# SQL and NoSQL Tutorial

Design Lab
Jan 15, 2026

# SQL: A Brief Introduction

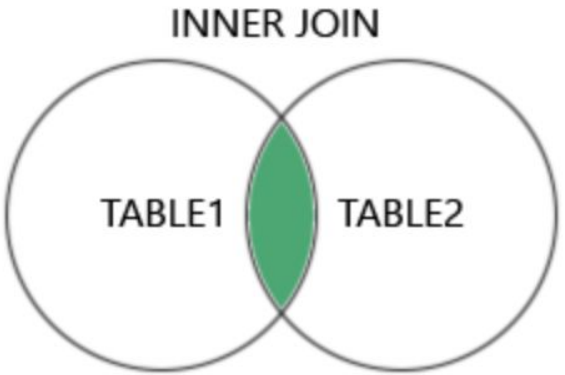
# Standard SQL Queries

- CREATE
- INSERT
- SELECT
- DELETE
- UPDATE

# JOINS in SQL

- INNER JOIN: The INNER JOIN keyword returns only rows with a match in both tables.

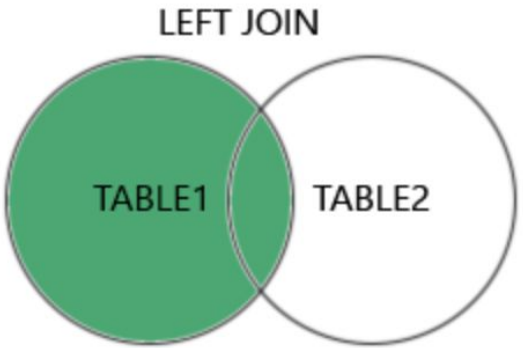
```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```



# JOINS in SQL

- LEFT JOIN: The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2).

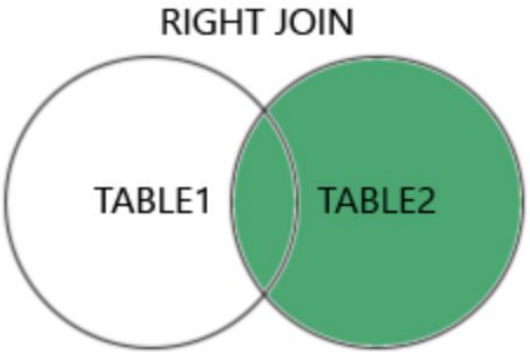
```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```



# JOINS in SQL

- RIGHT JOIN: The RIGHT JOIN keyword returns all records from the right table, even if there are no matches in the left table.

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```



## Group By and HAVING clause

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

# ANY/ALL Operators

ANY operator:

- returns TRUE if ANY of the subquery values meet the condition

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

ALL operator:

- returns TRUE if ALL of the subquery values meet the condition

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

# CREATE VIEW

- In SQL, a view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table.
- You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

# Commit and Rollback

**SQL Commit**

COMMIT is the SQL command that is used for storing changes performed by a transaction.

**SQL RollBack**

ROLLBACK is the SQL command that is used for reverting changes performed by a transaction.

# PARTITION BY in SQL

**What is PARTITION BY?**

- **Divides rows into groups**
- Used with **window functions**
- Calculation **resets for each group**
- **Does not reduce rows**

## Syntax

```
FUNCTION() OVER (
  PARTITION BY column
  ORDER BY column
)
```

**`PARTITION BY` splits data into groups without collapsing rows**

# ROW_NUMBER() in SQL

**What is ROW_NUMBER()?**

- Assigns a **serial number** to each row
- Numbers start from **1**
- Order is decided by **ORDER BY**
- **Does not remove rows**

## Syntax

```
ROW_NUMBER() OVER
(
 PARTITION BY
columns
 ORDER BY column
)
```

**`ROW_NUMBER()` + filtering = controlled row selection**

# LEAD() in SQL

**What is LEAD()?**

- Accesses the **next row's value**
- No self-join needed
- Works within an **ordered set**

**Ordered set = rows arranged in a specific order before the function runs.**

**`LEAD()` looks ahead without joins**

## Syntax

```
LEAD(column) OVER
(ORDER BY column)
```

# RANK() in SQL

**What is RANK()?**

- Assigns **rank** to rows
- **Same values → same rank**
- **Skips numbers** after ties

**`RANK()` handles ties but leaves gaps**

## Syntax

```
RANK() OVER (ORDER
BY column DESC)
```

# CASE in SQL

**What is CASE?**

- SQL's **if–else logic**
- Returns values based on conditions
- Works inside `SELECT`, `WHERE`, `ORDER BY`

**Syntax**

```
CASE
 WHEN condition THEN value
 ELSE value
END
```

**`CASE` brings decision-making into SQL**

# NoSQL Tutorial

## Map-Reduce

# SQL vs NoSQL

- SQL has tables with fixed rows and columns whereas NoSQL has dynamic data storage models (can be altered anytime).

