ChatGPT

CockroachDB Study Plan (Beginner → Advanced)

Stage 1 – Beginner

- **Core concepts & architecture:** CockroachDB is a *distributed SQL* database designed for horizontal scalability and strong consistency [1] [2]. It uses a shared-nothing architecture: data is stored in many **ranges** (contiguous key-space chunks) which are automatically sharded, replicated, and rebalanced across nodes [3] [4]. Each range is replicated (3× by default) using the Raft consensus protocol so that any node can serve reads/writes without a single "master" [3] [5]. CockroachDB exposes a PostgreSQL-compatible SQL layer, so applications can use standard SQL drivers (libpq, etc.) to talk to any node [6] [7].
- **Setup & basics:** Practice installing CockroachDB locally. For example, run `cockroach start --insecure --listen-addr=localhost:26257 --join=localhost:26257` and initialize with `cockroach init` [1]. Use the built-in `cockroach sql` shell or GUI **DB Console** to issue SQL commands. The command `cockroach demo` spins up an in-memory cluster with sample data (e.g. **MovR** or **bank** schema) for exploration [8]. Use `cockroach demo --nodes=3` to simulate a 3-node cluster. Beginners should focus on core SQL: creating databases and tables, inserting and selecting rows, defining **PRIMARY KEY**s (required on every table in CockroachDB [9]), and experimenting with multi-row `INSERT` and `UPSERT` statements (multi-row DML is much faster than single-row loops [10]). Learn the basic CockroachDB commands (`cockroach node`, `cockroach sql`, etc.) [8] [11].
- **Hands-on projects:** Try a simple app or "hello world" example. For instance, create a small "to-do" or banking schema and write a script to do basic CRUD via Cockroach's SQL API. Use `cockroach demo` or a local 3-node cluster and deliberately shut down one node to see failover (writes still succeed on remaining replicas). Experiment with SQL features like **RETURNING** clauses (e.g. `INSERT ... RETURNING *`) and test that reads/writes succeed on any node. Import a sample CSV (using `IMPORT INTO`) to load data. Write a few JOINs and transactions to see how CockroachDB behaves under concurrent access.
- **Key resources:**
- **Official docs & tutorials:** CockroachDB Quickstart guides for local clusters, the SQL Tutorial for basic SQL syntax, and the Architecture Overview [1].
- **Cockroach University:** Free courses (e.g. "Intro to Distributed SQL") on Cockroach Labs' online learning portal [12].
- **Books:** *CockroachDB: The Definitive Guide* (free O'Reilly book) covers all levels (including architecture, SQL, deployment, backup) [13] [14].
- **Community tools:** Use `cockroach demo` (with built-in "MovR" sample) [8]; Cockroach Labs Slack and forum; YouTube tutorials like "Getting Started with CockroachDB".
- **Beginner practice tasks:**
    - Install and run a single-node cluster (insecure and secure modes).
    - Spin up a 3-node cluster (e.g. on local VMs or Docker) and watch the **DB Console** (http://localhost:8080) as nodes join.
    - Create tables with composite primary keys and experiment with inserting/updating rows.
    - Use `IMPORT` or `cockroach sql < schema.sql` to load data (e.g. the MovR dataset from examples).
    - Perform basic DML and SELECT queries, including multi-row INSERT and UPSERT (per [17†L252-L261]).

- Compare to PostgreSQL: note that `INT` is 64-bit by default [15] and that you must define a primary key (no automatic sequence column unless you use `SERIAL` or `UNIQUE` row ID).

