SE446: Big Data Engineering

Week 4B: HiveQL Queries in Practice

Prof. Anis Koubaa

SE 446
Alfaisal University
https://github.com/aniskoubaa/big_data_course

Spring 2026

Today's Agenda

Recap: Hive Fundamentals

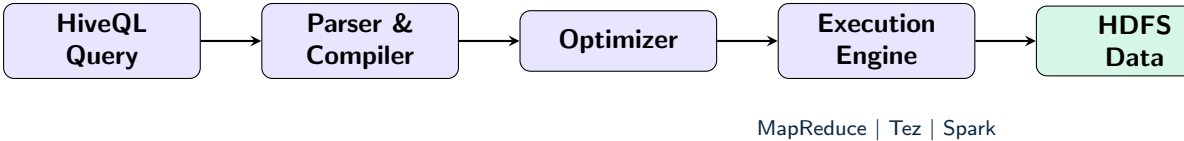
What we learned in 4A:

- Hive = SQL on HDFS
- Schema-on-read paradigm
- Architecture: Driver + Metastore
- Managed vs. External tables
- Partitioning & Bucketing
- File formats: ORC, Parquet

Today: Hands-on HiveQL

- DDL: Create databases & tables
- Loading data into Hive
- DQL: SELECT, WHERE, GROUP BY
- JOINs across tables
- Built-in functions
- Performance tips

Hive Query Execution Pipeline



- **Parser**: Validates SQL syntax, builds AST
- **Compiler**: Generates logical plan from AST
- **Optimizer**: Rewrites plan (predicate push-down, partition pruning)
- **Execution Engine**: Translates plan into MapReduce / Tez / Spark jobs

Accessing Hive: Beeline

Beeline

The recommended CLI for Hive. Connects to HiveServer2 via JDBC.

```
-- SSH into the cluster master
ssh student01@master-node

-- Start Beeline
beeline -u "jdbc:hive2://master-node:10000"

-- Or with authentication
beeline -u "jdbc:hive2://master-node:10000" \
        -n student01 -p password
```

Alternative Hive Interfaces

Beyond Beeline:

- **Hue**: Web-based SQL editor (browser)
  - Graphical query builder, result visualization
- **DBeaver**: Desktop SQL client (JDBC)
  - Multi-database tool, ERD diagrams
- **PyHive**: Python library for programmatic access
  - Integrate Hive into data pipelines

Creating a Database

```
-- Create a database
CREATE DATABASE IF NOT EXISTS crime_analytics
COMMENT 'Chicago crime data for SE446';

-- List databases
SHOW DATABASES;

-- Switch to database
USE crime_analytics;

-- See current database
SELECT current_database();
```

Dropping a Database

```
-- Drop database (must be empty)
DROP DATABASE IF EXISTS crime_analytics;

-- Drop database and all its tables
DROP DATABASE IF EXISTS crime_analytics CASCADE;
```

CASCADE

`CASCADE` drops all tables inside the database. Use with caution!

Creating a Managed Table (CSV input)

```
CREATE TABLE crimes (
    case_number    STRING,
    date_str       STRING,
    primary_type   STRING,
    description    STRING,
    location_desc  STRING,
    arrest         BOOLEAN,
    domestic       BOOLEAN,
    district       INT,
    latitude       DOUBLE,
    longitude      DOUBLE
)
```

Creating a Managed Table (continued)

```
CREATE TABLE crimes (
    case_number STRING, date_str STRING,
    primary_type STRING, ...
)
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    LINES  TERMINATED BY '\n'
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");
```

Note

`skip.header.line.count` tells Hive to ignore the CSV header row.

Creating an External Table

```
CREATE EXTERNAL TABLE taxi_trips (
    vendor_id         INT,
    pickup_datetime   STRING,
    dropoff_datetime  STRING,
    passenger_count   INT,
    trip_distance     DOUBLE,
    fare_amount       DOUBLE,
    tip_amount        DOUBLE,
    total_amount      DOUBLE,
    payment_type      INT
)
```

Creating an External Table (continued)

```
CREATE EXTERNAL TABLE taxi_trips (...)
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/data/nyc_taxi/'
TBLPROPERTIES ("skip.header.line.count"="1");
```

Key Difference

`EXTERNAL` + `LOCATION`: data stays in HDFS. `DROP TABLE` only removes metadata, **not the files**.

