1) What Hive actually is (in simple words)

Think of **Hive as "SQL on top of files"**.

- Your **data is stored as files** (mostly on **HDFS**) — not inside a traditional database.
- Hive lets you **query that file data** using **HiveQL** (SQL-like), so you don't have to write MapReduce code.

Hive is not a "normal database"

Hive is designed for **batch analytics** (big data queries), not for quick transactional work like banking apps.

Hive is **NOT**:

- **OLTP / real-time system**
- It has **high latency** (query takes time to start/finish)
- But it can have **high throughput** (process lots of data per batch)

Exam-style phrasing:

> Hive is for *analysis* over large datasets stored as files, not for frequent row-by-row updates like an RDBMS.

---

2) The "big idea" behind Hive: Schema-on-read

What it means

In Hive, schema is checked **when you read/query the data**, not when you store it.

- **Schema-on-read:** you can store raw files first, decide structure later at query time.

Why it matters (real-life example)

Imagine you have raw web logs (CSV/TSV). You dump them daily into HDFS.
Later:

- marketing team reads them with one schema (campaign focus)
- security team reads them with another schema (IP focus)

Same files, different "views".

---

3) Where tables “live”: Hive Warehouse

Hive organizes data like folders.

- Default warehouse path in HDFS is: **`/user/hive/warehouse`**
- **Each table = a folder**
- **Each partition = sub-folder inside table folder**

So visually:

- `/warehouse/sales/` (table)
- `/warehouse/sales/year=2024/month=10/` (partition)

---

4) Hive Metastore (the “dictionary” of Hive)

Hive needs a place to store **metadata** (info *about* tables), because the real data is in files.

Metastore stores:

- Database namespace
- Table schema (columns/types)
- Storage format + SerDe info
- Partition info

Important: Metastore database modes

Your slides mention:

- Default uses **Derby** (embedded, single-user, mostly for testing) and only one active connection.
- For production, use **MySQL/Postgres**.

Types:

- **Embedded:** metastore inside Hive service process (good for dev/tests)
- **Local:** each Hive service has its own metastore process; uses shared external DB (multi-user)
- **Remote:** metastore runs as standalone service; all clients connect to it

Exam trap: “Why metastore is needed?”
Because Hive needs to know **what columns exist** and **where the files are** before it can query.

---

5) Hive architecture (how a query runs)

When you write a Hive query, it passes through stages:

A) Driver (the boss/manager)

“All commands and queries go to the Driver” — it coordinates compile + optimize + execution.

B) Compiler (turns query into a plan)

Compiler does multiple steps:

- Parser → makes parse tree
- Semantic analyzer → checks meaning
- Plan generator → logical plan
- Optimizer → improves plan (like pushing filters early, pruning columns, balancing load, combining joins)

C) Execution (actually runs work)

Slides say Hive usually executes using **MapReduce jobs** and communicates with JobTracker to start them.

Simple mental model:

Query → Driver → Compile/Optimize → Execute (MR jobs) → Output

---

6) Tables: Internal vs External (VERY exam-important)

Internal/Managed table

- Hive “owns” the data folder.
- If you drop the table, it can delete the underlying files (in managed style).

External table (your sir explicitly said focus this)

External table means:

- Data is in HDFS somewhere.
- Hive just keeps a **reference** to it.
- **Dropping table does NOT delete the data** (because Hive doesn't own it).
- You can even specify the exact location.

Perfect exam one-liner:

> External table = metadata in Hive, data stays independent in HDFS; DROP removes metadata only, not files.

Real-world example:
Company has a shared HDFS folder of raw logs used by multiple tools (Spark, custom scripts, Hive). If Hive drops the table, you **still need logs**, so you use **external**.

---

7) Partitioning (your sir said this is important)

What partitioning means

Partitioning = **split table data into folders by a column** (like year, month, country).

- Makes some queries faster
- You divide data based on partition column
- You use `PARTITIONED BY` when creating
- `SHOW PARTITIONS` shows partitions

Why it speeds up queries

Because Hive can do **partition pruning**:
If query says `year=2024`, Hive reads only that folder, not everything.

Slides literally show this idea: "Only 1 partition will be read – no need to scan the whole dataset."

Real-world example:
E-commerce orders table partitioned by date. "Sales of Pakistan on 2025-11-02" reads just that partition folder.