Stage 2 – Intermediate

- **Replication & consensus (Raft):** Learn how CockroachDB replicates data. Each range's replicas form a Raft group: one **leader/leaseholder** coordinates writes, replicating them to followers via Raft [5]. A quorum (2 of 3) must agree to commit a change [16]. CockroachDB can tolerate *F = (N-1)/2* node failures (e.g. one failure in a 3-replica group) [16]. When a node fails or is added, CockroachDB automatically rebalances ranges across the remaining/added nodes [17]. Hands-on: reconfigure a running cluster (e.g. shut down a node) and verify data availability and self-healing. Use `cockroach node status` and the DB Console "Liveness" and "Ranges" dashboards to observe replication.
- **Built-in SQL features:** Practice CockroachDB's SQL capabilities beyond basics:
- **Indexes:** CockroachDB supports PRIMARY and secondary indexes. Learn how to create and use unique vs non-unique indexes. Use `EXPLAIN` and `EXPLAIN (OPT)` to examine query plans.
- **Functions & operators:** Familiarize yourself with CockroachDB's built-in SQL functions (string, date/time, JSONB, ARRAY functions, etc.) [18]. Note CockroachDB's PostgreSQL compatibility: most PostgreSQL functions work, but some advanced features differ (e.g. limited `ALTER COLUMN` support). Refer to "Features that differ from PostgreSQL" in the docs.
- **Transactions & consistency:** CockroachDB provides full ACID transactions with **serializable** isolation by default [19]. (This is stronger than PostgreSQL's default READ COMMITTED.) Practice simple multi-statement transactions with `BEGIN ... COMMIT`. See how CockroachDB avoids write-skew anomalies that can occur in weaker isolation [19]. For intermediate tasks, experiment with stale reads: use the `AS OF SYSTEM TIME` clause to read historical data from a local replica (faster, but potentially stale) [20].
- **Performance tuning:** Learn CockroachDB-specific tuning guidelines: for example, use multi-row DML and `UPSERT` efficiently [10]; avoid hotspots by distributing writes (e.g. avoid sequential ID as primary key [10]). Use `EXPLAIN (DISTSQL)` to see if a query is distributed or co-located [21]. Practice identifying slow queries via the DB Console's "Insights" or `crdb_internal` tables. Review CockroachDB's Performance Best Practices (e.g. use `TRUNCATE` over large `DELETE` [22], use indexes to avoid full scans [23]).
- **Security features:** Set up encryption and authentication. By default, CockroachDB uses TLS for all inter-node and client communications [24]. Experiment with both secure and insecure modes (production requires TLS certificates). Learn user/role management (SQL statements `CREATE USER/ROLE`, `GRANT`, etc.). CockroachDB supports SCRAM-SHA-256 password auth and, in cloud/enterprise editions, advanced auth (OIDC/SSO) [24]. Explore row-level and client-level security: configure IP allowlists and TLS client cert auth [24]. Review audit logging options (CockroachDB Advanced) for compliance.
- **Hands-on projects:**
- **Replication test:** Deploy a 3-node cluster (multi-AZ or multi-region if possible). Create a table and insert data, then kill the leader node. Verify that writes continue (a new leader is elected) and the data remains replicated.
- **Use MOLT tools:** Practice using Cockroach's MOLT toolkit. For example, take a simple PostgreSQL schema (e.g. Northwind or a small sample), run the Schema Conversion Tool to translate it [25], and load data into CockroachDB. Note the need for an explicit PK on each table (unlike some legacy DBs) [9].
- **Secure cluster:** Reinitialize a cluster in secure mode (generate certificates), create multiple users with different privileges, and test authentication.
- **Key resources:**

- **Replication & Consensus:** CockroachDB docs on Replication Layer (Raft) [26] [5]. A blog post "Raft Explained" or Dan The Engineer's Raft tutorial may help.
- **Performance:** The Performance Best Practices and Performance Tuning Recipes pages [10] [23].
- **SQL & Tools:** CockroachDB SQL docs, including `cockroach demo` and `cockroach dump`. Cockroach University Labs on DML tuning.
- **Security:** Security Overview (encryption/auth/authz) [24] [27]. Cockroach Labs blog posts on security features (e.g. "Encryption in CockroachDB").
- **Intermediate practice tasks:**
    1. **Test replication:** Run `cockroach start` on 3 servers (each in different AZ if possible) and use `cockroach node status`. Insert data, then stop one node; observe that cluster remains available.
    2. **Migration experiment:** Use MOLT to convert a small PostgreSQL schema (add explicit PRIMARY KEYs) [9], and migrate data.
    3. **Optimize a query:** Create a table with a secondary index, run a SELECT with and without the index. Use `EXPLAIN ANALYZE` to see the difference.
    4. **Security setup:** Enable TLS (generate CA and client certs per Cockroach docs), create a user, and connect with `cockroach sql --certs-dir=...`.

