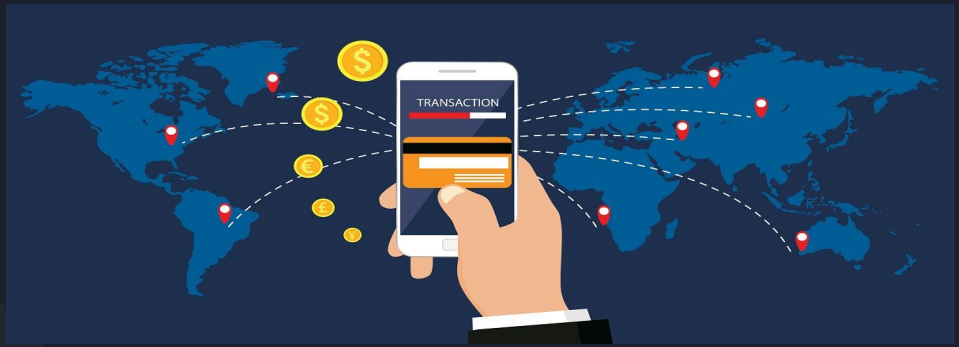
BLOCKCHAIN



Using Blockchain as Database Solution

Joseph R. Tursi

Table of Contents

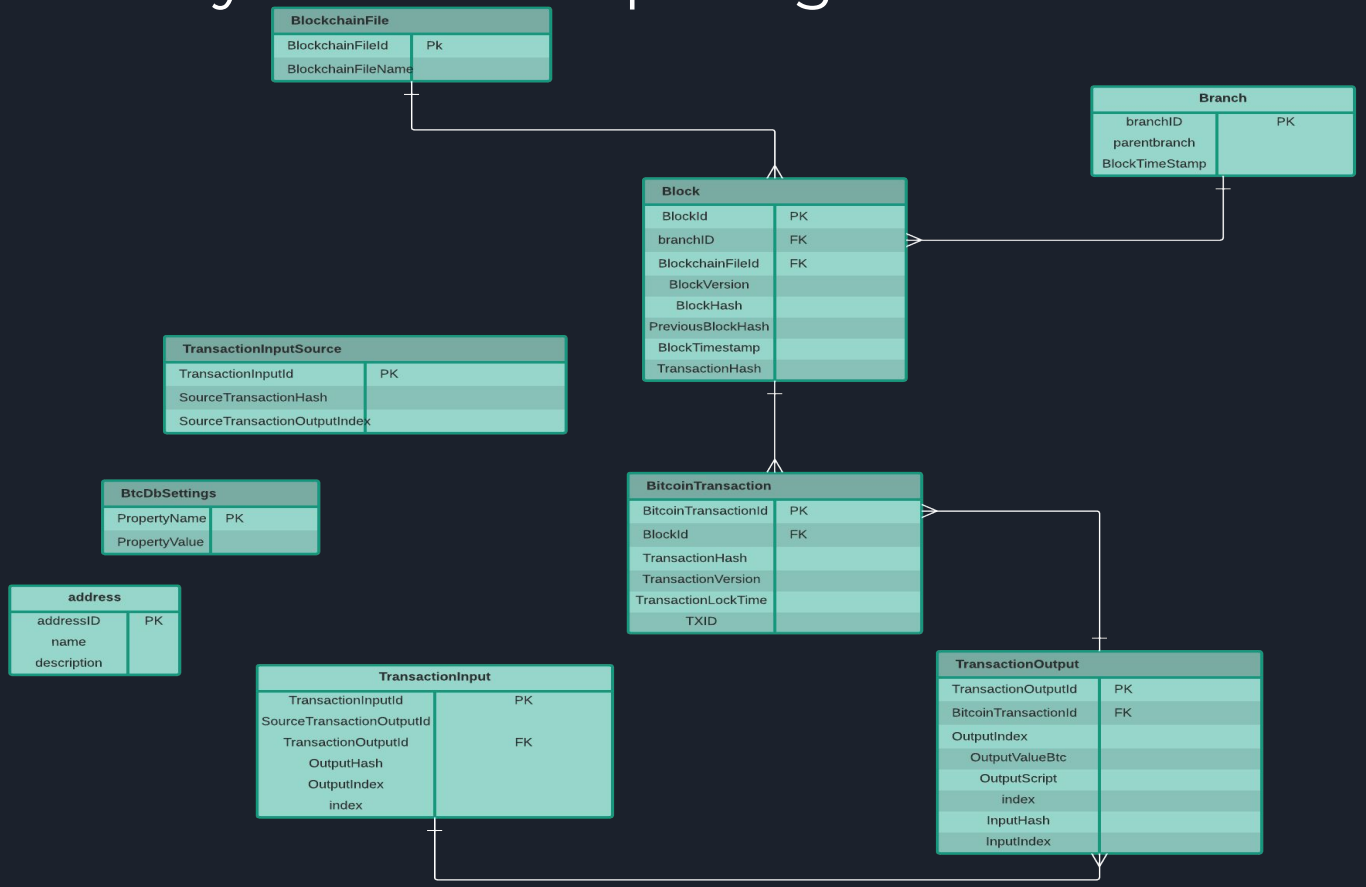
Executive Summary

By design, the blockchain concept is a decentralized technology. Blockchain stores data across its network, so anything that happens on it is a function of the network as a whole. Ever since blockchain burst onto the scene it's been a technology surrounded by hype, with the practical uses of cryptocurrencies.

This document outlines the design of a database which holds data on the cryptocurrency, bitcoin. The design focuses on the transaction of bitcoin, which could be altered to be any product or service. An overview of the database will be presented with explanation of each table and their purposes. The views, reports, triggers, and stored procedures will be implemented, explained, and contain sample data.

This implementation is intended to be used as a test case to demonstrate that blockchain could be easily integrated into your product or service. The blockchain could potentially get rid of the middleman for various transactions . This project is just scratching the surface of the concept of blockchain, which will change the way business is conducted in the future. The design does need improvements and will be built upon.

Entity Relationship Diagram



BlockchainFile Table

Implement a table to populate cryptocurrency data

```
CREATE TABLE BlockchainFile (
  BlockchainFileId          INT PRIMARY KEY      NOT NULL,
  BlockchainFileName          VARCHAR (300)      NOT NULL
);
```

Functional Dependencies

BlockchainFileId>BlockchainFileName

