Advance Relational Databases

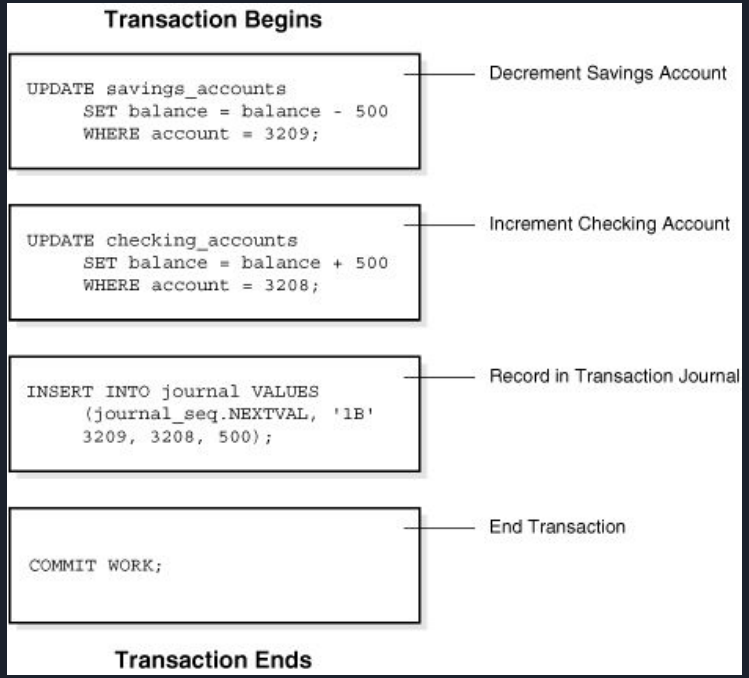
Musab & Maiia

Agenda

1. Transactions
2. ACID Properties
3. Isolation Levels
4. CAP Theorem
5. Indexing
6. Replication
7. Partitioning

1.1: Database Transactions

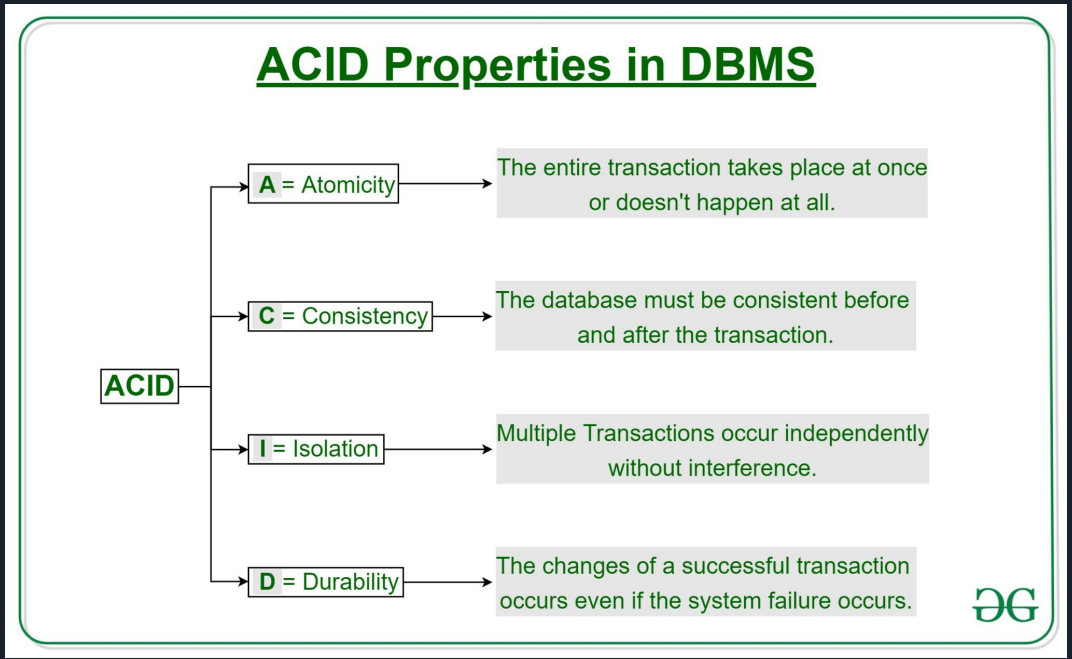
- Collection of Queries
- One Unit of Work
- Begin Transaction …….(queries)....Commit Transaction
- Reasons for Rollbacks: Explicit Rollback, Crashes



