

0) The “why Hive exists” story (start from scratch)

The pain before Hive = MapReduce was painful for analysts

In Hadoop days, if you wanted “total sales by country”, you had to write **verbose Java MapReduce**, understand distributed systems, and it was slow/overkill for simple queries. The lecture lists problems like: complex Java, imperative programming, no SQL, high latency, long job stack, no schema management, and weak BI connectivity.

Hive’s purpose

Hive is a data warehouse layer on top of Hadoop that lets you run **SQL-like queries (HiveQL / HQL)** on data stored in **HDFS**.

So analysts can query large data like they do in a relational DB: **SELECT ... FROM ... WHERE ... etc.**

1) What Hive actually is (exam definition)

Definition (write this in exam)

Apache Hive is a data warehouse infrastructure built on Hadoop that stores data in HDFS and provides HiveQL (SQL-like language) to summarize and query large datasets.

One-liner: “Hive gives SQL on HDFS”

- Data physically sits in **HDFS files**
- Schema/metadata sits in **Hive Metastore (RDBMS)** → this is your “logical layer” idea

That “split” is SUPER important for your exam.

2) The MOST IMPORTANT concept your sir hinted: Logical vs Physical layer

Physical layer (real data)

Your table’s rows are not “rows in a database server”.
They are **files in HDFS**.

Logical layer (schema)

Hive stores:

- table name, columns + types, file format, SerDe, partitions info, ownership, etc.
in the **Metastore**, which is an **RDBMS-based system catalog**.

“How HDFS files link to tables” (EXAM ANSWER)

When you `CREATE TABLE`, Hive creates/points to an **HDFS directory** for that table, and the Metastore stores metadata telling Hive:

- where the directory is
- how to read files (format + SerDe)
- what columns exist

So when you query, Hive reads the files from that directory using that metadata.

3) Hive data organization: Table → Partition → Bucket (super exam)

A) Tables

A Hive table corresponds to an **HDFS directory**.

B) Partitions (must know)

Partitioning = split table into subfolders based on a partition column.

Example from slides:

If table T is in `/wh/T`, and partitioned by `ds` and `ctry`, a partition is stored like:

`/wh/T/ds=20090101/ctry=US`

Why it matters: Query becomes faster because Hive reads only the needed partition(s), not whole dataset. The example shows filtering by country/date reads only that partition.

Exam-ready 3 points for Partitioning

1. Reduces data scanned (partition pruning)
2. Improves performance for filter queries on partition columns
3. Data is physically organized as directories per partition

C) Bucketing (must know)

Bucketing = inside a partition, data is split into a fixed number of files (“buckets”) using hash(column) mod N.

Slide example: `hash(customer_id) mod 8` → goes into one of 8 bucket files.

Why bucketing helps (exam)

- Helps **joins and aggregations** because matching keys land in the same bucket file, enabling parallelization.
- Helps **sampling** (TABLESAMPLE), because Hive can pick a bucket without scanning everything.

Partitioning vs Bucketing (common compare question)

- Partitioning = directory-level split based on column values (coarse)
 - Bucketing = file-level split using hash into fixed N files (fine)
-

4) Managed (Internal) vs External tables (VERY IMPORTANT)

Managed / Internal table

Hive “owns” both:

- metadata
- and actual HDFS data inside warehouse path

If you **DROP** it → metadata + data deleted (lifecycle controlled by Hive).

External table

Hive manages **only metadata**; data is outside Hive’s control and typically in some existing HDFS path you specify. Dropping external table deletes **only metadata**, not the data.

When to use external (exam answer)

- When data is shared with other Hadoop tools
 - When you want multiple schemas on same data
 - When you don’t want Hive to delete the files
-

5) Schema-on-read (a core Hive identity)

Hive is **schema-on-read**:

- schema is enforced when data is read at query time
 - same raw data can be read with different schemas
- RDBMS is schema-on-write (schema enforced at load time).

