Analyzing Bitcoin Blockchain Data DMQL

Project Milestone-2

Gouri Ramdas Menon UBID: gouriram

Rohan Shrikant Thorat UBID: rohanshr

I. Introduction

The primary goal of this project is to analyze Bitcoin blockchain data to identify transaction patterns, understand the distribution of funds, and explore the flow of Bitcoin across the network. To facilitate this, a database system will be developed to efficiently store and query blockchain data, enabling faster insights and analyses.

II. Problem statement

As the volume of Bitcoin transactions grows, efficiently storing, querying, and analyzing blockchain data has become increasingly challenging. The decentralized and cryptographically secure nature of blockchain adds complexity to data management and transaction analysis. Current tools often struggle with scalability and real-time processing, limiting the ability to extract meaningful insights from vast amounts of data.

The proposed database system will solve the problem of efficiently managing and analyzing large-scale Bitcoin blockchain data. Blockchain data is highly complex and rapidly growing, consisting of millions of transactions spread across thousands of blocks. Storing and processing such vast amounts of data in a structured way is essential for extracting insights, such as tracking transaction patterns, detecting anomalies, and analyzing the flow of Bitcoin.

A. Why do you need a database instead of an Excel file?

• Excel files are limited in their ability to handle large datasets. Bitcoin blockchain data involves gigabytes or terabytes of information, far exceeding Excel's capacity.A database system can store and manage vast amounts of data while ensuring efficient access and performance.

• Databases allow for complex querying using structured query languages to retrieve specific information quickly. Excel lacks such advanced query capabilities, making it cumbersome to search, filter, and analyze large datasets.

• As data size increases, Excel becomes slow and prone to crashes. A database, however, is optimized for performance and can handle large data volumes without compromising speed or reliability.

• Blockchain data is sensitive, and maintaining its integrity is crucial. Databases have features like constraints which are crucial to maintain the integrity of complex blockchain data. Excel lacks such robust features.

• Databases can be automated and integrated with other systems, such as data analytics tools or machine learning models, allowing for seamless, real-time processing of blockchain data. Excel is not designed for these types of interactions.

III. Target User

The target users of the proposed Bitcoin blockchain database system will be individuals and organizations involved in cryptocurrency analysis, financial services, and blockchain technology. These may include:

• Blockchain Analysts and Researchers
• Cryptocurrency Exchanges
•Regulatory and Compliance Teams
• Financial Institutions

A. Database Administrator

The database will likely be administered by Data Engineers or Database Administrators (DBAs) working within the IT departments of organizations like exchanges, financial firms, or blockchain research firms. They will ensure the database operates efficiently, and data is streamed appropriately from the blockchain. These administrators will also manage database backups, data updates, and user access controls.

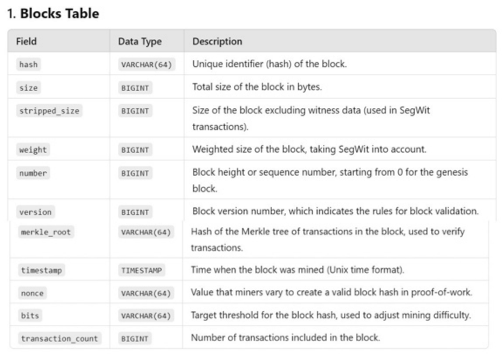
B. Real-life Scenario

When there's a large cryptocurrency exchange, "CryptoTradeX," that handles millions of Bitcoin transactions daily, relies on fast data access to ensure real-time price calculations, monitor fraud patterns, and comply with financial regulations. Their blockchain analysts regularly analyze transaction trends, while compliance officers use the system to detect and investigate potentially fraudulent transactions. Data engineers at CryptoTradeX would administer the database, ensuring smooth operation and handling any technical issues that arise, keeping the data up-to-date for various departments across the organization.

In this scenario, the database serves multiple users—analysts, compliance officers, and financial executives—while being managed by an IT team skilled in handling large-scale blockchain data systems.

IV. DATABASE SCHEMA

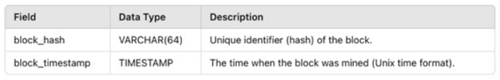
1) The database contains the following four 5 tables. The table block reference is new addition created after milestone 1. The descriptions of each field are specified in the images.



1. Blocks Table

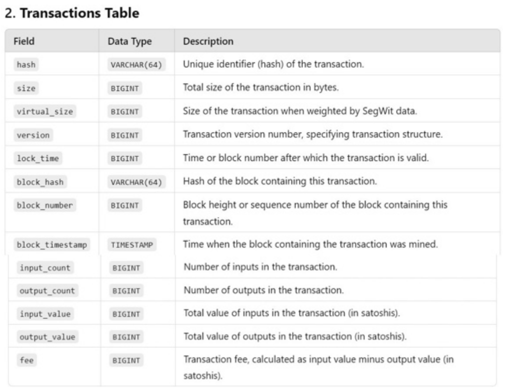
Field	Data Type	Description
hash	VARCHAR(64)	Unique identifier (hash) of the block.
size	BIGINT	Total size of the block in bytes.
stripped_size	BIGINT	Size of the block excluding witness data (used in SegWit transactions).
weight	BIGINT	Weighted size of the block, taking SegWit into account.
number	BIGINT	Block height or sequence number, starting from 0 for the genesis block.
version	BIGINT	Block version number, which indicates the rules for block validation.
merkle_root	VARCHAR(64)	Hash of the Merkle tree of transactions in the block, used to verify transactions.
timestamp	TIMESTAMP	Time when the block was mined (Unix time format).
nonce	VARCHAR(64)	Value that miners vary to create a valid block hash in proof-of-work.
bits	VARCHAR(64)	Target threshold for the block hash, used to adjust mining difficulty.
transaction_count	BIGINT	Number of transactions included in the block.

Fig. 1. Blocks Table Attributes



Field	Data Type	Description
block_hash	VARCHAR(64)	Unique identifier (hash) of the block.
block_timestamp	TIMESTAMP	The time when the block was mined (Unix time format).

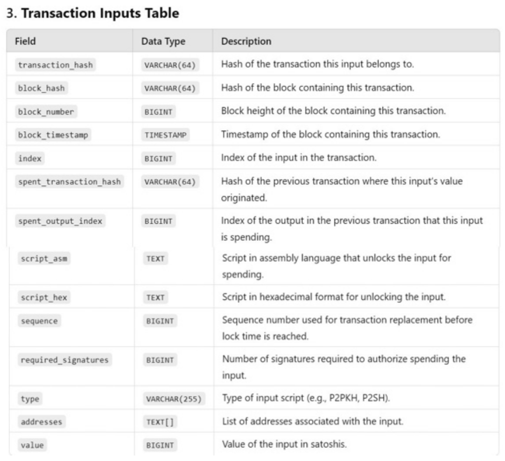
Fig. 2. Block Refernce Table Attributes



2. Transactions Table

Field	Data Type	Description
hash	VARCHAR(64)	Unique identifier (hash) of the transaction.
size	BIGINT	Total size of the transaction in bytes.
virtual_size	BIGINT	Size of the transaction when weighted by SegWit data.
version	BIGINT	Transaction version number, specifying transaction structure.
lock_time	BIGINT	Time or block number after which the transaction is valid.
block_hash	VARCHAR(64)	Hash of the block containing this transaction.
block_number	BIGINT	Block height or sequence number of the block containing this transaction.
block_timestamp	TIMESTAMP	Time when the block containing the transaction was mined.
input_count	BIGINT	Number of inputs in the transaction.
output_count	BIGINT	Number of outputs in the transaction.
input_value	BIGINT	Total value of inputs in the transaction (in satoshis).
output_value	BIGINT	Total value of outputs in the transaction (in satoshis).
fee	BIGINT	Transaction fee, calculated as input value minus output value (in satoshis).