Exam trap:
Partition column is not "magic indexing"; it is **physical organization (folders)**.

---

8) Bucketing (your sir said bucketing too)

What it is

Bucketing = split data into a fixed number of files (“buckets”) using a **hash** of a column.

Why do it?

- Helps in sampling, joins, and aggregations.
- Hive can parallelize work because data is already distributed.

Slides say:

- Bucketing can speed up sampling; without bucketing Hive might scan whole dataset.
- Uses `CLUSTERED BY ... INTO N BUCKETS`

The key concept (super exam-worthy)

“All records for the same customer always go to the same bucket file (=hash(cid) mod 8). A single customer is completely in 1 bucket, not spread out.”

Real-world example:
If you bucket by `customer_id`, then counting orders per customer is faster because all of that customer’s rows are together.

---

9) Loading/Getting data into Hive (concept only)

Two common ways:

LOAD DATA

- Loads from file/directory (HDFS by default, or local if you specify local)
- Appends unless overwrite

INSERT INTO / INSERT OVERWRITE

- Inserts using result of a query
- Again: overwrite vs append matters

---

10) HiveQL limitations (conceptual exam points)

Hive is SQL-like, but not full SQL.

Slides list missing parts:

- Correlated subqueries (better to use JOIN)
- Stored procedures not allowed (use UDFs)
- Materialized/updatable views issues

Also: Hive traditionally doesn't support row-level update/delete in the way RDBMS does (it's more "load data / rewrite files").

A) Core "Why Hive" and What It Is

1) Why was Hive introduced on top of Hadoop/MapReduce? Give 3 challenges of MapReduce it fixes.

Answer: MapReduce is (1) **verbose/complex** to write (imperative, distributed computing burden), (2) **no SQL** support so analysts can't do ad-hoc queries easily, (3) **batch latency** + long job stack even for simple queries; also lacks schema management and BI connectivity. Hive provides SQL-like HiveQL and translates it to MR/Tez/Spark plans. **Example:** business analyst wants "total sales by country" without writing Java MR.

2) Define Hive in one paragraph and mention where data and schema live.

Answer: Hive is a **data warehouse infrastructure on Hadoop** enabling summarization + ad-hoc SQL-like queries (HiveQL). Data is stored as **files in HDFS** while schema is stored as **metadata in an RDBMS metastore (logical layer)**; queries are compiled into an execution plan and run on a distributed engine. **Example:** querying clickstream logs stored in HDFS using HiveQL.

3) Hive is "not OLTP / not real-time." Explain with 3 differences from RDBMS.

Answer: (1) Hive is optimized for **batch processing**; **latency** to start/finish a query is high vs RDBMS. (2) It targets **high throughput** per batch vs quick per-row transactions. (3) Traditionally not designed for frequent row updates/deletes like OLTP (though ACID exists with constraints). **Example:** bank transfer system fits OLTP; monthly sales reporting fits Hive.

B) Architecture & Components (exam loves “define each part”)

4) Draw/explain Hive architecture components: External interfaces, Driver, Metastore, Execution, Storage.

Answer: Users connect via **CLI/Web UI/JDBC/ODBC**. The **Driver** manages compilation/optimization/execution lifecycle. **Metastore** holds table/partition metadata. The execution engine (MR/Tez/Spark/LLAP) runs jobs on the cluster (submitted to YARN), and data is stored in **HDFS** (warehouse/external locations). **Example:** BI tool uses JDBC to run query; driver compiles; execution runs on Tez; reads Parquet from HDFS.

5) What does the Driver do? Give 3 responsibilities.

Answer: (1) Accepts commands/queries, (2) **compiles** and **optimizes** them, (3) **executes** them by coordinating with the execution engine (traditionally MapReduce via JobTracker/YARN). **Example:** you run a query in Beeline; driver builds plan and submits tasks to cluster.

6) What does the Metastore store? Give at least 4 metadata items.

Answer: Metastore stores **database(namespace)**, **table schema (columns & types)**, owner, storage format and **SerDe** info, plus **partition metadata** (partition columns, locations, storage/SerDe per partition). **Example:** a table points to `/user/hive/warehouse/sales` and partitions to subfolders.

