Parallel processing

Case study

Starburst

Lesson objectives

Parallel processing: Case study

1. Explain the difference between sequential processing and parallel processing.
2. Describe a real-world example where parallel processing is useful.
3. Describe at a high-level how parallel processing operates in a Starburst cluster.

Case study - Retail transactions

Order records on the data lake with a schema on read table referencing them

- Total data size of 20GB
- Average record size is 500 bytes
- That yields 40 million log records
- 10K records per second processing speed

Case study - Retail transactions

Order records on the data lake with a schema on read table referencing them

- Total data size of 20GB
- Average record size is 500 bytes
- That yields 40 million log records
- 10K records per second processing speed

Agreed, not really BIG data, but using that math sequentially processing this amount of data would take…

Case study - Retail transactions

Order records on the data lake with a schema on read table referencing them

- Total data size of 20GB
- Average record size is 500 bytes
- That yields 40 million log records
- 10K records per second processing speed

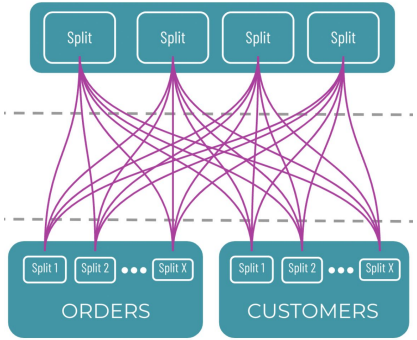
Agreed, not really BIG data, but using that math sequentially processing this amount of data would take…

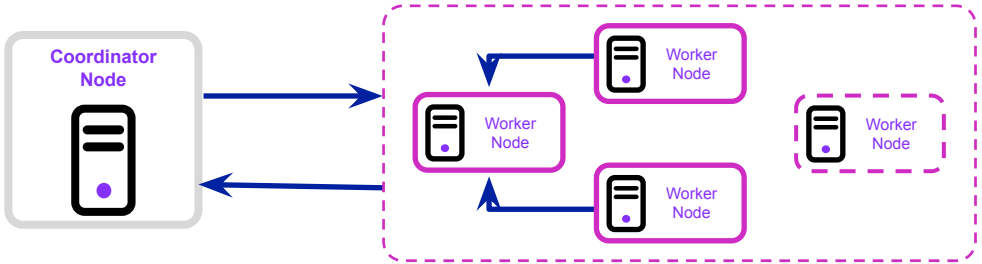
4000 SECONDS! Yep, that's > 1 hour

Parallel processing to the rescue!

Query the data lake with Starburst and Trino

Starburst





Lesson summary

Parallel processing: Case study

1. Sequential processing deals with operations as a long chain acted upon row-by-row in a single thread of execution.

2. Parallel processing splits the workload across multiple processing units and executes many tasks at once.

3. There are may real-world examples in which parallel processing is beneficial, and any large dataset will typically benefit from the approach.

4. Starburst Galaxy and Starburst Enterprise both incorporate parallel processing at their core, and the efficiencies this brings is one of the main benefits of the technology.