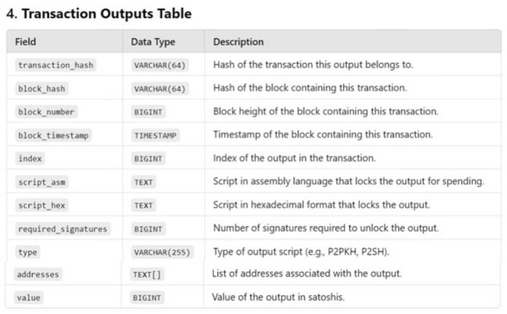
Fig. 3. Transaction Table Attributes



3. Transaction Inputs Table

Field	Data Type	Description
transaction_hash	VARCHAR(64)	Hash of the transaction this input belongs to.
block_hash	VARCHAR(64)	Hash of the block containing this transaction.
block_number	BIGINT	Block height of the block containing this transaction.
block_timestamp	TIMESTAMP	Timestamp of the block containing this transaction.
index	BIGINT	Index of the input in the transaction.
spent_transaction_hash	VARCHAR(64)	Hash of the previous transaction where this input's value originated.
spent_output_index	BIGINT	Index of the output in the previous transaction that this input is spending.
script_asm	TEXT	Script in assembly language that unlocks the input for spending.
script_hex	TEXT	Script in hexadecimal format for unlocking the input.
sequence	BIGINT	Sequence number used for transaction replacement before lock time is reached.
required_signatures	BIGINT	Number of signatures required to authorize spending the input.
type	VARCHAR(255)	Type of input script (e.g., P2PKH, P2SH).
addresses	TEXT[]	List of addresses associated with the input.
value	BIGINT	Value of the input in satoshis.

Fig. 4. Transaction Inputs Table Attributes



4. Transaction Outputs Table

Field	Data Type	Description
transaction_hash	VARCHAR(64)	Hash of the transaction this output belongs to.
block_hash	VARCHAR(64)	Hash of the block containing this transaction.
block_number	BIGINT	Block height of the block containing this transaction.
block_timestamp	TIMESTAMP	Timestamp of the block containing this transaction.
index	BIGINT	Index of the output in the transaction.
script_asm	TEXT	Script in assembly language that locks the output for spending.
script_hex	TEXT	Script in hexadecimal format that locks the output.
required_signatures	BIGINT	Number of signatures required to unlock the output.
type	VARCHAR(255)	Type of output script (e.g., P2PKH, P2SH).
addresses	TEXT[]	List of addresses associated with the output.
value	BIGINT	Value of the output in satoshis.

Fig. 5. Transaction Outputs Table Attributes

V. DATABASE DESIGN

The database design for this project focuses on capturing the essential components of the Bitcoin blockchain, including blocks, transactions, and their associated inputs and outputs. Five tables were created: blocks, transactions, transaction inputs, and transaction outputs. Each table reflects a key aspect of the blockchain, with foreign key relationships linking transactions to blocks, and inputs/outputs to transactions. This structure ensures efficient data organization and allows for detailed queries between blocks, transactions, and value movements. The schema was implemented using PostgreSQL, with careful attention to indexing and relational integrity for performance and scalability. In addition to the database design, we have also provided an ER diagram to visually represent the relationships between the key entities (tables). This diagram highlights the one-to-many relationships between blocks and transactions, as well as between transactions and their respective inputs and outputs, providing a clear overview of the data structure and dependencies within the system. The script to create all the tables is available in the TableCreation.sql file and should be run when creating the instance of the db.

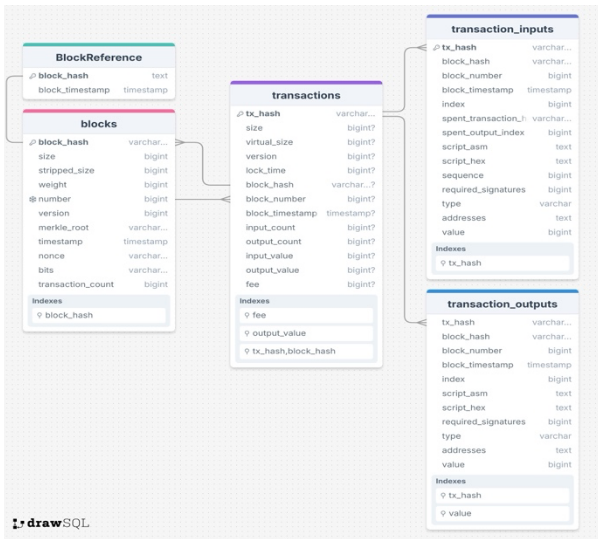


Fig. 6. Entity Relationship Diagram

IV. DATA ACQUISITION PROCESS

The blockchain data for this project was acquired using a free RPC endpoint provided by Getblocks.io and the opensource library bitcoin-etl. The following steps describe the data acquisition process:

A. Setup

A free RPC endpoint from *GetBlocks.io* was used to connect to the Bitcoin network. The *bitcoin-etl* library was used to facilitate the extraction of blockchain data in JSON format.

B. Data Collection Command

The following command was used to export blocks and transactions for block numbers ranging from 861700 to 871668:

```
!bitcoinetl export_blocks_and_transactions \
--start-block 861700 --end-block 871668 \
   --provider-uri <RPC-uri> \
   --blocks-output blocks.json \
   --transactions-output transactions.json
```

Parameters:

- --start-block: Specifies the starting block number for data collection.
- --end-block: Specifies the ending block number for data collection.
- --provider-uri: Provides the RPC endpoint URI for the Bitcoin network.
- --blocks-output: The output file for block data (blocks.json).
- --transactions-output: The output file for transaction data (transactions.json).

Output:

- blocks.json: A JSON file containing details about each block, such as block hash, size, timestamp, and transaction count.

.

C. Data Validation

After exporting the data, it was reviewed for integrity to ensure:

- All blocks within the specified range were exported.
- Transaction data matched the block data in terms of transaction counts.

D. Data Insertion into the Database

The exported JSON files (blocks.json and transactions.json) were parsed and loaded into the database in batches. This is because the transactions.json is a total of 68 GB and batching the data line by line significantly improved performance. The data insertion scripts are available in the collection_loading.ipynb file.

