — 64-Bit Range, Preventing `2.14B` `INT4` Exhaustion & FK Type Matching — Why high-growth events, logs, and primary keys standardize on 8-byte signed `BIGINT` (`-9.22e18` to `+9.22e18`) to eliminate catastrophic 32-bit `INTEGER` (`2,147,483,647`) exhaustion. • ✓ PostgreSQL `SERIAL` vs `GENERATED ALWAYS AS IDENTITY` — Sequence Mechanics, Gaps & `setval` Desync — Understanding how legacy `SERIAL` expands into `INTEGER + CREATE SEQUENCE + DEFAULT nextval(...)`, why explicit ID inserts desync `SERIAL` sequences, and why PostgreSQL 10+ recommends `GENERATED ALWAYS AS IDENTITY`. • ✓ Build, verify, and ship Module Capstone: Overflow-Proof `BIGINT GENERATED ALWAYS AS IDENTITY` Order Sequence & Gap Behaviour Lab.

7.1 PostgreSQL `BIGINT` (`int8`) — 64-Bit Range, Preventing `2.14B` `INT4` Exhaustion & FK Type Matching

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: 8-byte signed BIGINT (9.22 quintillion max) & 32-bit INTEGER 2.14B overflow outage hazard • Core Syntax, CLI & Production APIs: id BIGINT PRIMARY KEY, user_id BIGINT NOT NULL REFERENCES users(id) • Deep-Dive Mechanics: Upsert ON CONFLICT burning sequence numbers & Strict BIGINT FK == BIGINT PK type matching • Advanced Patterns: Integer division truncation vs ::numeric cast & 64-bit native CPU register speed • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: 32-Bit `INT4` Ceiling vs 64-Bit `BIGINT` Headroom Verification • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL `BIGINT` (`int8`)

Purpose: To understand and apply PostgreSQL `BIGINT` (`int8`) — 64-Bit Range, Preventing `2.14B` `INT4` Exhaustion & FK Type Matching in real PostgreSQL systems — specifically: Wh...

What Topic Is Next → Next Topic 7.2: PostgreSQL `SERIAL` vs `GENERATED ALWAYS AS IDENTITY` — Sequence Mechanics, Gaps & `setval` Desync

7.2 PostgreSQL `SERIAL` vs `GENERATED ALWAYS AS IDENTITY` — Sequence Mechanics, Gaps & `setval` Desync [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: GENERATED ALWAYS AS IDENTITY vs legacy SERIAL & SERIAL explicit-insert sequence desync bug • Core Syntax, CLI & Production APIs: id BIGINT GENERATED ALWAYS AS IDENTITY PRIMARY KEY /* modern replacement for BIGSERIAL */ • Deep-Dive Mechanics: OVERRIDING SYSTEM VALUE explicit seed escape & Sequences are non-transactional (gaps on ROLLBACK) • Advanced Patterns: Pg_get_serial_sequence() + setval() repair & SEQUENCE CACHE n high-concurrency tuning • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: SQL-Standard `GENERATED ALWAYS AS IDENTITY` & Non-Rollback Sequence Proof • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL `SERIAL` vs `GENERATED ALWAYS AS IDENTITY...

Purpose: To understand and apply PostgreSQL `SERIAL` vs `GENERATED ALWAYS AS IDENTITY` — Sequence Mechanics, Gaps & `setval` Desync in real PostgreSQL systems — specifically: Unde...

What Topic Is Next → Module 7 Capstone (Overflow-Proof `BIGINT GENERATED ALWAYS AS IDENTITY` Order Sequence & Gap Behaviour Lab) → Next Topic 8.1: PostgreSQL `TIMESTAMPTZ` vs `TIMESTAMP` — 8-Byte UTC Microseconds, `AT TIME ZONE` & SARGable Ranges

Module 7 Hands-On Capstone Project (100 Marks): Overflow-Proof `BIGINT GENERATED ALWAYS AS IDENTITY` Order Sequence & Gap Behaviour Lab — Build an order ledger using `BIGINT GENERATED ALWAYS AS IDENTITY` that rejects accidental manual ID overrides (unless `OVERRIDING SYSTEM VALUE` is explicit) and demonstrates why rolled-back...

MODULE 8 OF 16 • STAGE 2 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: Timezone-Safe Temporal Engineering (`TIMESTAMPTZ` & `INTERVAL`)

Store instants unambiguously in UTC microseconds with `TIMESTAMPTZ` (never naive `TIMESTAMP`!), preserve B-Tree index SARGability on date ranges, and compute calendar-aware `INTERVAL` math.

Objectives: ✓ PostgreSQL `TIMESTAMPTZ` vs `TIMESTAMP` — 8-Byte UTC Microseconds, `AT TIME ZONE` & SARGable Ranges — Why PostgreSQL's "Don't Do This" wiki explicitly warns against naive `TIMESTAMP` (`without time zone`): `TIMESTAMPTZ` takes the exact same 8 bytes, stores pure UTC, and formats per session `TimeZone`. • ✓ PostgreSQL `INTERVAL` — 16-Byte Calendar/Day/Microsecond Triplet, DST Awareness & `justify_interval` — Performing precise SLA, retention, and billing-cycle date math using PostgreSQL's 16-byte `INTERVAL` (`months` + `days` + `microseconds`), `AGE()`, `EXTRACT(EPOCH FROM ...)`, and `make_interval()`. • ✓ Build, verify, and ship Module Capstone: DST-Safe Cohort Subscription Scheduler (`TIMESTAMPTZ` + `INTERVAL` + SARGable Ranges).