Creating a Partitioned Table with ORC

```
CREATE EXTERNAL TABLE crimes_partitioned (
    case_number    STRING,
    primary_type   STRING,
    description    STRING,
    district       INT,
    arrest         BOOLEAN,
    latitude       DOUBLE,
    longitude      DOUBLE
)
PARTITIONED BY (year INT)
STORED AS ORC
LOCATION '/data/crimes_orc/';
```

Partitioned Table: Directory Structure

On HDFS this creates:

```
/data/crimes_orc/year=2022/part-00000.orc
/data/crimes_orc/year=2023/part-00000.orc
/data/crimes_orc/year=2024/part-00000.orc
```

Benefits:

- Query `WHERE year=2023` reads only one directory
- Massive performance improvement on large datasets
- Partition pruning happens automatically

Bucketing: Hash-Based Splits

What Is Bucketing?

Divide data within each partition into a **fixed number of files** using a hash function on a chosen column.

```
CREATE TABLE crimes_bucketed (
    case_number   STRING,
    primary_type  STRING,
    district      INT,
    arrest        BOOLEAN
)
CLUSTERED BY (district) INTO 8 BUCKETS
STORED AS ORC;
```

Benefits: Efficient joins (sort-merge), sampling (`TABLESAMPLE(BUCKET 1 OUT OF 8)`), consistent file sizes.

Table Inspection Commands

```
-- List all tables in current database
SHOW TABLES;

-- Show table structure
DESCRIBE crimes;

-- Show detailed metadata (location, format, etc.)
DESCRIBE FORMATTED crimes;
```

More Inspection Commands

```
-- Show partitions
SHOW PARTITIONS crimes_partitioned;

-- Show create statement (reverse-engineer DDL)
SHOW CREATE TABLE crimes;
```

Pro Tip

`DESCRIBE FORMATTED` is your best friend for debugging. It shows file location, SerDe, input format, and table properties.

Loading Data into Hive

Method 1: LOAD DATA

```
-- From local filesystem
LOAD DATA LOCAL INPATH '/home/student01/crimes.csv'
INTO TABLE crimes;

-- From HDFS (moves the file!)
LOAD DATA INPATH '/staging/crimes.csv'
INTO TABLE crimes;

-- Overwrite existing data
LOAD DATA LOCAL INPATH '/home/student01/crimes.csv'
OVERWRITE INTO TABLE crimes;
```

LOAD DATA: Important Warning

Warning: INPATH Moves Files

`LOAD DATA INPATH` (without LOCAL) **moves** HDFS files — the original is gone!

`LOCAL INPATH`: Copies from local to HDFS `INPATH`: Moves within HDFS

Loading Data: INSERT and CTAS

Method 2: INSERT...SELECT (ETL)

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

INSERT OVERWRITE TABLE crimes_partitioned
    PARTITION (year)
SELECT case_number, primary_type, description,
       district, arrest, latitude, longitude,
       YEAR(date_str) AS year
FROM crimes;
```

CTAS: Create Table As Select

Method 3: CTAS

```
CREATE TABLE theft_crimes
STORED AS ORC AS
SELECT case_number, district, arrest
FROM crimes
WHERE primary_type = 'THEFT';
```

CTAS

Creates a new table **and** populates it in one step. Very useful for ETL pipelines.

Loading into a Specific Partition

```
-- Manual partition loading (static)
LOAD DATA LOCAL INPATH 'crimes_2023.csv'
INTO TABLE crimes_partitioned
PARTITION (year=2023);

-- Add an empty partition
ALTER TABLE crimes_partitioned
ADD PARTITION (year=2025)
LOCATION '/data/crimes_orc/year=2025';
```

Static vs. Dynamic Partitions

Static: You specify the value (`PARTITION (year=2023)`).

Dynamic: Hive infers from data (`INSERT...PARTITION (year)`).

Basic SELECT Queries

```sql
-- Select all columns (avoid on large tables!)
SELECT * FROM crimes LIMIT 10;

-- Select specific columns
SELECT case_number, primary_type, arrest
FROM crimes
LIMIT 20;
```

Filtering and Sorting

```sql
-- Filter with WHERE
SELECT case_number, primary_type, district
FROM crimes
WHERE arrest = TRUE
  AND district = 11;

-- Distinct values
SELECT DISTINCT primary_type
FROM crimes
ORDER BY primary_type;
```

Aggregation: GROUP BY

```sql
-- Count crimes by type
SELECT primary_type, COUNT(*) AS crime_count
FROM crimes
GROUP BY primary_type
ORDER BY crime_count DESC
LIMIT 10;
```

Advanced Aggregation Example