Batch Processing: Transactions were was inserted into the database in batches of 100,000 lines for efficiency using Python scripts.

```
def load_transactions(batch_size=100000):
        transactions_batch = []
        inputs_batch = []
        outputs_batch = []

        with open(transactions_file, 'r') as f:
                with engine.connect() as conn:
                        for line_num, line in enumerate(f,\
                        start=1):
                                transaction = json.loads(line) #
Insert batches into the database if line_num %
batch_size == 0: pd.DataFrame(transactions_batch)
.to_sql('transactions', conn, if_exists='append',
index=False) pd.DataFrame(inputs_batch)
.to_sql('transactioninputs', conn, if_exists='append',
index=False) pd.DataFrame(outputs_batch)
.to_sql('transactionoutputs', conn, if_exists='append',
index=False)

        # Clear batches transactions_batch.clear()
        inputs_batch.clear() outputs_batch.clear()
        print(f"Processed {line_num} transactions..)
```

VII. NORMALIZATION TO BOYCE-CODD NORMAL FORM (BCNF)

- block timestamp, block nonce, block bits, block transaction count}

- BlockReference: {block hash, block timestamp}
- Transactions: {tx hash, tx size, tx virtual size, tx version, tx lock time, block hash, tx _is coinbase, tx index, tx input count, tx output count, txinput value, tx output value, tx fee}
- TransactionInput: {tx hash, input index, prev tx hash, prev output index, script _ asm, script _ hex, sequence}
- TransactionOutput: {tx _hash, output index, value, script asm, script hex}

B. Functional Dependencies

The functional dependencies for each relation are listed below:

C. Functional Dependencies

The functional dependencies for each relation are as follows:

Blocks:

- block_hash determines:
  - block_size
  - block_stripped_size
  - block_weight
  - block_number
  - block_version
  - block_merkle_root
  - block_timestamp
  - block_nonce
  - block_bits
  - block_transaction_count

BlockReference:

- block_hash determines:
  - block_timestamp

Transactions:

- tx_hash determines:
  - tx_size
  - tx_virtual_size
  - tx_version
  - tx_lock_time
  - block_hash
  - tx_is_coinbase – tx_index
  - tx_input_count
  - tx_output_count
  - tx_input_value
  - tx_output_value
  - tx_fee

TransactionInputs:

- {tx_hash, input_index} determines:
  - prev_tx_hash
  - prev_output_index
  - script_asm
  - script_hex
  - sequence

TransactionOutputs:

- {tx_hash, output_index} determines:
  - value
  - script_asm
  - script_hex

D. Boyce-Codd Normal Form (BCNF) Verification

The schema was analyzed to ensure all relations are in BCNF. A relation is in BCNF if, for every non-trivial functional dependency $X \rightarrow Y$, the determinant X is a superkey.

1) Block Table: - Attributes: {block hash, block _size, block stripped size, block weight, block number, block _version, block _merkle root, block timestamp, block nonce, block bits, block transaction count}. Analysis: The primary key block _hash is a superkey, and all non-trivial dependencies satisfy the BCNF condition. - Conclusion: The Block table is in BCNF.

2) BlockReference Table: - Attributes: {block hash, block timestamp}. - Analysis: The primary key block hash is a superkey, ensuring BCNF compliance. Conclusion: The BlockReference table is in BCNF. *3) Transactions Table:* - Attributes: {tx hash, txsize, tx virtual size, tx version, tx _lock time, block_hash, tx _is coinbase, tx index, tx input count, tx output count, tx input value, tx output value, tx fee}. Analysis: The primary key tx hash is a superkey, and no further decomposition is required. - Conclusion: The Transactions table is in BCNF.

4) TransactionInput Table: - Attributes: {tx hash, input index, prev tx hash, prev output index, script asm, script hex, sequence}. - Analysis: The composite primary key {tx hash, input index} is a superkey, there are only two attributes in the table, hence the table is automatically in BCNF form. - Conclusion: The TransactionInput table is in BCNF.

5) TransactionOutput Table: - Attributes: {tx hash, output index, value, script asm, script hex}. - Analysis: The composite primary key {tx hash, output index} is a superkey, ensuring BCNF compliance. - Conclusion: The TransactionOutput table is in BCNF.

E. Conclusion

The final schema remains the same as the initial schema. All relations are in BCNF, ensuring that there is no redundancy or update anomalies.

VIII. CHALLENGES AND SOLUTIONS FOR HANDLING LARGE DATASET (Indexing)

While handling the 70 GB blockchain dataset, a performance issue was identified during query execution:

A. Challenge: Query Performance

A query filtering transactions by block_hash took approximately 40 seconds to execute. The reasons for the poor performance were:

The Transactions table contained millions of rows.

No index was present on the block_hash column. • A full table scan was required to retrieve matching rows.

B. Solution: Indexing with Clustered Index

To address the performance issue, a clustered index was implemented on the block_hash column. A clustered index physically organizes the table rows to store data with the same key value close together. This significantly improved query performance.

The steps involved in implementing the clustered index included:

- Creating an index on the block_hash column.
- Clustering the table using the created index.

C. Results

After implementing the clustered index:

- Query execution time reduced from 40 seconds to less than 1 second.
- Queries filtering by block_hash became significantly faster.

D. Conclusion

Various other indexes were also implemented but none provided large significant performance improvements. All index creation code is available in the indexes.sql script.

The clustered index optimized the physical data storage in the Transactions table, enabling efficient query execution for large datasets. This solution demonstrates the importance of indexing and storage optimization, particularly for:

- Filtering queries, such as those using block_hash. • Join operations, where block_hash is used as a foreign key.

IX. DATABASE CONSTRAINTS

A. Database Constraints

To ensure data integrity and consistency, the following constraints were implemented in the database:

- Foreign Key Constraints with Cascading Deletes:
  - In the Transactions table, the block_hash column references Block(block_hash). The ON DELETE CASCADE clause ensures that when a block is deleted, all associated transactions are automatically removed. This maintains referential integrity by preventing orphaned transactions.
  - In the TransactionOutput table, the tx_hash column references Transactions(tx_hash). The ON DELETE CASCADE clause ensures that when a transaction is deleted, all its outputs are also removed, avoiding inconsistencies in the output data.
- Check Constraints:
  - In the Transactions table, the tx_fee must be greater than or equal to 0. This constraint ensures that transaction fees are always non-negative, which aligns with blockchain principles where fees cannot be negative.
  - In the Block table, the block_weight must be greater than 0. This constraint ensures that blocks have a valid positive weight, reflecting their size and significance in the blockchain network. Blocks with zero or negative weight would not be valid in real-world blockchain systems.

B. Validation with Real-World Data

The schema and constraints were validated using realworld blockchain data samples. The validation process ensured that:

- The foreign key constraints maintained referential integrity between related tables, ensuring no orphaned records in the database.
- The check constraints enforced logical consistency, preventing invalid entries such as negative fees or invalid block weights.
- The schema adhered to database normalization principles, minimizing redundancy and ensuring scalability for large datasets.