1.2: ACID Properties

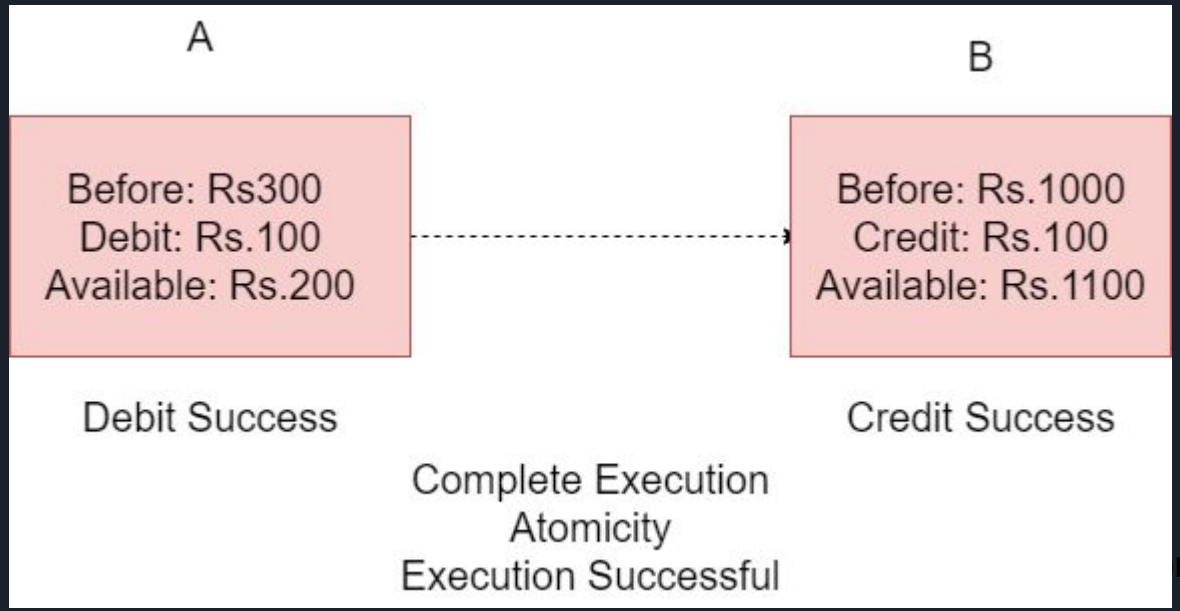
ACID consists of 4 properties that transactions should follow in order to maintain Database Integrity.

- Atomicity
- Consistency
- Isolation
- Durability



Atomicity

- States that a Transaction must complete or fail.
- If at least one query fails or the database crashes, a rollback is triggered.



mic-transaction

Consistency

- Consistency is defined by the user (referential integrity and constraints)
- When changing/modifying, the database must be the same before and after the commit.
- So after a commit, a new transaction should be able to see the change.

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X − 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

Durability

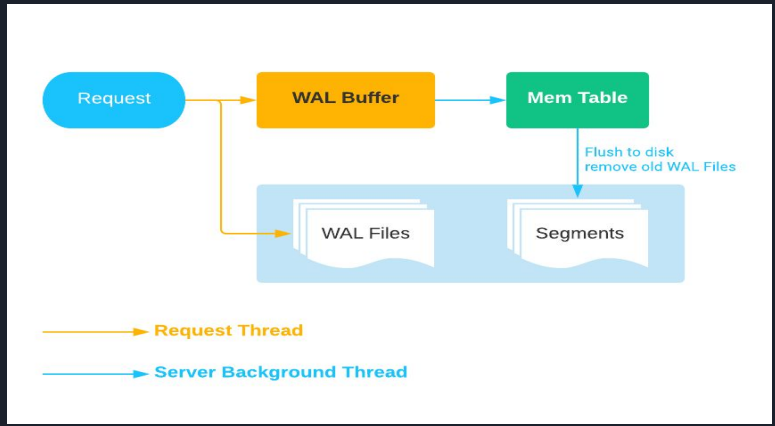
- Ensures committed transactions are saved permanently and do not accidentally disappear or get erased, even during a database crash.
- Achieved by saving all transactions to a non-volatile storage medium
- Ex: Write Ahead Logging(WAL)

