1. Why Hive Was Needed

- Challenges of MapReduce
- Limitations of raw MapReduce for analysts
- Why SQL-like systems were required

---

2. Introduction to Hive

- What is Hive?
- Why Facebook built it
- Hive vs MapReduce
- Batch processing nature of Hive
- Hive execution engines:
    - MapReduce
    - Tez
    - Spark
    - LLAP

---

3. Hive Architecture

- Components:
    - CLI / Beeline
    - Web UI
    - JDBC / ODBC

    - Thrift Server
    - Driver
    - Compiler
    - Metastore
- How Query flows through Hive
- Why Hive uses a DAG
- Role of YARN

---

4. Hive Data Model

- Tables
- HDFS directories
- Schema stored in Metastore
- Serialization and Deserialization
- Lazy serialization

---

5. Hive Partitions

- What is partitioning?
- Purpose of partitioning
- How partitioning improves query performance
- Directory structure of partitions
- Partitioned table examples

- Partition filters in queries

---

6. Hive Bucketing

- What is bucketing?
- Hash-based partitioning inside partitions
- Difference between partitioning and bucketing
- Bucketing for joins and aggregations
- Reducer allocation for buckets
- BUCKETED table examples

---

7. Hive Data Types

- Primitive data types
- Complex data types:
    - ARRAY
    - MAP
    - STRUCT
- How to access elements

---

8. DDL in Hive

- CREATE TABLE
- DROP TABLE

- ALTER TABLE
- TRUNCATE
- Difference between Managed vs External tables
- What happens to data during DROP

---

9. Managed Tables vs External Tables

- Difference in data ownership
- When to use which
- Directory behavior
- Data deletion behavior
- Use cases

---

10. Hive Schema Concepts

- Schema-on-read vs Schema-on-write
- Comparison with RDBMS

---

11. Loading Data into Hive

- LOAD DATA
- LOAD LOCAL DATA
- OVERWRITE vs APPEND
- Partitioned data loading

---

12. Inserting Data

- INSERT INTO
- INSERT OVERWRITE
- Multi-partition insert
- CTAS (Create Table As Select)

---

13. Hive Querying

- SELECT
- WHERE
- GROUP BY
- HAVING
- DISTINCT vs ALL
- ORDER BY vs SORT BY
- LIMIT

---

14. Hive Joins

- Join types
- Why join condition must not be in WHERE
- Cartesian Product issue
- Multiple table joins

- Join optimization rule (big table last)
- Left Semi Join

---

15. HiveQL & Functions

- HiveQL basics
- UDF (User Defined Functions)
- UDAF (User Defined Aggregation Function)
- UDTF
- TRANSFORM queries

---

16. Hive Metastore

- What is metastore?
- Role of metadata
- Embedded / Local / Remote Metastore
- Apache Derby default
- Production setup (MySQL, PostgreSQL)

---

17. Hive Warehouse

- Default warehouse directory
- How tables and partitions stored
- External data locations

---

18. Serialization & Deserialization (SerDe)

- What is serialization?
- What is deserialization?
- Why it is required
- Examples with JSON and Python object

---

19. Hive ACID Support

- What operations Hive supports:
    - INSERT
    - UPDATE
    - DELETE
    - MERGE
- Requirements:
    - ORC/Parquet
    - Bucketing
    - transactional=true
    - Tez/Spark engine

---

20. Hive Limitations

- No correlated subqueries in WHERE

- No stored procedures
- No real-time usage
- Slow startup time
- OLTP vs OLAP mismatch

---

21. Advanced Hive Concepts

- Hive Stinger
- MPP style execution
- Hive without MapReduce

topic1

TOPIC 1 — Challenges of MapReduce & Why Hive Was Needed

(From slides: pages 3–4)

BDA.Lecture.10

⭐ 1. What is MapReduce? (Quick recap)

MapReduce = a programming model in Hadoop used to process large datasets by writing **Java code** where you manually define the Map and Reduce steps.

Sounds simple, but…

In reality? **PAINFULLY complex for analysts.**

⭐ 2. Why MapReduce Was a Problem

Your teacher will expect *very specific* points from the slide.
Here is each one **explained simply + with examples**:

---

1. Requires complex, verbose Java

- Writing MapReduce jobs means writing **dozens of lines of Java** even for simple things.
- Example:
  Counting records means writing a Mapper class, Reducer class, config code, etc.

This makes MapReduce **not analyst-friendly.**

---

2. Requires understanding distributed computing

To write MR jobs, you must understand:

- Distributed systems
- Partitioning
- Shuffle/sort
- Reducers
- Fault tolerance
  This is too much for someone who just wants to run a query like:
  `SELECT AVG(price) FROM sales;`

---

3. Imperative programming (step-by-step)

MapReduce forces you to **tell the system EXACTLY HOW to compute something**, not just what you want**.**

Example:
If you want average salary, you must:

- Write mapper → emit (salary, 1)
- Write reducer → sum salaries + sum counts
- Divide inside reducer

This is extra work compared to SQL's declarative style**.**

---

4. No SQL support

MapReduce has **no SQL layer.**
Analysts cannot run:

- SELECT

- JOIN
- GROUP BY
- WINDOW functions

Everything must be coded manually.

This was a MAJOR motivation for Hive.

---

5. Optimized only for batch → High Latency

MapReduce jobs take **time to start** because:

- JVM starts
- Containers allocated
- Task trackers sync

Even small jobs (<100MB) feel slow.
This makes MR unsuitable for interactive queries.

---

6. Job execution stack is too long for simple queries

Small queries like:

```
SELECT COUNT(*) FROM table;
```

still require:

- MapJob
- ReduceJob
- Shuffle
- Writes to HDFS

So even tiny operations have **long setup time.**

---

7. No schema management

MapReduce works on raw files.
There is:

- No table structure
- No columns
- No metadata

Analysts must manually parse data.

---

8. No connectivity with BI tools

You cannot connect:

- Tableau
- Power BI
- Excel
- JDBC

MapReduce cannot integrate with enterprise tools.

Hive solves this through JDBC/ODBC.

---

9. Maintenance burden

- Lots of manual code
- Hard to debug

- Hard to upgrade
- Job failures difficult to trace

MR becomes painful at scale.

⭐ 3. How Hive Solves These Problems

(You don't need to memorize this now — just overview)

Hive provides:

- SQL-like language (HiveQL)
- Schema and table metadata
- Query optimizer
- Runs on top of MR/Tez/Spark
- Much easier for analysts
- Interactive-like querying (compared to MR)

⭐ SUMMARY — 5 POINTS TO WRITE IN THE QUIZ

If teacher asks:

"Why do we need Hive?" / "Problems with MapReduce?"

Write any **5 of these** (safe, high-quality):

1. MapReduce requires **complex, verbose Java code**, even for simple queries.

2. It demands deep **understanding of distributed systems**, making it hard for analysts.
3. It follows **imperative programming**, forcing users to specify steps manually.
4. No built-in **SQL support**, so common queries require writing full programs.
5. Very **high latency**, because MR is batch-oriented and slow to start.
6. No **schema management**; data is raw and unstructured for MR.
7. No easy **integration with BI tools** (Tableau/Power BI).
8. Long and heavy **job execution pipeline** even for tiny operations.

You only need 3–4 in the exam — but here you have 8 strong ones.

⭐ Exam-Style Questions (with 4–5 points each)

Q1. Why is MapReduce difficult for data analysts?

Answer (your 5 points):

1. Requires writing complex Java code.
2. No SQL interface, everything must be coded manually.
3. Imperative programming makes simple tasks lengthy.
4. Analysts must understand distributed computing details.
5. MR's high latency makes it unsuitable for interactive analysis.

Q2. List any 4 limitations of MapReduce.

Answer:

1. High latency due to batch processing.
2. No schema or metadata management.
3. Not compatible with BI tools.
4. Maintenance overhead for large MR codebases.

Q3. Why was Hive developed?

Answer:

1. To give analysts a SQL-like interface on Hadoop.
2. To remove complexity of writing MapReduce code.
3. To add schema, metadata, and table management.
4. To optimize queries using Tez/Spark instead of raw MR.
5. To allow easier integration with BI systems.

Q4. Explain “imperative vs declarative” in context of MapReduce and Hive.

Answer:

1. MapReduce is imperative → user specifies HOW to compute results.
2. Hive is declarative → user only specifies WHAT data they want.
3. Declarative style reduces coding effort.
4. Query optimizers can handle execution plans better.

Q5. Why is MapReduce unsuitable for real-time analytics?

Answer:

1. High job startup time.
2. Batch execution model.
3. JVM/container initialization overhead.
4. Cannot support sub-second or low-latency queries.

topic2

TOPIC 2 — Hive Basics & Hive Execution Engines

This topic usually gives questions like:
• *"What is Hive?"*
• *"Explain the base of Hive."*
• *"Name and explain Hive execution engines."*
• *"Hive is not a real-time system — why?"*

Your teacher will nitpick words like **batch**, **warehouse**, **throughput**, **latency**, and **DAG**. I'll explain those very clearly.

🔷 1. What is Apache Hive?

From your slides:
Hive is a data warehouse infrastructure built on top of Hadoop.
(Page 5)

In simple words:

Hive is a tool that sits on top of Hadoop and lets users run **SQL-like queries** on huge data stored in **HDFS**, without writing complex MapReduce code.

If Hadoop is the *engine*, Hive is the *steering wheel*.

Key points from your lecture:

Hive:

1. Is built **on top of Hadoop**
2. Stores data inside **HDFS**
3. Provides a **SQL-like language** called **HiveQL (HQL)**
4. Converts Hive queries into **execution plans** for Hadoop
5. Supports **ad-hoc querying** (like `SELECT * FROM table`)

6. Was originally developed at **Facebook**
7. Is mainly for **big data analytics**, not real-time

How Hive actually works (basic flow):

When you write:

```
SELECT SUM(price) FROM sales;
```

Hive does this internally:

1. Parses your query
2. Checks schema from metastore
3. Optimizes query
4. Converts it into MapReduce/Tez/Spark jobs
5. Executes it on Hadoop
6. Returns result to you

So:
You write SQL → Hive converts → Hadoop executes.

🔷 2. Why Hive is Called a Data Warehouse

From the slide:
"Data warehouse infra on top of Hadoop"

Hive behaves like a **data warehouse**, because:

1. It stores **historical data**
2. It supports **aggregation and summarization**

3. It is used for **analysis**, not daily transactions
4. It focuses on **querying large volumes of data**

This means Hive is **OLAP** (analytics), not **OLTP** (transactions).

🔷 3. Hive Execution Engines

Your slides list 4 main execution engines (Page 5–6):

1. **MapReduce (older default)**
2. **Apache Tez**
3. **Apache Spark**
4. **LLAP (Live Long and Process)**

Let's break them one by one.

🔹 1. MapReduce Engine

Definition:
Oldest execution engine for Hive.

How it works:

- Every Hive query is converted into MapReduce jobs.
- Reliable but **slow.**

Key Points:

1. Very stable
2. High latency

3. Good for batch processing
4. Not good for interactive queries

🔹 2. Apache Tez Engine

From slide:

> "DAG-based engine — faster"

What this means:
Instead of breaking the query into many MR jobs, Tez creates a **DAG (Directed Acyclic Graph).**

This allows:

1. Query optimization
2. Combining multiple steps
3. Faster execution
4. Lower latency

Example advantage:
In MapReduce → you may need 3–4 jobs
In Tez → it becomes one optimized graph

🔹 3. Apache Spark Engine

Instead of MapReduce, Hive can use **Apache Spark** as backend.

Why Spark is better:

1. In-memory processing
2. Faster than disk-based MR
3. Good for iterative tasks

Hive just delegates execution to Spark instead of MR.

🔹 4. LLAP (Live Long and Process)

From slide:

> "In-memory caching for low-latency"

LLAP is used when:
• You want **faster interactive queries**
• Data is frequently accessed
• You want Hive to behave more like a near real-time system

LLAP keeps frequently accessed data cached in memory.

🔷 4. Hive Key Principles

From slides 12–15:

Hive:

1. Enables relational-style querying
2. Provides SQL-like language
3. Allows custom mappers & reducers
4. Supports schema-on-read
5. Enables ETL easily
6. Handles partitioning & bucketing

It basically **bridges the gap** between Hadoop and SQL users.

🔷 5. Hive is NOT…

Very important slide (Page 16):

Hive is **NOT**:

1. Real-time system
2. Transaction-heavy system
3. OLTP engine

Instead, Hive is:
• **High throughput** (can process huge data in one go)
• **High latency** (slow to start query)

Even small data (<100MB) still runs slowly because Hive is batch oriented.

Exam phrase you should remember:

> Hive has **high throughput but high latency**

Your teacher will LOVE this line.

✅ Exam-Style Questions (With Strong Answers)

Q1: What is Apache Hive?

Answer (write any 4–5 of these):

1. Hive is a data warehouse tool built on top of Hadoop.
2. It allows users to query big data using SQL-like HiveQL.
3. It stores data in HDFS and manages schema via metastore.
4. It converts HQL queries into MapReduce/Tez/Spark jobs.
5. It was developed by Facebook for large-scale analytics.

Q2: Name and explain Hive execution engines.

Answer:

1. **MapReduce –** Default, slow, disk based, batch engine.
2. **Tez –** DAG-based execution engine, faster and optimized.
3. **Spark –** Uses Apache Spark for in-memory processing.
4. **LLAP –** Uses in-memory caching for low-latency querying.

Q3: Why Hive is not suitable for real-time systems?

Answer:

1. It is designed for batch processing.
2. It has high query startup latency.
3. Each query passes through heavy execution pipeline.
4. Even small datasets take time to execute.
5. It prioritizes throughput instead of low latency.

Q4: Explain why Hive is called a data warehouse infrastructure.

Answer:

1. It stores large historical data.
2. It supports summarization and analytics.
3. It works on structured data stored in HDFS.
4. It is used for reporting and analysis, not daily transactions.

Q5: Difference between latency and throughput in Hive.

Answer:

1. Latency = time taken to start + finish a query.
2. Throughput = amount of data processed per batch.
3. Hive has high latency but very high throughput.
4. RDBMS has low latency but lower throughput than Hive.

topic3

🌟 TOPIC 3: Hive Architecture

You told me the components you want covered:

✔ CLI / Beeline
✔ Web UI
✔ JDBC / ODBC
✔ Thrift Server
✔ Driver
✔ Compiler
✔ Metastore
✔ Query Flow
✔ Why Hive uses a DAG
✔ Role of YARN

Let's go in order.