7) Explain Embedded vs Local vs Remote Metastore. When would you use each?

Answer: Embedded: metastore inside Hive process using embedded Derby—good for dev/testing, single user. **Local**: each Hive service has its own metastore process; shared external DB—multi-user. **Remote**: metastore runs as separate service; clients connect to it—best for centralized production. **Example:** classroom laptop uses embedded; company cluster uses remote with MySQL/Postgres.

8) Why is Derby “bad” for production? Give 3 reasons.

Answer: Default metastore (Derby) is embedded + **single-user**, allows **only one active connection**, intended for local/testing; production needs Postgres/MySQL with concurrency. **Example:** multiple analysts running Hive queries simultaneously will fail/lock with Derby.

C) Execution Engines (concepts, not code)

9) Hive can run on MapReduce/Tez/Spark/LLAP. Compare MR vs Tez/Spark in 3 points.

Answer: MR is original/stable but **slow**, heavy startup. Tez is **DAG-based** and optimized for Hive (faster). Spark engine uses Spark's in-memory and DAG execution model. LLAP adds in-memory caching for lower latency on Hive-on-Tez. **Example:** interactive dashboards prefer LLAP/Tez over MR.

10) What are the 6 steps from HiveQL to execution (pipeline)?

Answer: (1) Parse to AST, (2) Semantic analysis checks schema via metastore, (3) Logical plan operator tree, (4) Optimization (rule/cost), (5) Physical plan -> MR/Tez/Spark DAG, (6) Execution submitted to YARN. **Example:** query with filter gets predicate pushed down before execution.

D) Data organization: Warehouse, Tables, Partitions, Buckets

11) What is Hive "warehouse"? How are tables and partitions stored physically?

Answer: Warehouse is default HDFS area for Hive tables (e.g., `/user/hive/warehouse`). Each **table is a subdirectory**, and **partitions are subdirectories** inside the table directory. Data is stored as **flat files** (Parquet/ORC/Text/etc.). **Example:** `/warehouse/sales/year=2024/month=10/`.

12) Define Partitioning in Hive. Give 3 benefits.

Answer: Partitioning divides a table into **folders by partition column(s)** (horizontal partitions). Benefits: (1) **faster queries** via partition pruning (read fewer files), (2) better manageability (data lifecycle by partition), (3) enables parallelism by partition. **Example:** partition ecommerce_sales by country and date; query one country+date reads only that folder.

13) Explain "partition pruning" with an example.

Answer: When query filters on partition columns, Hive can skip reading other partitions and only scan the matching partition directory. **Example:** `WHERE country='Pakistan' AND` `order_date='2025-11-02'` reads only `/country=Pakistan/order_date=2025-11-02/` rather than the full table.

14) Partitioning vs Bucketing: compare in 3 points.

Answer: (1) Partitioning = **directory-level split** by partition columns; bucketing = **file-level split** by hash. (2) Partitioning helps when queries filter by partition column; bucketing helps joins/aggregations/sampling on bucket column. (3) Partitions can be many (by dates); buckets are fixed number (e.g., 8). **Example:** partition by date, bucket by customer_id for faster `GROUP BY customer_id`.

15) Define Bucketing. Why does "hash(cid) mod 8" matter?

Answer: Bucketing divides data into a fixed number of **bucket files** using **hash(column) mod N**. This ensures (1) **same key goes to same bucket**, (2) parallel reducers can process buckets independently, (3) joins/aggregations are faster because data is pre-distributed. **Example:** all rows for customer 123 end up in one bucket file.

16) In bucketing, why does "Hive assigns each bucket to one reducer only" help performance?

Answer: (1) Each reducer can process one bucket independently, (2) reduces reshuffling for group-by/join because data already grouped by hash, (3) enables parallelism without extra partitioning work. **Example:** counting orders per customer runs across multiple reducers, each handling a subset of customers.

17) Scenario: You need random sampling 1% of a huge table daily. Why bucketing helps?

Answer: Without bucketing, Hive may scan full dataset to sample. With bucketing, data is split into bucket files, so sampling can pick some buckets/files efficiently. Benefits: less scan, faster sampling, repeatable sample. **Example:** TABLESAMPLE on a bucketed table for quick QA checks.

E) Managed vs External Tables (sir's favorite)

18) Managed (internal) table vs External table: give 4 differences.