```sql
-- Arrest rate by district
SELECT district,
       COUNT(*) AS total,
       SUM(CASE WHEN arrest THEN 1 ELSE 0 END)
           AS arrests,
       ROUND(SUM(CASE WHEN arrest THEN 1 ELSE 0 END)
             * 100.0 / COUNT(*), 2) AS arrest_pct
FROM crimes
GROUP BY district
ORDER BY arrest_pct DESC;
```

Familiar?

Same analysis as MapReduce (Week 3) — now in 8 lines of SQL!

Filtering Aggregations: HAVING

```sql
-- Districts with more than 100 crimes
SELECT district, COUNT(*) AS crime_count
FROM crimes
GROUP BY district
HAVING COUNT(*) > 100
ORDER BY crime_count DESC;
```

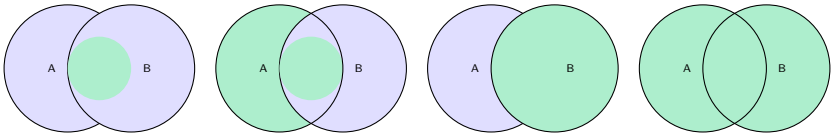
WHERE vs. HAVING

`WHERE` filters **rows before** grouping. `HAVING` filters **groups after** aggregation.

HAVING: Complex Example

```sql
-- Crime types with arrest rate below 20%
SELECT primary_type,
       COUNT(*) AS total,
       ROUND(SUM(CASE WHEN arrest THEN 1 ELSE 0 END)
             * 100.0 / COUNT(*), 2) AS arrest_pct
FROM crimes
GROUP BY primary_type
HAVING COUNT(*) > 50
   AND SUM(CASE WHEN arrest THEN 1 ELSE 0 END)
       * 100.0 / COUNT(*) < 20
ORDER BY arrest_pct;
```

Types of JOINs



- **INNER JOIN**: Only matching rows from both tables
- **LEFT JOIN**: All rows from left + matching from right
- **RIGHT JOIN**: All rows from right + matching from left
- **FULL OUTER JOIN**: All rows from both, NULLs where no match

JOIN Examples

```
-- Join crimes with weather data on date
SELECT c.primary_type,
       w.avg_temp_f,
       COUNT(*) AS num_crimes
FROM crimes c
JOIN nyc_weather w
    ON c.date_str = w.date_str
GROUP BY c.primary_type, w.avg_temp_f
ORDER BY w.avg_temp_f;
```

LEFT JOIN Example

```
-- LEFT JOIN: include all crimes, even without
-- matching weather records
SELECT c.case_number, c.primary_type,
       w.avg_temp_f
FROM crimes c
LEFT JOIN nyc_weather w
    ON c.date_str = w.date_str;
```

Result: All crime records appear. `avg_temp_f` is NULL when no weather match.

Map-Side Join (Optimization)

Problem

Regular JOINs require a **shuffle** phase (expensive network I/O).

Solution: Map-Side Join

If one table is **small enough to fit in memory**, Hive can broadcast it to all mappers. No shuffle needed!

```
-- Automatic (Hive decides)
SET hive.auto.convert.join=true;
SET hive.mapjoin.smalltable.filesize=25000000;
  -- 25 MB threshold
```

Map-Side Join Example

```sql
-- Manual hint for map-side join
SELECT /*+ MAPJOIN(d) */
    c.primary_type, d.district_name
FROM crimes c
JOIN districts d
    ON c.district = d.district_id;
```

Map-side joins can be **10–100x faster** than regular shuffle joins.

String Functions (Part 1)

Common functions: LENGTH, LOWER, UPPER, SUBSTR, TRIM

```
-- Extract year from date string
SELECT SUBSTR(date_str, 7, 4) AS year,
       COUNT(*) AS crimes
FROM crimes
GROUP BY SUBSTR(date_str, 7, 4);

-- Uppercase crime type
SELECT UPPER(primary_type) AS crime_type,
       LENGTH(primary_type) AS len
FROM crimes
LIMIT 10;
```

String Functions (Part 2)

Function	Description & Example
`CONCAT(s1,s2,...)`	Concatenate strings
`CONCAT_WS(sep,...)`	Join: `CONCAT_WS('-','2023','06','15')`
`SPLIT(s,pattern)`	Split → `ARRAY<STRING>`
`REGEXP_EXTRACT(s,p,g)`	Extract regex match

Date and Time Functions

```
YEAR(ts)
MONTH(ts)
DAY(ts)
HOUR(ts)
DATEDIFF(d1, d2)
DATE_ADD(d, n)
FROM_UNIXTIME(t)
TO_DATE(ts)
```