Exam 3 points for schema-on-read

1. Flexible for messy/big logs
 2. Faster ingestion (no heavy validation at load time)
 3. Validation/parsing happens at query time (can increase query cost)
-

6) Hive architecture (what components do)

Main components (learn these labels)

- **External interfaces:** CLI (Beeline), Web UI, JDBC/ODBC (BI tools connect here)
- **Driver:** manages the lifecycle of a HiveQL statement (compile → optimize → execute)
- **Metastore:** metadata catalog (DB, table schema, partitions, file format, SerDe info)
- Execution happens on Hadoop via YARN (engine runs distributed).

Query lifecycle (high-yield, short)

Hive does:

1. Parse HQL → AST
 2. Semantic analysis using Metastore
 3. Logical plan
 4. Optimization
 5. Physical plan → MR/Tez/Spark
 6. Execute on YARN
-

7) Execution engines (know names + one line each)

Hive can execute using:

- **MapReduce** (original, slow)
- **Tez** (DAG-based faster)
- **Spark** (uses Spark engine)
- **LLAP** (in-memory caching for lower latency; Hive-on-Tez idea)

8) Hive warehouse path (where tables live)

Default Hive warehouse in HDFS: `/user/hive/warehouse`

- tables are subdirectories
 - partitions are subdirectories inside table directory
-

9) Loading vs Inserting data (must distinguish)

LOAD DATA

Moves/loads a file into Hive table (from HDFS by default; LOCAL loads from local FS). Appends by default; OVERWRITE replaces.

INSERT

Loads data from a query result into a table/partition.

CTAS (Create Table As Select)

Create a table and populate it at the same time.

10) Hive is NOT for real-time OLTP (very common theory Q)

Hive is designed for **batch processing**:

- high latency (slow start/finish compared to RDBMS)
 - high throughput (process lots of data per batch)
-

11) ACID (only the idea you need)

Old Hive had weak row updates, but newer Hive supports **UPDATE/DELETE/MERGE only for ACID/transactional tables**, with requirements like ORC/Parquet, bucketing,

'transactional' = 'true', and Tez/Spark engine.
(You just need to know: **row-level update/delete needs ACID setup.**)

12) Key “terms list” you MUST memorize (these are your marks words)

- Hive, HiveQL/HQL, data warehouse, HDFS, schema-on-read, Metastore, Driver
 - Managed/Internal vs External table
 - Partitioning (partition pruning), Bucketing (hash mod N, TABLESAMPLE)
 - Warehouse directory (`/user/hive/warehouse`)
 - LOAD DATA vs INSERT vs CTAS
 - Execution engines: MapReduce, Tez, Spark, LLAP
- Everything above is directly in your lecture slides.
-

Exam-style mini Q/A (your teacher style: 3 points + example)

Q1) Why do we use Hive instead of writing MapReduce?

Answer (3 points):

1. Hive provides SQL-like querying (HiveQL) so analysts don't write verbose MapReduce.
 2. It manages schema via Metastore (schema + table metadata), which raw MapReduce lacks.
 3. It supports data warehouse tasks: summarization and ad-hoc queries at scale.
- Example:** “Total sales by country/date” is a simple Hive query but would need custom MR jobs.

Q2) Explain how Hive links HDFS data to tables (logical vs physical).

Answer:

- Physical data = flat files stored in HDFS directories (warehouse/table folders).
- Logical schema = Metastore stores table/partition metadata, file format, SerDe, etc.
- Query uses Metastore metadata to read/deserialize correct HDFS files at runtime.

Q3) Partitioning vs Bucketing (with an example).

Answer:

- Partitioning stores data in separate directories by column values (e.g., country/date), so queries read only relevant partitions.
- Bucketing hashes rows into fixed number of bucket files, improving joins/aggregations and sampling.

Example: `country` as partition, `customer_id` as bucketed key.

Q4) Managed vs External tables (and when to use external).

Answer:

- Managed: Hive controls data + metadata; DROP deletes both.
 - External: Hive controls metadata only; DROP keeps data files.
- Use external** when data is shared or you don't want accidental deletion.