C. Conclusion

The finalized schema, supported by the described constraints, ensures that the database is robust, consistent, and capable of handling large-scale blockchain data efficiently. The constraints enforce data integrity by:

- Automatically maintaining relationships between blocks, transactions, and outputs through cascading deletes.
- Ensuring logical validity by preventing negative fees and invalid block weights.

This approach not only enforces data integrity but also provides a flexible framework for future extensions and scalability.

D. Conclusion

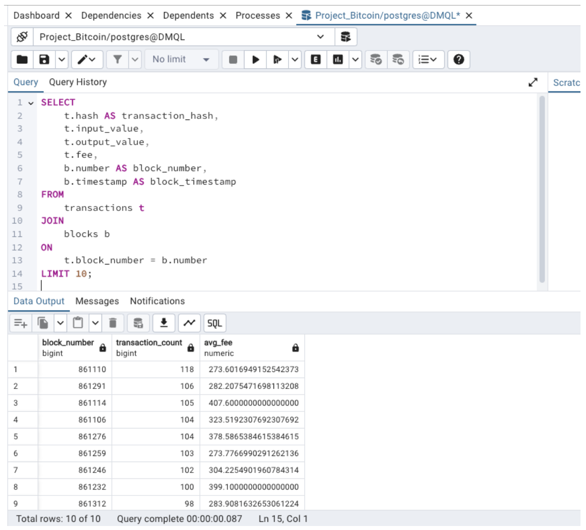
The finalized schema, supported by the described constraints, ensures that the database is robust, consistent, and capable of handling large-scale blockchain data efficiently. The constraints enforce data integrity while maintaining flexibility for future extensions.

X. QUERY EXECUTION

We can use various queries in the Bitcoin Database to uncover insights into transactions occurring on the blockchain. This Bitcoin blockchain database allows for a deep analysis of transactional and block-level insights. By querying the data, we can gather information on transaction patterns, such as the average transaction fees and input/output values. We can also track the frequency and size of blocks over time, understand trends in mining difficulty (using block size and weight), and analyze wallet activity by exploring transaction inputs and outputs. Furthermore, insights into the distribution of BTC across addresses, transaction volumes, and correlations between block times and transaction types can provide valuable knowledge about the network's performance and behavior. Here we show four examples of queries on the subset data by using various SQL queries that demonstrate the use of Group BY, subqueries, Join, and aggregate functions.

Note: Insertion queries are performed during data insertion. Since fundamentally blockchains are immutable we cannot perform any update or delete operations. Updates and deletes are prohibited to prevent tampering or fraud, ensuring the integrity of historical data. As a result, any changes must be handled by appending new data, rather than modifying existing records.

A. Using Joins Retrieve details of transactions along with their corresponding block information.



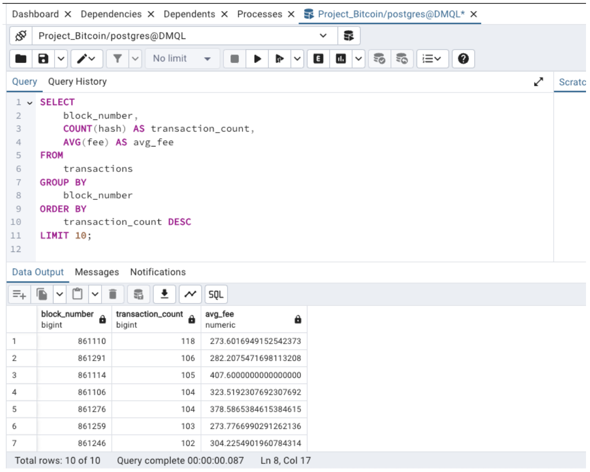
```sql
SELECT
    t.hash AS transaction_hash,
    t.input_value,
    t.output_value,
    t.fee,
    b.number AS block_number,
    b.timestamp AS block_timestamp
FROM
    transactions t
JOIN
    blocks b
ON
    t.block_number = b.number
LIMIT 10;
```

	block_number bigint	transaction_count bigint	avg_fee numeric
1	861110	118	273.6016949152542373
2	861291	106	282.2075471698113208
3	861114	105	407.6000000000000000
4	861106	104	323.5192307692307692
5	861276	104	378.5865384615384615
6	861259	103	273.7766990291262136
7	861246	102	304.2254901960784314
8	861232	100	399.1000000000000000
9	861312	98	283.9081632653061224

```
Total rows: 10 of 10    Query complete 00:00:00.087    Ln 15, Col 1
```

Fig. 7. Using simple SELECT JOIN

B. Using Group By finding the total number of transac-tions and the average fee for each block.



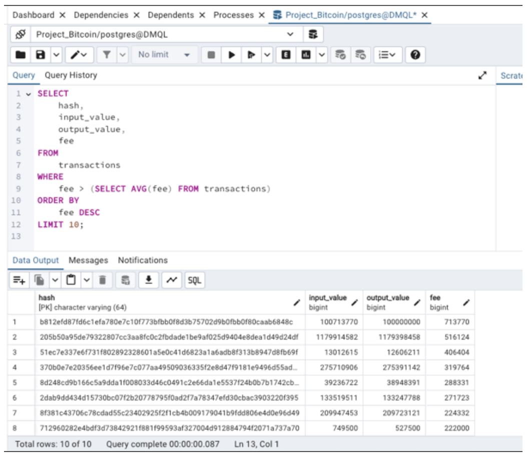
```sql
SELECT
    block_number,
    COUNT(hash) AS transaction_count,
    AVG(fee) AS avg_fee
FROM
    transactions
GROUP BY
    block_number
ORDER BY
    transaction_count DESC
LIMIT 10;
```

	block_number bigint	transaction_count bigint	avg_fee numeric
1	861110	118	273.6016949152542373
2	861291	106	282.2075471698113208
3	861114	105	407.6000000000000000
4	861106	104	323.5192307692307692
5	861276	104	378.5865384615384615
6	861259	103	273.7766990291262136
7	861246	102	304.2254901960784314

```
Total rows: 10 of 10    Query complete 00:00:00.087    Ln 8, Col 17
```

Fig. 8. Using Group By

C. Using Sub-query to retrieve transactions where the fee is greater than the average fee across all transactions.



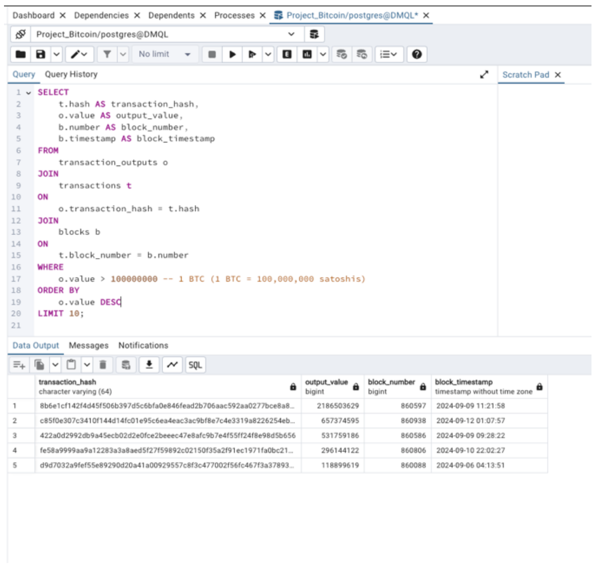
```sql
SELECT
    hash,
    input_value,
    output_value,
    fee
FROM
    transactions
WHERE
    fee > (SELECT AVG(fee) FROM transactions)
ORDER BY
    fee DESC
LIMIT 10;
```