Stage 3 – Advanced

- **Multi-region & geo-partitioning:** Explore CockroachDB's global features. First, practice configuring node localities: start nodes with `--locality=region=...` flags to designate regions. Use `ALTER DATABASE <db> ADD REGION "<region>"` to make a multi-region database. Experiment with table **localities**: create some tables as `GLOBAL`, some as `REGIONAL BY ROW` or `REGIONAL BY TABLE`. For example:
- A **GLOBAL** table has replicas in every region (good for read-mostly data) [28].
- A **REGIONAL BY ROW** table has each row tied to a "home" region, with follower reads elsewhere [29].
- Test **survival goals**: by default CockroachDB provides Zone survival (survive one AZ failure); you can `ALTER DATABASE ... SURVIVAL GOAL = REGION` for full region fault-tolerance (writes become higher-latency) [30] [31].
  Use the How to Choose a Multi-Region Configuration guide [30] [32]. Also see Multi-Region Capabilities Overview [33] [34] (e.g. Cockroach recommends lowering `--max-offset` clock skew to ~250ms for geo clusters [35]).
- **Distributed SQL execution:** Dive into Cockroach's query planner and DistSQL. Understand how a query is parsed, optimized, and possibly distributed: simple queries run on the gateway, but large scans are split across nodes (via **DistSQL**) [21] [36]. Use `EXPLAIN (DISTSQL)` and `EXPLAIN (VEC)` to see the physical plan. Advanced tuning: enable or disable vectorized execution and benchmark. Learn to interpret the `vectorized` vs `row` execution in EXPLAIN.
- **Backup & restore / disaster recovery:** Practice CockroachDB's backup features. Use `BACKUP TO 's3://bucket...'` (or `BACKUP TO 'nodelocal://...` for local FS) to take a full backup of a database or cluster [37]. Create an incremental backup, and then use `RESTORE FROM ...` to restore to a new cluster. Test **point-in-time recovery** by taking periodic backups with `WITH revision_history` and then restoring to a timestamp within GC TTL [38]. Learn about **scheduled backups**: use `CREATE SCHEDULE FOR BACKUP` to automate daily backups [39].
- **Fault tolerance & internals:** Study deep internals. Learn how CockroachDB uses **Hybrid Logical Clocks (HLCs)** for transaction timestamps [40]. Cockroach enforces clock sync: if a node's clock drifts beyond a threshold (default 500ms), the node will crash to prevent anomalies [41]. Investigate the **timestamp cache** and read/write intent resolution in the transaction layer [42].

Advanced tasks: instrument a query using `SHOW TRACE FOR` to see how requests flow through DistSender and Raft. Explore the Sys tables (`crdb_internal`) for metrics on contention or latency.

- **Troubleshooting:** Familiarize with CockroachDB's debugging tools. Use `cockroach debug zip` to collect logs and diagnostics across the cluster [43]. If a query fails, check the CockroachDB **Console** for hotspots, contention metrics, or view the **Transactions** and **Statement Diagnostics** pages. Review the Troubleshooting Overview steps: checking logs, examining the `crdb_internal` tables, and using the "critical nodes" endpoint [43] [44]. Simulate issues: disable a node's network, create a checksum error (using `cockroach debug rewrite-locality`), etc., and practice resolving.
- **Key resources:**
- **Multi-Region Guides:** Cockroach docs on Multi-Region Overview [33] [34] and blog posts like "Multi-Region patterns" or "Low Latency in a Global Cluster".
- **Internal Architecture:** The detailed Transaction Layer and SQL Layer docs [21] [40]. Cockroach University courses on internals.
- **Backup/Restore:** Backup and Restore Overview [45] [46] and Storage Architecture docs.
- **Troubleshooting:** Troubleshooting Overview [43] [44], "common errors and solutions" docs, and the DB Console's built-in monitors.
- **Advanced practice tasks:**
    1. **Multi-region app:** Modify an app to connect to a multi-region CockroachDB (e.g. in AWS us-east1 and us-west1). Create a REGIONAL BY ROW table, insert data from different regions, and verify low-latency writes in home region vs higher-latency cross-region writes.
    2. **Backup/PITR:** On a live cluster, enable scheduled backups with revision history. Perform updates/deletes, then `RESTORE` to a point before the change to see data recovered.
    3. **Chaos testing:** Use tools like `cockroach debug decommission` and `cockroach node decommission` to simulate node failures, or adjust clock sync to trigger an offset crash. Verify the cluster self-heals (e.g. leases transfer to new leaders).
    4. **Query profiling:** Run a heavy workload (e.g. `cockroach demo --with-load`) and then use the "Insights" page or `SHOW TRACE` to analyze a slow query. Use **cockroach debug** tools to capture profiles.