```
-- Extract year from timestamp
SELECT YEAR(pickup_datetime)
         AS trip_year,
       COUNT(*) AS trips
FROM taxi_trips
GROUP BY YEAR(pickup_datetime);
```

Numeric Functions

```
ROUND(x, d)
CEIL(x) / FLOOR(x)
ABS(x)
SQRT(x) / POW(x,n)
GREATEST(a,b,...)
LEAST(a,b,...)
```

```
-- Average fare rounded
SELECT vendor_id,
  ROUND(AVG(fare_amount),2)
    AS avg_fare,
  ROUND(AVG(tip_amount),2)
    AS avg_tip
FROM taxi_trips
GROUP BY vendor_id;
```

Conditional and NULL Functions

```sql
-- CASE WHEN (like SQL)
SELECT case_number,
       CASE
           WHEN arrest = TRUE THEN 'Arrested'
           WHEN domestic = TRUE THEN 'Domestic'
           ELSE 'Other'
       END AS category
FROM crimes;

-- IF shorthand
SELECT primary_type,
       IF(arrest, 'Yes', 'No') AS arrested
FROM crimes;
```

Handling NULL Values

```
-- Handle NULLs
SELECT COALESCE(district, -1) AS district,
       NVL(location_desc, 'UNKNOWN') AS location
FROM crimes;
```

- `COALESCE(val1, val2, ...)`: Returns first non-NULL value
- `NVL(val, default)`: Returns `default` if `val` is NULL

Window Functions (Advanced)

What Are Window Functions?

Perform calculations **across rows** related to the current row, without collapsing the result set (unlike GROUP BY).

Common window functions:

- `ROW_NUMBER()`, `RANK()`, `DENSE_RANK()`
- `LAG(col, n)`, `LEAD(col, n)`
- `SUM() OVER(...)`, `AVG() OVER(...)`

Window Functions: Ranking Example

```sql
-- Rank crime types by count within each district
SELECT district,
       primary_type,
       COUNT(*) AS crime_count,
       RANK() OVER (
           PARTITION BY district
           ORDER BY COUNT(*) DESC
       ) AS rank
FROM crimes
GROUP BY district, primary_type;
```

Window Function: Running Total

```sql
-- Running total of crimes by date
SELECT date_str,
       COUNT(*) AS daily_crimes,
       SUM(COUNT(*)) OVER (
           ORDER BY date_str
           ROWS BETWEEN UNBOUNDED PRECEDING
                    AND CURRENT ROW
       ) AS running_total
FROM crimes
GROUP BY date_str
ORDER BY date_str;
```

Window Function: Top N per Group

```sql
-- Top 3 crime types per district
SELECT * FROM (
    SELECT district, primary_type, crime_count,
           ROW_NUMBER() OVER (
               PARTITION BY district
               ORDER BY crime_count DESC
           ) AS rn
    FROM crime_summary
) ranked
WHERE rn <= 3;
```

File Format Comparison

Format	Layout	Compression	Splittable	Best For
TextFile	Row	None	Yes	Quick loading, debugging
ORC	Column	Zlib/Snappy	Yes	Hive-optimized analytics
Parquet	Column	Snappy/Gzip	Yes	Cross-engine (Spark, Impala)

Rule of thumb:

- Use **ORC** when working primarily with Hive (best predicate push-down)
- Use **Parquet** when sharing data across Spark, Impala, or Presto
- Use **TextFile** only for initial CSV/TSV ingestion

HiveQL Performance Tips

1. **Use partitioned tables** for time-series / categorical data
   - Always include partition column in `WHERE` clause
2. **Use columnar formats** (ORC or Parquet) instead of TextFile
   - Compression + column pruning = much faster reads
3. **Avoid** `SELECT *` on large tables
   - Select only the columns you need
4. **Enable map-side joins** for small dimension tables
   - `SET hive.auto.convert.join=true;`
5. **Use** `LIMIT` during development
   - Test queries on small samples first
6. **Use Tez or Spark** engine instead of MapReduce
   - `SET hive.execution.engine=tez;`

EXPLAIN: Understanding Query Plans

EXPLAIN

Shows the **execution plan** without running the query. Essential for understanding and optimizing performance.