- SQL is a general purpose database whereas NoSQL is a database used for handling documents, key-value pairs, wide-column data and graphs.

- SQL is fixed whereas NoSQL is flexible. (SQL has RDBMS schema consisting of relationship between tables and constraints defined initially which is not so for NoSQL - can be altered without handling relationships so any data can be handled with ease)

# SQL vs NoSQL

- SQL has vertical scaling (increase resource of server where SQL is present - Central). NoSQL has horizontal scaling (data distributed among other local machines - Distributed).
- SQL has ACID properties. NoSQL has CAP properties (Consistency, Availability, Partition Tolerance)
- NoSQL has the power to handle very large amounts of data unlike SQL.

# Examples

- SQL - MySQL, PostgreSQL, MS SQL Server
- NoSQL - MongoDB, Cassandra, HBase

# Advantages of NoSQL

- **Flexible Databases:** Allows you to easily make changes to your database as requirements change.

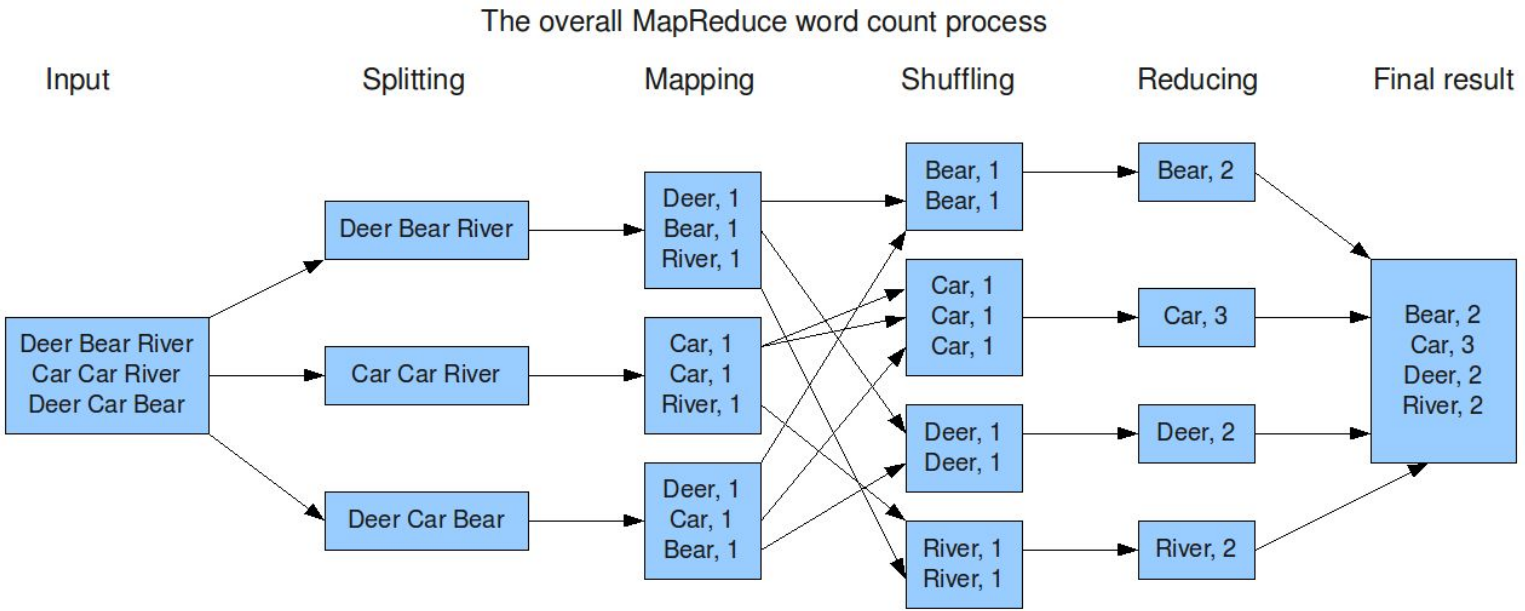
- **Horizontal scaling :** Allow you to scale-out horizontally, meaning you can add cheaper commodity servers whenever you need to.