1️⃣ External Interfaces (How users interact with Hive)

These are the **entry points** to Hive.

A. CLI / Beeline

From slide:

> CLI allows creating, inspecting schema and query tables etc. All commands and queries go to the Driver
>
> BDA.Lecture.10

Explanation:

- CLI = Command Line Interface
- Beeline is a **newer CLI** that connects to HiveServer2.
- It allows you to:
    - Run queries

    - Create / drop tables
    - Inspect metadata
    - View partitions

Why teacher cares:
He may ask: *Why is Beeline replacing old CLI?*

Answer points:

- Beeline works with HiveServer2.
- Supports authentication & multi-user.
- Better for production.

B. Web UI

Slide text:

Web UI: Management interface

BDA.Lecture.10

Explanation:

- A graphical interface used to:
    - Browse tables
    - Monitor queries
    - Manage schemas
- Example: Hue in Cloudera.

Exam phrasing tip:
Say: *Web UI provides graphical management of Hive, useful for administrators and analysts.*

C. JDBC / ODBC

From slides:

> API: JDBC, ODBC
>
> BDA.Lecture.10

Explanation:
These allow external applications to connect to Hive.

Examples:

- Power BI
- Tableau
- Java Apps
- Python apps

Important definition to memorize:

- **JDBC**: Java Database Connectivity
- **ODBC**: Open Database Connectivity

These send queries to HiveServer → then into Driver.

D. Thrift Server

From slide 29:

> Thrift is a framework for cross-language services… supports clients in other languages
>
> BDA.Lecture.10

Simple explanation:
Thrift Server allows:

- Clients written in Python, C++, Java, etc.
- To communicate with Hive using a common protocol.

Basically:
👉 It acts as a **translator & communication bridge.**

2️⃣ Core Hive Components

E. Driver

From slide:

> Driver: manages the life cycle of a HiveQL statement during compilation, optimization and execution
>
> BDA.Lecture.10

Think of Driver like a manager.

It does:

1. Receives query from CLI/JDBC/etc.
2. Sends to compiler
3. Tracks execution status
4. Returns results to user

F. Compiler

From slides:

> Compiler: translates HiveQL into a plan → DAG of MapReduce jobs
>
> BDA.Lecture.10

The compiler does several steps:

1. Parser – converts query into parse tree
2. Semantic analyzer – checks tables, columns, types
3. Plan generator – builds logical plan
4. Optimizer – optimizes the plan
5. Physical plan generator – creates DAG of jobs

Important optimization examples from slide:

- Combine joins into multiway joins
- Predicate pushdown
- Column pruning

Teacher LOVES this line:
👉 *Compiler converts HiveQL into a DAG of MapReduce/Tez/Spark jobs.*

G. Metastore

From slide:

> Metastore: System catalog – contains metadata about Hive tables
>
> BDA.Lecture.10

This is CRUCIAL. Teacher WILL test this.

Metastore stores:

- Database names
- Table schemas
- Column names and types
- Partitions

- Storage format
- SerDe info

Types of Metastore according to slides:

1. **Embedded –** inside Hive, uses Apache Derby, single-user
2. **Local** – shared DB, multiple Hive services
3. **Remote –** standalone server for big clusters

Exam phrase to remember:

> Metastore is the centralized repository of Hive metadata.

3️⃣🔄 How a Hive Query Flows Through the System

This is direct from slides page 6 + architecture diagrams.

Let's do it like a story:

Step-by-Step Query Flow

1. User writes a query
   through CLI / Web UI / JDBC / ODBC
2. Query goes to **Driver**
3. Driver sends it to **Compiler**
4. Compiler:
    - Parses it
    - Validates schema with **Metastore**
    - Generates logical plan
    - Optimizes

    - Creates physical plan (DAG)
5. Physical Plan is handed to **Execution Engine**
6. Execution Engine submits jobs to **YARN**
7. YARN schedules jobs on cluster (MapReduce / Tez / Spark)
8. Output written to HDFS
9. Driver sends results back to User

📌 This flow is shown in Hive Architecture diagrams (page 22 & 28)

BDA.Lecture.10

4️⃣🧱 Why Hive Uses a DAG

From slide page 25:
Reasons listed:

1. Dependency management
2. Optimize – eliminate unnecessary steps
3. Parallel execution
4. Fault tolerance
5. Effective resource allocation

BDA.Lecture.10

Simple Explanation:

DAG = Directed Acyclic Graph

Hive queries are broken into steps (operators):

- Filter

- Join
- Group By
- Aggregate

These steps are connected like a graph.

Why DAG is useful:

1. **Dependency Management**
   Each step waits only for its dependencies.
2. **Parallel Execution**
   Independent steps can run together.
3. **Optimization**
   Hive removes useless steps.
4. **Fault Tolerance**
   If one node fails, only dependent tasks rerun.
5. **Better Resource Use**
   YARN knows dependencies, so better scheduling.

5⚙️ Role of YARN in Hive

From slide:

> Query Execution is submitted to YARN for distributed execution
>
> BDA.Lecture.10

What YARN actually does:

1. Allocates resources (CPU, memory)
2. Schedules containers
3. Manages cluster-wide execution

4. Handles failures

Think of YARN as:
👉 The **resource manager of the Hadoop cluster.**

🧠 Exam-Style Short Questions

I'll give you exactly the kind of questions your teacher may ask:

Q1. List the main components of Hive architecture.

Answer (write 4–5):

1. CLI/Beeline & Web UI
2. Driver
3. Compiler
4. Metastore
5. Thrift Server
6. Execution Engine
7. YARN
8. HDFS

Q2. What is the role of the Hive Metastore?

Answer:

1. Stores metadata for all Hive databases and tables.
2. Contains schema, column datatype, partitions, and storage format.

3. Helps compiler verify query correctness.
4. Centralized catalog used by all Hive components.

Q3. Explain the role of Driver in Hive.

Answer:

1. Receives queries from client interfaces.
2. Manages query lifecycle.
3. Monitors execution progress.
4. Returns final result to the user.

Q4. Why does Hive use a DAG?

Answer:

1. Manages dependencies between query tasks.
2. Supports parallel execution of tasks.
3. Enables optimization by removing unnecessary operations.
4. Improves fault tolerance.
5. Helps efficient resource allocation.

Q5. What is the role of YARN in Hive?

Answer:

1. Allocates cluster resources.

2. Manages execution of MapReduce/Tez/Spark jobs.
3. Handles scheduling across nodes.
4. Provides fault tolerance.

Q6. Explain query flow in Hive architecture.

Answer:

1. Query sent through CLI/JDBC/Web UI.
2. Driver receives it.
3. Compiler parses and optimizes it.
4. Metadata verified via Metastore.
5. Physical plan generated as DAG.
6. Jobs executed via YARN.
7. Results returned to user.

topic4

✅ Now: Topic 4 — Hive Data Model

This topic appears mainly across:

- Page 7: Hive table organization
- Page 8–10: Data model, partitions, buckets
- Page 20–21: Serialization / Deserialization
- Page 58–60: Hive Warehouse & Schema-on-read

BDA.Lecture.10

We'll go **section by section.**

4.1 Hive Tables

Basic Idea:

In Hive, data is organized into **tables**, just like SQL databases – but with a big Hadoop twist.

Key points your teacher expects:

1. A Hive table is **not stored inside Hive**
   It is actually stored as a **directory in HDFS.**
2. A table has **two parts**:
   - **Data** → stored as files in HDFS
   - **Schema** → stored separately in the Hive Metastore
3. Hive tables:
   - Can have **primitive data types** (int, string, float etc.)

- Also support **complex types** like Array, Map, Struct
  (shown in the data type slides)

📌 Example from lecture:
A Hive table `T` is stored at:

```
/wh/T/
```

Each table automatically gets its own HDFS folder.

How teacher might trap you:

He might ask:

> "Where is the data of a Hive table stored, and where is the schema stored?"

Answer structure (ideal):

1. Data stored in **HDFS files**
2. Schema stored in **Hive Metastore (RDBMS)**
3. Hive links them during query execution

4.2 HDFS Directories

Your teacher emphasized this heavily.

Every Hive table corresponds to an HDFS directory.

Example from your slides:

If table = `ecommerce_sales`

Its data is in:

```
/warehouse/ecommerce_sales/
```

If it's partitioned:

```
/warehouse/ecommerce_sales/country=Pakistan/order_date=2025-11-02/
```

BDA.Lecture.10

Why this matters:

1. Hive **does not store data itself**
2. It just manages it *logically*, while HDFS stores it *physically*
3. This design allows Hive to scale for huge data

4.3 Schema Stored in Metastore

This is where students lose marks.

Your teacher's wording:

> "Schema is stored as metadata in RDBMS."

Meaning:

1. Hive keeps metadata in a database like:
    - MySQL
    - PostgreSQL
    - Apache Derby (default)
2. This includes:
    - Column names
    - Data types

- Table owner
- Location in HDFS
- Partition information
- Serialization format

3. The Hive Metastore acts like a **catalog**

He literally explains it like:

> Metastore is the system catalog which contains metadata about tables.

BDA.Lecture.10

Important phrase to remember:

> Hive separates **metadata from actual data**, unlike traditional DBMS.

4.4 Serialization & Deserialization (SerDe)

This is a favorite exam area.

What it means:

Serialization = Convert data into storable format
Deserialization = Convert it back for reading

Hive uses SerDe to:

1. Read data from files
2. Convert it into table format
3. Send results to user

From your lecture:

- Supports Parquet
- ORC
- TextFile
- SequenceFile

Also supports complex structures like:

- JSON
- Maps
- Arrays

BDA.Lecture.10

Why your teacher cares:

1. Hive stores raw data in files
2. SerDe tells Hive how to interpret it
3. Without SerDe, Hive doesn't know how to read columns

4.5 Lazy Serialization

This term is **100% exam bait.**

Meaning:

Hive does **not deserialize everything immediately**.

Instead:

1. It waits until the column is actually needed.

2. Saves CPU and memory.
3. Makes Hive efficient for big data.

Example:
If your query reads only 2 columns out of 50, Hive only deserializes those 2.

Why? Because Big Data is too large to waste resources.

This concept is directly mentioned in your data model slide:

> Hive provides built-in serialization formats (lazy serialization – delayed until needed)
>
> BDA.Lecture.10

📝 Exam-Style Questions (With 4–5 Solid Points)

I'll give you 7 questions now.

Q1: What is the Hive Data Model?

Answer (pick best 3-4):

- Data is organized into tables
- Each table maps to an HDFS directory
- Schema is stored separately in Hive Metastore
- Supports primitive and complex data types
- Uses schema-on-read approach

Q2: How does Hive store tables physically?

Answer:

- As directories in HDFS
- Each partition becomes a subdirectory
- Buckets are stored as files inside partition folders
- Metadata for these is stored in Metastore

Q3: What is the role of Metastore in Hive's data model?

Answer:

- Stores schema and metadata
- Keeps information about table structure and location
- Acts as system catalog
- Allows different Hive components to share metadata

Q4: Explain Serialization & Deserialization in Hive.

Answer:

- Serialization converts data to storable format
- Deserialization reconstructs data for queries
- Enables Hive to read complex data formats
- Uses SerDe libraries for flexible file formats

Q5: What is Lazy Serialization?

Answer:

- Hive delays deserialization until column is accessed
- Improves performance
- Saves memory and CPU
- Useful for wide tables with many columns

Q6: Difference between Data and Schema storage in Hive?

Answer:

- Data stored in HDFS
- Schema stored in Metastore (RDBMS)
- Enables schema changes without touching data
- Supports schema-on-read flexibility

Q7: How does Hive manage large datasets efficiently?

Answer:

- Uses lazy deserialization
- Uses partitioning and bucketing
- Uses HDFS for scalable storage
- Separates schema from actual data files

topic5

5.1 What is Hive Partitioning?

Definition (exam-ready):
Partitioning in Hive is a technique of **dividing a table into subdirectories in HDFS based on the values of one or more columns**.

Instead of storing all rows in one big folder, Hive stores them as:

```
table_name/partition_column=value/
```

Example (simple):

If you partition by `year`, data is stored as:

```
/warehouse/sales/year=2022/
/warehouse/sales/year=2023/
/warehouse/sales/year=2024/
```

Each partition holds only rows for that year.

5.2 Purpose of Partitioning

Your teacher will expect performance + management points.

Main purposes:

1. **Reduce scanning of irrelevant data**
   Instead of reading the whole table, Hive reads only relevant partitions.

2. **Improve query performance**
   Queries become faster because less data is processed.

3. **Organize large datasets efficiently**
   Makes data management easier for very large tables.

4. **Faster data loading**
   You can load data partition-wise instead of loading the whole table.

5. **Better data maintenance**
   You can drop old partitions without touching whole table.

5.3 How Partitioning Improves Query Performance

This is the most important conceptual part.

Imagine this table:

- Sales table
- 10 years of data
- Billions of rows

If NOT partitioned:

Query scans entire dataset.

If partitioned by `year`:

Query only scans that year's folder.

How exactly performance improves:

Partition Pruning
Hive checks the WHERE clause.
If it sees:

```
WHERE year = 2023
```

it only reads:

```
/warehouse/sales/year=2023/
```

1.
2. **I/O Reduction**
   Fewer files = less disk reading = faster execution.

3. **Smaller input size for MapReduce/Spark**
   This reduces job startup and processing cost.

4. **Better parallelism**
   Different partitions can be processed in parallel.

5.4 Directory Structure of Partitions

From your slide diagrams, partitioning looks like this in HDFS:

For single partition:

```
/warehouse/sales/year=2024/
/warehouse/sales/year=2023/
```

For multiple partitions:

If table is partitioned by `country` and `year`:

```
/warehouse/sales/country=Pakistan/year=2023/
/warehouse/sales/country=Pakistan/year=2024/
/warehouse/sales/country=USA/year=2023/
/warehouse/sales/country=USA/year=2024/
```

Each level represents a partition column.

This structure is shown in the partitioned table storage diagram in your slides.