PostgreSQL→CockroachDB Migration Strategy

- **Compatibility:** CockroachDB is PostgreSQL-wire-compatible and largely syntactically compatible, but there are key differences [47] [7]. For example, every table **must** have an explicit PRIMARY KEY [9] (unlike Postgres, where you can have a table without one). CockroachDB's `INT` defaults to 64-bit (`INT8`) [15], so adjust schemas expecting 32-bit.
- **Schema conversion:** Use Cockroach's MOLT Schema Conversion Tool or `cockroach dump` with `--dump-mode=schema`. The MOLT tool can ingest a Postgres DDL and automatically convert incompatible syntax [25]. Common edits include adding missing primary keys, rewriting certain Postgres-specific data types or functions, and adjusting sequence/serial columns (Cockroach supports `SERIAL` or `GENERATED AS IDENTITY`).
- **Data migration:** Export your Postgres data (e.g. CSV dumps or logical replication). Cockroach recommends using MOLT FETCH to do minimal-downtime migration: it loads initial data into CockroachDB and then streams ongoing changes via Postgres logical replication [48] [25]. Alternatively, for a one-time migration, bulk COPY/IMPORT your data.
- **Validation:** After migrating schema and data, run consistency checks. CockroachDB can compare row counts or checksums using SQL queries. Use the DB Console or SQL to verify critical data. Test application workloads against the new cluster.

- **Deployment cutover:** Once CockroachDB is in sync, update your application's database connection to point to CockroachDB (via the PG driver). Because CockroachDB is designed for geo-distribution, you can consider a phased cutover by region or DC. Also plan to set up CockroachDB monitoring and backups (see Stage 3 above).
- **Practical example:** For instance, migrating a simple `users` and `orders` schema from Postgres: run `ALTER TABLE users ADD PRIMARY KEY (id)`, export data to CSV, use `cockroach sql < schema.sql` to create tables in Cockroach, then `IMPORT INTO users CSV DATA ('nodelocal://.../users.csv')`. Use `SET CLUSTER SETTING kv.rangefeed.enabled = false;` to disable CDC if not needed. After loading, run a few JOIN queries to ensure behavior matches Postgres.
- **Resources:** CockroachDB's migration guides – the Migration Overview and "Migrate from PostgreSQL" tutorial – and community blog posts (e.g. "Migrating Our Multicloud Banking App from Postgres to CockroachDB" from RoachFest). Use the official Features Differences doc as a checklist.

CockroachDB Interview Questions (by difficulty)

Beginner

- **What is CockroachDB and what are its core features?**
  Answer hint: A distributed SQL/NewSQL database with **horizontal scalability** and strong consistency [1] [2]. Key features: ACID transactions (serializable by default [19]), geo-replication, and **automatic failover** via Raft consensus [3] [4]. It is PostgreSQL-wire compatible [6] [7].
- **Explain "distributed SQL" in the context of CockroachDB.**
  CockroachDB shards and replicates data across all nodes automatically. The database accepts SQL queries on any node (active/active), translating them into distributed key–value operations [3] [21]. Developers use familiar SQL, while Cockroach handles data partitioning, consensus, and distributed execution under the hood.
- **How does CockroachDB achieve strong consistency?**
  CockroachDB uses the [Raft consensus algorithm] to replicate each data range to multiple nodes [5]. A write is committed only when a majority of replicas agree, ensuring linearizability. Transactions use a distributed timestamp (via HLC) and a 2-phase commit protocol for atomicity [3] [40]. This yields serializable isolation by default (no anomalies allowed) [19].
- **What is a "range" in CockroachDB?**
  A range is a contiguous span of the table's key-space (rows). CockroachDB partitions tables into many ranges (default ~64 MB each). Each range is replicated to multiple nodes; the set of replicas forms a Raft group [5]. Ranges allow Cockroach to split, move, and replicate data flexibly.
- **Why must every table have a primary key in CockroachDB?**
  CockroachDB requires an explicit primary key to uniquely identify rows and to place them in the distributed key–value store [9]. Unlike Postgres, CockroachDB cannot have a table without a unique key, because it uses the primary key to organize and lookup rows in its KV engine.
- **How do you connect to CockroachDB?**
  You can use the `cockroach sql` CLI or any PostgreSQL-compatible client library (Go, Python `psycopg2`, Java JDBC, etc.), because CockroachDB speaks the Postgres wire protocol [6] [7]. Cockroach also provides a web-based **DB Console** for monitoring and basic SQL.