- **Fast Queries** : NoSQL does not require joins, unlike SQL, making execution of queries faster. Data in SQL databases is typically normalized, so queries for a single object or entity require you to join data from multiple tables. Increasing table size implies faster joins.

## MapReduce

- MapReduce is a programming paradigm used for efficient processing in parallel over large data-sets in a distributed manner.
- The data is first split and then combined to produce the final result.

# MapReduce - Working



# MapReduce - Phases

- Its main use is to map the input data in key-value pairs.
- The input to the map may be a key-value pair where the key can be the id of some kind of address and value is the actual value that it keeps.
- The Map() function will be executed in its memory repository on each of these input key-value pairs and generates the intermediate key-value pair which works as input for the Reducer or Reduce() function.

## MapReduce - Phases

- The intermediate key-value pairs that work as input for Reducer are shuffled and sort and send to the Reduce() function.
- Reducer aggregate or group the data based on its key-value pair as per the reducer algorithm written by the developer.

# MapReduce - Working

- Input Phase − Here we have a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.

- Map − Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.

- Intermediate Keys − They key-value pairs generated by the mapper are known as intermediate keys.

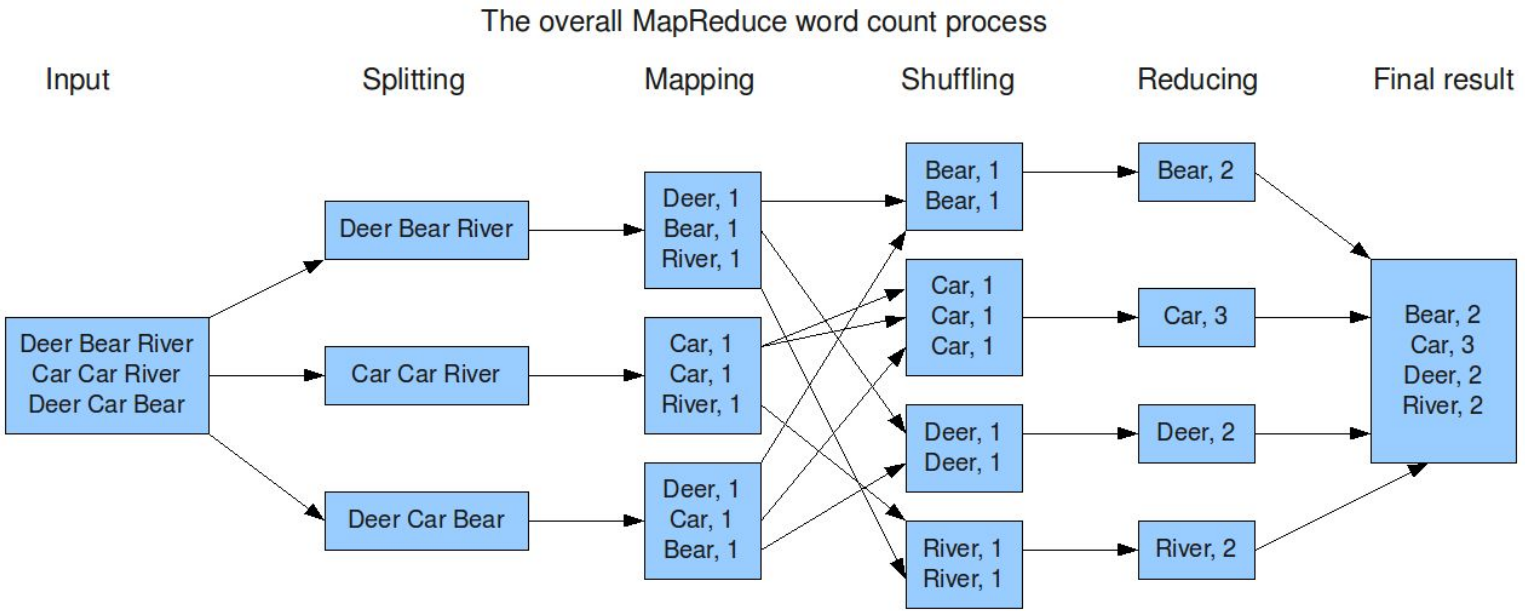
- Shuffle and Sort − The Reducer task starts with the Shuffle and Sort step. It downloads the grouped key-value pairs onto the local machine, where the Reducer is running. The individual key-value pairs are sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.