8.1 PostgreSQL `TIMESTAMPTZ` vs `TIMESTAMP` — 8-Byte UTC Microseconds, `AT TIME ZONE` & SARGable Ranges [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: TIMESTAMPTZ 8-byte UTC storage internals & Why naive TIMESTAMP silently drops offsets • Core Syntax, CLI & Production APIs: created_at TIMESTAMPTZ NOT NULL DEFAULT now() /* WHERE created_at >= $1 AND created_at < $2... • Deep-Dive Mechanics: Now() (transaction start) vs clock_timestamp() & SARGable half-open [>= start, < end) index scan • Advanced Patterns: AT TIME ZONE IANA conversion (Asia/Kolkata) & Date_trunc() time-bucket aggregation • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: UTC Normalization & IANA `Asia/Kolkata` Conversion with `TIMESTAMPTZ` • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL `TIMESTAMPTZ` vs `TIMESTAMP`

Purpose: To understand and apply PostgreSQL `TIMESTAMPTZ` vs `TIMESTAMP` — 8-Byte UTC Microseconds, `AT TIME ZONE` & SARGable Ranges in real PostgreSQL systems — specifically: Why...

What Topic Is Next → Next Topic 8.2: PostgreSQL `INTERVAL` — 16-Byte Calendar/Day/Microsecond Triplet, DST Awareness & `justify_interval`

8.2 PostgreSQL `INTERVAL` — 16-Byte Calendar/Day/Microsecond Triplet, DST Awareness & `justify_interval` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: 16-byte (months, days, microseconds) triplet & INTERVAL '1 month' end-of-month clamping • Core Syntax, CLI & Production APIs: expires_at = enrolled_at + INTERVAL '10 weeks', EXTRACT(EPOCH FROM (ended_at - started_at)) • Deep-Dive Mechanics: INTERVAL '1 day' vs '24 hours' across DST & Make_interval(days => $1) parameterized safety • Advanced Patterns: EXTRACT(EPOCH FROM interval) total seconds & Justify_days / justify_hours / justify_interval • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Calendar Month Clamping & Parameter-Safe `make_interval` Math • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL `INTERVAL`

Purpose: To understand and apply PostgreSQL `INTERVAL` — 16-Byte Calendar/Day/Microsecond Triplet, DST Awareness & `justify_interval` in real PostgreSQL systems — specifically: Pe...

What Topic Is Next → Module 8 Capstone (DST-Safe Cohort Subscription Scheduler (`TIMESTAMPTZ` + `INTERVAL` + SARGable Ranges)) → Next Topic 9.1: PostgreSQL `UUID` — 16-Byte Binary Storage, `gen_random_uuid()`, & `UUIDv4` B-Tree Fragmentation vs `UUIDv7`

Module 8 Hands-On Capstone Project (100 Marks): DST-Safe Cohort Subscription Scheduler (`TIMESTAMPTZ` + `INTERVAL` + SARGable Ranges) — Write a timezone-safe PostgreSQL billing query computing `Asia/Kolkata` and `America/New_York` views over UTC `TIMESTAMPTZ` instants using SARGable range bounds and calendar `INTERVAL` renew...

MODULE 9 OF 16 • STAGE 3 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: Native 16-Byte `UUID` Keys (`UUIDv4` vs `UUIDv7`) & Binary `JSONB` + GIN Indexing

Store compact 128-bit `UUID` identifiers natively (understanding random `UUIDv4` B-Tree page-split bloat vs time-ordered `UUIDv7`!), and accelerate semi-structured `JSONB` queries by 1000x with GIN `@>` indexes.

Objectives: ✓ PostgreSQL `UUID` — 16-Byte Binary Storage, `gen_random_uuid()`, & `UUIDv4` B-Tree Fragmentation vs `UUIDv7` — Generating collision-free distributed identifiers without leaking sequential row counts, storing them in 16 binary bytes (not 36 ASCII chars), and avoiding random `UUIDv4` B-Tree page-split write amplification on massive tables. • ✓ PostgreSQL `JSON` vs `JSONB` — Decomposed Binary Storage, `->` vs `->>`, Containment `@>` & GIN Indexes — Combining relational ACID rigor with schemaless flexibility using binary `JSONB`, containment (`@>`), key-existence (`?`), `jsonb_set()`, SQL/JSON path (`@?`), and `GIN (col jsonb_path_ops)` indexing. • ✓ Build, verify, and ship Module Capstone: Native `UUID` Event Store with GIN-Indexed `JSONB` Containment

(`@>`) Search.

9.1 PostgreSQL `UUID` — 16-Byte Binary Storage, `gen_random_uuid()`, & `UUIDv4` B-Tree Fragmentation vs `UUIDv7` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: 16-byte native binary UUID vs 36-byte VARCHAR & Built-in gen_random_uuid() (PG 13+ core) • Core Syntax, CLI & Production APIs: public_id UUID NOT NULL DEFAULT gen_random_uuid() /* 16 bytes binary */ • Deep-Dive Mechanics: UUIDv4 random B-Tree page splits & WAL bloat & UUIDv7 48-bit millisecond prefix sequential locality • Advanced Patterns: Internal BIGINT PK + external UUID unique pattern & Preventing IDOR enumeration attacks on public URLs • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: 16-Byte Binary `UUID` vs `TEXT` Footprint & Version Check • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL `UUID`

Purpose: To understand and apply PostgreSQL `UUID` — 16-Byte Binary Storage, `gen_random_uuid()`, & `UUIDv4` B-Tree Fragmentation vs `UUIDv7` in real PostgreSQL systems — specific...