	hash [PK] character varying (64)	input_value bigint	output_value bigint	fee bigint
1	b812efd87fd6c1efa780e7c10f773bfbb0f8d3b75702d9b0fbb0f80caab6848c	100713770	100000000	713770
2	205b50a95de79322807cc3aa8fc0c2fbdade1be9af025d9404e8dea1d49d24df	1179914582	1179398458	516124
3	51ec7e337e6f731f802892328601a5e0c41d6823a1a6adb8f313b8947d8fb69f	13012615	12606211	406404
4	370b0e7e20356ee1d7f96e7c077aa49509036335f2e8d47f9181e9496d55ad...	275710906	275391142	319764
5	8d248cd9b166c5a9dda1f008033d46c0491c2e66da1e5537f24b0b7b1742cb...	39236722	38948391	288331
6	2dab9dd434d15730bc07f2b20778795f0ad2f7a78347efd30cbac3903220f395	133519511	133247788	271723
7	8f381c43706c78cdad55c23402925f2f1cb4b009179041b9fdd806e4d0e96d49	209947453	209723121	224332
8	712960282e4bdf3d73842921f881f99593af327004d912884794f2071a737a70	749500	527500	222000

Total rows: 10 of 10 Query complete 00:00:00.087 Ln 13, Col 1

Fig. 9. Using Sub-queries

D. Using Joins with filtering to list all transactions with an output value greater than 1 BTC and show the associated block details.

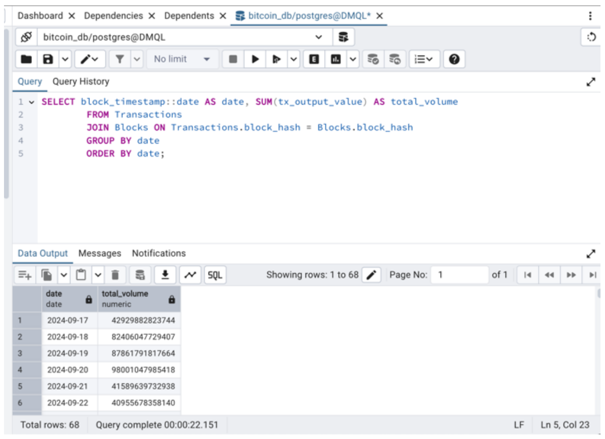


```sql
SELECT
    t.hash AS transaction_hash,
    o.value AS output_value,
    b.number AS block_number,
    b.timestamp AS block_timestamp
FROM
    transaction_outputs o
JOIN
    transactions t
ON
    o.transaction_hash = t.hash
JOIN
    blocks b
ON
    t.block_number = b.number
WHERE
    o.value > 100000000 -- 1 BTC (1 BTC = 100,000,000 satoshis)
ORDER BY
    o.value DESC
LIMIT 10;
```

	transaction_hash character varying (64)	output_value bigint	block_number bigint	block_timestamp timestamp without time zone
1	8b6e1cf142f4d45f506b397d5c6bfa0e846fead2b706aac592aa0277bce8a8...	2186503629	860597	2024-09-09 11:21:58
2	c85f0e307c3410f144d14fc01e95c6ea4eac3ac9bf8e7c4e3319a8226254eb...	657374595	860938	2024-09-12 01:07:57
3	422a0d2992db9a45ecb02d2e0fce2beeec47e8afc9b7e4f55ff24f8e98d5b656	531759186	860586	2024-09-09 09:28:22
4	fe58a9999aa9a12283a3a8aed5f27f59892c02150f35a2f91ec1971fa0bc21...	296144122	860806	2024-09-10 22:02:27
5	d9d7032a9fef55e89290d20a41a00929557c8f3c477002f56fc467f3a37893...	118899619	860088	2024-09-06 04:13:51

Fig. 10. Using Joins with Filtering

E. Daily Transaction Volume in Satoshis.



```sql
SELECT block_timestamp::date AS date, SUM(tx_output_value) AS total_volume
        FROM Transactions
        JOIN Blocks ON Transactions.block_hash = Blocks.block_hash
        GROUP BY date
        ORDER BY date;
```

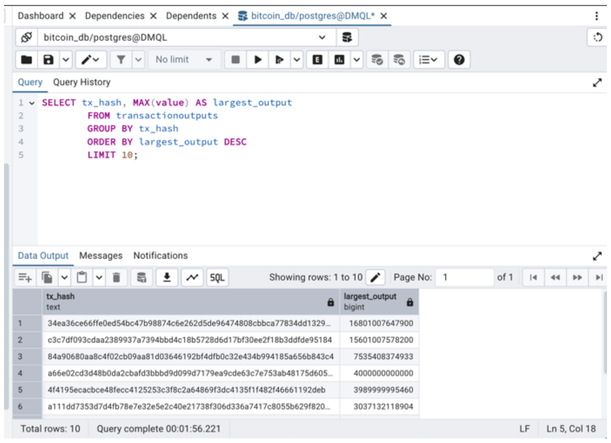
Showing rows: 1 to 68 Page No: 1 of 1

	date date	total_volume numeric
1	2024-09-17	42929882823744
2	2024-09-18	82406047729407
3	2024-09-19	87861791817664
4	2024-09-20	98001047985418
5	2024-09-21	41589639732938
6	2024-09-22	40955678358140

Total rows: 68 Query complete 00:00:22.151 LF Ln 5, Col 23

Fig. 11. Daily Transaction Volume

F. Largest Transaction by output value in Satoshis



```sql
SELECT tx_hash, MAX(value) AS largest_output
        FROM transactionoutputs
        GROUP BY tx_hash
        ORDER BY largest_output DESC
        LIMIT 10;
```

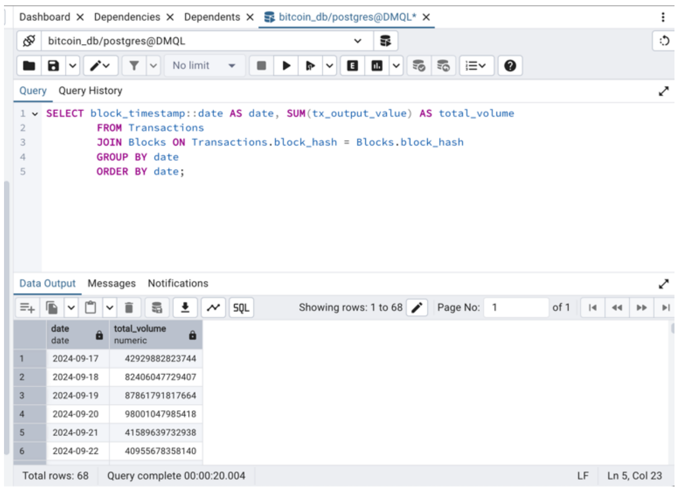
Showing rows: 1 to 10 Page No: 1 of 1

	tx_hash text	largest_output bigint
1	34ea36ce66ffe0ed54bc47b98874c6e262d5de96474808cbbca77834dd1329...	16801007647900
2	c3c7df093cdaa2389937a7394bbd4c18b5728d6d17bf30ee2f18b3ddfde95184	15601007578200
3	84a90680aa8c4f02cb09aa81d03646192bf4dfb0c32e434b994185a656b843c4	7535408374933
4	a66e02cd3d48b0da2cbafd3bbbd9d099d7179ea9cde63c7e753ab48175d605...	4000000000000
5	4f4195ecacbce48fecc4125253c3f8c2a64869f3dc4135f1f482f46661192deb	3989999995460
6	a111dd7353d7d4fb78e7e32e5e2c40e21738f306d336a7417c8055b629f820...	3037132118904