blockchainfileid integer	**blockchainfilename** character varying (300)
2	AlanLabouseur

Branch Table

```
create table Branch(
branchID            int,
parentbranch        int,
BlockTimeStamp  date,
primary key(branchID)
);
```

Functional Dependencies

branchID>parentbranch, BlockTimeStamp

branchid integer	**parentbranch** integer	**blocktimestamp** date
12	11	2017-01-01

Block Table

Contains information about the Bitcoin blocks.

```
CREATE TABLE Block (
   BlockId                 BIGINT PRIMARY KEY   NOT NULL,
   branchID                INT,
   BlockchainFileId        INT                  NOT NULL,
   BitcoinTransactionId    BIGINT               NOT NULL,
   BlockVersion            INT                  NOT NULL,
   BlockHash               VARCHAR (200)        NOT NULL,
   PreviousBlockHash       VARCHAR (200)        NOT NULL,
   BlockTimestamp          DATE                 NOT NULL,
   TransactionHash         VARCHAR (32)         NOT NULL
);
```

Functional Dependencies

BlockId> branchID, BlockchainFileId, BitcoinTransactionId, BlockVersion, BlockHash, PreviousBlockHash, BlockTimestamp,TransactionHash

blockid bigint	**branchid** integer	**blockchainfileid** integer	**bitcointransactionid** bigint	**blockversion** integer	**blockhash** character varying (200)	**previousblockhash** character varying (200)	**blocktimestamp** date	**transactionhash** character varying (32)
100	1	1	25	1	ab	aabc	2017-07-19	cb
101	1	2	26	1	abc	abcd	2017-12-05	ca
102	1	3	27	1	abd	abde	2017-02-22	cd
103	1	4	28	1	abe	aabef	2017-05-12	ce
104	1	5	29	1	abf	abg	2017-07-19	cf

BitcoinTransaction Table

Contains information about the Bitcoin transactions.