What Topic Is Next → Next Topic 9.2: PostgreSQL `JSON` vs `JSONB` — Decomposed Binary Storage, `->` vs `->>`, Containment `@>` & GIN Indexes

9.2 PostgreSQL `JSON` vs `JSONB` — Decomposed Binary Storage, `->` vs `->>`, Containment `@>` & GIN Indexes [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: JSON (raw text) vs JSONB (decomposed binary) & -> (returns JSONB) vs ->> (returns TEXT) • Core Syntax, CLI & Production APIs: CREATE INDEX idx_meta_gin ON events USING gin (meta jsonb_path_ops); SELECT ... WHERE meta @... • Deep-Dive Mechanics: @> containment operator + GIN index synergy & GIN jsonb_ops vs compact jsonb_path_ops • Advanced Patterns: Jsonb_set() & - key deletion immutability & JSON null ('null'::jsonb) vs SQL NULL distinction • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: GIN-Ready `JSONB` Containment (`@>`), `->>` Extraction & `jsonb_set` • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL `JSON` vs `JSONB`

Purpose: To understand and apply PostgreSQL `JSON` vs `JSONB` — Decomposed Binary Storage, `->` vs `->>`, Containment `@>` & GIN Indexes in real PostgreSQL systems — specifically:...

What Topic Is Next → Module 9 Capstone (Native `UUID` Event Store with GIN-Indexed `JSONB` Containment (`@>`) Search) → Next Topic 10.1: PostgreSQL `TEXT` — TOAST Internals (`lz4` Compression, Out-of-Line Chunks), `SELECT *` Cost & Full-Text Search

Module 9 Hands-On Capstone Project (100 Marks): Native `UUID` Event Store with GIN-Indexed `JSONB` Containment (`@>`) Search — Create an event store keyed by native 16-byte `UUID` (`DEFAULT gen_random_uuid()`) with a `JSONB` metadata payload indexed via `GIN (meta jsonb_path_ops)` and queried cleanly using `@>` and...

MODULE 10 OF 16 • STAGE 3 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Large `TEXT` TOAST Compression / Full-Text Search & Zero-Accident `ALTER SCHEMA` Governance

Understand how PostgreSQL stores values up to 1 GB inside 8 KB pages via TOAST (`pglz` / `lz4`), accelerate search with `tsvector` + `pg_trgm` GIN indexes, and govern `ALTER SCHEMA` safely without breaking `SECURITY DEFINER` / view dependencies.

Objectives: ✓ PostgreSQL `TEXT` — TOAST Internals (`lz4` Compression, Out-of-Line Chunks), `SELECT *` Cost & Full-Text Search — How PostgreSQL stores `TEXT` strings up to 1 GB transparently across 8 KB heap pages via TOAST (`COMPRESSION lz4`), why `SELECT id, title` avoids detoasting out-of-line chunks, and indexing text via `tsvector` / `pg_trgm`. • ✓ PostgreSQL `ALTER SCHEMA` — `RENAME TO`, `OWNER TO`, OID Dependency Tracking & Zero-Downtime Cutovers — Managing schema lifecycle changes (`ALTER SCHEMA staging RENAME TO v2_live; ALTER SCHEMA v2_live OWNER TO ...`) safely inside atomic transactions while understanding catalog OID binding vs string-based client references. • ✓ Build, verify, and ship Module Capstone: Full-Text `tsvector` Search Engine & Schema Ownership / Rename Migration Script.

10.1 PostgreSQL `TEXT` — TOAST Internals (`lz4` Compression, Out-of-Line Chunks), `SELECT *` Cost & Full-Text Search [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: TOAST (~2 KB threshold, 1 GB max value) & COMPRESSION lz4 vs default pglz speed • Core Syntax, CLI & Production APIs: body TEXT COMPRESSION lz4, search_vec TSVECTOR GENERATED ALWAYS AS (to_tsvector('english', b... • Deep-Dive Mechanics: Avoiding SELECT * detoasting I/O penalty & GENERATED ALWAYS AS (to_tsvector(...)) STORED • Advanced Patterns: Websearch_to_tsquery @@ GIN full-text search & Pg_trgm GIN index for fast ILIKE %substring% • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Generated `tsvector` Full-Text Search with `websearch_to_tsquery` • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL `TEXT`

Purpose: To understand and apply PostgreSQL `TEXT` — TOAST Internals (`lz4` Compression, Out-of-Line Chunks), `SELECT *` Cost & Full-Text Search in real PostgreSQL systems — speci...

What Topic Is Next → Next Topic 10.2: PostgreSQL `ALTER SCHEMA` — `RENAME TO`, `OWNER TO`, OID Dependency Tracking & Zero-Downtime Cutovers

10.2 PostgreSQL `ALTER SCHEMA` — `RENAME TO`, `OWNER TO`, OID Dependency Tracking & Zero-Downtime Cutovers [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: ALTER SCHEMA ... RENAME TO atomic swap & ALTER SCHEMA ... OWNER TO privilege governance • Core Syntax, CLI & Production APIs: BEGIN; ALTER SCHEMA current_api RENAME TO archive_api; ALTER SCHEMA next_api RENAME TO curre... • Deep-Dive Mechanics: OID-bound Views/FKs vs name-bound client SQL & Atomic blue/green schema cutover in BEGIN...COMMIT • Advanced Patterns: Pg_depend catalog dependency inspection & Updating role search_path alongside schema rename • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Atomic Blue/Green `ALTER SCHEMA ... RENAME TO` Swap & View OID Survival • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL `ALTER SCHEMA`

Purpose: To understand and apply PostgreSQL `ALTER SCHEMA` — `RENAME TO`, `OWNER TO`, OID Dependency Tracking & Zero-Downtime Cutovers in real PostgreSQL systems — specifically: M...

What Topic Is Next → Module 10 Capstone (Full-Text `tsvector` Search Engine & Schema Ownership / Rename Migration Script) → Next Topic 11.1: Tuple Visibility (xmin/xmax), Dead Tuples, HOT Updates & Table Bloat

Module 10 Hands-On Capstone Project (100 Marks): Full-Text `tsvector` Search Engine & Schema Ownership / Rename Migration Script — Build a PostgreSQL 17 documentation search table using a stored generated `tsvector` column (`GENERATED ALWAYS AS (to_tsvector('english', ...)) STORED`) and execute an atomic transactional `...