Total rows: 10 Query complete 00:01:56.221 LF Ln 5, Col 18

Fig. 12. Largest Transaction by output value

G. Transactions with the highest fees



```sql
SELECT block_timestamp::date AS date, SUM(tx_output_value) AS total_volume
        FROM Transactions
        JOIN Blocks ON Transactions.block_hash = Blocks.block_hash
        GROUP BY date
        ORDER BY date;
```

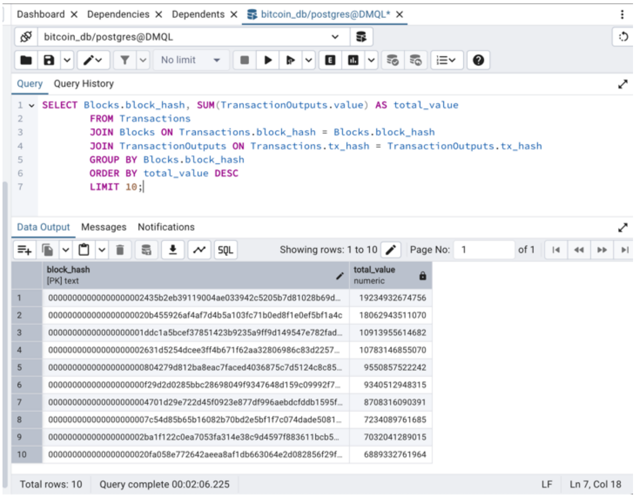
Showing rows: 1 to 68 Page No: 1 of 1

	date date	total_volume numeric
1	2024-09-17	42929882823744
2	2024-09-18	82406047729407
3	2024-09-19	87861791817664
4	2024-09-20	98001047985418
5	2024-09-21	41589639732938
6	2024-09-22	40955678358140

Total rows: 68 Query complete 00:00:20.004 LF Ln 5, Col 23

Fig. 13. Largest Transaction by output value

H. Top 10 Blocks by Total Transaction Value



```sql
SELECT Blocks.block_hash, SUM(TransactionOutputs.value) AS total_value
        FROM Transactions
        JOIN Blocks ON Transactions.block_hash = Blocks.block_hash
        JOIN TransactionOutputs ON Transactions.tx_hash = TransactionOutputs.tx_hash
        GROUP BY Blocks.block_hash
        ORDER BY total_value DESC
        LIMIT 10;
```

	block_hash [PK] text	total_value numeric
1	00000000000000000002435b2eb39119004ae033942c5205b7d81028b69d...	19234932674756
2	0000000000000000000020b455926af4af7d4b5a103fc71b0ed8f1e0ef5bf1a4c	18062943511070
3	00000000000000000001ddc1a5bcef37851423b9235a9ff9d149547e782fad...	10913955614682
4	00000000000000000002631d5254dcee3ff4b671f62aa32806986c83d2257...	10783146855070
5	000000000000000000000804279d812ba8eac7faced4036875c7d5124c8c85...	9550857522242
6	0000000000000000000f29d2d0285bbc28698049f9347648d159c09992f7...	9340512948315
7	000000000000000000004701d29e722d45f0923e877df996aebdcfddb1595f...	8708316090391
8	0000000000000000000007c54d85b65b16082b70bd2e5bf1f7c074dade5081...	7234089761685
9	00000000000000000002ba1f122c0ea7053fa314e38c9d4597f883611bcb5...	7032041289015
10	000000000000000000020fa058e772642aeea8af1db663064e2d082856f29f...	6889332761964

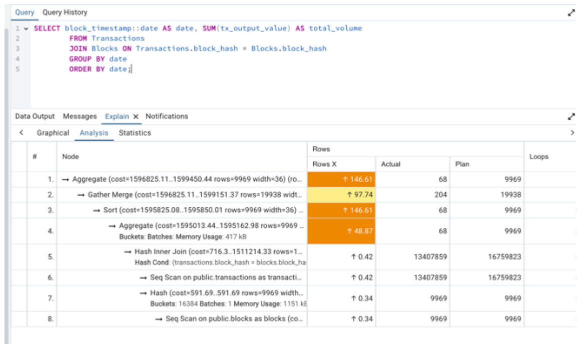
Total rows: 10 Query complete 00:02:06.225 LF Ln 7, Col 18

Fig. 14. Largest Transaction by output value

XI. QUERY EXECUTION ANALYSIS AND OPTIMIZATION

This section analyzes the execution plans for three queries to identify performance bottlenecks and proposes optimizations to improve query efficiency.

A. *Query 1: Daily Transaction Volume*



```sql
SELECT block_timestamp::date AS date, SUM(tx_output_value) AS total_volume
        FROM Transactions
        JOIN Blocks ON Transactions.block_hash = Blocks.block_hash
        GROUP BY date
        ORDER BY date;
```

#	Node	Rows			Loops
		Rows X	Actual	Plan	
1.	→ Aggregate (cost=1596825.11..1599450.44 rows=9969 width=36) (ro...	↑ 146.61	68	9969	
2.	→ Gather Merge (cost=1596825.11..1599151.37 rows=19938 widt...	↑ 97.74	204	19938	
3.	→ Sort (cost=1595825.08..1595850.01 rows=9969 width=36) ...	↑ 146.61	68	9969	
4.	→ Aggregate (cost=1595013.44..1595162.98 rows=9969 ... Buckets: Batches: Memory Usage: 417 kB	↑ 48.87	68	9969	
5.	→ Hash Inner Join (cost=716.3..1511214.33 rows=1... Hash Cond: (transactions.block_hash = blocks.block_ha	↑ 0.42	13407859	16759823	
6.	→ Seq Scan on public.transactions as transacti...	↑ 0.42	13407859	16759823	
7.	→ Hash (cost=591.69..591.69 rows=9969 width... Buckets: 16384 Batches: 1 Memory Usage: 1151 kE	↑ 0.34	9969	9969	
8.	→ Seq Scan on public.blocks as blocks (co...	↑ 0.34	9969	9969	

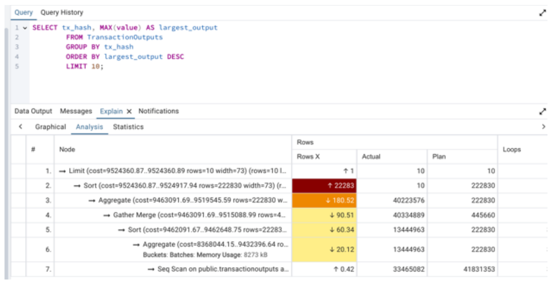
Query Objective: Calculate the total transaction output value for each day. Issues Identified:

- **Sequential Scans**: Sequential scans on the Blocks and Transactions tables increase the cost significantly.
- **Hash Joins**: The hash join between Blocks and Transactions consumes memory and processing resources due to the large number of rows.
- **Sorting Overhead**: Sorting by date before grouping adds to the computational cost.