CREATE TABLE BitcoinTransaction (

CREATE TABLE BitcoinTransaction (

```
  BitcoinTransactionId     BIGINT PRIMARY KEY     NOT NULL,
  BlockId                  BIGINT                 NOT NULL,
  TXID                     BIGINT                 NOT NULL,
  TransactionHash          VARCHAR (32)           NOT NULL,
  TransactionVersion       INT                    NOT NULL,
  TransactionLockTime      INT                    NOT NULL
);
```

Functional Dependencies

BitcoinTransactionId> BlockId, TXID, TransactionHash, TransactionVersion, TransactionLockTime

bitcointransactionid bigint	**blockid** bigint	**txid** bigint	**transactionhash** character varying (32)	**transactionversion** integer	**transactionlocktime** integer
25	100	50	a	1	419382
26	101	51	ab	1	419382
27	102	52	abc	1	419382
28	103	53	abcd	1	419382
29	104	54	abcde	1	419382

TransactionInput Table

Contains information about the Bitcoin transaction inputs.

```
CREATE TABLE TransactionInput (
  TransactionInputId          BIGINT PRIMARY KEY     NOT NULL,
SourceTransactionOutputId    BIGINT                          NULL,
  TransactionOutputId         BIGINT                         NULL,
  OutputHash                  TEXT                      NOT NULL,
  OutputIndex                  INT                      NOT NULL,
  index                        INT                     NOT NULL
);
```

Functional Dependencies

TransactionInputId> SourceTransactionOutputId, TransactionOutputId, OutputHash, OutputIndex, index

transactioninputid bigint	sourcetransactionoutputid bigint	transactionoutputid bigint	outputhash text	outputindex integer	index integer	bitcointransactionid bigint
335	25	100	zzzf	0	0	25
336	26	101	abcd	0	0	26
337	27	102	dcba	0	0	27
338	28	103	badd	0	0	28
339	29	104	badc	0	0	29

TransactionInputSource Table

Contains information about the source of the Bitcoin transaction inputs. This table contains links between transaction inputs and their corresponding source outputs. a stage where data from this table is processed and a more direct link is calculated and saved in TransactionInput.SourceTransactionOutputId.

```
CREATE TABLE TransactionInputSource (
  TransactionInputId          BIGINT PRIMARY KEY          NOT NULL,
  SourceTransactionHash       VARCHAR (32)                NOT NULL,
  SourceTransactionOutputIndex   INT                          NULL
);
```

Functional Dependencies

TransactionInputId> SourceTr

transactioninputid bigint	sourcetransactionhash character varying (32)	sourcetransactionoutputindex integer
335	a	400
336	a	401
337	a	402
338	a	403
339	a	404

TransactionOutput Table

Contains information about the Bitcoin transaction outputs.

```
CREATE TABLE TransactionOutput (
   TransactionOutputId        BIGINT PRIMARY KEY        NOT NULL,
   BitcoinTransactionId       BIGINT                    NOT NULL,
   OutputIndex                INT                       NOT NULL,
   OutputValueBtc             NUMERIC(20,8)             NOT NULL,
   OutputScript               VARCHAR (300)             NOT NULL,
   index                      INT                       NOT NULL,
   InputHash                  VARCHAR(64),
   InputIndex                 INT
);
```

Functional Dependencies

TransactionOutputId>BitcoinTransactionId, OutputIndex, OutputValueBtc, OutputScript, index, InputHash, InputIndex

transactionoutputid bigint	bitcointransactionid bigint	outputindex integer	outputvaluebtc numeric (20,8)	outputscript character varying (300)	index integer	inputhash character varying (64)	inputindex integer
7	25	0	5.60000000	Complete	0	a	0
8	26	0	14.20000000	Complete	0	a	0
9	27	0	201.30000000	Complete	0	a	0
10	28	0	32.00000000	Complete	0	a	0
11	29	0	8.20000000	Complete	0	a	0