BDA.Lecture.10

5.5 Partitioned Table Examples

Example 1: Table Partitioned by Single Column

```
CREATE TABLE sales (
  order_id INT,
  amount FLOAT
```

```
)
PARTITIONED BY (year INT);
```

Then data for:

```
year = 2023
```

goes into:

```
/warehouse/sales/year=2023/
```

Example 2: Multiple Partition Columns

```
CREATE TABLE sales (
  order_id INT,
  amount FLOAT
)
PARTITIONED BY (country STRING, year INT);
```

Hive creates folder structure like:

```
/warehouse/sales/country=Pakistan/year=2023/
/warehouse/sales/country=Pakistan/year=2024/
```

5.6 Partition Filters in Queries

This is where students lose marks.

Partition filters are just **conditions on partition columns in WHERE clause.**

Example:

```
SELECT SUM(amount)
FROM sales
WHERE year = 2023;
```

Here:

- `year` is a partition column
- Hive reads only the `year=2023` partition

Important exam point:

If you DON'T include a partition filter:
Hive scans **ALL partitions** → very slow.

So always filter by partition columns.

📚 Exam-Style Questions (Short Answer Type)

I'll give you 8 strong ones.

Q1: What is partitioning in Hive?

Answer (write any 4):

- Dividing a table into subdirectories based on column values
- Each partition is a folder in HDFS
- Helps in faster query execution
- Improves data management
- Reduces the amount of data scanned

Q2: State 4 advantages of partitioning.

Answer:

- Faster query performance due to partition pruning
- Less I/O as only relevant partitions are scanned
- Easier data management and deletion
- Efficient loading of large datasets
- Improves parallel processing

Q3: How does partitioning improve query performance?

Answer:

- Reduces data scanned using partition filters
- Enables partition pruning
- Reduces I/O overhead
- Smaller dataset passed to execution engine
- Improves job execution time

Q4: What is partition pruning?

Answer:

- Process where Hive skips unnecessary partitions
- Occurs when query uses partition filter
- Reduces number of partitions read

- Improves performance significantly

Q5: Show HDFS structure of a partitioned table.

Answer:
For partitioned by `country` and `year`:

```
/warehouse/table/country=Pakistan/year=2023/
/warehouse/table/country=India/year=2024/
```

Explain:

- Each partition column creates folder structure
- Makes partition access efficient

Q6: What happens if partition filter is not used?

Answer:

- Hive scans all partitions
- Increases execution time
- Large I/O and CPU overhead
- Reduces performance significantly

Q7: Difference between partition column and normal column?

Answer:

- Partition column determines HDFS directory structure

- Normal column is just a field inside row
- Partition column affects physical data storage
- Normal column does not affect storage layout

Q8: Give an example of partitioned table creation.

Answer:

```
CREATE TABLE sales (
  id INT,
  price FLOAT
)
PARTITIONED BY (year INT);
```

Explain:

- Creates separate folders for each year
- Data separated physically in HDFS

topic6

✅ Topic 6: Hive Bucketing

6.1 What is Bucketing?

Definition (exam-ready):
Bucketing in Hive is a technique to **divide data inside each partition into fixed number of files (called buckets) based on a hash of a column.**

From your slide text:

> "Buckets: Data in each partition may be divided into buckets based on the hash of a column in the table. Each bucket is stored as a file in the partition directory."
>
> BDA.Lecture.10

So:

- Partition = folder
- Bucket = file inside that folder

Simple idea:

Partitioning divides data **horizontally** (by values like year, country).

Bucketing divides data **within a partition** into multiple files using hashing.

6.2 Hash-Based Partitioning Inside Partitions

Hive uses a hash function to decide **which row goes into which bucket.**

From your slide (page 10):

> Each bucket file corresponds to a hash(cid) mod 8

BDA.Lecture.10

How it works:

If:

- Number of buckets = 8
- Bucketing column = `customer_id`

Hive calculates:

```
hash(customer_id) % 8
```

So:

- If result = 0 → goes to bucket 0
- If result = 1 → goes to bucket 1
  ...
- If result = 7 → goes to bucket 7

Important slide line:

> All records for the same customer always go to the same bucket.
> A single customer is completely in 1 bucket – not spread out.
>
> BDA.Lecture.10

This is **very important for joins and aggregations** (your next subtopic).

6.3 Difference Between Partitioning and Bucketing

This is a **guaranteed exam question.**
Here are super clear points.

Partitioning	Bucketing

Divides data into folders	Divides data into files
Based on partition column	Based on hash of bucket column
Affects physical directory structure	Affects file structure inside directory
May create many folders	Creates fixed number of buckets
Good for filtering/pruning	Good for joins and aggregation

Short exam comparsion (write any 4):

1. Partitioning creates directories, bucketing creates files.
2. Partitioning uses column values, bucketing uses hash of column.
3. Partitioning reduces data scanned, bucketing organizes data inside partitions.
4. Partition count can grow large, bucket count is fixed.

6.4 Bucketing for Joins and Aggregations

This is where chunk marks are given.

From slide:

> Data pre-hashed by cid, so Hive parallelizes aggregations and joins using bucket files.
> Hive assigns each bucket to one reducer only.
>
> BDA.Lecture.10

Let's break that.

Why bucketing helps joins:

If two tables are bucketed on the **same column** and have the **same number of buckets**, Hive can:

- Join bucket 1 with bucket 1

- Join bucket 2 with bucket 2
- And so on…

So instead of shuffling massive data across network, it directly joins related buckets.

This makes joins:

1. Faster
2. Less network-shuffle heavy
3. More parallel

Why bucketing helps aggregations (GROUP BY):

Since all values of a key go into the same bucket:

- Aggregations become easier
- Less data movement between reducers
- Faster grouping

Example from slide:

```
SELECT customer_id, COUNT(order_id)
FROM ecommerce_sales_bucketed
WHERE country = 'Pakistan'
GROUP BY customer_id;
```

Here every customer_id is already grouped in buckets.

6.5 Reducer Allocation for Buckets

From your slide again:

Hive assigns each bucket to one reducer only.

BDA.Lecture.10

This means:

If you have:

- 8 buckets
  Hive will allocate:
- 8 reducers

Each reducer:

- Processes one bucket
- Handles its assigned file

So bucket count = reducer count (ideally)

6.6 Bucketed Table Examples

Straight from your slide (page 10):

```
CREATE TABLE ecommerce_sales_bucketed (
 order_id STRING,
 customer_id STRING,
 category STRING,
 product_id STRING,
 price FLOAT,
 payment_type STRING
)
PARTITIONED BY (country STRING)
CLUSTERED BY (customer_id) INTO 8 BUCKETS
STORED AS ORC;
```

It creates files like:

```
/warehouse/ecommerce_sales_bucketed/country=Pakistan/000000_0
/warehouse/ecommerce_sales_bucketed/country=Pakistan/000001_0
...
/warehouse/ecommerce_sales_bucketed/country=Pakistan/000007_0
```

BDA.Lecture.10

Another example from slide 71:

```
CREATE TABLE employees (
 id INT,
 name STRING,
 department STRING
)
CLUSTERED BY (id) INTO 4 BUCKETS
STORED AS TEXTFILE;
```

Here:

- `id` is used for bucketing
- Data is split into 4 bucket files

✍️ Exam-Style Short Questions (With 4–5 Points Each)

You may get any of these:

Q1: What is bucketing in Hive?

Answer:

- Bucketing divides data into fixed number of files

- It uses hash function on a column
- All similar keys go to same bucket
- Helps in efficient joins and aggregations
- Each bucket is stored as a file in HDFS

Q2: How is bucketing different from partitioning?

Answer:

- Partitioning creates directories, bucketing creates files
- Partitioning uses column values, bucketing uses hash values
- Bucketing ensures fixed number of files
- Partition count can grow dynamically, bucket count is predefined
- Bucketing improves query parallelism

Q3: How does Hive assign data into buckets?

Answer:

- Uses hash function on bucketing column
- Applies hash(column) mod number_of_buckets
- All same keys go into same bucket
- Each bucket is stored as a separate file

Q4: Why is bucketing useful for joins?

Answer:

- Matching keys already in same bucket
- Reduces shuffle during join
- Enables bucket map join optimization
- Makes joins faster and more scalable

Q5: Explain role of reducers in bucketing.

Answer:

- Each bucket is assigned to one reducer
- Reducer processes data from its single bucket
- Enables parallel execution
- Improves performance in group-by and joins

Q6: Give SQL syntax for creating a bucketed table.

Answer:

```
CREATE TABLE employees (
 id INT,
 name STRING
)
CLUSTERED BY (id) INTO 4 BUCKETS
STORED AS TEXTFILE;
```

Points:

- Uses CLUSTERED BY keyword

- INTO N BUCKETS specifies bucket count
- Data is hashed into N files

Q7: How does bucketing help sampling?

Answer (from slide 70):

- Hive can sample data efficiently using buckets
- No need to scan full dataset
- TABLESAMPLE works efficiently on bucketed tables
- Reduces I/O and improves speed

Q8: What happens if bucket count is different in two tables during join?

Answer:

- Hive cannot perform bucketed map join
- Data shuffling increases
- Join becomes slower
- Bucketing benefit is lost

topic7

7.1 Primitive Data Types (Basics first)

These are the **simple, built-in types** similar to SQL.

From your slide table:

Type	Meaning
TINYINT	1-byte integer
SMALLINT	2-byte integer
INT	4-byte integer
BIGINT	8-byte integer
BOOLEAN	TRUE / FALSE
FLOAT	Single precision real number
DOUBLE	Double precision real number
STRING	Text / character data
TIMESTAMP	Date + time value
DECIMAL	High precision decimal
BINARY	Raw bytes

Important things your teacher may nitpick:

1. Hive supports **both numeric and non-numeric** types.
2. **STRING is used instead of VARCHAR** in most Hive systems.
3. `DECIMAL` is used for financial data because it avoids floating-point errors.
4. `TIMESTAMP` stores date + time in Unix epoch or datetime form.

Exam sentence to remember:

> Hive supports standard primitive data types such as INT, STRING, BOOLEAN, FLOAT, and TIMESTAMP for structured data storage.

7.2 Complex Data Types

Hive supports **nested data structures**, which are very important in Big Data systems.

Your slide calls them:

> Associative arrays, Lists, and Struct.

These are:

1. ARRAY
2. MAP
3. STRUCT

7.3 ARRAY

What is an ARRAY?

From your slide:

> ARRAY is an indexed list of elements of the same type.

That means:

- All elements must be **of the same data type**
- You access them using **index number**

Example from slide:

If:

```
A = ['a', 'b', 'c']
```

Then:

```
A[0] → 'a'
A[1] → 'b'
A[2] → 'c'
```

Hive Table Example:

```
CREATE TABLE employees (
  name STRING,
  skills ARRAY<STRING>
);
```

Example row data:

```
("Ali", ["Python", "SQL", "Java"])
```

7.4 MAP

What is a MAP?

From your slide:

> MAP is a key-value pair data structure.

Here:

- Every key maps to exactly one value
- Keys must be unique
- Values can repeat

Example from slide:

If:

```
M = {'age': 25, 'salary': 50000}
```

Then you access:

```
M['age'] → 25
M['salary'] → 50000
```

Hive Table Example:

```
CREATE TABLE employees (
  name STRING,
  deductions MAP<STRING, FLOAT>
);
```

Example:

```
{"tax": 5000, "insurance": 1200}
```

7.5 STRUCT

What is STRUCT?

From slide:

> STRUCT is a collection of named fields, possibly of different data types.

It works like a structure or object.

Example from your slide:

If:

```
STRUCT { street:STRING, city:STRING, zip:INT }
```

Access:

```
address.city
address.zip
```

Example Table (from page 63):

```
CREATE TABLE employees (
```

```
 name STRING,
 salary FLOAT,
 subordinates ARRAY<STRING>,
 deductions MAP<STRING, FLOAT>,
 address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
);
```

So:

```
employees.address.city
```

will give you the city.

7.6 How to Access Complex Elements

Your teacher will 100% ask this.

From slide:

Type	Access Format
ARRAY	`array[index]`
MAP	`map['key']`
STRUCT	`struct.field_name`

Example Queries:

```
SELECT skills[0] FROM employees;
SELECT deductions['tax'] FROM employees;
SELECT address.city FROM employees;
```

🚨 Common Traps Examiner Tries

1. ARRAY = same type elements only
2. MAP = key-value structure
3. STRUCT = mixed data types allowed
4. All complex types help represent hierarchical data
5. Used for JSON-like Big Data

He may also ask:

Why are complex data types useful in Big Data?

Possible answer:
Because they support semi-structured and nested real-world data like logs, JSON, and IoT streams.

📝 Exam-Style Short Questions

Here are 8 strong questions with **4–5 marking points** each, exactly how your quiz demands:

Q1: What are primitive data types in Hive?

Answer:

- Built-in basic data types
- Include INT, STRING, FLOAT, BOOLEAN
- Used for simple structured data
- Similar to SQL data types
- Support numeric and non-numeric values

Q2: Name any 5 primitive data types in Hive.

Answer:

1. INT
2. STRING
3. BOOLEAN
4. FLOAT
5. TIMESTAMP

Q3: What are complex data types in Hive?

Answer:

- Used to store nested and structured data
- Include ARRAY, MAP, STRUCT
- Support real-world hierarchical data
- Useful for JSON-like data
- Increase flexibility of data modeling

Q4: Explain ARRAY data type with example.

Answer:

- Stores ordered list of same type elements
- Accessed using index
- Example: `skills ARRAY<STRING>`

- `skills[0]` retrieves first element
- Used for lists like skills, product IDs

Q5: Explain MAP data type with example.

Answer:

- Stores key-value pairs
- Each key maps to one value
- Example: `deductions MAP<STRING, FLOAT>`
- `deductions['tax']` accesses tax value
- Useful for dynamic attributes

Q6: Explain STRUCT data type.

Answer:

- Stores fields of different data types
- Similar to C/Java struct
- Accessed using dot operator
- Example: `address.city`
- Used for hierarchical records

Q7: How do you access elements from Hive complex types?

Answer:

- ARRAY: using index e.g. `arr[0]`
- MAP: using key e.g. `map['key']`
- STRUCT: using field name e.g. `struct.col`
- Enables retrieval from nested structures

Q8: Why does Hive support complex data types?

Answer:

- To handle nested real-world data
- To support semi-structured formats
- Useful for JSON and web logs
- Makes Hive flexible for Big Data
- Supports modern application data

topic8

8.1 CREATE TABLE

Used to create a new table in Hive.