Proposed Optimizations:

- Create indexes on block_hash in the Transactions and Blocks tables to speed up joins.
- Use a materialized view to precompute and store daily transaction volumes.
- Apply filters earlier to reduce the dataset size before joining and grouping.

B. *Query 2: Transactions with Largest Output*



```sql
SELECT tx_hash, MAX(value) AS largest_output
        FROM TransactionOutputs
        GROUP BY tx_hash
        ORDER BY largest_output DESC
        LIMIT 10;
```

#	Node	Rows			Loops
		Rows X	Actual	Plan	
1.	→ Limit (cost=9524360.87..9524360.89 rows=10 width=73) (rows=10 l...	↑ 1	10	10	
2.	→ Sort (cost=9524360.87..9524917.94 rows=222830 width=73) (r...	↑ 22283	10	222830	
3.	→ Aggregate (cost=9463091.69..9519545.59 rows=222830 w...	↓ 180.52	40223576	222830	
4.	→ Gather Merge (cost=9463091.69..9515088.99 rows=4...	↓ 90.51	40334889	445660	
5.	→ Sort (cost=9462091.67..9462648.75 rows=22283...	↓ 60.34	13444963	222830	
6.	→ Aggregate (cost=8368044.15..9432396.64 ro... Buckets: Batches: Memory Usage: 8273 kB	↓ 20.12	13444963	222830	
7.	→ Seq Scan on public.transactionoutputs a...	↑ 0.42	33465082	41831353	

Query Objective: Retrieve the top 10 transactions with the highest output value.

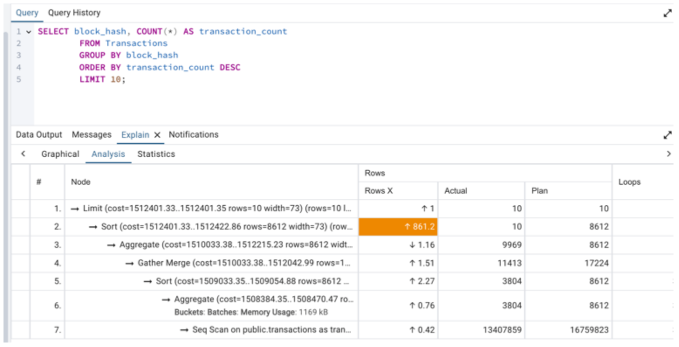
Issues Identified:

- **Sequential Scans**: The query performs a sequential scan on the TransactionOutputs table, which is inefficient given its large size.
- **Aggregation Overhead**: The MAX function applied on a large dataset increases the computation time.
- **Sorting Cost**: Sorting all transaction outputs by value before applying the LIMIT adds unnecessary overhead.

Proposed Optimizations:

- Create an index on the value column in the TransactionOutputs table to reduce sorting time.
- Partition the table by tx_hash to improve the efficiency of grouping and aggregation.

C. *Query 3: Blocks with the Most Transactions*



```sql
SELECT block_hash, COUNT(*) AS transaction_count
        FROM Transactions
        GROUP BY block_hash
        ORDER BY transaction_count DESC
        LIMIT 10;
```

#	Node	Rows			Loops
		Rows X	Actual	Plan	
1.	→ Limit (cost=1512401.33..1512401.35 rows=10 width=73) (rows=10 l...	↑ 1	10	10	
2.	→ Sort (cost=1512401.33..1512422.86 rows=8612 width=73) (row...	↑ 861.2	10	8612	
3.	→ Aggregate (cost=1510033.38..1512215.23 rows=8612 widt...	↓ 1.16	9969	8612	
4.	→ Gather Merge (cost=1510033.38..1512042.99 rows=1...	↑ 1.51	11413	17224	
5.	→ Sort (cost=1509033.35..1509054.88 rows=8612 ...	↑ 2.27	3804	8612	
6.	→ Aggregate (cost=1508384.35..1508470.47 ro... Buckets: Batches: Memory Usage: 1169 kB	↑ 0.76	3804	8612	
7.	→ Seq Scan on public.transactions as tran...	↑ 0.42	13407859	16759823	

Query Objective: Identify the top 10 blocks with the highest transaction count.

Issues Identified:

- **Sequential Scans**: Sequential scans on the Transactions table increase cost.
- **Aggregation and Sorting Overhead**: Aggregating and sorting the transaction counts for all blocks add significant computation time.

- **Lack of Indexing:** Absence of indexes on block_hash leads to inefficient data access during grouping and sorting.

Proposed Optimizations:

- Create an index on the block_hash column in the Transactions table to optimize grouping.
- Use a materialized view to precompute transaction counts for each block.
- Implement parallel processing for sorting and aggregation operations.

D. Conclusion

The analysis identified sequential scans, hash joins, and aggregation overhead as the primary performance bottlenecks. The proposed optimizations, such as indexing, materialized views, and parallel query execution, are expected to significantly reduce query execution times and improve overall performance for large datasets.

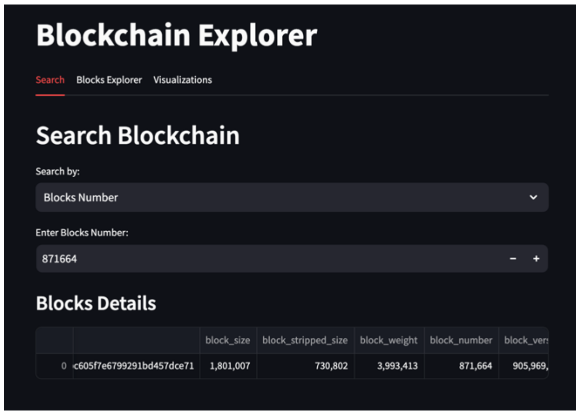
XI. Bonus Website

A. Overview

A web-based Blockchain Explorer was developed to interface with the blockchain database. The website allows users to perform various queries, visualize data, and explore blockchain blocks and transactions. Below are the key features and their explanations

B. Search Blockchain:

This page allows users to search the blockchain by block attributes, such as block number or block hash. The user enters a block number, and the website retrieves and displays detailed information about the block, including its size, stripped size, weight, and version. Note Since the database is 70 GB is size, we are working on hosting the database in the cloud. The website link will be ready by final submission.