Write Ahead Log (WAL)

- Database original content preserved in DB file, while changes are appended into WAL file.
- Commits occur without actually affecting the database
- When Checkpoints occur, the WAL file transactions are moved back into the database
- In SQLite, after a WAL file reaches a 1000 pages a checkpoint is applied.

Read more:

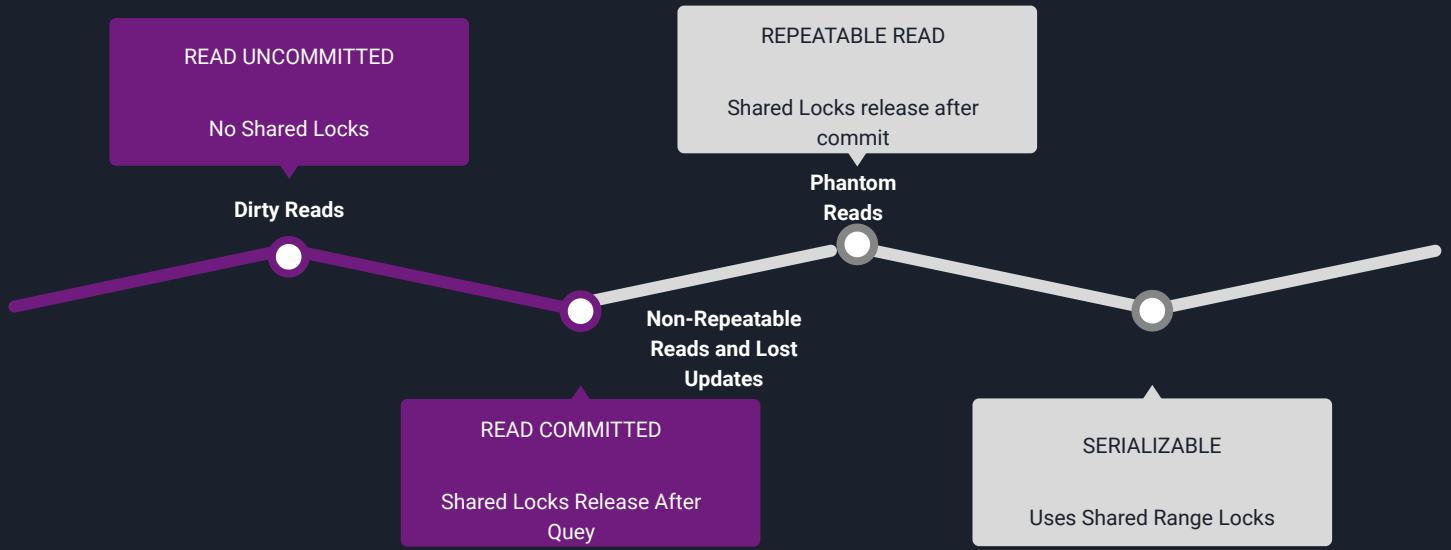
https://sqlite.org/wal.html



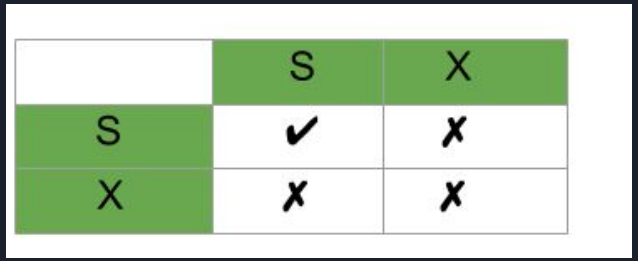
1.3: Isolation Levels

Isolation level \ Read phenomena	Dirty reads	Lost updates[inconsistent]	Non-repeatable reads	Phantoms
Serializable	-	-	-	-
Repeatable Read	-	-	-	+
Read Committed	-	+	+	+
Read Uncommitted	+	+	+	+

- States that multiple transactions can occur without interfering with each other



What are Shared Locks and Exclusive Locks?



