Hive Practicals

Quick setup (session & DB)

```
-- start a session in beeline or hive CLI;
create/use DB
CREATE DATABASE IF NOT EXISTS demo_hive;
USE demo_hive;
```

1. Create managed table (TEXTFILE)

```
CREATE TABLE employees(
  id INT,
  name STRING,
  dept STRING,
  salary DOUBLE,
  doj STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

2. Create external table

```
CREATE EXTERNAL TABLE ext_events(
  event_id STRING,
  ts STRING,
  payload STRING
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LOCATION '/user/hive/external/events';
```

3. Create table as select (CTAS)

```
CREATE TABLE top_paid AS
SELECT id, name, salary FROM employees WHERE salary
> 100000;
```

4. Load data (local -> Hive)

```
-- loads local file into managed table
LOAD DATA LOCAL INPATH '/local/path/employees.csv'
OVERWRITE INTO TABLE employees;
-- or load HDFS file into external table path (no
LOAD needed if file in LOCATION)
```

5. Partitions (static)

```
CREATE TABLE sales(
  order_id STRING, amount DOUBLE
)
PARTITIONED BY (year INT, month INT)
STORED AS PARQUET;

-- add partition and load
ALTER TABLE sales ADD PARTITION (year=2024, month=8)
LOCATION '/data/sales/2024/08';
```

6. Dynamic partition insert

```
SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode=nonstrict;

INSERT INTO TABLE sales PARTITION (year, month)
SELECT order_id, amount, year(order_dt),
month(order_dt) FROM staging_orders;
```

7. Bucketing

```
CREATE TABLE users_bucketed(
  user_id INT,
  name STRING
)
CLUSTERED BY (user_id) INTO 8 BUCKETS
STORED AS ORC;

-- enable bucketing reads/writes
SET hive.enforce.bucketing=true;
INSERT INTO TABLE users_bucketed SELECT * FROM
users;
```

8. ORC / Parquet / SequenceFile example

```
CREATE TABLE orc_table(id INT, data STRING) STORED
AS ORC;
```

```
CREATE TABLE parquet_table(id INT, data STRING)
STORED AS PARQUET;
CREATE TABLE seq_table(id INT, data STRING) STORED
AS SEQUENCEFILE;
```

9. Compressed ORC/Parquet writes

```
SET parquet.compression=SNAPPY;
SET hive.exec.compress.output=true;
SET hive.exec.orc.compression=ZLIB;

INSERT OVERWRITE TABLE orc_table SELECT * FROM
some_source;
```

10. Partition pruning demonstration

```
-- assume sales is partitioned by year,month
SELECT SUM(amount) FROM sales WHERE year=2024 AND
month=8; -- prunes partitions
EXPLAIN SELECT SUM(amount) FROM sales WHERE
year=2024 AND month=8;
```

11. Basic SELECT, WHERE, GROUP BY, HAVING

```
SELECT dept, COUNT(*) cnt, AVG(salary) avg_sal
FROM employees
WHERE salary > 20000
GROUP BY dept
HAVING COUNT(*) > 2
```

```
ORDER BY avg_sal DESC;
```

12. Joins (inner, left, right, full)

```
SELECT e.id, e.name, d.dept_name
FROM employees e
JOIN departments d ON e.dept = d.dept_code;

-- left join
SELECT * FROM employees e LEFT JOIN dept_budget b ON
e.dept=b.dept_code;
```

13. Map-side join (mapjoin hint) for small table

```
-- force the smaller table into memory to avoid
shuffle
SELECT /*+ MAPJOIN(departments) */ e.id, e.name,
d.dept_name
FROM employees e JOIN departments d ON e.dept =
d.dept_code;
```

14. Lateral view + explode (UDTF use)

```
CREATE TABLE user_tags(user_id INT, tags
ARRAY<STRING>) STORED AS ORC;

-- explode tags to rows
SELECT u.user_id, t.tag
FROM user_tags u
```

```
LATERAL VIEW EXPLODE(tags) exploded_table as tag;
```

15. Window functions

```
SELECT id, name, dept, salary,
       ROW_NUMBER() OVER (PARTITION BY dept ORDER BY
salary DESC) rn,
       RANK() OVER (PARTITION BY dept ORDER BY
salary DESC) rnk,
       NTILE(4) OVER (ORDER BY salary) quartile
FROM employees;
```

16. Grouping sets / rollup / cube

```
-- rollup
SELECT dept, year, SUM(amount)
FROM sales
GROUP BY ROLLUP(dept, year);

-- cube
SELECT dept, year, SUM(amount)
FROM sales
GROUP BY CUBE(dept, year);

-- grouping sets
SELECT dept, year, SUM(amount)
FROM sales
GROUP BY GROUPING SETS ((dept, year), (dept),
(year), ());
```

17. Subqueries (IN / EXISTS / scalar)

```
SELECT name FROM employees WHERE id IN (SELECT
emp_id FROM managers);

SELECT * FROM departments d WHERE EXISTS (SELECT 1
FROM employees e WHERE e.dept=d.dept_code);
```

18. Views and materialized views

```
CREATE VIEW v_high_salary AS SELECT id,name,salary
FROM employees WHERE salary>100000;

CREATE MATERIALIZED VIEW mv_sales_monthly
AS SELECT year,month, SUM(amount) total FROM sales
GROUP BY year,month;
-- refresh depends on Hive version: REFRESH
MATERIALIZED VIEW mv_sales_monthly;
```

19. Transactions and ACID (insert/update/delete)