MODULE 11 OF 16 • STAGE 3 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

MVCC Heap Internals, VACUUM/Autovacuum Tuning & Lock Contention

Production-grade PostgreSQL Training engineering module covering MVCC Heap Internals, VACUUM/Autovacuum Tuning & Lock Contention with hands-on labs and capstone build.

Objectives: ✓ Tuple Visibility (xmin/xmax), Dead Tuples, HOT Updates & Table Bloat — Purpose: Design update patterns and fillfactor settings that keep PostgreSQL heaps lean and fast. • ✓ Tuning Per-Table Autovacuum & Preventing Transaction ID Wraparound — Purpose: Ensure high-churn queue/session tables vacuum continuously without emergency downtime. • ✓ Lock Modes, lock_timeout Guards & Zero-Downtime DDL Habits — Purpose: Add columns, NOT NULL check constraints, and CREATE INDEX CONCURRENTLY without locking traffic!. • ✓ SKIP LOCKED Transactional Queues & Advisory Locks — Purpose: Build rock-solid concurrent job claimers right inside PostgreSQL.

11.1 Tuple Visibility (xmin/xmax), Dead Tuples, HOT Updates & Table Bloat

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Dead Tuples • Integration with Tuning Per • Live Debugging, Telemetry & Technical Interview Trade • What Is Tuple Visibility (xmin/xmax), Dead Tuples, HOT Updates & Table Bloat & Why Industry Teams Use It • In PostgreSQL Training, Tuple Visibility (xmin/xmax), Dead Tuples, HOT Updates & Table Bloat exists for one clear r... • Even if you come from a non • Runnable Code Implementation (SQL)

Purpose: To understand and apply Tuple Visibility (xmin/xmax), Dead Tuples, HOT Updates & Table Bloat in real PostgreSQL systems — specifically: Design update patterns and fillfac...

What Topic Is Next → Next Topic 11.2: Tuning Per-Table Autovacuum & Preventing Transaction ID Wraparound

11.2 Tuning Per-Table Autovacuum & Preventing Transaction ID Wraparound

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Preventing Transaction ID Wraparound • Integration with Lock Modes, lock_timeout Guards & Zero • Live Debugging, Telemetry & Technical Interview Trade • What Is Tuning Per • In PostgreSQL Training, Tuning Per • Even if you come from a non • Runnable Code Implementation (SQL)

Purpose: To understand and apply Tuning Per-Table Autovacuum & Preventing Transaction ID Wraparound in real PostgreSQL systems — specifically: Ensure high-churn queue/session tabl...

What Topic Is Next → Next Topic 11.3: Lock Modes, lock_timeout Guards & Zero-Downtime DDL Habits

11.3 Lock Modes, lock_timeout Guards & Zero-Downtime DDL Habits

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of lock_timeout Guards • Integration with SKIP LOCKED Transactional Queues & Advisory Locks • Live Debugging, Telemetry & Technical Interview Trade • What Is Lock Modes, lock_timeout Guards & Zero • In PostgreSQL Training, Lock Modes, lock_timeout Guards & Zero • Even if you come from a non • Runnable Code Implementation (SQL)

Purpose: To understand and apply Lock Modes, lock_timeout Guards & Zero-Downtime DDL Habits in real PostgreSQL systems — specifically: Add columns, NOT NULL check constraints, and...

What Topic Is Next → Next Topic 11.4: SKIP LOCKED Transactional Queues & Advisory Locks

11.4 SKIP LOCKED Transactional Queues & Advisory Locks

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Advisory Locks • Integration with Tuple Visibility (xmin/xmax), Dead Tuples, HOT Updates & Table Bloat • Live Debugging, Telemetry & Technical Interview Trade • What Is SKIP LOCKED Transactional Queues & Advisory Locks & Why Industry Teams Use It • In PostgreSQL Training, SKIP LOCKED Transactional Queues & Advisory Locks exists for one clear real • Even if you come from a non • Runnable Code Implementation (SQL)

Purpose: To understand and apply SKIP LOCKED Transactional Queues & Advisory Locks in real PostgreSQL systems — specifically: Build rock-solid concurrent job claimers right inside...

What Topic Is Next → Module 11 Capstone (Zero-Bloat High-Write Financial Ledger on PostgreSQL 16) → Next Topic 12.1: Reading EXPLAIN (ANALYZE, BUFFERS): Seq/Bitmap/Index-Only Scans & Sort Spills

Module 11 Hands-On Capstone Project (100 Marks): Zero-Bloat High-Write Financial Ledger on PostgreSQL 16 — Design, implement, test, and document 'Zero-Bloat High-Write Financial Ledger on PostgreSQL 16' applying every production concept from Module 11 (MVCC Heap Internals, VACUUM/Autovacuum Tunin...

MODULE 12 OF 16 • STAGE 3 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Advanced Indexing (B-Tree, GIN, GiST, BRIN) & EXPLAIN BUFFERS Mastery

Production-grade PostgreSQL Training engineering module covering Advanced Indexing (B-Tree, GIN, GiST, BRIN) & EXPLAIN BUFFERS Mastery with hands-on labs and capstone build.

Objectives: ✓ Reading EXPLAIN (ANALYZE, BUFFERS): Seq/Bitmap/Index-Only Scans & Sort Spills — Purpose: Spot disk work_mem spills and stale statistics immediately from execution plans. • ✓ Partial Indexes, Expression Indexes & Covering INCLUDE Columns — Purpose: Index only the 2% active unpaid orders for tiny RAM-resident sub-millisecond lookups. • ✓ JSONB Containment (@>), jsonb_path_ops GIN Indexes & Full-Text tsvector — Purpose: Query flexible attributes and ranked full-text search without adding separate search infra. • ✓ BRIN Indexes for Billion-Row Append-Only Logs & pg_stat_statements Audits — Purpose: Rank the top 10 DB CPU-consuming queries in production and fix them systematically.