Blockchain Explorer

Search Blockchain

Search by: Blocks Number

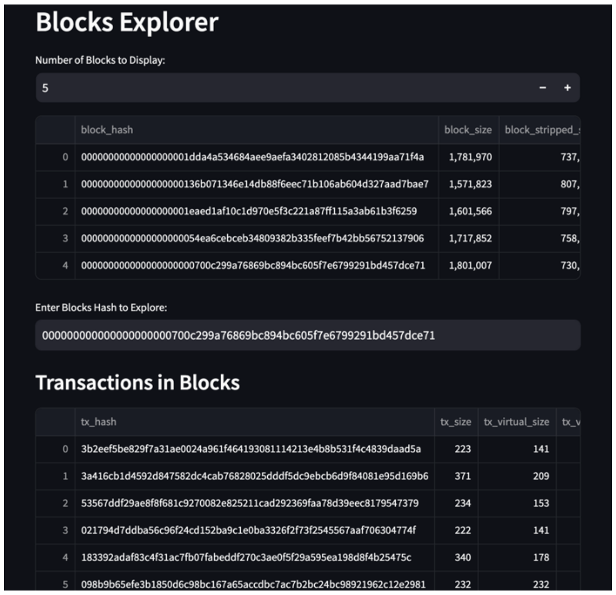
Enter Blocks Number: 871664

Blocks Details

		block_size	block_stripped_size	block_weight	block_number	block_vers
0	c605f7e6799291bd457dce71	1,801,007	730,802	3,993,413	871,664	905,969,

C. Block Explorer

This section includes interactive visualizations of blockchain data. For example, the "Transaction Volume Over Time" line chart illustrates daily transaction volumes, highlighting trends and spikes in activity over a given period.



Blocks Explorer

Number of Blocks to Display: 5

	block_hash	block_size	block_stripped_
0	00000000000000000001dda4a534684aee9aefa3402812085b4344199aa71f4a	1,781,970	737,
1	0000000000000000000136b071346e14db88f6eec71b106ab604d327aad7bae7	1,571,823	807,
2	00000000000000000001eaed1af10c1d970e5f3c221a87ff115a3ab61b3f6259	1,601,566	797,
3	0000000000000000000054ea6cebceb34809382b335feef7b42bb56752137906	1,717,852	758,
4	000000000000000000000700c299a76869bc894bc605f7e6799291bd457dce71	1,801,007	730,

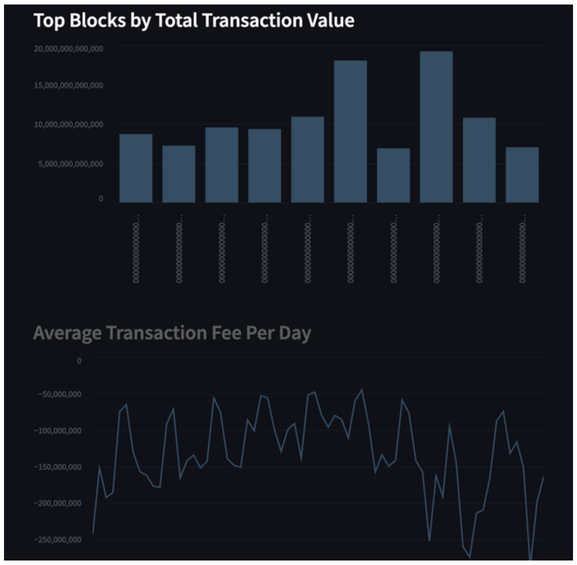
Enter Blocks Hash to Explore: 000000000000000000000700c299a76869bc894bc605f7e6799291bd457dce71

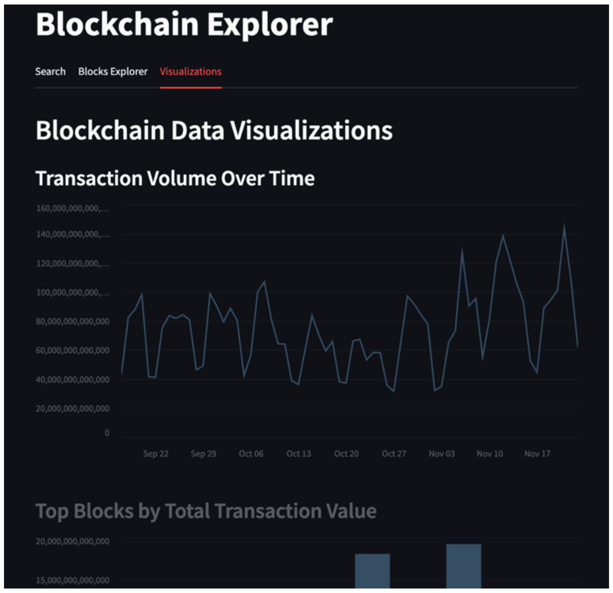
Transactions in Blocks

	tx_hash	tx_size	tx_virtual_size	tx_v
0	3b2eef5be829f7a31ae0024a961f464193081114213e4b8b531f4c4839daad5a	223	141	
1	3a416cb1d4592d847582dc4cab76828025dddf5dc9ebcb6d9f84081e95d169b6	371	209	
2	53567ddf29ae8f8f681c9270082e825211cad292369faa78d39eec8179547379	234	153	
3	021794d7ddba56c96f24cd152ba9c1e0ba3326f2f73f2545567aaf706304774f	222	141	
4	183392adaf83c4f31ac7fb07fabeddf270c3ae0f5f29a595ea198d8f4b25475c	340	178	
5	098b9b65efe3b1850d6c98bc167a65accdbc7ac7b2bc24bc98921962c12e2981	232	232	

D. Blockchain Data Visualizations

This section includes interactive visualizations of blockchain data. For example, the "Transaction Volume Over Time" line chart illustrates daily transaction volumes, highlighting trends and spikes in activity over a given period. This bar chart ranks blocks by their total transaction output value, allowing users to identify the most valuable blocks in the blockchain. This line chart visualizes the average transaction fee per day, providing insights into the cost dynamics of blockchain transactions over time.




Blockchain Explorer
Search
Blocks Explorer
Visualizations
Blockchain Data Visualizations
Transaction Volume Over Time
Top Blocks by Total Transaction Value

Implementation Details

- **Framework Used**: The website is built using using streamlit
- **Database Connection**: The database connection is created with sql alchemy module.
- **Visualization Tools**: Interactive charts are presented with the help of pandas and visualization functions from streamlit framework.

XII. PROJECT CONTRIBUTIONS

Work/Student	**Rohan Shrikant Thorat**	**Gouri Ramdas Menon**
Data Set Collection	Planned and Collected Data	
DB Design	Designed and Populated the Database	
Queries		Created queries for various SQL operations
Documentation and Project Planning		Documented and Planned project along with finding use cases

REFERENCES

[1] Get Block.io https://getblock.io/nodes/btc/
[2] Bitcoin-etl https://github.com/blockchain-etl/bitcoin-etl
[3] Blockstream explorer https://blockstream.info/
[4] https://medium.com/google-cloud/the-fastest-way-to-learn-sql-with-bitcoin-data-on-a-live-database-from-google-part-2-c00ff1067343
[5] Claude LLM for generating descriptions of attributes in Tables.