```
-- See the execution plan
EXPLAIN
SELECT primary_type, COUNT(*) AS cnt
FROM crimes_partitioned
WHERE year = 2023
GROUP BY primary_type;
```

EXPLAIN: What to Look For

```
-- Extended plan with more details
EXPLAIN EXTENDED
SELECT primary_type, COUNT(*) AS cnt
FROM crimes_partitioned
WHERE year = 2023
GROUP BY primary_type;
```

What to look for:

- `filterExpr: (year = 2023)` — Partition pruning!
- Number of `Map` and `Reduce` stages
- `Map Join` vs. `Reduce Join`

Saving Query Results

```
-- Method 1: INSERT OVERWRITE DIRECTORY (HDFS)
INSERT OVERWRITE DIRECTORY '/output/crime_summary'
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
SELECT primary_type, COUNT(*) AS cnt
FROM crimes
GROUP BY primary_type
ORDER BY cnt DESC;

-- Method 2: Create a new table from query
CREATE TABLE crime_summary
STORED AS ORC AS
SELECT primary_type, COUNT(*) AS cnt
FROM crimes
GROUP BY primary_type;

-- Method 3: From Beeline to local file
!sh hdfs dfs -get /output/crime_summary .
```

Common Errors & Debugging (Part 1)

1. **Schema mismatch**: CSV columns don't match `CREATE TABLE`
   - All columns read as `NULL` → check delimiter and column order
2. **Header row in data**: Forgot `skip.header.line.count`
   - First row appears as garbage values
3. **Boolean casting**: CSV strings `"true"/"false"` may need `CAST`

Common Errors & Debugging (Part 2)

4. **NULL explosion**: `JOIN` on nullable columns produces unexpected NULLs
   - Use `COALESCE` or filter NULLs before joining
5. **Partition not found**: Data loaded but query returns 0 rows
   - Run `MSCK REPAIR TABLE` to sync partitions with HDFS

Lab Exercise: Chicago Crimes Analysis

Using the `chicago_crimes_sample.csv` dataset:

1. **Setup**: Create database `se446_<teamname>` and load data
2. **Basic queries**: Count records, find top 5 crime types, calculate arrest rate
3. **Aggregation**: Arrest rate per district, crimes per month

Lab Exercise (continued)

4. **Advanced**:
   - Join with weather data — do more crimes happen on hot days?
   - Rank crime types per district using window functions
5. **Optimization**:
   - Convert to ORC format
   - Compare query times: TextFile vs. ORC

Quick Reference: Setup & Load

```
-- 1. Create database
CREATE DATABASE IF NOT EXISTS se446_team01;
USE se446_team01;

-- 2. Create table & load data
CREATE TABLE crimes (...)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");

LOAD DATA LOCAL INPATH 'crimes.csv'
INTO TABLE crimes;
```

Quick Reference: Query & Optimize

```
-- 3. Quick check
SELECT COUNT(*) FROM crimes;
SELECT * FROM crimes LIMIT 5;

-- 4. Analytics
SELECT primary_type, COUNT(*) AS cnt
FROM crimes GROUP BY primary_type
ORDER BY cnt DESC LIMIT 10;

-- 5. Convert to ORC for performance
CREATE TABLE crimes_orc STORED AS ORC
AS SELECT * FROM crimes;
```

Key Takeaways

1. **DDL**: `CREATE DATABASE`, `CREATE TABLE`, `DESCRIBE FORMATTED`
2. **Data Loading**: `LOAD DATA`, `INSERT...SELECT`, `CTAS`
3. **Queries**: Standard SQL — `SELECT`, `WHERE`, `GROUP BY`, `HAVING`
4. **JOINs**: `INNER`, `LEFT`, `RIGHT`, `FULL OUTER` + map-side optimization
5. **Functions**: String, date, numeric, conditional, window
6. **Performance**: Partition pruning, ORC, EXPLAIN, map-side joins

HiveQL Workflow Summary

The HiveQL Workflow

Create → Load → Query → Optimize → Export

Best Practices:

- Always use `DESCRIBE FORMATTED` to verify table structure
- Convert to ORC format for better performance
- Use `EXPLAIN` to understand query execution
- Partition large tables by date/category
- Enable map-side joins for small dimension tables

What's Next

Coming up:

- **Milestone M2**: Hive-based analysis on Chicago crimes
- **Week 5–6**: Midterm 1 review + Apache Spark

Deliverables for this week:

- Complete the Hive lab exercises
- Load your project dataset into Hive
- Write at least 5 analytical queries
- Commit HiveQL scripts to your team GitHub repo

Preparation for Next Week

- Review Milestone M2 requirements
- Experiment with ORC vs. TextFile performance
- Practice window functions