BtcDbSettings Table

Contains information cocnerning Key-value pairs containing system data.

```
CREATE TABLE BtcDbSettings (
   PropertyName          VARCHAR (32) PRIMARY KEY     NOT NULL,
   PropertyValue         VARCHAR (500)                NOT NULL
);
```

Functional Dependencies

PropertyName> PropertyValue

propertyname character varying (32)	**propertyvalue** character varying (500)
ALAN	Backup

address Table

Contains information about the Private/Public Keys aka. address

```
create table address(
addressID     bigint,
name          varchar(255),
description   varchar(255),
primary key(addressID)
);
```

Functional Dependencies

addressID>name, description,

addressid bigint	**name** character varying (255)	**description** character varying (255)
12	Depression	So much time invested,...

Stored Procedure

link_txs() is intended for every transaction input to link and existing output to it.

```
CREATE OR REPLACE FUNCTION link_txs() RETURNS TRIGGER AS $$
BEGIN
  UPDATE TransactionOutput
  SET InputHash = (SELECT TransactionHash FROM BitcoinTransaction WHERE BlockId =
NEW.BitcoinTransactionId
    ,InputIndex = NEW.index
  WHERE BitcoinTransactionId = (SELECT BlockId FROM BitcoinTransaction WHERE BlockHash =
NEW.OutputHash)
    AND index = NEW.OutputIndex;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Store Procedures

update_Blochain_status() is intended to update BlockchainFileId

```
CREATE OR REPLACE FUNCTION update_Blockchain_status()
RETURNS TRIGGER AS
$$
BEGIN
IF NEW.BlockchainFileId  is NOT NULL THEN
UPDATE BlockchainFile
SET available = FALSE
WHERE NEW.BlockchainFileId  = BlockchainFile.BlockchainFileId;
END IF;
RETURN NEW;
END;
$$
LANGUAGE PLPGSQL;
```

Triggers - t_input_linkoutput is intended to update existing Transactions

```
CREATE TRIGGER t_input_linkoutput
BEFORE INSERT ON TransactionInput
  FOR EACH ROW
EXECUTE PROCEDURE link_txs();
```

Triggers

update_Blockchain_status_trigger intended to update the blockchain file.

```
CREATE TRIGGER update_Blockchain_status_trigger

BEFORE INSERT ON BlockchainFile

FOR EACH ROW

EXECUTE PROCEDURE update_Blockchain_status();
```

TransactionOutput

```
UPDATE TransactionOutput
SET InputHash = data.InputHash,
 InputIndex = data.InputIndex
FROM (SELECT TransactionHash AS InputHash, index AS InputIndex, OutputIndex,
OutputHash
    FROM TransactionInput LEFT JOIN BitcoinTransaction
    ON TransactionInput.BitcoinTransactionId = BitcoinTransaction.BlockId) data
WHERE TransactionOutput.BitcoinTransactionId = (SELECT BlockId FROM BitcoinTransaction
WHERE TransactionHash = data.OutputHash)
AND TransactionOutput.index = data.OutputIndex;
```

Views TransactionAggregated

use this view to retrieve aggregated data for a transaction including total input, output and transaction fees.

