```
#check that java is installed
!java -version
```

```
openjdk version "11.0.27" 2025-04-15
OpenJDK Runtime Environment (build 11.0.27+6-post-Ubuntu-0ubuntu122.04)
OpenJDK 64-Bit Server VM (build 11.0.27+6-post-Ubuntu-0ubuntu122.04, mixed mode, sharing)
```

```
#install pyspark
!pip install pyspark
```

```
Requirement already satisfied: pyspark in /usr/local/lib/python3.11/dist-packages (3.5.1)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.11/dist-packages (from pyspark) (0.10.9.7)
```

```
# ============================================
# STEP 1: Install PySpark in Colab
# ============================================
!pip install pyspark --quiet

from pyspark.sql import SparkSession
from pyspark.sql.functions import broadcast
import time
```

```
# ============================================
# STEP 2: Create Spark Session
# ============================================
spark = SparkSession.builder \
    .appName("BroadcastJoinComparison") \
    .config("spark.sql.autoBroadcastJoinThreshold", -1) \
    .getOrCreate()
```

```
# ============================================
# STEP 3: Create Large DataFrame (Fact Table)
# ============================================
# Large dataset with 5 million sales records
large_data = [(i, f"Product_{i % 1000}", i % 50) for i in range(5_000_000)]
df_large = spark.createDataFrame(large_data, ["sale_id", "product_name", "store_id"])
```

```
# ============================================
# STEP 4: Create Small DataFrame (Dimension Table)
# ============================================
# Small lookup table with store information (50 rows)
small_data = [(i, f"Store_{i}", f"Region_{i % 5}") for i in range(50)]
df_small = spark.createDataFrame(small_data, ["store_id", "store_name", "region"])
```

```
# ============================================
# STEP 5: Join without broadcast
# ============================================
print("\n=== WITHOUT BROADCAST JOIN ===")
start_time = time.time()

df_normal_join = df_large.join(df_small, on="store_id", how="inner")
df_normal_join.count()  # Trigger execution

end_time = time.time()
print(f"Execution Time (No Broadcast): {round(end_time - start_time, 2)} sec")

# Show physical plan
df_normal_join.explain(extended=False)
```

```
=== WITHOUT BROADCAST JOIN ===
Execution Time (No Broadcast): 25.27 sec
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- Project [store_id#2L, sale_id#0L, product_name#1, store_name#7, region#8]
   +- SortMergeJoin [store_id#2L], [store_id#6L], Inner
      :- Sort [store_id#2L ASC NULLS FIRST], false, 0
      :  +- Exchange hashpartitioning(store_id#2L, 200), ENSURE_REQUIREMENTS, [plan_id=220]
      :     +- Filter isnotnull(store_id#2L)
      :        +- Scan ExistingRDD[sale_id#0L,product_name#1,store_id#2L]
```

```
            +- Sort [store_id#6L ASC NULLS FIRST], false, 0
               +- Exchange hashpartitioning(store_id#6L, 200), ENSURE_REQUIREMENTS, [plan_id=221]
                  +- Filter isnotnull(store_id#6L)
                     +- Scan ExistingRDD[store_id#6L,store_name#7,region#8]
```

```
# ============================================
# STEP 6: Join with broadcast
# ============================================
print("\n=== WITH BROADCAST JOIN ===")
start_time = time.time()

df_broadcast_join = df_large.join(broadcast(df_small), on="store_id", how="inner")
df_broadcast_join.count()  # Trigger execution

end_time = time.time()
print(f"Execution Time (With Broadcast): {round(end_time - start_time, 2)} sec")

# Show physical plan
df_broadcast_join.explain(extended=False)
```

```
=== WITH BROADCAST JOIN ===
Execution Time (With Broadcast): 11.15 sec
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- Project [store_id#2L, sale_id#0L, product_name#1, store_name#7, region#8]
   +- BroadcastHashJoin [store_id#2L], [store_id#6L], Inner, BuildRight, false
      :- Filter isnotnull(store_id#2L)
      :  +- Scan ExistingRDD[sale_id#0L,product_name#1,store_id#2L]
      +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, false]),false), [plan_id=372]
         +- Filter isnotnull(store_id#6L)
            +- Scan ExistingRDD[store_id#6L,store_name#7,region#8]
```

```
# ============================================
# STEP 7: Verify sample output
# ============================================
df_broadcast_join.show(5, truncate=False)
```

```
+--------+-------+------------+----------+--------+
|store_id|sale_id|product_name|store_name|region  |
+--------+-------+------------+----------+--------+
|0       |0      |Product_0   |Store_0   |Region_0|
|1       |1      |Product_1   |Store_1   |Region_1|
|2       |2      |Product_2   |Store_2   |Region_2|
|3       |3      |Product_3   |Store_3   |Region_3|
|4       |4      |Product_4   |Store_4   |Region_4|
+--------+-------+------------+----------+--------+
only showing top 5 rows
```

Start coding or generate with AI.