Intermediate

- **Explain transaction isolation in CockroachDB.**
  CockroachDB provides **serializable snapshot isolation** by default [19]. This is stronger than Postgres' default; it prevents all ANSI-SQL anomalies (e.g. write skew) [19]. Internally it uses

optimistic MVCC with Raft to enforce serializability. (Cockroach also supports `READ COMMITTED` if explicitly set, but serializable is standard.)

- **What happens when you INSERT with `RETURNING` or `UPSERT`?**
  An `UPSERT` performs a write without a prior read when no secondary index exists, making it faster than `INSERT ON CONFLICT` on simple tables [49]. Using `RETURNING` returns values of inserted rows. On a table with secondary indexes, there's no performance difference. Multi-row `INSERT ... RETURNING` batches multiple rows in one transaction – recommended for bulk loads [10] [50].
- **How does CockroachDB replicate and fail over data?**
  Each range is replicated (default factor 3). Raft elects one replica as the leader (leaseholder) [5]. Writes go through the leader; once a majority commit, they are durable. If a node fails, the cluster detects the missing replica and rebalances by adding replicas on other nodes [17]. Another replica automatically becomes leader, so reads/writes continue. The cluster tolerates one failure per Raft group (e.g. one node out of three) [16].
- **What is `cockroach demo` and how is it used?**
  The `cockroach demo` command starts a temporary in-memory cluster for experimentation [8]. It can preload example databases (like `movr`, `bank`, `kv`). It's useful for trying CockroachDB features quickly without installing a full cluster. It also opens an SQL shell and local DB Console.
- **How do you handle SSL/TLS in CockroachDB?**
  By default, CockroachDB uses TLS for all inter-node and client communications (node-to-node authentication requires TLS certificates) [24]. You must generate a CA and certificates (Cockroach provides tools in the `cockroach cert` suite). For clients, you can also use password auth (SCRAM) or client certs [24]. The Cockroach Cloud service enforces TLS by default.
- **Performance tuning:**
  Key points: batch DML (multi-row inserts), use `UPSERT`, avoid hotspots (e.g. randomize primary keys) [10]. Use `TRUNCATE` instead of `DELETE` for whole-table clears [22]. Analyze query plans (`EXPLAIN`) to ensure indexes are used (avoid full scans) [23]. Adjust cluster settings (`kv.concurrency`, etc.) if needed.

Advanced

- **How do you deploy a multi-region CockroachDB cluster?**
  Assign each node a `--locality` (e.g. region=us, zone=...). Use `CREATE DATABASE ... PRIMARY REGION` and `ALTER DATABASE ... ADD REGION` for each region. Tables default to a locality; you can set a table to `GLOBAL`, `REGIONAL BY ROW`, or `REGIONAL BY TABLE`. For strong fault tolerance, set `DATABASE ... SURVIVAL GOAL = REGION` (quorum survives region failure) [51]. Cockroach recommends setting `--max-offset=250ms` on new clusters to reduce global write latency [34]. Multi-region reads can use follower reads for lower latency; this requires a closed timestamp (max clock skew) [40].
- **What is DistSQL (distributed SQL) and vectorized execution?**
  Cockroach's query planner may split a query into parts that run on multiple nodes (DistSQL) [21]. Large scans are parallelized: each node scans its local range of data, performs filters/aggregations, and sends partial results back to the coordinator [52] [36]. This reduces data transfer and speeds up large queries. The vectorized engine processes data in columnar batches for CPU efficiency [53]. Use `EXPLAIN (VEC)` to see if a query uses the vectorized engine, and tune if needed.
- **Fault tolerance & clocks:**
  CockroachDB requires loosely synchronized clocks (via NTP). Each transaction gets an HLC timestamp [40]. If a node's clock drifts too far (above the max-offset setting), it will crash to avoid consistency violations [41]. The cluster maintains a **timestamp cache** to detect stale reads and