12.1 Reading EXPLAIN (ANALYZE, BUFFERS): Seq/Bitmap/Index-Only Scans & Sort Spills [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of BUFFERS) • Integration with Partial Indexes, Expression Indexes & Covering INCLUDE Columns • Live Debugging, Telemetry & Technical Interview Trade • What Is Reading EXPLAIN (ANALYZE, BUFFERS) • In PostgreSQL Training, Reading EXPLAIN (ANALYZE, BUFFERS) • Even if you come from a non • Runnable Code Implementation (SQL)

Purpose: To understand and apply Reading EXPLAIN (ANALYZE, BUFFERS): Seq/Bitmap/Index-Only Scans & Sort Spills in real PostgreSQL systems — specifically: Spot disk work_mem spills...

What Topic Is Next → Next Topic 12.2: Partial Indexes, Expression Indexes & Covering INCLUDE Columns

12.2 Partial Indexes, Expression Indexes & Covering INCLUDE Columns [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Expression Indexes • Integration with JSONB Containment (@>), jsonb_path_ops GIN Indexes & Full • Live Debugging, Telemetry & Technical Interview Trade • What Is Partial Indexes, Expression Indexes & Covering INCLUDE Columns & Why Industry Teams Use It • In PostgreSQL Training, Partial Indexes, Expression Indexes & Covering INCLUDE Columns exists for one clear real • Even if you come from a non • Runnable Code Implementation (SQL)

Purpose: To understand and apply Partial Indexes, Expression Indexes & Covering INCLUDE Columns in real PostgreSQL systems — specifically: Index only the 2% active unpaid orders f...

What Topic Is Next → Next Topic 12.3: JSONB Containment (@>), jsonb_path_ops GIN Indexes & Full-Text tsvector

12.3 JSONB Containment (@>), jsonb_path_ops GIN Indexes & Full-Text tsvector [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of jsonb_path_ops GIN Indexes • Integration with BRIN Indexes for Billion • Live Debugging, Telemetry & Technical Interview Trade • What Is JSONB Containment (@>), jsonb_path_ops GIN Indexes & Full • In PostgreSQL Training, JSONB Containment (@>), jsonb_path_ops GIN Indexes & Full • Even if you come from a non • Runnable Code Implementation (SQL)

Purpose: To understand and apply JSONB Containment (@>), jsonb_path_ops GIN Indexes & Full-Text tsvector in real PostgreSQL systems — specifically: Query flexible attributes and r...

What Topic Is Next → Next Topic 12.4: BRIN Indexes for Billion-Row Append-Only Logs & pg_stat_statements Audits

12.4 BRIN Indexes for Billion-Row Append-Only Logs & pg_stat_statements Audits [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of pg_stat_statements Audits • Integration with Reading EXPLAIN (ANALYZE, BUFFERS) • Live Debugging, Telemetry & Technical Interview Trade • What Is BRIN Indexes for Billion • In PostgreSQL Training, BRIN Indexes for Billion • Even if you come from a non • Runnable Code Implementation (SQL)

Purpose: To understand and apply BRIN Indexes for Billion-Row Append-Only Logs & pg_stat_statements Audits in real PostgreSQL systems — specifically: Rank the top 10 DB CPU-consum...

What Topic Is Next → Module 12 Capstone (100x Query Speedup Lab on 10M-Row JSONB + Relational Data) → Next Topic 13.1: Declarative Range/Hash Partitioning, Partition Pruning & Instant Detach

Module 12 Hands-On Capstone Project (100 Marks): 100x Query Speedup Lab on 10M-Row JSONB + Relational Data — Design, implement, test, and document '100x Query Speedup Lab on 10M-Row JSONB + Relational Data' applying every production concept from Module 12 (Advanced Indexing (B-Tree, GIN, GiST, BRIN...

MODULE 13 OF 16 • STAGE 4 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Declarative Partitioning, Row-Level Security (RLS) & pgvector AI Search

Production-grade PostgreSQL Training engineering module covering Declarative Partitioning, Row-Level Security (RLS) & pgvector AI Search with hands-on labs and capstone build.

Objectives: ✓ Declarative Range/Hash Partitioning, Partition Pruning & Instant Detach — Purpose: Manage 500GB+ time-series tables cleanly and drop old monthly partitions in 5ms. • ✓ Multi-Tenant Row-Level Security (RLS) Policies with Session Context — Purpose: Enforce tenant isolation inside the PostgreSQL kernel so app bugs never leak cross-tenant rows!. • ✓ Lateral Joins, Grouping Sets, Window Frames & PL/pgSQL Audit Triggers — Purpose: Compute rich analytical rollups and tamper-evident compliance audit trails. • ✓ pgvector Embeddings, HNSW Indexes & Hybrid Relational + Semantic Queries — Purpose: Power modern AI RAG retrieval directly inside ACID PostgreSQL transactions.

13.1 Declarative Range/Hash Partitioning, Partition Pruning & Instant Detach [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Partition Pruning • Integration with Multi • Live Debugging, Telemetry & Technical Interview Trade • What Is Declarative Range/Hash Partitioning, Partition Pruning & Instant Detach & Why Industry Teams Use It • In PostgreSQL Training, Declarative Range/Hash Partitioning, Partition Pruning & Instant Detach exists for one clea... • Even if you come from a non • Runnable Code Implementation (SQL)

Purpose: To understand and apply Declarative Range/Hash Partitioning, Partition Pruning & Instant Detach in real PostgreSQL systems — specifically: Manage 500GB+ time-series table...