```
CREATE VIEW View_TransactionAggregated AS
SELECT
  BitcoinTransactionId,
  BlockId,
  TransactionHash,
  TransactionVersion,
  TransactionLockTime,
  TransactionInputCount,
  TotalInputBtc,
  TransactionOutputCount,
  TotalOutputBtc,
  TotalUnspentOutputBtc
FROM (
  SELECT
    BitcoinTransaction.BitcoinTransactionId,
    BitcoinTransaction.BlockId,
    BitcoinTransaction.TransactionHash,
    BitcoinTransaction.TransactionVersion,
    BitcoinTransaction.TransactionLockTime,
    (  SELECT COUNT(1)
      FROM TransactionInput
      WHERE BitcoinTransaction.BitcoinTransactionId = TransactionInput.BitcoinTransactionId
    ) AS TransactionInputCount,
    (  SELECT SUM(TransactionOutput.OutputValueBtc)
      FROM TransactionInput
      INNER JOIN TransactionOutput ON TransactionOutput.TransactionOutputId = TransactionInput.SourceTransactionOutputId
      WHERE TransactionInput.BitcoinTransactionId = BitcoinTransaction.BitcoinTransactionId
    ) AS TotalInputBtc,
    (  SELECT COUNT(1)
      FROM TransactionOutput
      WHERE BitcoinTransaction.BitcoinTransactionId = TransactionOutput.BitcoinTransactionId
    ) AS TransactionOutputCount,
    (  SELECT SUM(TransactionOutput.OutputValueBtc)
      FROM TransactionOutput
      WHERE TransactionOutput.BitcoinTransactionId = BitcoinTransaction.BitcoinTransactionId
    ) AS TotalOutputBtc,
    (  SELECT SUM(TransactionOutput.OutputValueBtc)
      FROM TransactionOutput
      LEFT OUTER JOIN TransactionInput ON TransactionInput.SourceTransactionOutputId = TransactionOutput.TransactionOutputId
      WHERE
        TransactionOutput.BitcoinTransactionId = BitcoinTransaction.BitcoinTransactionId
        AND TransactionInput.TransactionInputId IS NULL
    ) AS TotalUnspentOutputBtc
  FROM BitcoinTransaction) AS TransactionAggregated
```

bitcointransactionid bigint	blockid bigint	transactionhash character varying (32)	transactionversion integer	transactionlocktime integer	transactioninputcount bigint	transactionoutputcount bigint	totaloutputbtc numeric
25	100	a	1	419382	1	1	5.60000000

Sample Query
SELECT * from view_transactionAggregated WHERE BitcoinTransactionId= 25

totalunspentoutputbtc numeric
5.60000000

Views BlockAggregated

Use this view retrieve aggregated data for a block including the total input, output and transaction fees.

```
DROP VIEW IF EXISTS View_BlockAggregated;
CREATE VIEW View_BlockAggregated AS
SELECT
  Block.BlockId,
  Block.BlockchainFileId,
  Block.BlockVersion,
  Block.BlockHash,
  Block.PreviousBlockHash,
  Block.BlockTimestamp,
  BlockAggregated.TransactionCount,
  BlockAggregated.TransactionInputCount,
  BlockAggregated.TotalInputBtc,
  BlockAggregated.TransactionOutputCount,
  BlockAggregated.TotalOutputBtc,
 -- BlockAggregated.TransactionFeeBtc, This is = (TotalInputBtc - TotalOutputBtc)
  BlockAggregated.TotalUnspentOutputBtc
FROM Block
INNER JOIN (
  SELECT
    Block.BlockId,
    SUM(1) AS TransactionCount,
    SUM(TransactionInputCount) AS TransactionInputCount,
    SUM(TotalInputBtc) AS TotalInputBtc,
    SUM(TransactionOutputCount) AS TransactionOutputCount,
    SUM(TotalOutputBtc) AS TotalOutputBtc,
    --SUM(TransactionFeeBtc) AS TransactionFeeBtc,
    SUM(TotalUnspentOutputBtc) AS TotalUnspentOutputBtc
  FROM Block
  INNER JOIN View_TransactionAggregated ON Block.BlockId = View_TransactionAggregated.BlockId
  GROUP BY Block.BlockId
  ) AS BlockAggregated ON BlockAggregated.BlockId = Block.BlockId
```

blockid bigint	blockchainfileid integer	blockversion integer	blockhash character varying (200)	previousblockhash character varying (200)	blocktimestamp date	transactioncount bigint	transactioninputcount numeric	tran num
100	1	1	ab	aabc	2017-07-19	1	1	

transactionoutputcount numeric	totaloutputbtc numeric	totalunspentoutputbtc numeric
1	5.60000000	5.60000000

Sample Query: SELECT * FROM View_BlockAggregated WHERE BlockId = 100

Views View_BlockchainFile Counts

Use this view retrieve data about a blockchain file.