Parallel processing

Divide and conquer

Lesson objectives

Parallel processing: Divide and conquer

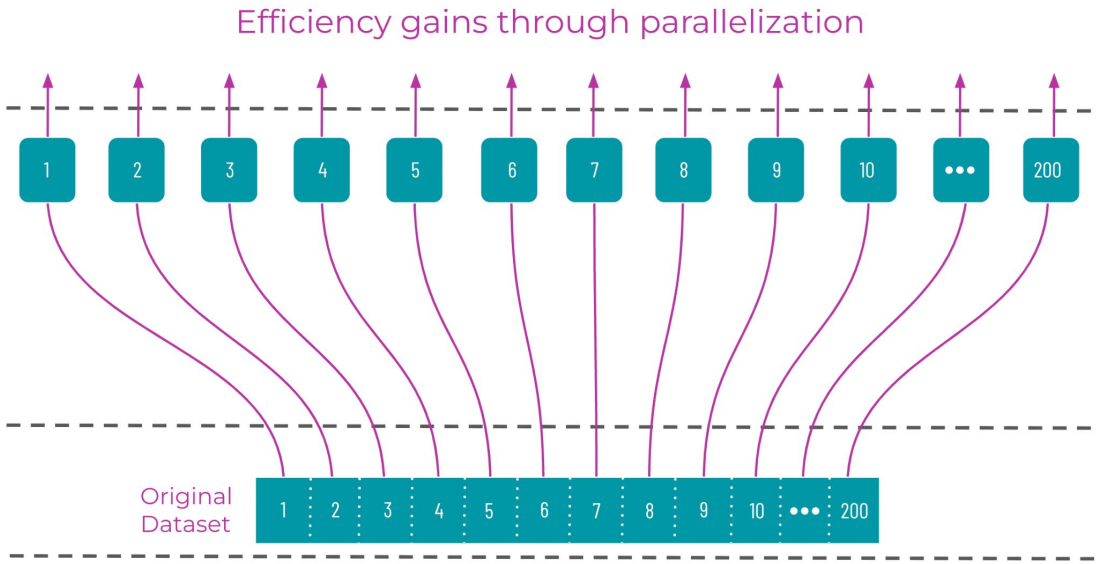
1. Explain the nature of splits and the role they play in parallelization.
2. Describe how pipelining operations help facilitate parallelization.
3. Explore some common steps in a simple parallelized data pipeline.
4. Describe how tasks and stages operate, including the role that data exchanges play in this process.

Understanding splits

Parallelization divides a single workload into independent units running concurrently

Process efficiently

- Imagine that you have the same 20GB of data as before
- Now imagine that you split this data into equal parts and divided the workload evenly
- With 200 splits, each with 200K records, the 10K/sec processing speed yields 20 secs/split

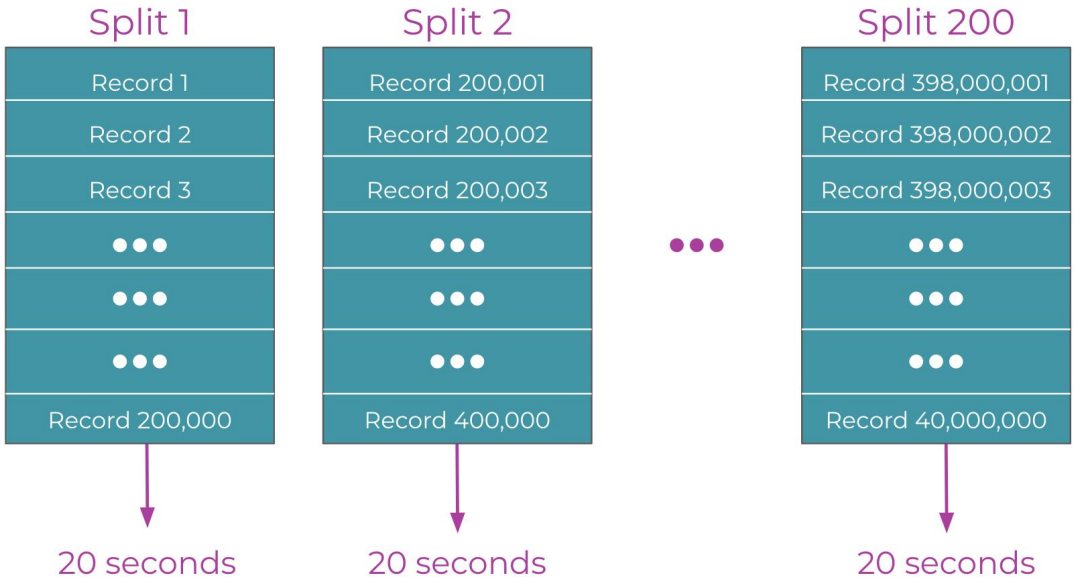


Understanding splits

Parallelization divides a single workload into independent units running concurrently

Work simultaneously

- Each split can be processed in parallel
- 20 seconds for all splits at the same time - instead of 4000 seconds if processed sequentially
- Possible if sufficient resources to run all splits at the same time



Starburst

Pipelining operations

A series of mutually exclusive operations can easily be processed in parallel.