What Topic Is Next → Next Topic 13.2: Multi-Tenant Row-Level Security (RLS) Policies with Session Context

13.2 Multi-Tenant Row-Level Security (RLS) Policies with Session Context [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with Lateral Joins, Grouping Sets, Window Frames & PL/pgSQL Audit Triggers • Live Debugging, Telemetry & Technical Interview Trade • What Is Multi • In PostgreSQL Training, Multi • Even if you come from a non • Runnable Code Implementation (SQL)

Purpose: To understand and apply Multi-Tenant Row-Level Security (RLS) Policies with Session Context in real PostgreSQL systems — specifically: Enforce tenant isolation inside the...

What Topic Is Next → Next Topic 13.3: Lateral Joins, Grouping Sets, Window Frames & PL/pgSQL Audit Triggers

13.3 Lateral Joins, Grouping Sets, Window Frames & PL/pgSQL Audit Triggers [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Grouping Sets • Integration with pgvector Embeddings, HNSW Indexes & Hybrid Relational + Semantic Queries • Live Debugging, Telemetry & Technical Interview Trade • What Is Lateral Joins, Grouping Sets, Window Frames & PL/pgSQL Audit Triggers & Why Industry Teams Use It • In PostgreSQL Training, Lateral Joins, Grouping Sets, Window Frames & PL/pgSQL Audit Triggers exists for one clear.... • Even if you come from a non • Runnable Code Implementation (SQL)

Purpose: To understand and apply Lateral Joins, Grouping Sets, Window Frames & PL/pgSQL Audit Triggers in real PostgreSQL systems — specifically: Compute rich analytical rollups a...

What Topic Is Next → Next Topic 13.4: pgvector Embeddings, HNSW Indexes & Hybrid Relational + Semantic Queries

13.4 pgvector Embeddings, HNSW Indexes & Hybrid Relational + Semantic Queries [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of HNSW Indexes • Integration with Declarative Range/Hash Partitioning, Partition Pruning & Instant Detach • Live Debugging, Telemetry & Technical Interview Trade • What Is pgvector Embeddings, HNSW Indexes & Hybrid Relational + Semantic Queries & Why Industry Teams Use It • In PostgreSQL Training, pgvector Embeddings, HNSW Indexes & Hybrid Relational + Semantic Queries exists for one cle... • Even if you come from a non • Runnable Code Implementation (SQL)

Purpose: To understand and apply pgvector Embeddings, HNSW Indexes & Hybrid Relational + Semantic Queries in real PostgreSQL systems — specifically: Power modern AI RAG retrieval...

What Topic Is Next → Module 13 Capstone (Multi-Tenant RLS-Protected SaaS & Vector Search Database) → Next Topic 14.1: Transaction-Mode Connection Pooling with PgBouncer

Module 13 Hands-On Capstone Project (100 Marks): Multi-Tenant RLS-Protected SaaS & Vector Search Database — Design, implement, test, and document 'Multi-Tenant RLS-Protected SaaS & Vector Search Database' applying every production concept from Module 13 (Declarative Partitioning, Row-Level Securit...

MODULE 14 OF 16 • STAGE 4 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Streaming Replication, PgBouncer Pooling, WAL PITR & High Availability

Production-grade PostgreSQL Training engineering module covering Streaming Replication, PgBouncer Pooling, WAL PITR & High Availability with hands-on labs and capstone build.

Objectives: ✓ Transaction-Mode Connection Pooling with PgBouncer — Purpose: Multiplex 2,000 microservice pods smoothly onto 80 physical PostgreSQL backend processes. • ✓ Physical Streaming Replication, Replication Slots & Logical CDC — Purpose: Scale read replicas safely while monitoring slot lag to prevent WAL disk overflow. • ✓ Continuous WAL Archiving (pgBackRest) & Point-in-Time Recovery (PITR) — Purpose: Practice restoring a corrupted cluster to 14:32:10 UTC with verified integrity. • ✓ SCRAM-SHA-256, SSL Enforcement, Role Hierarchies & Patroni HA Failover — Purpose: Automate sub-15-second primary failover and pass enterprise DB security audits.

14.1 Transaction-Mode Connection Pooling with PgBouncer [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with Physical Streaming Replication, Replication Slots & Logical CDC • Live Debugging, Telemetry & Technical Interview Trade • What Is Transaction • In PostgreSQL Training, Transaction • Even if you come from a non • Runnable Code Implementation (SQL)

Purpose: To understand and apply Transaction-Mode Connection Pooling with PgBouncer in real PostgreSQL systems — specifically: Multiplex 2,000 microservice pods smoothly onto 80 p...

What Topic Is Next → Next Topic 14.2: Physical Streaming Replication, Replication Slots & Logical CDC

14.2 Physical Streaming Replication, Replication Slots & Logical CDC [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Replication Slots • Integration with Continuous WAL Archiving (pgBackRest) & Point • Live Debugging, Telemetry & Technical Interview Trade • What Is Physical Streaming Replication, Replication Slots & Logical CDC & Why Industry Teams Use It • In PostgreSQL Training, Physical Streaming Replication, Replication Slots & Logical CDC exists for one clear real • Even if you come from a non • Runnable Code Implementation (SQL)

Purpose: To understand and apply Physical Streaming Replication, Replication Slots & Logical CDC in real PostgreSQL systems — specifically: Scale read replicas safely while monito...

What Topic Is Next → Next Topic 14.3: Continuous WAL Archiving (pgBackRest) & Point-in-Time Recovery (PITR)

14.3 Continuous WAL Archiving (pgBackRest) & Point-in-Time Recovery (PITR) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Point • Integration with SCRAM • Live Debugging, Telemetry & Technical Interview Trade • What Is Continuous WAL Archiving (pgBackRest) & Point • In PostgreSQL Training, Continuous WAL Archiving (pgBackRest) & Point • Even if you come from a non • Runnable Code Implementation (SQL)

Purpose: To understand and apply Continuous WAL Archiving (pgBackRest) & Point-in-Time Recovery (PITR) in real PostgreSQL systems — specifically: Practice restoring a corrupted cl...