Answer: (1) Managed: Hive owns metadata + data; External: Hive owns **only metadata**. (2) Managed LOAD typically moves data into warehouse; External does not necessarily move. (3) Dropping managed table deletes metadata + data; dropping external deletes **metadata only**. (4)

External is used for shared data / multiple schemas / other tools. **Example:** raw logs shared with Spark => external.

19) When should you use an External table? Give 3 reasons + example.

Answer: Use external when (1) data is shared with other Hadoop apps (Spark/Presto), (2) you don't want Hive to control lifecycle of files, (3) you might apply multiple schemas to same files. **Example:** `/user/staging/page_view` used by ETL pipeline and queried by Hive.

20) Scenario: A team drops a Hive table and suddenly the underlying HDFS data vanished. What type of table was it? How to prevent?

Answer: It was a **managed/internal** table because DROP removed both metadata and data. Prevent by creating an **EXTERNAL table** so DROP removes only metadata, or by storing critical data outside warehouse and referencing it. **Example:** shared "gold" dataset should be external.

F) Schema-on-read, SerDe, and formats (conceptual but testable)

21) Define schema-on-read and contrast with schema-on-write (3 points).

Answer: Hive is schema-on-read: (1) schema enforced at **query time**, (2) flexible—same data can be read with multiple schemas, (3) faster loading but potentially more work at query. RDBMS schema-on-write: enforced during load, can speed queries but slows load and is rigid. **Example:** raw JSON logs stored first, structured later.

22) What is SerDe in Hive? Why do we need it? Give a real example.

Answer: SerDe = Serialize/Deserialize: (1) converts raw stored bytes into row/column representation for Hive, (2) lets Hive read/write different formats consistently, (3) supports complex structures by encoding/decoding. **Example:** reading JSON/CSV/ORC/Parquet with proper SerDe so Hive can interpret fields.

23) Give 3 considerations when choosing a SerDe/format.

Answer: (1) Overhead (encoding/decoding cost), (2) data integrity/ACID needs, (3) compatibility with engines/tools (Spark/Presto). **Example:** Parquet/ORC are columnar and better for analytics than raw text.

24) Why do ORC/Parquet get recommended for ACID / analytics workloads?

Answer: (1) Columnar storage improves scan performance, (2) supports metadata/statistics and compression, (3) required/recommended for transactional (ACID) tables with bucketing and configuration. **Example:** warehouse fact tables stored in ORC for faster SUM/GROUP BY.

G) DDL/DML behavior (but no heavy code)

25) Explain LOAD DATA behavior in 3 points (LOCAL, OVERWRITE, partition requirement).

Answer: (1) LOAD reads from HDFS unless **LOCAL** keyword used. (2) It **appends** unless **OVERWRITE** specified. (3) If target table is partitioned, you must specify the **PARTITION** destination. **Example:** load daily logs into partition (date,country).

26) INSERT INTO vs INSERT OVERWRITE: explain difference + example use-case.

Answer: INSERT INTO appends new files/rows to table/partition; INSERT OVERWRITE replaces data in the table/partition. **Example:** daily incremental loads use INSERT INTO; rebuilding a partition after correction uses INSERT OVERWRITE.

27) Why is “no row-level insert/update/delete” considered a limitation in classic Hive?

Answer: (1) Hive historically writes new files rather than changing individual rows, (2) updating single row would require rewriting large files/partitions, (3) analytics workloads prefer append/batch. **Example:** correcting one record in 1TB table is expensive vs OLTP DB.

28) What must be true for UPDATE/DELETE to work in Hive (ACID tables)? List 3 requirements.

Answer: Needs transactional setup: (1) table property `'transactional'='true'`, (2) bucketing (for full ACID tables), (3) execution engine Tez/Spark (not MR) + HiveServer2 + Metastore concurrency config; ORC/Parquet recommended. **Example:** slowly-changing dimension table requiring updates must be ACID.

H) Compiler & Optimization (light concepts, still testable)

29) List the 4 compile stages: Parser → Semantic analyzer → Plan generator → Optimizer. What does each do?

Answer: Parser converts query to parse tree; Semantic analyzer validates schema/columns via metastore; Plan generator makes logical plan (operators); Optimizer rewrites plan for efficiency (multi-join combine, repartition, prune columns, predicate pushdown). **Example:** pushing filters early reduces data read.