Pipelining operations

A series of mutually exclusive operations can easily be processed in parallel. *Initial examples from queries would include:*

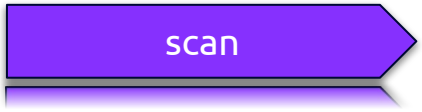
```
SELECT split_part(name, ',', 1) AS last_name,
       nationality, year_of_mission
  FROM astronauts
 WHERE military_civilian = 'military'
   AND year_of_mission BETWEEN 1960 AND 1970
   AND upper(occupation) = 'PILOT';
```

Pipelining operations

A series of mutually exclusive operations can easily be processed in parallel. Initial examples from queries would include:

- **Scanning**

```
SELECT split_part(name, ',', 1) AS last_name,
       nationality, year_of_mission
  FROM astronauts
 WHERE military_civilian = 'military'
   AND year_of_mission BETWEEN 1960 AND 1970
   AND upper(occupation) = 'PILOT';
```

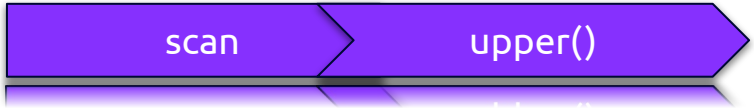


Pipelining operations

A series of mutually exclusive operations can easily be processed in parallel. Initial examples from queries would include:

- Scanning
- **Scalar functions**
- Filtering
- Projection

```
SELECT split_part(name, ',', 1) AS last_name,
       nationality, year_of_mission
  FROM astronauts
 WHERE military_civilian = 'military'
   AND year_of_mission BETWEEN 1960 AND 1970
   AND upper(occupation) = 'PILOT';
```

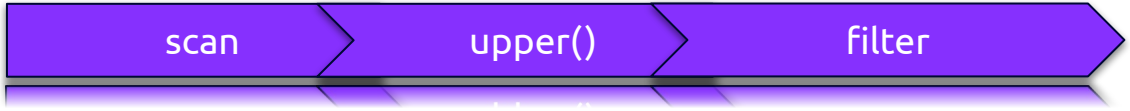


Pipelining operations

A series of mutually exclusive operations can easily be processed in parallel. Initial examples from queries would include:

- Scanning
- Scalar functions
- **Filtering**
- Projection

```
SELECT split_part(name, ',', 1) AS last_name,
       nationality, year_of_mission
  FROM astronauts
 WHERE military_civilian = 'military'
   AND year_of_mission BETWEEN 1960 AND 1970
   AND upper(occupation) = 'PILOT';
```



Pipelining operations

A series of mutually exclusive operations can easily be processed in parallel. Initial examples from queries would include:

- Scanning
- **Scalar functions**
- Filtering
- Projection

```
SELECT split_part(name, ',', 1) AS last_name,
       nationality, year_of_mission
  FROM astronauts
 WHERE military_civilian = 'military'
   AND year_of_mission BETWEEN 1960 AND 1970
   AND upper(occupation) = 'PILOT';
```

Pipelining operations

A series of mutually exclusive operations can easily be processed in parallel. Initial examples from queries would include:

- Scanning
- Scalar functions
- Filtering
- **Projection**

```
SELECT split_part(name, ',', 1) AS last_name,
       nationality, year_of_mission
  FROM astronauts
 WHERE military_civilian = 'military'
   AND year_of_mission BETWEEN 1960 AND 1970
   AND upper(occupation) = 'PILOT';
```

Pipelining operations

A series of mutually exclusive operations can easily be processed in parallel. Initial examples from queries would include:

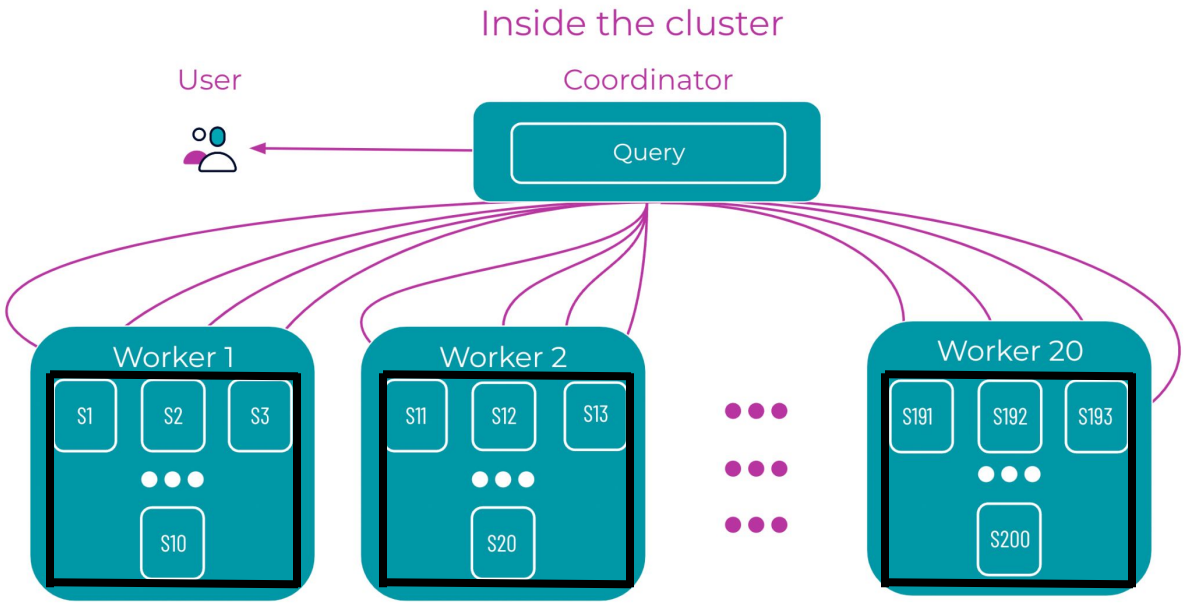
- Scanning
- Scalar functions
- Filtering
- Projection

```
SELECT split_part(name, ',', 1) AS last_name,
       nationality, year_of_mission
  FROM astronauts
 WHERE military_civilian = 'military'
   AND year_of_mission BETWEEN 1960 AND 1970
   AND upper(occupation) = 'PILOT';
```

Visualizing tasks

Tasks operate on splits

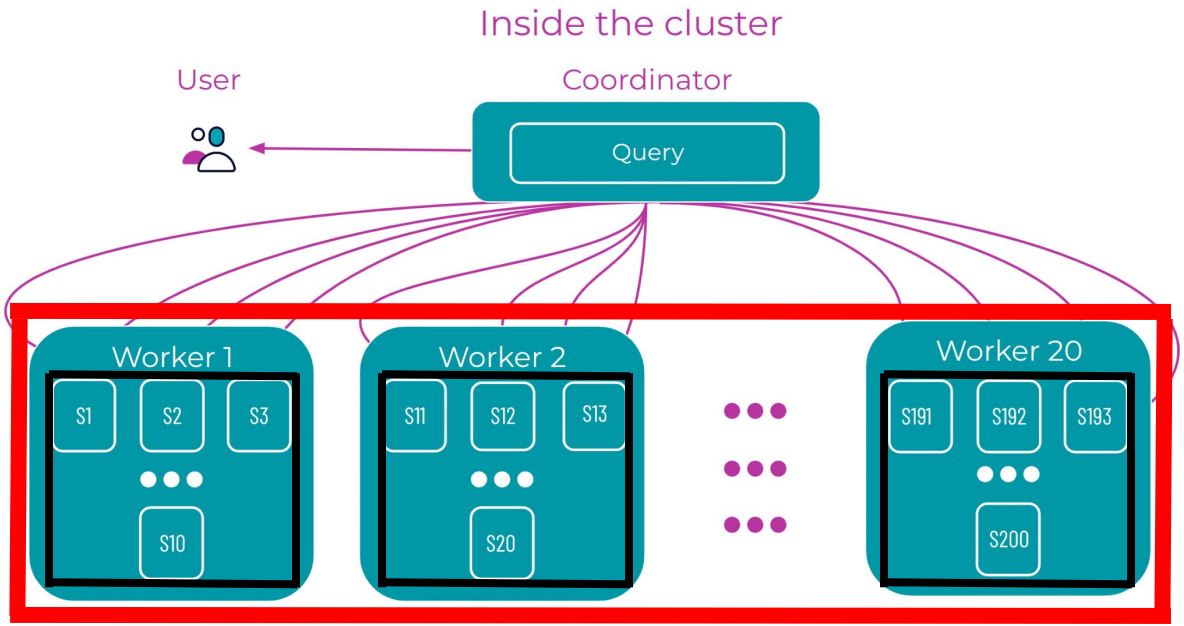
- Each worker participating in the parallelized processing of multiples splits is assigned a single task.
- Each task (again, one per worker) is assigned 1+ Splits.