What Topic Is Next → Next Topic 14.4: SCRAM-SHA-256, SSL Enforcement, Role Hierarchies & Patroni HA Failover

14.4 SCRAM-SHA-256, SSL Enforcement, Role Hierarchies & Patroni HA Failover [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of SSL Enforcement • Integration with Transaction • Live Debugging, Telemetry & Technical Interview Trade • What Is SCRAM • In PostgreSQL Training, SCRAM • Even if you come from a non • Runnable Code Implementation (SQL)

Purpose: To understand and apply SCRAM-SHA-256, SSL Enforcement, Role Hierarchies & Patroni HA Failover in real PostgreSQL systems — specifically: Automate sub-15-second primary f...

What Topic Is Next → Module 14 Capstone (HA Primary–Standby Cluster with PgBouncer & PITR Drills) → Next Topic 15.1: MVCC Tuple Visibility, Dead Tuples, Autovacuum Tuning & Transaction ID Wraparound Safety

Module 14 Hands-On Capstone Project (100 Marks): HA Primary–Standby Cluster with PgBouncer & PITR Drills — Design, implement, test, and document 'HA Primary–Standby Cluster with PgBouncer & PITR Drills' applying every production concept from Module 14 (Streaming Replication, PgBouncer Pooling, WA...

MODULE 15 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

PostgreSQL MVCC Internals, Autovacuum Tuning, Row-Level Security & Partitioning

Self-explanatory list-and-bullet production mastery module covering PostgreSQL MVCC Internals, Autovacuum Tuning, Row-Level Security & Partitioning in PostgreSQL Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ MVCC Tuple Visibility, Dead Tuples, Autovacuum Tuning & Transaction ID Wraparound Safety — Purpose: Keep high-write PostgreSQL tables free from table/index bloat and vacuum stalls. • ✓ PostgreSQL Row-Level Security (RLS) Policies & Zero-Downtime CREATE INDEX CONCURRENTLY — Purpose: Enforce tenant isolation directly inside Postgres and evolve live schemas without write locks. • ✓ Declarative Range/Hash Partitioning, pg_stat_statements & Connection Pooling (PgBouncer) — Purpose: Scale to thousands of concurrent app connections and isolate slow queries scientifically. • ✓ Build, verify, and document 'RLS Multi-Tenant SaaS Schema with Declarative Partitioning & Bloat Monitor' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

15.1 MVCC Tuple Visibility, Dead Tuples, Autovacuum Tuning & Transaction ID Wraparound Safety

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for MVCC Tuple Visibility, Dead Tuples, Autovacuum Tuning & Transaction ID Wra... • Performance Optimization, Complexity & Latency Impact of MVCC Tuple Visibility, Dead Tuples, Autovacuum Tuning & Tr... • Live Debugging, Stack • Runnable Code Implementation (SQL) • Hands • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for MVCC Tuple Visibility • Production Hardening & Security: Row-Level Security (RLS), SCRAM-SHA-256 roles & lock_timeout guards

Purpose: To understand and apply MVCC Tuple Visibility, Dead Tuples, Autovacuum Tuning & Transaction ID Wraparound Safety in real PostgreSQL systems — specifically: Keep high-writ...

What Topic Is Next → Next Topic 15.2: PostgreSQL Row-Level Security (RLS) Policies & Zero-Downtime CREATE INDEX CONCURRENTLY

15.2 PostgreSQL Row-Level Security (RLS) Policies & Zero-Downtime CREATE INDEX CONCURRENTLY

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for PostgreSQL Row • Performance Optimization, Complexity & Latency Impact of PostgreSQL Row • Live Debugging, Stack • Runnable Code Implementation (SQL) • Hands • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for PostgreSQL Row-Level Security (RLS) Policies • Production Hardening & Security: Row-Level Security (RLS), SCRAM-SHA-256 roles & lock_timeout guards

Purpose: To understand and apply PostgreSQL Row-Level Security (RLS) Policies & Zero-Downtime CREATE INDEX CONCURRENTLY in real PostgreSQL systems — specifically: Enforce tenant i...

What Topic Is Next → Next Topic 15.3: Declarative Range/Hash Partitioning, pg_stat_statements & Connection Pooling (PgBouncer)

15.3 Declarative Range/Hash Partitioning, pg_stat_statements & Connection Pooling (PgBouncer)

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Declarative Range/Hash Partitioning, pg_stat_statements & Connection Pooli... • Performance Optimization, Complexity & Latency Impact of Declarative Range/Hash Partitioning, pg_stat_statements &.... • Live Debugging, Stack • Runnable Code Implementation (SQL) • Hands • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for Declarative Range/Hash Partitioning • Production Hardening & Security: Row-Level Security (RLS), SCRAM-SHA-256 roles & lock_timeout guards

Purpose: To understand and apply Declarative Range/Hash Partitioning, pg_stat_statements & Connection Pooling (PgBouncer) in real PostgreSQL systems — specifically: Scale to thous...

What Topic Is Next → Module 15 Capstone (RLS Multi-Tenant SaaS Schema with Declarative Partitioning & Bloat Monitor) → Next Topic 16.1: Full-Text Search (tsvector / GIN) + pgvector HNSW Embeddings for AI/RAG Workloads

Module 15 Hands-On Capstone Project (100 Marks): RLS Multi-Tenant SaaS Schema with Declarative Partitioning & Bloat Monitor — Architect, implement, test, and document 'RLS Multi-Tenant SaaS Schema with Declarative Partitioning & Bloat Monitor' applying every bullet-wise engineering concept from Module 15 (PostgreSQ...