Basic Syntax (from slide 61):

```
CREATE TABLE table_name (
 col1 data_type,
 col2 data_type
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

BDA.Lecture.10

What happens internally when you run CREATE TABLE?

1. Hive creates **metadata** entry in the Metastore.

Hive creates a directory in HDFS under:

```
/user/hive/warehouse/
```

2.
3. The **table schema is stored**, not the actual data yet.

From slide 64:

> CREATE TABLE → metadata + HDFS directory created
>
> BDA.Lecture.10

Example from your lecture:

```
CREATE TABLE page_view (
```

```
 viewTime INT,
 userid BIGINT,
 page_url STRING,
 referrer_url STRING,
 ip STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;
```

BDA.Lecture.10

8.2 DROP TABLE

Used to delete a table.

Syntax:

```
DROP TABLE table_name;
```

Now the important part:

📌 What happens depends on table type (Managed vs External).

From slide 64:

Table Type	What happens on DROP
Managed	Metadata + Data deleted
External	Only metadata deleted

BDA.Lecture.10

8.3 ALTER TABLE

ALTER modifies the structure of an existing table.

Common ALTER operations from your slide (page 43):

Operation	Syntax
Rename table	`ALTER TABLE old_name RENAME TO new_name;`
Add column	`ALTER TABLE t ADD COLUMNS (age INT);`
Drop partition	`ALTER TABLE t DROP PARTITION(age=70);`

BDA.Lecture.10

Explanation:

- ALTER does **not delete data**
- Only changes metadata or table structure
- Used to adapt schema with changing data needs

8.4 TRUNCATE

Used to delete all data from a table **but keep the table structure**.

Syntax:

```
TRUNCATE TABLE table_name;
```

What it does:

1. Deletes all data files inside table directory.
2. Keeps schema and metadata intact.

3. Faster than DELETE since it doesn't scan rows.

Your teacher may directly ask:

> Difference between DROP and TRUNCATE?

DROP	TRUNCATE
Removes table completely	Keeps table, removes only data
Deletes schema + metadata	Only deletes rows
Cannot be undone	Sometimes recoverable if backed

8.5 Managed vs External Tables (Very Important)

This part is **guaranteed quiz material**.

🟦 Managed Table

From slide 65:

> Hive owns and manages both metadata and actual data.

Example:

```
CREATE TABLE employees (
 id INT,
 name STRING
)
STORED AS ORC;
```

Stored under:

```
/user/hive/warehouse/employees
```

Key properties:

1. Hive controls data and metadata.
2. Hive deletes **both** on DROP.
3. Used when Hive fully owns the data.

🟨 External Table

From slide 66:

> Hive only manages metadata; actual data is outside its control.

Example:

```
CREATE EXTERNAL TABLE employees (
 id INT,
 name STRING
)
STORED AS ORC
LOCATION '/user/data/employees';
```

BDA.Lecture.10

Key properties:

1. Data stored externally.
2. Hive only references it.
3. Dropping the table does **not delete the data.**

8.6 What Happens to Data During DROP?

This is where half the class loses marks.

Table Type	What happens to data?

Managed Table	Data is deleted from HDFS
External Table	Data remains in HDFS
Only metadata is removed	For external

From slide 64:

> “DROP: both metadata and data deleted (Managed)”
> “DROP: only metadata deleted (External)”

BDA.Lecture.10

✍️ Exam-Style Short Questions (With 4–5 Points Each)

Here are 8 solid questions tailored to your teacher’s style:

Q1: What is DDL in Hive?

Answer:

- Data Definition Language
- Used to create/modify table structure
- Defines metadata in Hive Metastore
- Does not modify actual data directly
- Includes CREATE, ALTER, DROP, TRUNCATE

Q2: What happens when you execute CREATE TABLE?

Answer:

- Creates metadata entry in Metastore
- Creates directory in Hive warehouse
- Defines schema
- Does not move actual data
- Prepares table for data loading

Q3: Difference between DROP and TRUNCATE.

Answer:

- DROP removes schema + metadata
- TRUNCATE only deletes data
- DROP removes table completely
- TRUNCATE preserves table structure
- DROP on managed tables deletes data physically

Q4: What is a Managed Table?

Answer:

- Hive manages both metadata and data
- Stored in Hive warehouse directory
- On DROP, data is deleted
- Suitable when Hive fully controls data

- Default type if EXTERNAL not mentioned

Q5: What is an External Table?

Answer:

- Hive manages only metadata
- Data stored outside Hive warehouse
- On DROP, data is not deleted
- Useful when data is shared
- Requires LOCATION keyword

Q6: What happens to data when you DROP an external table?

Answer:

- Only metadata is deleted
- Physical data remains on HDFS
- Hive loses reference to data
- Useful for shared datasets

Q7: Write syntax for creating an external table.

Answer:

```
CREATE EXTERNAL TABLE table_name (
 col1 INT,
 col2 STRING
)
```

```
STORED AS TEXTFILE
LOCATION '/user/data/path';
```

Points:

- Uses EXTERNAL keyword
- Specifies custom location
- Metadata only managed by Hive

Q8: Why use External tables instead of Managed tables?

Answer:

- When multiple apps share same data
- Prevent data deletion on DROP
- For raw/staging data
- Allows multiple schemas on same data
- Gives user control over data lifecycle

topic9

🔷 Topic 9: Managed Tables vs External Tables

You already saw this slightly in Topic 8, but now we're going **deeper, more exam-focused.**

Your teacher is likely to ask about:

1. Data ownership
2. When to use which
3. Directory behavior
4. What happens on DROP
5. Real use cases

Let's break each part properly.

9.1 Difference in Data Ownership

This is the **core conceptual difference.**

✅ Managed Table (Internal Table):

In Managed tables:

- Hive **owns both** the metadata AND the actual data.
- Hive is responsible for data storage, lifecycle, and deletion.

From slides:

> "Managed: Hive owns and manages both the metadata and the actual data stored on HDFS."

BDA.Lecture.10

✅ External Table:

In External tables:

- Hive only manages **metadata.**
- The actual data is owned and controlled outside Hive.

From slide:

> “Hive manages only metadata while actual data files are outside Hive's control.”
>
> BDA.Lecture.10

Key exam differentiation:

Feature	Managed Table	External Table
Data ownership	Hive owns data	Data owned externally
Metadata	Hive owns	Hive owns
Data control	Full	Only reference
DROP effect	Data deleted	Data remains

9.2 When to Use Which?

Your teacher will love conceptual questions here.

🟦 When to use Managed Table:

Use a Managed table when:

1. Hive is the **main and only tool** accessing the data.
2. You want Hive to **control the full lifecycle** of data.

3. You want automatic deletion when dropping the table.
4. The data is finalized and settled.

🟨 When to use External Table:

Use External table when:

1. Data is used by **multiple tools or applications**.
2. Data already exists in external storage.
3. You want to **protect data** from being deleted.
4. You want multiple Hive schemas for same dataset.

This line from your slide explains it:

> "Use this when sharing with other Hadoop applications, or when you want to use multiple schemas on the same data."

BDA.Lecture.10

9.3 Directory Behavior

This part is both conceptual **and** technical.

🟦 Managed Tables Directory:

When you create a managed table:

Hive creates directory under:

```
/user/hive/warehouse/
```

-

Example from your slide:

`/user/hive/warehouse/employees`

-

BDA.Lecture.10

Hive fully controls this directory.

🟨 External Tables Directory:

For external tables:

- Data stored outside Hive warehouse
- Stored at custom user-defined location using `LOCATION`.

Example from your slide:

```
CREATE EXTERNAL TABLE employees (...)
LOCATION '/user/data/employees'
```

Directory not inside `/user/hive/warehouse`

BDA.Lecture.10

So:

Point	Managed	External
Default location	Hive warehouse	User-specified
HDFS control	Hive	User/App

9.4 Data Deletion Behavior (VERY IMPORTANT)

This comes in almost every quiz.

What happens when DROP is used?

Table Type	Behavior
Managed	Metadata + data deleted
External	Only metadata deleted

From your lecture:

> “DROP: both metadata and data deleted (Managed)”
> “DROP: only metadata deleted (External)”
>
> BDA.Lecture.10

Why this matters:

If you accidentally drop a Managed table → data is **gone forever**
If you drop External table → data is still safe in HDFS.

9.5 Real Use Cases

This is where you impress the examiner.

Use Case 1: Logs Data

If web server logs are shared with:

- Spark
- Hive
- Another ETL pipeline

→ Use **External Table**
Because you don’t want Hive to delete logs accidentally.

Use Case 2: Reporting Table

If you have transformed final reporting data only used by Hive:

→ Use **Managed Table**
Because Hive controls full lifecycle.

Use Case 3: Staging data

Raw ingestion data in staging area:

→ Use **External Table**
Because you may reuse same data with different schemas.

🔑 Quick Memory Trick

Remember this during quiz:

> **Managed controls everything.**
> **External only points to the data.**

📘 Exam-Style Short Questions (With Extra Points)

Here are 8 guaranteed-style questions.

Q1: Differentiate Managed and External tables.

Answer (write at least 4):

- Managed table: Hive controls metadata & data

- External table: Hive controls only metadata
- Managed deletes data on DROP
- External preserves data on DROP
- Managed stored inside Hive warehouse

Q2: What happens when a Managed table is dropped?

Answer:

- Hive deletes metadata
- Data removed from HDFS
- Table directory removed
- Data cannot be recovered
- Full lifecycle controlled by Hive

Q3: What happens when an External table is dropped?

Answer:

- Only metadata removed
- Data remains intact
- Hive just removes naming/schema info
- Used for shared datasets
- Actual files remain in HDFS

Q4: Why choose External table over Managed?

Answer:

- To prevent accidental deletion of data
- When data is shared by multiple systems
- For raw/staging data
- To allow multiple schemas
- When data location is fixed externally

Q5: Where are Managed tables stored?

Answer:

- Inside Hive warehouse
- Default location: `/user/hive/warehouse`
- Hive manages directory
- Stored as subdirectories
- Each table has its folder

Q6: Give example use case for External table.

Answer:

- Log files in shared Hadoop cluster
- Sensor data used by multiple tools
- Data shared between Spark and Hive

- Raw ingestion data
- When storage ownership is external

Q7: How do you create an External table?

Answer:

```
CREATE EXTERNAL TABLE logs (
  id INT,
  message STRING
)
LOCATION '/user/logs/data';
```

Points:

- Uses EXTERNAL keyword
- Specifies custom location
- Data is not moved
- Hive only stores schema

Q8: Can you convert Managed table to External?

Answer:

- Yes, using ALTER TABLE properties
- Changes table type
- Metadata behavior changes
- Data location stays same
- Drop behavior changes

topic10

🔷 Topic 10: Hive Schema Concepts

From your slides, this comes under the section **“Hive Schemas”** (Page ~60).
It clearly says:

> Hive is **schema-on-read**
> RDBMS is **schema-on-write**
> Schema enforcement timing is the key difference.
>
> BDA.Lecture.10

Let’s break it down.

1️⃣ Schema-on-Read (Hive)

Meaning

In Hive, **schema is applied when you READ the data**, not when you store it.

This means:

- You can dump raw data into HDFS **without caring about structure.**
- Hive will only try to understand the structure **when you run a query.**

What happens practically?

Suppose you have a messy log file like this:

```
101, Ali, 20
102, Sara, abc
103, , 18
104, Ahmed, 22
```

Hive lets you load this file into HDFS without checking anything.

When you run:

```
SELECT * FROM students;
```

Hive applies your schema **at query time**.
If something doesn't match (like "abc" instead of age), it might:

- Return NULL
- Or give an error depending on SerDe.

Why does Hive do this?

Because big data:

- Comes from many sources
- Is often semi-structured or unstructured
- Is too expensive to clean before loading

Key advantages of schema-on-read:

1. **Flexibility –** same data can be read with *different schemas*.
2. **Faster loading –** no validation during load.
3. **Better for evolving data –** schema can change without reloading data.

Disadvantage:

1. Query time is slower because schema is applied then.
2. Data errors appear late (during analysis, not during ingestion).

2️⃣ Schema-on-Write (RDBMS)

This is how traditional databases like MySQL, Oracle, PostgreSQL work.

Meaning

Schema is applied **before data is stored.**

So:

- You must define table structure first.
- Data must strictly follow that structure.

Example:

```
CREATE TABLE students (
 id INT,
 name STRING,
 age INT
);
```

Now if you try to insert:

```
102, Sara, abc
```

It will **fail immediately** because `age` must be an integer.

Advantages:

1. **Strong data consistency**
2. **Queries are faster** since data is already structured
3. **Errors detected early**

Disadvantages:

1. Inflexible for big, messy data
2. Hard to change schema once data grows huge
3. Slower data loading

3️⃣ Hive vs RDBMS – Exam Comparison

Your teacher *will* ask this.

Here is a clean table you can memorize:

Feature	Hive	RDBMS
Schema type	Schema-on-read	Schema-on-write
Validation	During query time	During insert time
Flexibility	Very flexible	Rigid
Query latency	High	Low
Data type	Best for unstructured/semi-structured	Best for structured data
Data size	Designed for Big Data	Designed for smaller datasets
Performance	High throughput, high latency	Low latency, lower throughput

---

4️⃣ Words Your Teacher Might Pick On

Remember these phrases:

- **Schema-on-read**
- **Schema-on-write**
- **Data ingestion vs data querying**
- **Flexible schema**
- **Late binding of schema**
- **Query-time validation**
- **Data evolution support**

If you use these terms in answers, he'll feel you actually studied.

5️⃣ Example Short Exam Answer (How YOU Should Write)

Hive uses schema-on-read where data is stored without schema enforcement and the schema is applied only at query time.
This allows flexibility and faster data loading, but increases query complexity.
In contrast, RDBMS uses schema-on-write where data is validated before storage, resulting in faster queries but reduced flexibility.

📝 Practice Questions (With Answers)

Here are **8 exam-style questions** like your teacher loves.

Q1: What is Schema-on-Read in Hive?

Answer (write any 2–3 points):

- Schema is applied when data is read during query time.
- Data is stored without structure enforcement in HDFS.
- Allows flexible and evolving schema over time.

Q2: What is Schema-on-Write in RDBMS?

Answer:

- Schema is enforced before data is written into the table.
- Data must strictly follow predefined structure.
- Ensures strong consistency and clean data.

Q3: Why does Hive use Schema-on-Read instead of Schema-on-Write?

Answer:

- To handle unstructured and semi-structured big data.
- To allow flexible data ingestion pipelines.
- To avoid heavy validation cost at load time.

Q4: Give two differences between Schema-on-Read and Schema-on-Write.

Answer:

- Schema application timing: Query time vs Load time
- Flexibility: High in Hive vs Low in RDBMS
- Error detection: Late vs Early

Q5: Which is faster: Hive or RDBMS? Explain why.

Answer:

- RDBMS queries are faster because schema is pre-applied and indexed.
- Hive has higher latency because schema is resolved during query execution.
- Hive focuses on throughput instead of low latency.

Q6: Does Schema-on-Read improve data loading speed? Why?

Answer:

- Yes, because Hive skips checking structure during loading.

- Data is copied directly into HDFS.
- No schema validation cost occurs at ingestion.

Q7: Explain one disadvantage of Schema-on-Read.

Answer:

- Errors appear during query, not at load time.
- Query becomes more complex and slower.
- Data quality issues are delayed.

Q8: Which system is better for dynamic, evolving data: Hive or RDBMS? Why?

Answer:

- Hive because schema can be modified without rewriting old data.
- Different schemas can be applied to same data.
- Suitable for data lakes and big data pipelines.

topic11

11. Loading Data into Hive

Hive supports two main ways to load data into tables:

1. **LOAD DATA** → load data from files/directories
2. **INSERT** → load data from query results

Your lecture focuses heavily on the first part.

---

1. `LOAD DATA`

This command loads an existing file or directory into a Hive table.

Basic Syntax