Visualizing tasks, stages

Tasks operate on splits & a stage includes all those tasks

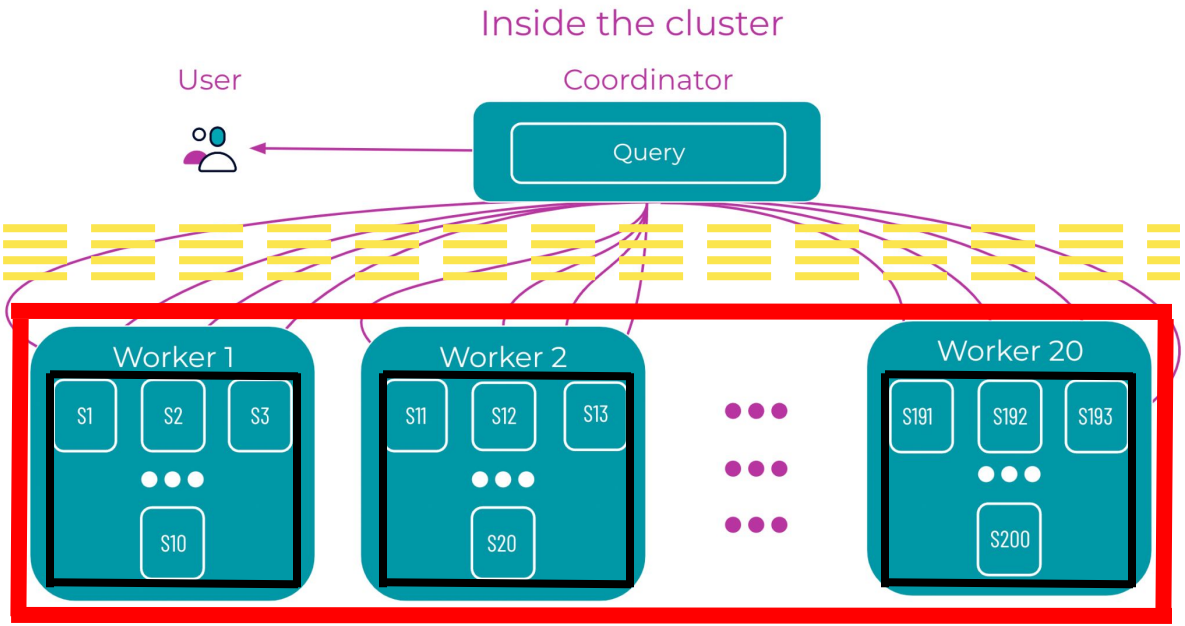
- Each worker participating in the parallelized processing of multiples splits is assigned a single task.
- Each task (again, one per worker) is assigned 1+ Splits.
- All the tasks across the cluster performing the same kind of work, just on different splits, are collectively called a stage.



Visualizing tasks, stages and exchanges

Tasks operate on splits & a stage includes all those tasks & stages exchange data

- Each worker participating in the parallelized processing of multiples splits is assigned a single task.
- Each task (again, one per worker) is assigned 1+ Splits.
- All the tasks across the cluster performing the same kind of work, just on different splits, are collectively called a stage.
- A stage transmits its results back to the coordinator, or another stage, via an exchange of data.