	S	X
S	✔	✗
X	✗	✗

Shared Locks:

- Are used when reading data
- Ensure that a record is not in process of being updated during a read-only request
- Prevent any kind of updates of record

Exclusive Locks:

- Also called write lock
- exclusive lock prevents any other locker from obtaining any sort of a lock on the object
- Can be owned by only one transaction at a time

READ UNCOMMITTED

- Changes are visible before transactions Commit
- Doesn’t use shared locks when reading data(SELECT)
- Problems: Dirty Reads,Non-Repeatable Reads,Lost Updates, Phantom Reads

Dirty Reads

TRANSACTION_READ_UNCOMMITTED

SALES

PID	QNT	PRICE
Product 1	15	$5
Product 2	20	$4

BEGIN TX1

```
SELECT PID, QNT*PRICE FROM SALES
```

Product 1, 50
Product 2, 80

```
SELECT SUM(QNT*PRICE) FROM SALES
```

We get $155 when it should be $130
We read a “dirty” value that has not been committed

BEGIN TX2

```
UPDATE SALES SET QNT = QNT+5
WHERE PID =1
```

READ COMMITTED

- Only changes from committed transactions are visible to other transactions
- Shared Locks are applied but release after SELECT query is executed
- Problems Fixed: Dirty Read
- Problems: Lost Updates, Non-Repeatable Read, Phantom Read

Non-repeatable read

TRANSACTION_READ_COMMITTED

SALES

PID	QNT	PRICE
Product 1	15	$5
Product 2	20	$4

BEGIN TX1

SELECT PID, QNT*PRICE **FROM** SALES

Product 1, 50
Product 2, 80

BEGIN TX2

UPDATE SALES **SET** QNT = QNT+5
WHERE PID =1

COMMIT TX2

SELECT SUM(QNT*PRICE) **FROM** SALES

We get $155 when it should be $130
We did read a committed value, but it gave us inconsistent results

COMMIT TX1

Lost Updates

```
--Initial price is 500

--Transaction #1 (starts first)
BEGIN TRANSACTION;
DECLARE @Price DECIMAL(18,0);
SELECT @Price = Price FROM dbo.Orders WHERE Id = 1 --Reads 500
WAITFOR DELAY '00:00:05'
UPDATE dbo.Orders SET Price = @Price + 1000 WHERE Id = 1 --Overwrites 2500 with 1500
COMMIT;

--Transaction #2 (starts immediatelly after the start of transaction #1)
BEGIN TRANSACTION;
DECLARE @Price DECIMAL(18,0);
SELECT @Price = Price FROM dbo.Orders WHERE Id = 1 -- Reads 500
UPDATE dbo.Orders SET Price = @Price + 2000 WHERE Id = 1 --Updates to 2500
COMMIT;

--The final price is 1500, not 3500 as users expect.
```

REPEATABLE READ

- Shared Locks release after Commits, so therefore rows being read won't be affected until the end of the transaction
- Problems Fixed: Dirty Read,Non-Repeatable Reads, Lost Updates
- Problems: Phantom Reads

Phantom read

TRANSACTION_REPEATABLE_READ

SALES

PID	QNT	PRICE
Product 1	10	$5
Product 2	20	$4
Product 3	10	$1

BEGIN TX1

SELECT PID, QNT*PRICE **FROM** SALES

Product 1, 50
Product 2, 80

BEGIN TX2

INSERT INTO SALES
VALUES ('Product 3', 10, 1)