# MapReduce - Working

- Reducer − The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.

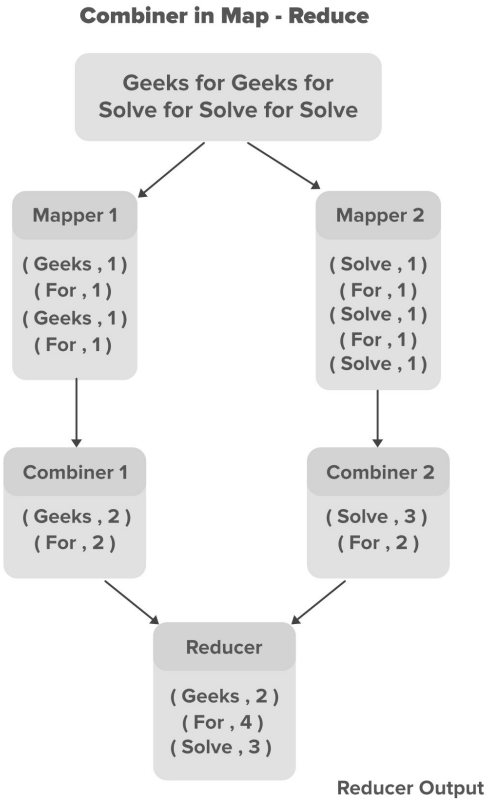
- Output Phase − In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

# MapReduce - Working



# MapReduce - Concept of Combiner

- A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is optional.



# EXAMPLE:

Find the total number of orders per country.

# Sample Dataset

```
order_id,customer_id,country,category,price,quantity,order_date
1,1293,FRA,Clothing,347.14,3,2020-09-17
2,370,GER,Clothing,291.38,4,2015-06-18
3,807,FRA,Electronics,118.83,5,2019-12-28
4,1938,USA,Sports,133.48,2,2016-11-23
5,1090,BRA,Electronics,350.16,2,2015-12-21
6,726,FRA,Books,247.64,2,2017-05-07
7,1637,GER,Clothing,459.33,3,2017-02-23
8,1759,FRA,Electronics,87.06,4,2016-10-17
9,374,IND,Clothing,275.97,2,2017-05-18
```

# Mapper Code

```
import sys

filename = sys.argv[1]

with open(filename) as f:
    for line in f:
        if line.startswith("order_id"):
            continue
        fields = line.strip().split(",")
        country = fields[2]
        print(f"{country}\t1")
```

# Combiner Code

```
import sys

current_key = None
current_count = 0

for line in sys.stdin:
    key, count = line.strip().split('\t')
    count = int(count)

    if current_key == key:
        current_count += count
    else:
        if current_key:
            print(f"{current_key}\t{current_count}")
        current_key = key
        current_count = count

if current_key:
    print(f"{current_key}\t{current_count}")
```

# Reducer Code

```
import sys

current_key = None
current_count = 0

for line in sys.stdin:
    key, count = line.strip().split('\t')
    count = int(count)

    if current_key == key:
        current_count += count
    else:
        if current_key:
            print(f"{current_key}\t{current_count}")
        current_key = key
        current_count = count

if current_key:
    print(f"{current_key}\t{current_count}")
```

# Bash File

```
#!/bin/bash

# Directory containing your data files
DATA_DIR="./data"

# Temporary file to store intermediate results
INTERMEDIATE_FILE="intermediate.txt"

# Output file for the final result
OUTPUT_FILE="result__.txt"

# Ensure the intermediate file is empty
> "$INTERMEDIATE_FILE"

# Loop through each text file in the data directory
for file in "$DATA_DIR"/*.txt; do
    # Process each file with the mapper, sort the output, then pass it to the combiner
    echo "Processing file: $file"
    python mapper.py "$file" | sort | python combiner.py >> "$INTERMEDIATE_FILE" &
done

# Wait for all background processes to complete
wait

# Sort the combined intermediate results and pass them to the reducer
sort -n "$INTERMEDIATE_FILE" | python reducer.py > "$OUTPUT_FILE"

# Clean up the intermediate file
rm "$INTERMEDIATE_FILE"
```

# Result

```
BRA 9944
FRA 9994
GER 10075
IND 9968
USA 10019
```

# THANK YOU