Lesson summary

Parallel processing: Divide and conquer

1. Parallelization operates using splits.

2. In this approach, large workloads are divided into smaller pieces and operated on in parallel.

3. Creating a chain of operations, a data pipeline, is a popular and efficient way of enacting parallelization.

4. Typical steps include: scanning, scalar functions, filtering, and projection.

5. Complex, parallel operations are executed in stages, with data transferred between each stage.

Parallel processing

Beyond single stage queries

Lesson objectives

Parallel processing: Beyond single stage queries

1. Explain how multi-stage queries operate, and describe the impact they have on parallelization.

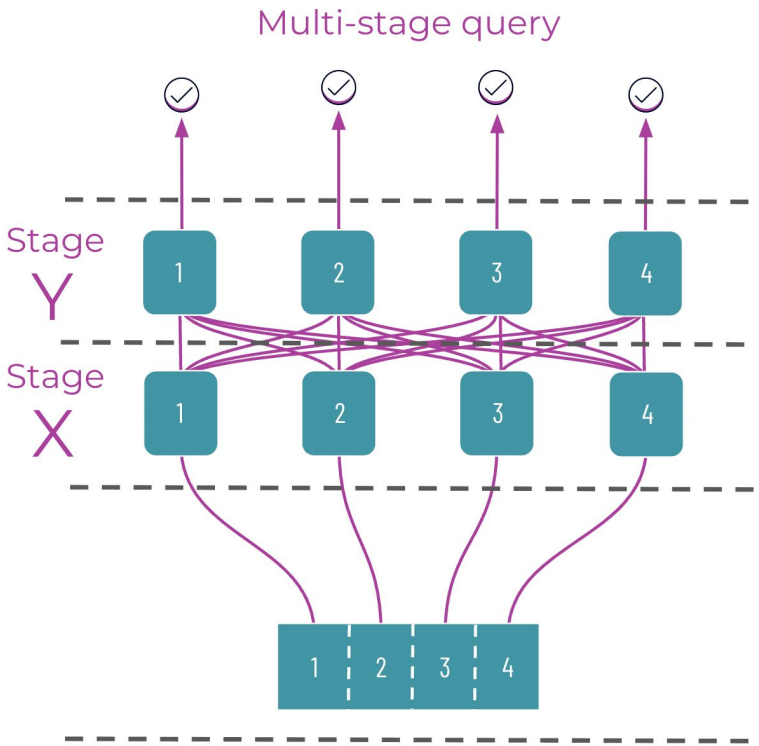
2. Describe common multi-stage scenarios and the syntax used to achieve them, including sorting, aggregation, partitioned joins, and broadcast joins.

3. Explain how a multi-stage query would function if it used all of the techniques previously described.

Multi-stage queries

Multi-stage queries

- Consider a 2 stage query.
- Two separate operation pipelines are needed.
- The first pipeline is processed in Stage X.
  - The system splits the data set.
  - Processes them in parallel.
- The system addresses the second pipeline in Stage Y.
  - The output of Stage X is exchanged to become the input of Stage Y.
  - This reordered dataset has splits, too.
  - Parallelization is employed a second time.
- After all of these operations are complete, the results of the entire query are returned to the user.



Sorting results with `ORDER BY`

```
SELECT Col_1, Col_2
  FROM table
 ORDER BY Col_1, Col_2 DESC
```

1. Scanning stage

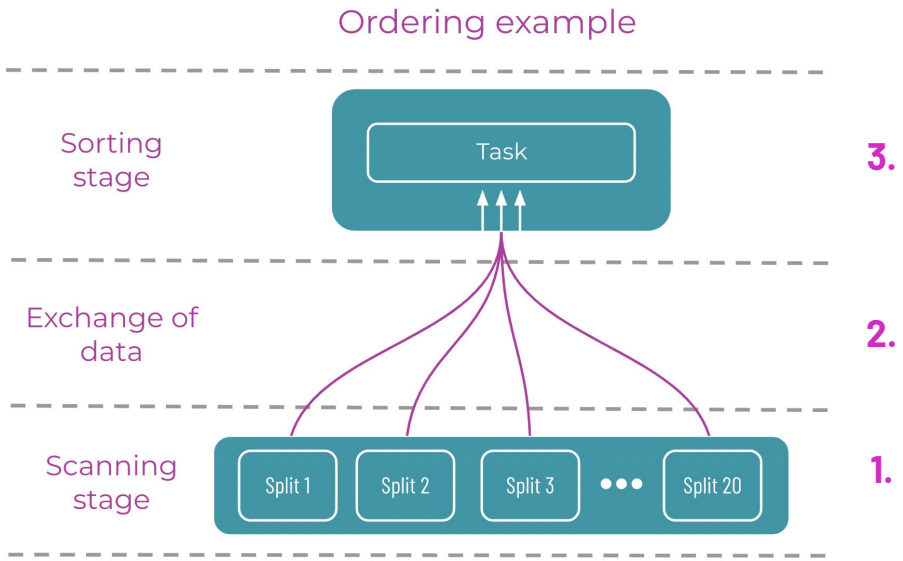
- Uses the `SELECT` command to select all of the data requested.
- This stage is divided into many splits that are operated on in parallel across the workers.