```
LOAD DATA INPATH '/path/in/hdfs/file.txt'
INTO TABLE table_name;
```

→ By default:

- It reads from **HDFS**
- The file is **moved**, not copied, into Hive warehouse
- Data is **appended** to the table

From your slides:

> "Use LOAD DATA to import data into a Hive table"
> "Hive will read data from HDFS unless LOCAL keyword is specified"
>
> BDA.Lecture.10

---

2. `LOAD LOCAL DATA`

If your file is on your **local machine**, use `LOCAL`.

Syntax:

```
LOAD DATA LOCAL INPATH '/home/user/data.txt'
INTO TABLE table_name;
```

Difference from normal LOAD:

Feature	LOAD DATA	LOAD LOCAL DATA
Reads from	HDFS	Local File System
File movement	Moves into HDFS	Copies then loads
Speed	Faster (already in HDFS)	Slower (needs copy)

Slide reference:

> "We can load data from Local file system by using LOCAL keyword."
>
> BDA.Lecture.10

---

3. `OVERWRITE` vs `APPEND`

By default, Hive **appends** data.

You can change this using `OVERWRITE`.

Option	Behavior
APPEND (default)	Adds data to existing files
OVERWRITE	Deletes old data and replaces it

Example:

```
LOAD DATA LOCAL INPATH '/tmp/data.txt'
INTO TABLE sales;
```

→ Appends data

```
LOAD DATA LOCAL INPATH '/tmp/data.txt'
OVERWRITE INTO TABLE sales;
```

→ Deletes old data and replaces it

Your lecture notes:

> “Use the word OVERWRITE to write over a file of the same name”
>
> BDA.Lecture.10

---

4. Loading Data into Partitioned Tables

For partitioned tables, you **must specify partition values.**

Example from slides:

```
LOAD DATA LOCAL INPATH '/tmp/pv_2008-06-8_us.txt'
OVERWRITE INTO TABLE page_view
PARTITION (date='2008-06-08', country='US');
```

What happens:

- Data goes into specific partition folders:

```
/user/hive/warehouse/page_view/date=2008-06-08/country=US/
```

Your slide explicitly says:

> “PARTITION required if destination table is partitioned”
> “LOAD DATA LOCAL INPATH … PARTITION(dt=..., country=...)”
>
> BDA.Lecture.10

---

5. LOAD vs INSERT

Hive also supports loading via SQL queries.

Slide summary table:

Method	Purpose
LOAD	Load data from files/directories
INSERT	Load data from a SQL query
CTAS	Create a table + load in one step

Example Insert:

```
INSERT OVERWRITE TABLE page_view
PARTITION (dt='2008-06-08', country='US')
SELECT viewTime, userid, page_url
FROM page_view_stg;
```

Slide reference:

> “Use INSERT to load data from a Hive query”
> “One partition at a time”
> “Use multiple INSERTs for multiple partitions”
>
> BDA.Lecture.10

---

6. Summary Table

Concept	Key Idea
LOAD DATA	Loads from HDFS
LOAD LOCAL DATA	Loads from local system
OVERWRITE	Replaces existing data
Default	AppEND
Partitioned table	Must specify partition
INSERT	Loads from query

CTAS	Create + load in one command

topic12

Topic 12: Inserting Data in Hive

Based directly on your lecture (slides around pages **96–103**), where Hive insert methods and examples are shown.

I'll cover each sub-topic in the same structure you asked for:

12.1 `INSERT INTO`

This is used to **add new data into an existing table without deleting old data.**

Basic Syntax:

```
INSERT INTO TABLE table_name
SELECT column1, column2, ...
FROM source_table;
```

Important points:

1. `INSERT INTO` **appends data**, it does not remove previous data.
2. It is used for **incremental loading.**
3. It reads data from another Hive table or query result.
4. It works similar to `APPEND` in LOAD.

From your lecture:

> "INSERT INTO will append data into an existing table or partition"

If table is partitioned:

```
INSERT INTO TABLE sales
PARTITION(country='US')
SELECT * FROM temp_sales;
```

12.2 INSERT OVERWRITE

Used when you want to **replace the existing contents of a table or partition** with new data.

Syntax:

```
INSERT OVERWRITE TABLE table_name
SELECT * FROM source_table;
```

Key behavior:

1. Deletes old data in target table/partition.
2. Replaces it with the new query results.
3. Useful for **refreshing data pipelines.**
4. Faster than DELETE + INSERT manually.

Lecture slide:

> “INSERT OVERWRITE replaces the table or partition data”

12.3 Multi-Partition Insert

Hive supports inserting data into **multiple partitions using one query.**

This is a high-yield exam topic because your teacher loves it.

Example from your lecture:

```
FROM page_view_stg
INSERT OVERWRITE TABLE page_view
PARTITION (dt='2008-06-08', country='US')
SELECT * WHERE country = 'US';

INSERT OVERWRITE TABLE page_view
```

```
PARTITION (dt='2008-06-08', country='IN')
SELECT * WHERE country = 'IN';
```

Key points:

1. Single source query.
2. Outputs go to different partitions.
3. Saves time compared to separate loads.
4. Good for big pipelines.

12.4 CTAS (Create Table As Select)

CTAS allows you to **create a table and load data into it in one command.**

Syntax:

```
CREATE TABLE new_table
AS
SELECT * FROM old_table;
```

From your lecture:

> "CTAS creates a table and populates it with the results of a SELECT query."

Important characteristics:

1. Creates table automatically based on SELECT schema.
2. No need to define table structure beforehand.
3. Faster in ETL pipelines.
4. Very useful in data warehouse setups.

🔑 Differences Summary Table

Feature	INSERT INTO	INSERT OVERWRITE	CTAS
Deletes old data?	❌ No	✅ Yes	Creates new table
Appends data?	✅ Yes	❌ No	New table only
Used for	Incremental load	Full refresh	Data transformation
Creates table?	❌ No	❌ No	✅ Yes

📘 Exam-Style Questions (With 4–5 Points Each)

Now as per your instruction, here are the **important short questions** your sir could ask.

Q1: Explain INSERT INTO in Hive.

Answer:

- Appends data into an existing table.
- Does not delete old data.
- Used for incremental data loads.
- Data comes from SELECT queries.
- Supports partition inserts.

Q2: Explain INSERT OVERWRITE with difference from INSERT INTO.

Answer:

- INSERT OVERWRITE deletes old data.
- INSERT INTO keeps old data.

- OVERWRITE refreshes the table.
- Both load data from queries.
- Used in ETL pipelines.

Q3: What is Multi-Partition Insert?

Answer:

- Insert into multiple partitions using one query.
- Single source table.
- Different conditions route data to partitions.
- Saves processing time.
- Used for partitioned tables.

Q4: What is CTAS in Hive?

Answer:

- Create Table As Select.
- Creates table + loads data.
- No need for schema definition.
- Uses query output schema.
- Useful for transformation pipelines.

Q5: When should we use INSERT OVERWRITE instead of INSERT INTO?

Answer:

- When old data must be removed.
- For daily refresh jobs.
- When updating aggregated tables.
- Avoids duplicates.
- Provides consistent new snapshot.

Q6: Difference between LOAD DATA and INSERT in Hive?

Answer:

- LOAD reads from files.
- INSERT reads from query results.
- LOAD moves/copies files.
- INSERT performs query processing.
- INSERT supports transformation during loading.

Q7: Write syntax for inserting data into partitioned table.

Answer:

```
INSERT INTO TABLE sales
PARTITION(region='Asia')
SELECT * FROM temp_sales;
```

Points:

- Uses PARTITION clause.

- Sends data to specific partition.
- Allows partition filtering.
- Works only on partitioned tables.

Q8: Advantages of CTAS command.

Answer:

- Saves time by creating & loading table together.
- Automatically creates schema.
- Easy for transformation queries.
- Reduces repeated DDL.
- Efficient for ETL workflows.

topic13

✅ Topic 13: Hive Querying

Your lecture groups this under **Relational Operators** and **Sample Select Clauses**.

It covers these core pieces:

- SELECT
- WHERE
- GROUP BY
- HAVING
- DISTINCT vs ALL
- ORDER BY vs SORT BY
- LIMIT

Let's go one by one.

13.1 SELECT

`SELECT` is used to **retrieve data from a table or partition.**

From your slide (page 78):

```
SELECT *
FROM sales
WHERE amount > 10 AND region = "US";
```

BDA.Lecture.10

Key points:

1. Used to fetch rows from one or more tables.
2. Supports column selection or wildcard `*`.
3. Used with filtering, grouping and ordering.
4. Can query both partitioned and non-partitioned tables.

Example from your slide (partitioned table):

```
SELECT page_views.*
FROM page_views
WHERE page_views.date >= '2008-03-01'
AND page_views.date <= '2008-03-31';
```

This also shows partition pruning when you filter by partition columns.

BDA.Lecture.10

13.2 WHERE

Your slide explicitly says:

> WHERE filters by expression
> Does **NOT support** correlated subqueries like SQL
> Does **NOT support** `IN`, `EXISTS` directly
>
> BDA.Lecture.10

Role of `WHERE`:

1. Filters rows before grouping.
2. Applied **before GROUP BY**.
3. Runs on individual rows.

Example:

```
SELECT *
FROM sales
WHERE region = 'US';
```

⚠️ Important difference:
Hive does **not** support correlated subqueries in WHERE. Your slide even gives an alternative using JOIN instead.

13.3 GROUP BY

From slide 80:

> GROUP BY – Group data by column values
> SELECT statement can only include columns included in GROUP BY
>
> BDA.Lecture.10

Meaning:

- It groups records based on column values.
- Used with aggregation functions like `COUNT`, `SUM`, `AVG`.

Example from your lecture:

```
SELECT department, COUNT(employee_id)
FROM employees
GROUP BY department;
```

Important rule:

In Hive:
You **cannot** select a column that is not either:

- In `GROUP BY`
  OR
- Inside an aggregate function.

13.4 `HAVING`

`HAVING` is used to **filter groups after GROUP BY.**

Your lecture example (page 51):

```
SELECT department, COUNT(employee_id) AS employee_count
FROM employees
GROUP BY department
HAVING employee_count > 5;
```

BDA.Lecture.10

Difference between WHERE and HAVING:

WHERE	HAVING
Filters rows	Filters groups
Before GROUP BY	After GROUP BY
Works on raw data	Works on aggregated data

13.5 `DISTINCT` vs `ALL`

From slide 79:

> ALL and DISTINCT
> ALL is the default
> DISTINCT removes duplicates

BDA.Lecture.10

DISTINCT:

- Removes duplicate rows.
- Used when you want unique values.

Example:

```
SELECT DISTINCT country
FROM sales;
```

ALL:

- Returns all rows including duplicates.
- Default behavior.

Example:

```
SELECT ALL country FROM sales;
```

(But `ALL` is often not written since it's default).

13.6 `ORDER BY` vs `SORT BY` 🔥 (Favorite Exam Trap)

From your lecture (page 80):

ORDER BY performs **total ordering**
SORT BY performs **partial ordering**
ORDER BY is slow
SORT BY sorts output of each reducer

BDA.Lecture.10

ORDER BY:

- Produces globally sorted output.
- Uses **single reducer** → slow.
- Good for small datasets.

Example:

```
SELECT * FROM sales ORDER BY amount DESC;
```

SORT BY:

- Sorts inside each reducer only.
- Faster for big data.
- Output is sorted per reducer, not globally.

Example:

```
SELECT * FROM sales SORT BY amount;
```

13.7 LIMIT

From slide 79:

> LIMIT indicates the number of rows to be returned
>
> BDA.Lecture.10

Example:

```
SELECT * FROM sales LIMIT 10;
```

Purpose:

1. Reduces output size.
2. Used for quick testing/debugging.
3. Avoids loading full dataset.

🔑 Quick Summary Table

Clause	Purpose
SELECT	Retrieve data
WHERE	Row filtering
GROUP BY	Grouping
HAVING	Filter groups
DISTINCT	Remove duplicates
ALL	Return all rows
ORDER BY	Global sorting
SORT BY	Local sorting
LIMIT	Restrict row count

📝 Exam-Style Questions (With Extra Points)

Here are **8 very likely short questions**:

Q1: Difference between WHERE and HAVING.

Answer:

- WHERE filters rows before grouping.
- HAVING filters after GROUP BY.
- WHERE cannot use aggregations.
- HAVING works on aggregated data.
- WHERE works on individual tuples.

Q2: Define GROUP BY in Hive.

Answer:

- Groups rows based on column values.
- Used with aggregate functions.
- Non-aggregate columns must be in GROUP BY.
- Performs summarization.

Q3: Difference between DISTINCT and ALL.

Answer:

- DISTINCT removes duplicates.
- ALL returns all records.
- ALL is default behavior.
- DISTINCT increases query cost.

Q4: Why is ORDER BY slower than SORT BY?

Answer:

- ORDER BY uses single reducer.
- ORDER BY sorts globally.
- SORT BY sorts per reducer.
- ORDER BY causes performance bottleneck.

Q5: What does LIMIT do?

Answer:

- Restricts number of rows returned.
- Used for sampling.
- Improves testing efficiency.
- Avoids large memory load.

Q6: Can Hive support subqueries in WHERE?

Answer:

- No correlated subqueries in WHERE.
- Must use JOIN instead.
- Supported only in FROM clause.

Q7: Write difference between ORDER BY and SORT BY.

Answer:

- ORDER BY global sort.
- SORT BY local sort.
- ORDER BY uses one reducer.
- SORT BY parallel execution.

Q8: What is the SELECT clause used for?

Answer:

- Retrieves data from Hive tables.
- Used with WHERE and GROUP BY.
- Can filter using LIMIT.
- Supports partitions and aggregations.

topic14

✅ Topic 14: Hive Joins

Hive joins are similar to SQL joins, but with **important big-data behavior differences** that your sir expects you to know.

Your slide covers:

- Join types
- Why join condition must not be in WHERE
- Cartesian product problem
- Multiple table joins
- Join optimization rule (big table last)
- Left Semi Join

Let's go step by step.

14.1 Join Types in Hive

Hive supports these join types:

1. **Inner Join**
2. **Left Outer Join**
3. **Right Outer Join**
4. **Full Outer Join**
5. **Left Semi Join** (special Hive join)
6. **Cross Join** (Cartesian product)

Basic Example:

```
SELECT *
FROM orders o
JOIN customers c
ON o.customer_id = c.customer_id;
```

14.2 Why Join Condition MUST be in `ON` (Not WHERE)

This is a **very important exam point** — your slide literally warns about this.

From your lecture:

> "Do not specify join conditions in the WHERE clause
> Hive does not know how to optimize such queries
> Will compute a full Cartesian product before filtering it"
>
> BDA.Lecture.10

WRONG way:

```
SELECT *
FROM a, b
WHERE a.id = b.id;
```

Hive treats this as:

1. First build **Cartesian product** of a × b
2. Then apply filter
   💀 Extremely slow and expensive.

CORRECT way:

```
SELECT *
FROM a
JOIN b
ON a.id = b.id;
```

Hive understands how to:

- Optimize join
- Reduce intermediate data
- Plan MapReduce jobs smartly

Exam line:

> Join conditions should be in the `ON` clause, not `WHERE`, to avoid full Cartesian product.

14.3 Cartesian Product Issue

A **Cartesian product** happens when:

- You join tables **without a join condition**, or
- Put the condition incorrectly.

Example:

If:

- Table A has 1,000 rows
- Table B has 1,000 rows

Cartesian product =
1,000 × 1,000 = 1,000,000 rows

This explodes memory and slows everything.

From your slide:

> Hive will compute a full Cartesian product before filtering
> if join condition is in WHERE

14.4 Multiple Table Joins

From your lecture (page 81):

> If only one column in each table is used in the join,
> then only one MapReduce job will run
>
> Example:
> `SELECT * FROM a JOIN b ON a.key = b.key JOIN c ON b.key = c.key`
>
> BDA.Lecture.10

Key idea:

If all joins use the **same join key**, Hive can:
✅ Combine them into **1 MapReduce job**

But if different keys are used:
→ Hive runs **multiple MapReduce jobs**

Example:

Query Type	MR Jobs
Same join key	1
Different join keys	Multiple

This is a very likely quiz question.

14.5 Join Optimization Rule: Big Table Last

From your lecture explicitly:

> “If multiple tables are joined, put the biggest table last
> and the reducer will stream the last table, buffer the others”

Meaning:

Hive buffers smaller tables in memory and streams the largest table row-by-row.

Example:

❌ Bad:

```
SELECT *
FROM large_table l
JOIN small_table s ON l.id = s.id;
```

✅ Good:

```
SELECT *
FROM small_table s
JOIN large_table l ON s.id = l.id;
```

Why?

- Hive loads the first tables into memory
- Streams the last table
- So small tables should come first, large table last

14.6 Left Semi Join

This is a **Hive-only join** (not common in SQL).

From your lecture:

```
SELECT a.key, a.val
FROM a
LEFT SEMI JOIN b
ON a.key = b.key;
```

BDA.Lecture.10

What it does:

- Returns only records from **left table**
- Checks existence in right table
- Does NOT return columns from right table

It behaves like `IN` or `EXISTS`.

Equivalent SQL:

```
SELECT *
FROM a
WHERE a.key IN (SELECT key FROM b);
```

But faster and optimized for big data.

📝 Exam-Style Questions (2–3 points expected)

Here are your **likely quiz questions**:

Q1. Why should join conditions not be written in WHERE for Hive?

Answer:

- Hive cannot optimize joins written in WHERE.
- It first computes Cartesian product.
- Then filters it → extremely slow.
- ON clause helps Hive optimize join plan.

Q2. What is a Cartesian product in Hive joins?

Answer:

- Occurs when no proper join condition is used.
- Every row of one table matches with every row of another.
- Causes huge intermediate datasets.
- Extremely inefficient.

Q3. State the join optimization rule in Hive.

Answer:

- Largest table should be placed last.
- Smaller tables get buffered in memory.
- Last table is streamed by reducer.
- Improves performance significantly.

Q4. When does Hive use only one MapReduce job for joins?

Answer:

- When all joins use the same join key.
- When joining on one common column.
- Allows multiway join optimization.
- Otherwise multiple MR jobs are used.

Q5. What is a Left Semi Join?

Answer:

- Joins left table with right table.
- Returns only left table records.
- Checks existence in right table.
- Similar to `IN` or `EXISTS` but optimized.

Q6. Difference between INNER JOIN and LEFT SEMI JOIN.

Answer:

- INNER JOIN returns data from both tables.
- LEFT SEMI only returns left table's columns.
- LEFT SEMI used for existence check.
- Faster for filtering.

Q7. Give a reason why Cartesian joins are dangerous in Hive.

Answer:

- Generates massive intermediate results.
- Wastes memory and computational power.
- Slows down cluster performance.
- Can even cause job failure.

Q8. Why should big tables come last in join order?

Answer:

- Hive buffers smaller tables.
- Streams the last (largest) table.
- Reduces memory pressure.
- Improves performance.

topic15

✅ Topic 15: HiveQL & Functions

Your slides say:

- Hive defines its own SQL-like language called **HiveQL (HQL)**
- HQL supports **DDL, DML, joins, aggregation, multi-inserts**
- Hive can be extended using **UDF, UDAF, UDTF**
- Hive can even run **external scripts using TRANSFORM**

  BDA.Lecture.10

Let's break it down.

15.1 HiveQL Basics

From your lecture:

> HiveQL is a SQL-like language used to query data stored in Hadoop using Hive.
>
> BDA.Lecture.10

What HiveQL Supports:

1. **DDL –** CREATE, DROP, ALTER, TRUNCATE tables
2. **DML** – INSERT, LOAD, SELECT
3. **Joins** – INNER, OUTER, LEFT SEMI
4. **Aggregations –** SUM, COUNT, GROUP BY
5. **Multi-table insert**
6. Subqueries (but only in **FROM** clause)

7. UNION ALL (not just UNION)

Important Lecture Line:

> HiveQL does **not** support row-level UPDATE or DELETE in normal tables
> (only allowed in ACID transactional tables)
>
> BDA.Lecture.10

Example HiveQL:

```
SELECT department, COUNT(*)
FROM employees
GROUP BY department;
```

15.2 UDF – User Defined Function

From slide 36:

> HiveQL can be extended with custom functions (UDFs)
>
> BDA.Lecture.10

What is a UDF?

A **UDF**:

- Takes **one input row**
- Returns **one output value**
- Works like built-in functions: `lower()`, `upper()`, `round()`

Key idea:

1 input row → 1 output value

Example use-case:

Suppose you want to calculate tax in a special way:

```
SELECT calculate_tax(salary) FROM employees;
```

You could write your own `calculate_tax()` as a UDF.

Important exam points:

1. Operates on single rows.
2. Returns one value per row.
3. Used for custom logic inside SELECT.

15.3 UDAF – User Defined Aggregation Function

From slide 36:

> User Defined Aggregation Function (UDAF)
>
> BDA.Lecture.10

What is a UDAF?

UDAF works like:

- COUNT()
- SUM()
- AVG()

But you define your own logic.

Key behavior:

Multiple input rows → One output value

Example:

If you want:

- Median
- Custom weighted average
- Special score calculation

Exam points:

1. Operates on a group of rows.
2. Returns single aggregated output.
3. Used with GROUP BY.

15.4 UDTF – User Defined Table Generating Function

From your lecture slide:

> User Defined Transformation Function (UDTF)
>
> BDA.Lecture.10

What is a UDTF?

Unlike UDF and UDAF:

Type	Input	Output
UDF	One row	One value
UDAF	Many rows	One value
UDTF	One row	**Multiple rows**

Example:

Imagine a column:

```
hobbies = ["reading","coding","sports"]
```

UDTF can convert it into:

hobby
reading
coding
sports

It **explodes** one row into many rows.

15.5 TRANSFORM Queries (Very Important & Unique Hive Feature 🔥)

From page 15:

> TRANSFORM: incorporate custom mapper and reducer scripts directly into Hive queries
>
> BDA.Lecture.10

This is a **big concept.**

What is TRANSFORM?

It allows you to:

- Use **external programs** (Python, Bash, Perl, etc.)
- Inside Hive queries
- As mappers or reducers

Example idea:

```
SELECT TRANSFORM (col1, col2)
USING 'python myscript.py'
AS (result1, result2)
FROM mytable;
```

Meaning:

- Hive sends each row to your script.
- Script processes it.
- Script sends output back to Hive.

So Hive becomes like a wrapper over custom MapReduce code.

📌 Quick Comparison Table

Function	Output Behavior
UDF	1 row → 1 value
UDAF	Many rows → 1 aggregated value
UDTF	1 row → Many rows
TRANSFORM	Uses external scripts for custom processing

📝 Likely Exam Questions (Short Answer Style)

Here are 8 very possible quiz questions with **4–5 solid points each**:

Q1: What is HiveQL?

Answer:

- SQL-like query language for Hive.
- Used to query data in Hadoop.
- Supports DDL and DML operations.
- Supports joins, grouping, and aggregation.

Q2: Define UDF in Hive.

Answer:

- User Defined Function.
- Processes one input row.
- Returns one output value.
- Used for custom row-level operations.

Q3: What is UDAF?

Answer:

- User Defined Aggregation Function.
- Works on group of rows.
- Produces one aggregated result.
- Similar to COUNT or SUM.

Q4: Explain UDTF.

Answer:

- User Defined Table Generating Function.
- Converts one row into multiple rows.
- Used for exploding complex data.
- Helps normalize nested structures.

Q5: What is TRANSFORM in Hive?

Answer:

- Allows execution of external scripts.
- Used for custom MapReduce logic.
- Supports Python, Shell, etc.
- Integrates external logic into HiveQL pipeline.

Q6: Difference between UDF and UDAF.

Answer:

- UDF works on single row.
- UDAF works on group of rows.
- UDF returns one value per row.
- UDAF returns one aggregated result.

Q7: Difference between UDF and UDTF.

Answer:

- UDF returns one value.
- UDTF can return multiple rows.
- UDTF expands data structure.
- Used in more complex transformations.

Q8: Why does Hive support TRANSFORM?

Answer:

- To support custom logic.
- To incorporate external tools.
- To handle complex transformations.
- To extend Hive functionality.

topic16

🔷 Topic 16: Hive Metastore

Your slide keywords:
What is metastore?
Role of metadata
Embedded / Local / Remote Metastore
Apache Derby default
Production setup (MySQL, PostgreSQL)

Let's break this properly.

---

1. What is the Hive Metastore?

The **Hive Metastore** is basically:

👉 The **central metadata repository** of Hive.

It does **NOT** store actual data.
It stores **information about your data.**

According to your slide:

> "Metastore: The system catalog which contains metadata about the tables stored in Hive." (page 31)
>
> BDA.Lecture.10

It stores metadata like:

- Databases
- Tables
- Columns & datatypes
- Partitions
- Storage format
- Location in HDFS

- SerDe information
- Owner, permissions, properties

So when you write:

```
SELECT * FROM sales;
```

Hive asks the **metastore**:

> "Where is the sales table stored? What are its columns? What format is it in?"

Example to Understand

When you create a table:

```
CREATE TABLE employees (
  id INT,
  name STRING
);
```

Hive stores:

- Table name: employees
- Columns: id, name
- Their datatypes
- HDFS path: `/user/hive/warehouse/employees`
- File format: TEXT/ORC/PARQUET
  in the **Metastore**, not inside the data file.

---

2. Role of Metadata (Why Metastore Matters)

Metadata = “data about data”.

According to your lecture (page 31):

It stores metadata about:

- Databases
- Tables
- Partitions
- SerDe and storage info
  BDA.Lecture.10

Why is this important?

Because Hive depends on metadata for:

1. **Query planning**
   - Without metadata, Hive doesn’t know how to read your table.
2. **Schema enforcement at read time**
   - Hive is **schema-on-read** (slide 60) → schema applied only when reading data.
3. **Optimization**
   - Query optimizer needs metadata to choose efficient plans.
4. **Partition pruning**
   - To skip irrelevant partitions.
5. **Security and governance**
   - Cloudera slide (page 32) shows metastore supports:
     - Access control
     - Auditing

- Data masking
- Centralized policy management

---

3. Types of Hive Metastore

Your slides classify three types:

👉 **Embedded Metastore**
👉 **Local Metastore**
👉 **Remote Metastore**

From page **56.**

BDA.Lecture.10

(A) Embedded Metastore

Description from slide:

- Runs **inside Hive process**
- Uses **Apache Derby**
- Only for **single-user** and development

Key Points:

1. Metastore and Hive run in the **same JVM/process.**
2. Uses **Apache Derby** (default DB).
3. Only **one connection allowed at a time.**
4. Not suitable for production.

Your slide literally says:

Embedded: For unit tests, development, single-user (uses embedded Derby).

BDA.Lecture.10

(B) Local Metastore

Local means:

- Hive runs its own metastore process.
- But multiple Hive clients connect to it.
- Uses **external DB** like MySQL.

Features:

1. Metastore runs separately from Hive in same machine.
2. Multiple clients can connect.
3. Better than embedded.
4. Uses shared external DB.

(C) Remote Metastore (Production)

From slide:

Remote: Metastore runs as a separate standalone service.
Each Hive client connects to the metastore server.

BDA.Lecture.10

This is used in **real production clusters**.

Features:

1. Metastore is a separate service.

2. Many Hive clients connect remotely.
3. Uses production DB like **MySQL or PostgreSQL.**
4. Supports concurrency, security, scaling.

---

4. Apache Derby (Default Metastore DB)

From page **55**:

> Default metastore database uses Apache Derby
> Embedded, single-user database
> Only one active connection allowed
> Used in local or testing environments
>
> BDA.Lecture.10

Summary:
Apache Derby is:

- Lightweight database
- Comes with Hive by default
- But **not scalable**
- Only suitable for:
    - Learning
    - Testing
    - Single-user setups

---

5. Production Metastore Setup (MySQL / PostgreSQL)

Your slide clearly says:

For production, replace Derby with MySQL or Postgres (configure hive-site.xml).

BDA.Lecture.10

Why replace Derby?
Because:

1. Derby supports **only one connection.**
2. Large clusters have **many Hive users.**
3. MySQL/PostgreSQL support:
    - Concurrency
    - Multiple connections
    - Reliability
    - Backups
    - Large-scale metadata.

Example real setup:

```
Hive Clients → Metastore Server → MySQL/PostgreSQL DB
```

---

6. Quick Diagram in Your Head

Think of Hive Metastore like this:

```
Query --> Hive --> Metastore --> Metadata DB
                      |
                      |
                   HDFS (Actual Data)
```

Metastore tells Hive:

- WHERE the data is

- HOW to read it

🔹 Important Keywords to Remember for Quiz

You must remember these exact terms:

- System Catalog
- Metadata Repository
- Schema-on-read
- Apache Derby
- Embedded Metastore
- Local Metastore
- Remote Metastore
- Production setup
- MySQL / PostgreSQL backend

📝 Topic 16 Quiz Questions (With Solid 4–5 Point Answers)

Here are **8 exam-style questions** like your teacher might ask.

Q1. What is the Hive Metastore?

Answer:

1. The Hive metastore is a system catalog that stores metadata of Hive objects.

2. It stores information about databases, tables, columns, partitions and file locations.
3. It does not store actual data, only metadata.
4. It is used by Hive during query compilation and optimization.
5. It enables schema management and partition handling.

Q2. What type of information is stored in the Hive Metastore?

Answer:

1. Table schema (column names and data types).
2. Partition information.
3. HDFS data locations.
4. Storage formats and SerDe details.
5. Ownership and table properties.

Q3. Explain Embedded Metastore.

Answer:

1. Runs within the Hive process itself.
2. Uses Apache Derby as backend.
3. Supports only one connection at a time.
4. Suitable for testing and development only.
5. Not suitable for multi-user or production environments.

Q4. Differentiate between Embedded and Remote Metastore.

Answer:

1. Embedded metastore runs inside Hive; remote runs as a separate service.
2. Embedded uses Apache Derby; remote uses MySQL/PostgreSQL.
3. Embedded supports single user; remote supports multiple clients.
4. Embedded is for testing; remote is for production.
5. Remote supports better concurrency and scalability.

Q5. Why is Apache Derby not suitable for production?

Answer:

1. It allows only one active connection.
2. Does not support multiple users well.
3. Not scalable for large systems.
4. Limited performance and reliability.
5. Not built for distributed environments.

Q6. Why do production systems use MySQL / PostgreSQL for the metastore?

Answer:

1. They support multiple concurrent connections.
2. More reliable for large metadata storage.
3. Provide backup and recovery features.

4. Suitable for high availability setups.
5. Better performance and scalability.

Q7. What is the difference between Local and Remote Metastore?

Answer:

1. Local metastore runs on same machine as Hive service.
2. Remote metastore runs as an independent server.
3. Local is semi-scalable; remote is fully scalable.
4. Remote supports multiple Hive clients.
5. Remote is preferred for production clusters.

Q8. Explain the role of Metastore in query execution.

Answer:

1. Provides schema details for query parsing.
2. Provides HDFS paths for data access.
3. Helps query optimizer generate execution plan.
4. Supports partition pruning to improve performance.
5. Ensures correct interpretation of stored data formats.

topic17

📌 Topic 17: Hive Warehouse

1. What is the Hive Warehouse?

The **Hive Warehouse** is the **default storage location in HDFS where Hive stores its managed tables and their data.**

From your lecture:

> "Hive tables are stored in the Hive *warehouse*. Tables are stored as sub-directories in the warehouse directory. Partitions are subdirectories of tables."
>
> BDA.Lecture.10

So basically:

Hive doesn't randomly store data anywhere.
It has a **dedicated base folder** in HDFS where everything goes unless you explicitly tell it otherwise.

---

2. Default Warehouse Directory

By default, Hive uses this HDFS path:

```
/user/hive/warehouse
```

Important points your teacher might test:

- This is the **root directory** for all **managed tables**.
- Every database inside Hive has its own folder inside this.
- Every table inside a database has its own subfolder.

Example structure:

If you create:

```
CREATE DATABASE sales_db;
CREATE TABLE sales_db.orders (...);
```

In HDFS, Hive creates:

```
/user/hive/warehouse/sales_db.db/orders/
```

So:

- `sales_db.db` → database folder
- `orders` → table folder

This is **automatic** for Managed tables.

---

3. How Tables and Partitions are Stored

Hive uses a **directory structure** to organize data.

From your lecture:

> "Tables are stored as sub-directories in the warehouse directory. Partitions are subdirectories of tables."
>
> BDA.Lecture.10

Example with partitions:

If you have:

```
CREATE TABLE page_view (
  userid STRING,
  url STRING
)
PARTITIONED BY (date STRING, country STRING);
```

And you insert data like:

```
LOAD DATA INPATH '/logs/us_2025_11_01'
INTO TABLE page_view
PARTITION (date='2025-11-01', country='US');
```

Then directory structure becomes:

```
/user/hive/warehouse/default.db/page_view/
  └── date=2025-11-01/
        └── country=US/
              ├── 000000_0
              ├── 000001_0
```

Key points:

- Table = folder
- Each partition column = hierarchical folders
- Partition values become folder names like `date=...`

This is **not just logical** — it's actual physical folders in HDFS.

---

4. How Data is Stored Inside

Another important line from your slides:

> "The actual data is stored in flat files."
>
> BDA.Lecture.10

This means:

- Hive doesn't store rows inside some internal DB engine like MySQL.
- It stores data as **files** in HDFS.
- These files can be:
    - TEXTFILE
    - ORC

    - PARQUET
    - SEQUENCEFILE

So:

- Metadata → in Metastore (DB)
- Actual rows → in files on HDFS

---

5. External Data Locations

If you use an **External Table**, the data location is **not inside** the warehouse**.**

Example:

```
CREATE EXTERNAL TABLE logs (
   userid STRING,
   activity STRING
)
LOCATION '/user/raw_logs/';
```

Now:

- Metadata is in Hive
- Data lives in: `/user/raw_logs/`
- NOT in `/user/hive/warehouse`

Key difference:

- Managed table → data inside Hive warehouse
- External table → data outside Hive warehouse

---

6. Why Your Teacher Cares About This

He might ask trick questions like:

- What happens in HDFS when you create a table?
- Why is warehouse important in Hive?
- What happens to files when you drop a managed vs external table?

These concepts connect directly with your **Managed vs External tables** topic also.

🔥 Exam-Focused Key Points to Remember (Write These)

Memorize these 4–5 points:

1. Hive warehouse is the **default HDFS directory** where managed tables are stored.
2. Default path: `/user/hive/warehouse`
3. Each table = one subdirectory inside the warehouse.
4. Partitions = subdirectories inside the table folder.
5. External tables can point to **locations outside** the warehouse directory.

📝 Short Quiz: Topic 17 (With Answers)

Here are typical short-answer questions your teacher might ask.

Q1. What is Hive Warehouse?

Answer points:

- Default storage directory for Hive tables in HDFS.
- Used for storing managed table data.

- Organizes tables and partitions as folders.

Q2. What is the default Hive warehouse location?

Answer points:

- Default location is: `/user/hive/warehouse`
- Located inside HDFS.

Q3. How are Hive tables and partitions stored physically?

Answer points:

- Each table is stored as a subdirectory.
- Each partition is a further subdirectory inside the table.
- Actual data stored as flat files inside these directories.

Q4. How is data stored in Hive warehouse?

Answer points:

- Data is stored as files (not rows like a database).
- These files can be ORC, TEXTFILE, PARQUET, etc.
- Hive reads metadata from Metastore and data from HDFS.

Q5. How is external table storage different from managed table storage?

Answer points:

- Managed tables: data stored inside Hive warehouse.
- External tables: data stored outside warehouse at a custom HDFS path.
- Dropping managed table deletes data.
- Dropping external table keeps data.

Q6. Why is warehouse location important?

Answer points:

- Controls where Hive stores its data.
- Helps organize data for different databases and tables.
- Important for storage management and backup.

Q7. What happens in HDFS when you create a table with partitions?

Answer points:

- Hive creates a directory for the table.
- Inside it creates sub-directories for each partition.
- Inside partitions it stores data files.

topic18

1. What is Serialization?

Serialization is the process of converting an object or data structure into a format that can be **stored or transmitted**.

In simple terms:

> Turning in-memory data into a storable or transferable form.

From your slides:
Serialization converts data into a format suitable for storage or transmission and allows complex data structures like lists and dictionaries to be saved as a single entity.

BDA.Lecture.10

Examples of serialized formats:

- JSON
- XML
- Binary (pickle in Python)
- Avro / Parquet / ORC (in Hadoop / Hive)

---

2. What is Deserialization?

Deserialization is the reverse process.

It means:

> Converting stored or transmitted data back into its original in-memory structure.

So if:

- Serialization = save/send
- Deserialization = load/use

The slides define this as reconstructing the original state of the data for application usage.

BDA.Lecture.10

---

3. Why Serialization & Deserialization is Required

From your lecture concepts, SerDe is important because:

1. **Storage –** Data must be stored on disk (HDFS, files, databases).
2. **Transmission –** Data needs to be transferred across networks or systems.
3. **Efficiency –** Optimized formats like ORC/Parquet reduce storage and speed up queries.
4. **Compatibility –** Helps different systems understand the same data format.
5. **Complex Data Handling –** Useful for structs, arrays, maps in Hive.

Slides also mention that it helps persist data state and restore it later efficiently.

BDA.Lecture.10

---

4. Examples

Example 1: JSON Serialization

Let's say we have this Python dictionary:

```
data = {
    "id": 1,
    "name": "John Doe",
    "email": "john.doe@example.com"
}
```

When converted to JSON:

```
{
  "id": 1,
  "name": "John Doe",
```

```
  "email": "john.doe@example.com"
}
```

This JSON is the **serialized form.**

Example 2: Python Object Serialization (Pickle)

Taken almost directly from your slides:

```
import pickle

class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __repr__(self):
        return f"Person(name={self.name}, age={self.age})"

person = Person("John Doe", 30)

# Serialize (Object → Bytes)
serialized_person = pickle.dumps(person)

# Deserialize (Bytes → Object)
deserialized_person = pickle.loads(serialized_person)

print(deserialized_person)
```

Here:

- `pickle.dumps()` → Serialization
- `pickle.loads()` → Deserialization

This example is shown in your lecture slides.

---

5. SerDe in Hive (Exam Focus)

In Hive:

- Data stored in HDFS needs to be converted into rows/columns when we query.
- Hive uses **SerDe libraries** to:
    - Deserialize data from files into Hive rows
    - Serialize Hive query output back to storage formats

Examples of Hive SerDe:

- JSON SerDe
- ORC SerDe
- Parquet SerDe

These SerDes allow Hive to understand how the data is structured and how it should be read/written.

---

6. Short Exam Summary (You can memorize)

Serialization: Converting object/data into a storable or transferable format.
Deserialization: Converting stored/transferred data back into original object form.
Why needed: For storage, communication, performance optimization, and handling complex structures.
Where used: Hive, Hadoop, databases, APIs, distributed systems.

topic19

✅ Topic 19: Hive ACID Support

Earlier versions of Hive were **append-only**.
No updates.
No deletes.
No real transactions.

But newer Hive versions support **ACID transactions** under strict conditions.

---

1. What Operations Does Hive Support in ACID Tables?

From your slide table (page 19), Hive supports these:

Operation	Supported ?	Notes
INSERT INTO	✅ Yes	Always supported
INSERT OVERWRITE	✅ Yes	Replace data
UPDATE	✅ Yes	Only for ACID tables
DELETE	✅ Yes	Only for ACID tables
MERGE	✅ Yes	ACID tables only

From your lecture:

> UPDATE / DELETE / MERGE are only supported in ACID transactional tables.
>
> BDA.Lecture.10

---

(1) INSERT INTO

- Adds new rows.

- Always supported, even without ACID.
- Appends new files to table.

---

(2) UPDATE

- Modifies existing rows.
- Requires full ACID support enabled.
- Only possible on ORC bucketed transactional tables.

---

(3) DELETE

- Removes rows from the table.
- Not physically deleting files like SQL.
- Hive marks rows as logically deleted (delta files).

---

(4) MERGE

- Used for “upserts” (update + insert).
- If row exists → update
- If row doesn’t exist → insert
- Works only in transactional setup.

---

2. Why Hive Needs Special Setup for ACID

Hive works on files, not row-by-row storage like MySQL.

So to make updates/deletes possible, Hive:

- Writes new delta files
- Keeps version history
- Merges when reading

This is more complex than normal databases.
Hence: **extra requirements.**

---

3. Requirements for Hive ACID Support

Your slide clearly lists these conditions (page 19):

Requirement Description
Format – ORC or Parquet recommended
Bucketing – Required for full ACID tables
Property – 'transactional' = 'true'
Execution engine – Tez or Spark (not MR)
HiveServer2 + Metastore – concurrency control

BDA.Lecture.10

Now let's break these deeply.

(A) ORC / Parquet Format

Hive ACID only works on columnar formats, mainly:

- **ORC** (Optimized Row Columnar)
- Parquet (newer versions)

Why?
Because:

1. They support fast reads.
2. They support versioning.
3. They store metadata efficiently.
4. They integrate with Hive's transaction logs.

But your slide hints:
👉 ORC is most commonly used for ACID tables.

(B) Bucketing is Mandatory

This is one of the biggest exam traps.

From your slide:

> Bucketing required for full ACID tables.
>
> BDA.Lecture.10

Why?
Because Hive needs deterministic file layout for:

- Row updates
- Deletes
- Transaction consistency

Example:

```
CREATE TABLE orders (
  order_id INT,
  amount DOUBLE
)
CLUSTERED BY (order_id) INTO 4 BUCKETS
STORED AS ORC
TBLPROPERTIES ('transactional' = 'true');
```

So no bucketing = ❌ no ACID.

(C) transactional=true

This is the magic line.

To enable ACID:

```
TBLPROPERTIES ('transactional'='true');
```

This tells Hive:

> Treat this table as transactional, not append-only.

Without this → Hive will **refuse** UPDATE/DELETE/MERGE.

(D) Execution Engine: Tez or Spark (Not MapReduce)

Very important from your slide:

> Execution engine: Tez or Spark, not MapReduce
>
> BDA.Lecture.10

Why?
Because:

- MapReduce is too slow and not suited for ACID transactions.
- Tez/Spark can handle fine-grained operations better.
- MapReduce doesn't support efficient file compaction.

So:

```
SET hive.execution.engine=tez;
```

or:

```
SET hive.execution.engine=spark;
```

(E) HiveServer2 + Metastore with Concurrency

Another hidden exam point.

For ACID to work:

1. HiveServer2 must be running.
2. Metastore must support concurrency.
3. Multiple users/transactions must be coordinated.

This ensures:

- Isolation
- Consistency
- Lock handling

---

4. Simple ACID Example

Here's how your ACID table would look like:

```
CREATE TABLE employees (
  id INT,
  name STRING,
  salary DOUBLE
)
CLUSTERED BY (id) INTO 4 BUCKETS
STORED AS ORC
TBLPROPERTIES ('transactional'='true');
```

Now this is allowed:

```
UPDATE employees SET salary = 60000 WHERE id = 102;

DELETE FROM employees WHERE id = 105;

MERGE INTO employees e
USING updates u
ON e.id = u.id
WHEN MATCHED THEN UPDATE SET salary = u.salary
WHEN NOT MATCHED THEN INSERT VALUES (u.id, u.name, u.salary);
```

Without ACID setup → Hive will throw error.

---

5. Important Exam Keywords

Memorize these words exactly:

- ACID Transactions
- Transactional Table
- ORC Format
- Bucketing
- Delta Files
- Tez / Spark Engine
- TBLPROPERTIES('transactional'='true')
- HiveServer2
- Concurrency Control

📝 Topic 19 Exam Questions (With 4–5 Bullet Answers)

Here are likely short questions your sir might ask.

Q1. What operations does Hive support under ACID?

Answer:

- INSERT
- UPDATE
- DELETE
- MERGE
- INSERT OVERWRITE
  (All on transactional tables)

Q2. What is required to enable ACID in Hive?

Answer:

- Table must be bucketed.
- It must be stored as ORC/Parquet.
- Must set `transactional=true`.
- Must use Tez or Spark engine.
- HiveServer2 + Metastore configured.

Q3. Why is bucketing mandatory for ACID tables?

Answer:

- Required for deterministic row storage.
- Helps track row-level updates/deletes.
- Supports transaction consistency.
- Enables Hive's delta file mechanism.

Q4. Why doesn't Hive use MapReduce for ACID?

Answer:

- MapReduce is too slow for row-level operations.
- Cannot efficiently manage delta files.
- Tez/Spark provide faster execution.
- ACID needs low-latency processing.

Q5. Explain the role of `transactional=true`.

Answer:

- Enables ACID behavior on table.
- Allows UPDATE and DELETE.
- Activates transaction log mechanism.
- Required for MERGE operations.

Q6. Why does Hive restrict ACID support?

Answer:

- Hive is file-based, not row-based.
- ACID needs complex version control.
- Requires additional storage management.
- Only supported on specific formats.

Q7. What is MERGE in Hive ACID?

Answer:

- Combines UPDATE + INSERT.
- Updates rows if they exist.
- Inserts if they don’t exist.
- Requires transactional table

topic20

✅ Topic 20: Hive Limitations

Hive is powerful, but **not perfect**.
It was designed for **batch analytics**, not real-time systems.

Your slide literally says:

> "Hive is not an OLTP or real-time system… Latency is high even for small data." (Page 16)
> "HiveQL is missing large parts of full SQL specification" (Page 50)
>
> BDA.Lecture.10

Let's break your listed limitations one by one.

---

1. ❌ No Correlated Subqueries in WHERE

From your slide:

> Hive does **not support correlated sub-queries in the WHERE clause**.
> Instead, you must rewrite them using JOINs.
>
> BDA.Lecture.10

What is a correlated subquery?

A correlated subquery is one that depends on values from the outer query.

Example (Not supported):

```
SELECT employee_id, salary
FROM employees e
WHERE salary > (
   SELECT AVG(salary)
   FROM employees
   WHERE department_id = e.department_id
);
```

Hive **does NOT support this**.

Instead, you must rewrite using JOIN:

```
SELECT e.employee_id, e.salary
FROM employees e
JOIN (
   SELECT department_id, AVG(salary) AS avg_salary
   FROM employees
   GROUP BY department_id
) dept_avg
ON e.department_id = dept_avg.department_id
WHERE e.salary > dept_avg.avg_salary;
```

Why Hive doesn't support it?

1. Complex query planning for distributed systems.
2. Inefficient execution in MapReduce.
3. Hive prefers **batch processing with joins** instead.

---

2. ❌ No Stored Procedures

From the HiveQL limitations slide:

> Stored procedures are **not allowed**. Instead, use UDFs.
>
> BDA.Lecture.10

What are Stored Procedures?

In databases like MySQL / Oracle:

- You write a block of reusable SQL logic
- It executes inside the database engine

Example:

```
CREATE PROCEDURE GetEmployees()
BEGIN
  SELECT * FROM employees;
END;
```

-

Hive **does not support this.**

Instead:

- You write **UDFs (User Defined Functions)** in Java or other languages.
- Or use external scripts with `TRANSFORM`.

---

3. ❌ No Real-Time Usage

From page 16:

> Hive is designed for batch processing of large datasets, not real-time systems.
> Latency is high compared to RDBMS.
>
> BDA.Lecture.10

Why Hive is not real-time?

Because:

1. Queries go through:
    - Parsing
    - Optimization
    - DAG generation
    - YARN scheduling

2. Even a simple query takes seconds to minutes.

3. Suitable for analytics, not live dashboards.

So Hive is used for:
✅ Historical analysis
✅ Reporting
✅ Big data processing

Not for:
❌ Live financial systems
❌ Online transaction processing
❌ Banking apps etc.

---

4. ❌ Slow Startup Time

From slide:

> Even for small datasets (<100MB), Hive query startup time is high.
>
> BDA.Lecture.10

Why?

1. Hive jobs need to start containers.

2. Interacts with YARN scheduler.

3. Job initialization overhead is high.

So:

- Even `SELECT * FROM small_table` can take 5–15 seconds to start.

- Unlike MySQL which responds immediately.

---

5. OLTP vs OLAP Mismatch

Nested inside your slide:

> Hive is not designed for OLTP workloads. It is an OLAP engine.
>
> BDA.Lecture.10

OLTP (Online Transaction Processing) systems:

- Example: Banking, ATM, E-commerce payments
- Many small transactions
- Update / insert / delete heavy
- Requires low latency

OLAP (Online Analytical Processing) systems:

- Example: Data warehouse, reporting, analytics
- Large datasets
- Few updates
- Heavy SELECT queries
- High throughput instead of low latency

Hive is:
✅ OLAP
❌ OLTP

Because Hive is optimized for:

- Large scans
- Aggregations
- Grouping

- Analytics

Not for fast row-level updates.

🔑 KEY PHRASES YOUR TEACHER WILL LOVE

Memorize these exact phrases:

- Hive is **not real-time**
- Hive is **batch oriented**
- Hive is designed for **OLAP, not OLTP**
- Hive has **high latency, high throughput**
- No correlated subqueries in WHERE
- No stored procedures (use UDF instead)

📝 Exam-Style Questions: Topic 20

Now your 5–10 short exam type questions with **4–5 points answers**:

Q1. List the major limitations of Hive.

Answer:

1. Hive does not support correlated subqueries in WHERE clause.
2. Stored procedures are not supported.
3. Hive cannot be used for real-time applications.

4. Query startup time and latency are high.
5. Hive is not suitable for OLTP workloads.

Q2. Why Hive cannot be used for real-time applications?

Answer:

1. Hive query execution involves MapReduce/Tez/Spark jobs.
2. High latency before query starts.
3. Resource scheduling through YARN delays execution.
4. Designed for batch analytics, not live systems.

Q3. Why Hive is not suitable for OLTP systems?

Answer:

1. OLTP requires fast, small transactions.
2. Hive is optimized for batch and large datasets.
3. Hive supports limited row-level operations.
4. Startup latency makes it unsuitable for transactional systems.

Q4. Why does Hive not support stored procedures?

Answer:

1. Hive is designed as a query engine, not a procedural DB.
2. Distributed execution makes stored procedures complex.

3. Instead Hive supports UDFs.
4. Procedures don’t scale well in Hadoop environment.

Q5. What is the problem with correlated subqueries in Hive?

Answer:

1. Hive SQL engine cannot optimize correlated subqueries.
2. Distributed execution model makes them inefficient.
3. Hive forces rewriting using JOINs.
4. It simplifies DAG execution planning.

Q6. Explain “High latency but high throughput” in Hive.

Answer:

1. Hive jobs take longer to start (high latency).
2. But once running, they process huge data efficiently.
3. Optimized for throughput, not quick response.
4. Suitable for analytics, not live transactions.

topic21

✅ Topic 21: Advanced Hive Concepts

You listed 3 sub-topics:

1. **Hive Stinger**
2. **MPP Style Execution**
3. **Hive without MapReduce**

These 3 are actually strongly connected. So I'll explain them in flow.

---

1. Hive Stinger

From your lecture slide:

> Hive Stinger
> • MPP-style execution of Hive queries
> • Available since Hive 0.13
> • No MapReduce
>
> BDA.Lecture.10

What is Hive Stinger?

Hive Stinger was a **major performance upgrade project** for Hive.

Before Stinger:
Hive ran everything on **MapReduce** → very slow.

After Stinger:
Hive started supporting:

- Faster execution
- Real-time style performance

- Interactive queries
- SQL-like latency

So basically:

👉 Hive Stinger = Project to make Hive *fast*.

Why did Hive introduce Stinger?

Because:

1. Traditional Hive was too slow for interactive analytics.
2. MapReduce startup overhead was very high.
3. Users wanted SQL-like performance from Hive.

So Apache introduced:

- Better optimizer
- LLAP (in-memory processing)
- Tez execution engine
- MPP-style query execution

All under **Hive Stinger.**

---

2. MPP Style Execution

Your slide says:

> Hive Stinger enables **MPP-style execution** of Hive queries.

What is MPP?

MPP = **Massively Parallel Processing**

It means:

- Query workload is divided across **many nodes**
- All nodes work **in parallel**
- Results are combined at the end
- No central single processing point

Difference: MapReduce vs MPP

Feature	MapReduce	MPP
Execution	Stage-based	Pipeline / parallel
Latency	High	Low
Startup time	Slow	Faster
Interactive	❌ No	✅ Yes
Used by Hive Stinger	❌ No	✅ Yes

In MPP:

- Data flows continuously between tasks
- No rigid map → shuffle → reduce stages

Why MPP helps Hive?

Because it:

1. Eliminates long MapReduce job chains.

2. Reduces execution latency drastically.
3. Allows multiple query operators to run concurrently.
4. Makes Hive suitable for interactive BI tools.

---

3. Hive Without MapReduce

This is a direct line from your slide:

> Hive Stinger: No MapReduce
>
> BDA.Lecture.10

What does it mean?

Earlier:

> Hive = SQL layer → MapReduce backend

After Stinger:

> Hive = SQL layer → Tez / LLAP / Spark backend

So Hive **no longer depends solely on MapReduce**.

How Hive runs without MapReduce?

After Hive 0.13:

1. Hive supports **Apache Tez** as execution engine.
2. Hive supports **Spark** execution engine.
3. Hive Stinger layered optimized DAG execution.
4. LLAP enables in-memory execution.

So instead of submitting MapReduce jobs every time, Hive now:
👉 Executes queries as DAGs directly
👉 Uses memory caching
👉 Uses long-running daemons

Result: Faster queries.

Why Hive moved away from MapReduce?

Because MapReduce:

1. Has long startup latency.
2. Writes intermediate data to disk.
3. Is inefficient for iterative or interactive queries.

Hive Stinger solves this by:

- Using in-memory compute
- Reducing disk IO
- Allowing pipeline execution

🔑 Important Keywords for Your Quiz

These are exact phrases you should memorize:

- Hive Stinger Project
- MPP-style execution
- No MapReduce
- LLAP (Long Live and Process)
- Apache Tez Execution Engine

- Interactive Queries in Hive
- Reduced Latency
- DAG-based Query Execution

📝 Topic 21 – Exam Style Questions

Here are your **quiz-ready questions**, with **4–5 points answers**.

Q1. What is Hive Stinger?

Answer:

1. Hive Stinger is a performance enhancement project for Hive.
2. It introduced MPP-style query execution.
3. It removed dependency on MapReduce.
4. It made Hive suitable for interactive queries.

Q2. What does MPP-style execution mean in Hive?

Answer:

1. Queries execute in parallel on multiple nodes.
2. Workload is distributed instead of single pipeline.
3. Reduces query execution time.
4. Improves scalability and throughput.

Q3. How does Hive Stinger improve performance?

Answer:

1. Replaces MapReduce with Tez/Spark.
2. Enables in-memory processing via LLAP.
3. Reduces startup latency.
4. Uses DAG-based execution.

Q4. Why was Hive Stinger introduced?

Answer:

1. To reduce Hive query latency.
2. To support interactive analytics.
3. To remove MapReduce dependency.
4. To improve SQL-on-Hadoop performance.

Q5. How does Hive work without MapReduce?

Answer:

1. Uses Apache Tez execution engine.
2. Supports Spark as backend.
3. Uses LLAP for in-memory execution.
4. Executes queries through DAG engine.

Q6. Difference between traditional Hive and Hive with Stinger?

Answer:

1. Traditional Hive used MapReduce, Stinger uses Tez/Spark.
2. Traditional had high latency, Stinger reduces latency.
3. Traditional was batch-only, Stinger supports interactive queries.
4. Traditional had slower execution pipelines.