MODULE 16 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

pgvector Hybrid Search, Logical Replication & PostgreSQL DBA/SDE Interviews

Self-explanatory list-and-bullet production mastery module covering pgvector Hybrid Search, Logical Replication & PostgreSQL DBA/SDE Interviews in PostgreSQL Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Full-Text Search (tsvector / GIN) + pgvector HNSW Embeddings for AI/RAG Workloads — Purpose: Run relational transactions, keyword search, and vector similarity inside a single ACID database. • ✓ Streaming & Logical Replication, WAL Archiving & Point-in-Time Recovery (PITR) — Purpose: Execute zero-data-loss failover drills and major-version upgrades with minimal downtime. • ✓ Reading Complex EXPLAIN (ANALYZE, BUFFERS) Plans & Postgres Interview

Scenarios — Purpose: Diagnose Nested Loop vs Hash Join misestimates, spill-to-disk sorts, and lock queues. • ✓ Build, verify, and document 'HNSW pgvector Semantic Search + Full-Text tsvector Hybrid Query Engine' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

16.1 Full-Text Search (tsvector / GIN) + pgvector HNSW Embeddings for AI/RAG Workloads [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Full • Performance Optimization, Complexity & Latency Impact of Full • Live Debugging, Stack • Runnable Code Implementation (SQL) • Hands • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for Full-Text Search (tsvector / GIN) + pgvector HNSW Em... • Production Hardening & Security: Row-Level Security (RLS), SCRAM-SHA-256 roles & lock_timeout guards

Purpose: To understand and apply Full-Text Search (tsvector / GIN) + pgvector HNSW Embeddings for AI/RAG Workloads in real PostgreSQL systems — specifically: Run relational transa...

What Topic Is Next → Next Topic 16.2: Streaming & Logical Replication, WAL Archiving & Point-in-Time Recovery (PITR)

16.2 Streaming & Logical Replication, WAL Archiving & Point-in-Time Recovery (PITR) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Streaming & Logical Replication, WAL Archiving & Point • Performance Optimization, Complexity & Latency Impact of Streaming & Logical Replication, WAL Archiving & Point • Live Debugging, Stack • Runnable Code Implementation (SQL) • Hands • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for Streaming • Production Hardening & Security: Row-Level Security (RLS), SCRAM-SHA-256 roles & lock_timeout guards

Purpose: To understand and apply Streaming & Logical Replication, WAL Archiving & Point-in-Time Recovery (PITR) in real PostgreSQL systems — specifically: Execute zero-data-loss f...

What Topic Is Next → Next Topic 16.3: Reading Complex EXPLAIN (ANALYZE, BUFFERS) Plans & Postgres Interview Scenarios

16.3 Reading Complex EXPLAIN (ANALYZE, BUFFERS) Plans & Postgres Interview Scenarios [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Reading Complex EXPLAIN (ANALYZE, BUFFERS) Plans & Postgres Interview Scen... • Performance Optimization, Complexity & Latency Impact of Reading Complex EXPLAIN (ANALYZE, BUFFERS) Plans & Postgre... • Live Debugging, Stack • Runnable Code Implementation (SQL) • Hands • Idiomatic PGSQ Implementation: B-Tree/GIN/GiST/BRIN indexes, CTEs, JSONB & declarative partitioning • PostgreSQL 16 Heap Tuples, MVCC XID Snapshots, WAL & Shared Buffers Execution & Memory Lifecycle for Reading Complex EXPLAIN (ANALYZE, BUFFERS) Plans • Production Hardening & Security: Row-Level Security (RLS), SCRAM-SHA-256 roles & lock_timeout guards

Purpose: To understand and apply Reading Complex EXPLAIN (ANALYZE, BUFFERS) Plans & Postgres Interview Scenarios in real PostgreSQL systems — specifically: Diagnose Nested Loop vs...

What Topic Is Next → Module 16 Capstone → End-of-Course Live Client Project: LedgerPG Multi-Tenant Row-Level Security, Partitioned Time-Series & pgvector Engine

Module 16 Hands-On Capstone Project (100 Marks): HNSW pgvector Semantic Search + Full-Text tsvector Hybrid Query Engine — Architect, implement, test, and document 'HNSW pgvector Semantic Search + Full-Text tsvector Hybrid Query Engine' applying every bullet-wise engineering concept from Module 16 (pgvector Hybr...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: LedgerPG Multi-Tenant Row-Level Security, Partitioned Time-Series & pgvector Engine • **Confirmed Role:** PostgreSQL 16 Database Architecture & Tuning Intern

Domain & Production Infra: Enterprise SaaS Analytics & AI Search Database Client (PostgreSQL 16 + Declarative Partitioning + Row-Level Security + PgBouncer + pg_stat_statements)

- Architected tenant-isolated PostgreSQL 16 schemas enforced at the database engine via Row-Level Security (RLS) policies.
- Designed B-Tree, GIN (JSONB/Full-Text), partial, and HNSW (pgvector) indexes verified with EXPLAIN (ANALYZE, BUFFERS).
- Implemented monthly declarative table partitioning, CTE analytics, MVCC autovacuum tuning, and PgBouncer pooling.
- Executed lock-safe online migrations and point-in-time recovery drills on a live production PostgreSQL instance.

Target Job Roles: PostgreSQL Database Engineer, Backend Data & Performance Engineer, Cloud Database Reliability Engineer (DBRE), SQL & Data Platform Specialist

Realistic Annual CTC Scope: **Fresher / Entry-Level Data & SQL Engineer (0–1 Yr):** ₹5.0 LPA – ₹9.2 LPA **Mid-Level PostgreSQL / Backend Engineer (1–3 Yrs):** ₹10.0 LPA – ₹20.0 LPA **Staff Database Architect / Principal DBRE (3–6+ Yrs):** ₹21.0 LPA – ₹42.0+ LPA

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 95400 07917 • Web: https://mockattempt.com/courses/postgresql-training

Try 2 Free Topic-Wise Timed Topic Mock Tests & Spoken MockAttempt AI Interviews daily on every topic above before enrolling.