write–read conflicts. Cockroach handles failures by transferring Raft leadership and leases to healthy nodes (via “leader leases”) [54] . Practice: disable a node’s clock sync (remove NTP) to see the node panic on startup.

- **Backup and restore strategies:**
  CockroachDB supports full and incremental backups [55] . You can back up an entire cluster, individual databases, or tables. For disaster recovery, use scheduled backups (`CREATE SCHEDULE FOR BACKUP`) to S3/GCS, and protect those timestamps from GC [39] . For point-in-time recovery, back up *with revision history* and use `RESTORE ... AS OF SYSTEM TIME` [55] . Test restoring to a new cluster regularly.
- **Troubleshooting:**
  When problems arise, first check per-node logs and use `cockroach debug zip` to gather diagnostics [43] . The DB Console provides metrics dashboards; use the “Insights” and “Transactions” pages to find slow queries or contention. Use the **Critical Nodes** HTTP endpoint to check cluster health [44] . Familiarize yourself with common errors (retry errors, etc.) and the procedures on the [Troubleshooting Overview] [43] [44] . In interviews, you might be asked how you’d use these tools to debug issues.

Sources: CockroachDB official documentation and tutorials [1] [4] [19] [8] [55] [24] cover these topics in detail.

---

[1] [3] [6] [12] Architecture Overview
https://www.cockroachlabs.com/docs/stable/architecture/overview/

[2] [7] Comparing CockroachDB and PostgreSQL
https://www.cockroachlabs.com/blog/postgresql-vs-cockroachdb/

[4] [5] [16] [17] [26] [54] Replication Layer
https://www.cockroachlabs.com/docs/stable/architecture/replication-layer

[8] [11] cockroach demo
https://www.cockroachlabs.com/docs/stable/cockroach-demo

[9] [15] [25] [47] [48] Migrate to CockroachDB
https://www.cockroachlabs.com/docs/molt/migrate-to-cockroachdb

[10] [22] [49] [50] SQL Performance Best Practices
https://www.cockroachlabs.com/docs/stable/performance-best-practices-overview

[13] [14] O'Reilly | CockroachDB: The Definitive Guide (2nd Edition)
https://www.cockroachlabs.com/guides/oreilly-cockroachdb-the-definitive-guide/

[18] Functions and Operators - CockroachDB
https://www.cockroachlabs.com/docs/stable/functions-and-operators

[19] Serializable Transactions
https://www.cockroachlabs.com/docs/stable/demo-serializable

[20] [40] [41] [42] Transaction Layer
https://www.cockroachlabs.com/docs/stable/architecture/transaction-layer

[21] [36] [52] [53] SQL Layer
https://www.cockroachlabs.com/docs/stable/architecture/sql-layer

[23] Performance Tuning Recipes
https://www.cockroachlabs.com/docs/stable/performance-recipes

24 27 CockroachDB Security Overview
https://www.cockroachlabs.com/docs/stable/security-reference/security-overview

28 32 33 34 35 Multi-Region Capabilities Overview
https://www.cockroachlabs.com/docs/stable/multiregion-overview

29 30 31 51 How to Choose a Multi-Region Configuration
https://www.cockroachlabs.com/docs/stable/choosing-a-multi-region-configuration

37 38 39 45 46 55 Backup and Restore Overview
https://www.cockroachlabs.com/docs/stable/backup-and-restore-overview

43 44 Troubleshooting Overview
https://www.cockroachlabs.com/docs/stable/troubleshooting-overview