2. Exchange of data

- When each task is complete, the results are sent from each worker to the a single split dataset.

3. Sorting stage

- After the unsorted data has been received, the results will be sorted according to the `ORDER BY` clause.



Aggregation with GROUP BY

```
SELECT Col_5, max(Col_3) FROM table_name
 WHERE Col_3 IN ('value_A', 'value_B', value_C')
   AND Col_5 LIKE ('bonus%')
 GROUP BY Col_5
```

1. Scanning stage

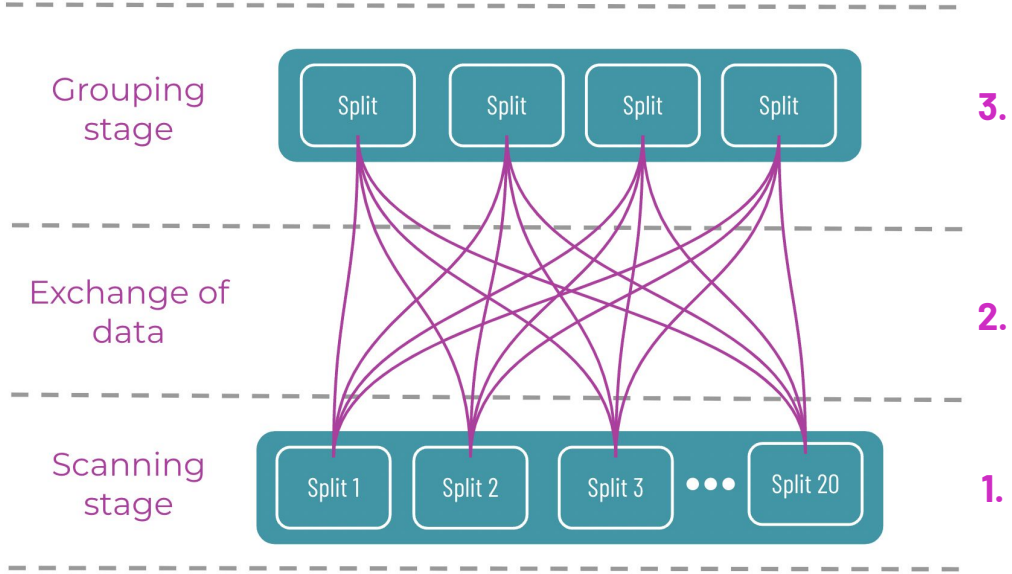
- `SELECT` command executes normally.
- Searches the database for all data that matches the query criteria.

2. Exchange of data

- Any recurrences of a given value are hashed on the aggregated column.
- For example, `Col_5`.

3. Grouping stage

- Massive transfer of hashed data.
- Collapses duplicate fields of values into individual, single values.



Lesson summary

Parallel processing: Beyond single stage queries

1. Complex queries require multiple stages.

2. Each stage is like a step, and data from a stage is transferred to the next stage.

3. ORDER BY clauses create sorting stages to sort results according to defined logical conditions.

4. GROUP BY clauses create aggregation stages to ensure data with the same key can be processed together.

5. Partitioned joins require scanning stages for both tables and a third stage for the actual join.

6. Broadcast joins cache the smaller table into all workers and only have to scan the larger table.