```
"""Plan

        Create a large fact table (sales) — 5M rows.
        Create a small dimension table (products) — 5 rows.

        Perform:
        Normal join (no broadcast hint).
        Broadcast join (force load small table into memory).

        Compare execution times.

"""
```

```
'Plan\n\n        Create a large fact table (sales) — 5M rows.\n        Create a small dimension table (products) — 5 rows.\n\n        P
erform:\n        Normal join (no broadcast hint).\n        Broadcast join (force load small table into memory).\n\n        Compare exec
ution times.\n\n'
```

```
# ============================
# STEP 1: Install & Import
```

```
# ============================
!pip install duckdb pyarrow pandas --quiet

import duckdb
import pandas as pd
import numpy as np
import time

# ============================
# STEP 2: Create Sample Data
# ============================

# Large Fact Table: 5 million sales records
num_sales = 5_000_000
sales_df = pd.DataFrame({
    'sale_id': np.arange(num_sales),
    'product_id': np.random.randint(1, 6, size=num_sales),  # IDs from 1 to 5
    'amount': np.random.uniform(10, 500, size=num_sales)
})

# Small Dimension Table: 5 products
products_df = pd.DataFrame({
    'product_id': [1, 2, 3, 4, 5],
    'product_name': ['Laptop', 'Mobile', 'Tablet', 'Camera', 'Headphones']
})

# ============================
# STEP 3: Save as Parquet (optional, but realistic)
# ============================
sales_df.to_parquet("/content/sales.parquet", compression='snappy')
products_df.to_parquet("/content/products.parquet", compression='snappy')

# ============================
# STEP 4: Connect DuckDB
# ============================
con = duckdb.connect()

# Register Parquet files as tables
con.execute("CREATE TABLE sales AS SELECT * FROM parquet_scan('/content/sales.parquet')")
con.execute("CREATE TABLE products AS SELECT * FROM parquet_scan('/content/products.parquet')")

# ============================
# STEP 5: Normal Join Benchmark
# ============================
start = time.time()
normal_join_df = con.execute("""
    SELECT s.sale_id, s.product_id, p.product_name, s.amount
    FROM sales s
    JOIN products p
    ON s.product_id = p.product_id
""").fetchdf()
end = time.time()
print(f"Normal Join Time: {round(end - start, 2)} sec")

# ============================
# STEP 6: Broadcast Join Benchmark (simulate)
# ============================
# In Spark, broadcast join sends small table to all nodes. In DuckDB,
# we'll emulate it by loading the small table fully into memory first.
products_mem = con.execute("SELECT * FROM products").fetchdf()

start = time.time()
broadcast_join_df = pd.merge(sales_df, products_mem, on='product_id', how='inner')
end = time.time()
print(f"Broadcast Join Time (simulated): {round(end - start, 2)} sec")

# ============================
# STEP 7: Validate Results
# ============================
print("\nSample Output (Broadcast Join):")
print(broadcast_join_df.head())
```

```
Normal Join Time: 1.08 sec
Broadcast Join Time (simulated): 0.46 sec

Sample Output (Broadcast Join):
```

	sale_id	product_id	amount	product_name
0	0	1	110.130285	Laptop
1	1	3	262.272151	Tablet
2	2	4	468.121586	Camera
3	3	5	86.877582	Headphones
4	4	4	374.854495	Camera

```
Start coding or generate with AI.
```