30) Explain "predicate pushdown" in 3 points + example.

Answer: (1) Apply WHERE filters **as close to table scan as possible**, (2) reduces data shuffled/processed later, (3) improves runtime for joins/group-by. **Example:** filtering `country='PK'` before join avoids joining irrelevant rows.

31) Explain "column pruning" in 3 points + example.

Answer: (1) Select only required columns early, (2) reduces IO (especially in columnar formats), (3) reduces network/shuffle. **Example:** query needs only `customer_id` and `price`, so prunes 30 other columns.

32) Why does Hive talk about "multiway join into single MR job"? When is it possible?

Answer: If multiple joins share the same join key, optimizer can combine them into one multiway join, reducing jobs/stages, improving performance. **Example:** joining three tables on `user_id` can be planned as one job instead of multiple.

I) "DAG" topic — sir said skip, but your lecture contains it; here are *concept-only* questions (in case he still asks)

33) Why represent query plan as a DAG? Give 4 reasons.

Answer: DAG helps (1) dependency management, (2) optimization by eliminating unnecessary steps, (3) parallel execution of independent branches, (4) fault tolerance: rerun only dependent nodes, and (5) better resource allocation because dependencies are known. **Example:** currency conversion and regional summaries run in parallel before final report merge.

J) Query behavior concepts (no long HQL)

34) ORDER BY vs SORT BY in Hive: explain difference + performance impact.

Answer: ORDER BY performs **total ordering** of entire result => slow/poor performance. SORT BY does **partial ordering per reducer** (each reducer output sorted), faster. **Example:** ORDER BY for final top-10 report; SORT BY for distributed preprocessing.

35) Why should join conditions NOT be placed in WHERE clause in Hive? (scenario)

Answer: Hive may not optimize it well; it can compute a full cartesian product before filtering, causing huge blow-up. Always put join condition in ON so optimizer understands join. **Example:** joining large clickstream tables incorrectly can generate massive intermediate data.

36) “Put the biggest table last in multi-table joins.” Why? Give 3 reasons.

Answer: When joining multiple tables, reducer can stream the last table and buffer smaller ones; placing biggest last reduces memory pressure, improves streaming efficiency, and avoids buffering huge table. **Example:** join small dimension tables first, stream big fact table last.

37) Hive limitation: subqueries only in FROM clause. Why does that matter? Give an example workaround idea.

Answer: You can’t use correlated subqueries freely; you restructure using derived tables/CTEs in FROM and JOIN. **Example:** compute department averages in a subquery in FROM then join back to employees.

K) Scenario/Numerical-style (strict, tricky)

38) Quasi-numerical: You have a table partitioned by (year, month). Data spans 5 years. Query filters `year=2024` only. Explain exactly what Hive reads and why.

Answer: Hive reads only partitions (directories) where `year=2024` (all months under that year). It prunes other years because partition columns are encoded in folder paths; metastore lists partition locations. **Example:** it skips `year=2023` folder entirely.

39) Bucketing numerical: You bucket by `customer_id` into 8 buckets. What does “hash(cid) mod 8” guarantee? Give 3 consequences.

Answer: Guarantees: (1) same `customer_id` always maps to same bucket file, (2) aggregations/group-by can be parallelized bucket-wise, (3) joins on `customer_id` can avoid full reshuffle if both tables bucketed similarly. **Example:** counting orders per customer runs each bucket independently.

40) Scenario: Two teams (Spark + Hive) use the same HDFS dataset. One team wants to change Hive schema without touching files. What Hive feature enables this and what table type is safest?

Answer: Use **schema-on-read** to interpret same files with multiple schemas, and use an **external table** so Hive changes don't risk deleting shared data. **Example:** raw JSON logs used by Spark; Hive creates external table with a new projection/schema for analysts.

Micro "must-hit" checklist (based on sir's exam message)

For Hive, you must be able to write confidently on:

- Partitioning + pruning
- Bucketing + hash(cid) mod N behavior
- External vs managed tables (DROP behavior)
- How Hive links HDFS files ↔ tables (warehouse + metastore metadata)
- Basic architecture terms (Driver/Metastore/CLI/JDBC/ODBC)