```
DROP VIEW IF EXISTS View_BlockchainFileCounts;
CREATE VIEW View_BlockchainFileCounts AS
SELECT
  BlockchainFile.BlockchainFileId,
  BlockchainFileName,
  ( SELECT COUNT(1) FROM Block WHERE Block.BlockchainFileId = BlockchainFile.BlockchainFileId ) AS BlockCount,
  ( SELECT COUNT(1)
    FROM BitcoinTransaction
    INNER JOIN Block ON Block.BlockId = BitcoinTransaction.BlockId
    WHERE Block.BlockchainFileId = BlockchainFile.BlockchainFileId
  ) AS TransactionCount,
  ( SELECT COUNT(1)
    FROM TransactionInput
    INNER JOIN BitcoinTransaction ON BitcoinTransaction.BitcoinTransactionId = TransactionInput.BitcoinTransactionId
    INNER JOIN Block ON Block.BlockId = BitcoinTransaction.BlockId
    WHERE Block.BlockchainFileId = BlockchainFile.BlockchainFileId
  ) AS TransactionInputCount,
  ( SELECT COUNT(1)
    FROM TransactionOutput
    INNER JOIN BitcoinTransaction ON BitcoinTransaction.BitcoinTransactionId = TransactionOutput.BitcoinTransactionId
    INNER JOIN Block ON Block.BlockId = BitcoinTransaction.BlockId
    WHERE Block.BlockchainFileId = BlockchainFile.BlockchainFileId
  ) AS TransactionOutputCount
FROM BlockchainFile
```

Sample Query

```
SELECT * FROM View_BlockchainFileCounts WHERE BlockchainFileId = 100
```

blockchainfileid integer	**blockchainfilename** character varying (300)	**blockcount** bigint	**transactioncount** bigint	**transactioninputcount** bigint	**transactionoutputcount** bigint
2	AlanLabouseur	1	1	1	1

Reports/ Interesting Queries

Query counts the number of blocks in the blockchain..

```
select count(*) from Block
```

	count bigint
1	5

Reports/ Interesting Queries

Selects the transaction with the highest value

SELECT * FROM transactionoutput

ORDER BY outputvalueBTC DESC

transactionoutputid bigint	**bitcointransactionid** bigint	**outputindex** integer	**outputvaluebtc** numeric (20,8)	**outputscript** character varying (300)	**index** integer	**inputhash** character varying (64)	**inputindex** integer
9	27	0	201.30000000	Complete	0	a	0
10	28	0	32.00000000	Complete	0	a	0
8	26	0	14.20000000	Complete	0	a	0
11	29	0	8.20000000	Complete	0	a	0
7	25	0	5.60000000	Complete	0	a	0

Security

```
CREATE ROLE ADMIN;
GRANT ALL ON ALL TABLES IN SCHEMA PUBLIC
TO ADMIN;

CREATE ROLE P_USER;
REVOKE ALL ON ALL TABLES IN SCHEMA PUBLIC
FROM P_USER;
GRANT SELECT ON BlockchainFile, Block, BitcoinTransaction,
TransactionInput, TransactionInputSource, TransactionOutput
```

Implementation Notes – Known Problems – Future Enhancements

Implementation Notes

- With a larger data sample this database would render inefficient not having referential data integrity
- Due to short timeline there is a lack of complex queries(No complex reports) and stored procedures which would've explored this blockchain concept further

Known Problems

- Maintaining referential data integrity
- Not apart of the blockchain network
- Create more stored procedures for a more efficient database
- Moving from a decentralized structured system to relational storage

Future Enhancements

- More Triggers, Stored Procedures
- Implement Queries to determine the time between successive blocks for a given time period
- Implement Queries to determine the distribution of UTXOs, or Unspent Transaction Output
- Creating more schemas concerning unlinked transactions, orphan blocks, and schemas connected the data chain.
- Create a client parallel, updating SQL database to be updated with the new blockdata