```
-- requires transactional table and ACID settings:
SET hive.support.concurrency=true;
SET hive.enforce.bucketing=true;
SET hive.exec.dynamic.partition.mode=nonstrict;
SET
hive.txn.manager=org.apache.hadoop.hive.ql.lockmgr.D
bTxnManager;
SET hive.compactor.initiator.on=true;
SET hive.compactor.worker.threads=1;
```

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

-- DML
INSERT INTO txn_users VALUES (1,'Alice');
UPDATE txn_users SET name='Alicia' WHERE id=1;
DELETE FROM txn_users WHERE id=1;
```

20. INSERT OVERWRITE vs INSERT INTO

```
-- overwrite entire table or partition
INSERT OVERWRITE TABLE sales PARTITION (year=2024,
month=8)
SELECT * FROM staging_sales WHERE year=2024 AND
month=8;

-- append
INSERT INTO TABLE sales PARTITION (year, month)
SELECT order_id, amount, year(order_dt),
month(order_dt) FROM staging_orders;
```

21. UDF (Java/JS) — example registering and using a simple UDF

(assumes udf jar already on HDFS/local)

```
-- add jar (local or hdfs)
ADD JAR /path/to/myudfs.jar;

-- create temporary function
CREATE TEMPORARY FUNCTION reverse_str AS
'com.example.hive.udf.ReverseStringUDF';

-- use
SELECT reverse_str(name) FROM employees LIMIT 5;
```

(If you want, I can provide sample Java code for a UDF/UDAF/UDTF.)

22. UDAF / UDTF brief examples

```
-- UDAF usage: assume a jar provides 'median_udaf'
ADD JAR /path/median_udaf.jar;
CREATE TEMPORARY FUNCTION median AS
'com.example.hive.udaf.Median';
SELECT dept, median(salary) FROM employees GROUP BY
dept;

-- UDTF example: explode() is a built-in UDTF used
with LATERAL VIEW above
```

23. SerDe usage (custom delimited or JSON SerDe)

```
-- JSON SerDe (builtin or external) example:
```

```
CREATE TABLE events_json(
  event_id STRING,
  ts STRING,
  payload MAP<STRING,STRING>
)
ROW FORMAT SERDE
'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS TEXTFILE;
```

24. ANALYZE / COMPUTE STATS & optimizer

```
ANALYZE TABLE employees COMPUTE STATISTICS;
ANALYZE TABLE sales PARTITION(year,month) COMPUTE
STATISTICS;
-- helps cost-based optimizer and partition pruning
```

25. EXPLAIN & PROFILE queries

```
EXPLAIN FORMATTED SELECT dept, AVG(salary) FROM
employees GROUP BY dept;
-- or use EXPLAIN EXTENDED / EXPLAIN COST if
supported
```

26. EXTERNAL TABLE over HDFS/Parquet/ORC files (schema-on-read)

```
CREATE EXTERNAL TABLE parquet_ext(
 id INT, name STRING
) STORED AS PARQUET LOCATION '/data/parquet/users';
```

27. File format-specific operations (ORC stats, vectorized reads)

```
SET hive.vectorized.execution.enabled=true;
SET hive.vectorized.execution.reduce.enabled=true;
-- ORC statistics example when creating ORC to get
min/max stats
INSERT OVERWRITE TABLE orc_table SELECT * FROM
employees;
```

28. Sampling

```
SELECT * FROM employees TABLESAMPLE(10 PERCENT);
-- or: SELECT * FROM employees TABLESAMPLE(BUCKET 3
OUT OF 32);
```

29. UNION and UNION ALL

```
SELECT id,name FROM employees WHERE dept='HR'
UNION ALL
SELECT id,name FROM contractors WHERE dept='HR';
```

30. SHOW / DESCRIBE / MSCK (repair) / ALTER

```
SHOW TABLES;
DESCRIBE EXTENDED employees;
ALTER TABLE sales ADD PARTITION(year=2023,
month=12);
MSCK REPAIR TABLE ext_events; -- recovers partitions
from HDFS for external tables
```

31. Indexes (historical / deprecated; still available in some versions)

```
-- note: indexes are rarely used and often
deprecated; example:
CREATE INDEX idx_emp_dept ON TABLE employees(dept)
AS 'COMPACT' WITH DEFERRED REBUILD;
ALTER INDEX idx_emp_dept ON employees REBUILD;
```

32. Hive configuration settings / session tuning

```
-- useful settings
SET hive.exec.reducers.bytes.per.reducer=268435456;
-- 256MB
SET hive.auto.convert.join=true;  -- convert to
mapjoin if small table
SET hive.exec.parallel=true;
```

33. Using BEELINE (JDBC) example

```
beeline -u "jdbc:hive2://namenode:10000/default" -n
hiveuser -p hivepassword
```

34. Export / Import table data

```
EXPORT TABLE employees TO
'/backup/employees_export';
IMPORT TABLE employees FROM
'/backup/employees_export';
```

35. Encryption / ACL hints (high-level)

- Hive itself integrates with HDFS encryption zones and external security (Kerberos, Ranger, Sentry). Typical steps: enable Kerberos, configure Ranger policies, ensure metastore/hive-server2 run under Kerberos principals. (Exact commands depend on your cluster/security tools.)