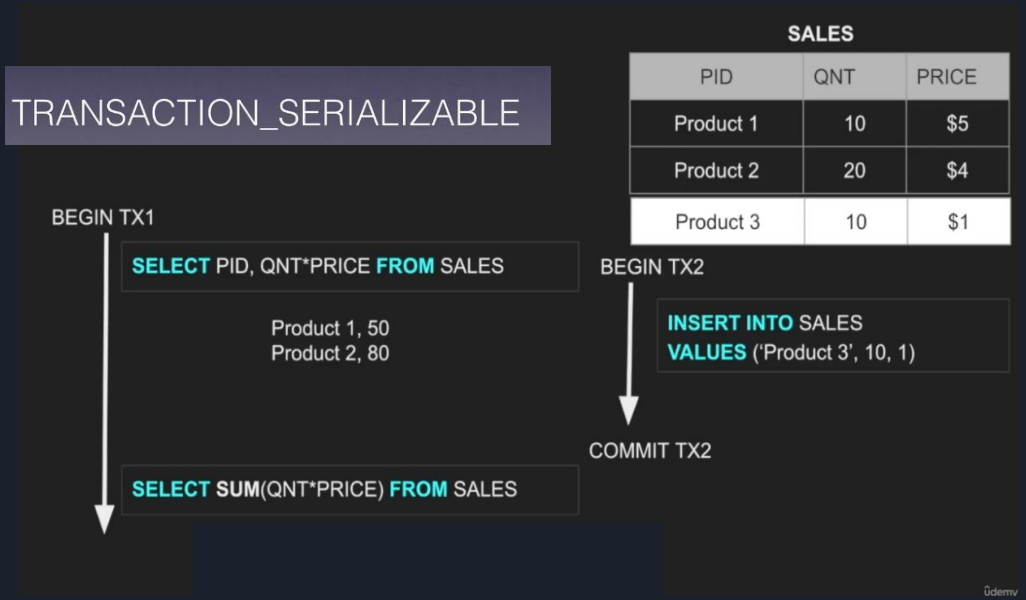
COMMIT TX2

SELECT SUM(QNT*PRICE) **FROM** SALES

We get $140 when it should be $130
We read a committed value that showed up in our range query

Serializable

- Shared Lock now account for the range of data they are reading, so they prevent modifications(Exclusive Locks) that affect the data in the range they are working with.



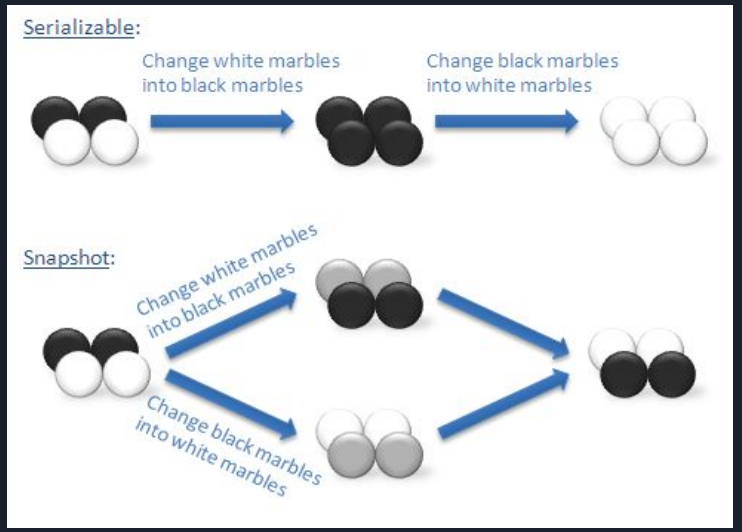
Snapshot Isolation

- Snapshot Isolation is different than other Isolation Levels since it uses Row Versioning not Locking.
- When data is being changed, committed data is copied to tempDB and are given version numbers.
- So when another transaction reads data there is no wait because of locking, since it will receive the version of data from the most recent committed transaction.

Serializable(Locking) vs Snapshot(Row-Versioning)

Serializable	
Pros	Cons
Guarantees Database Integrity	Decrease in Concurrency due to locking

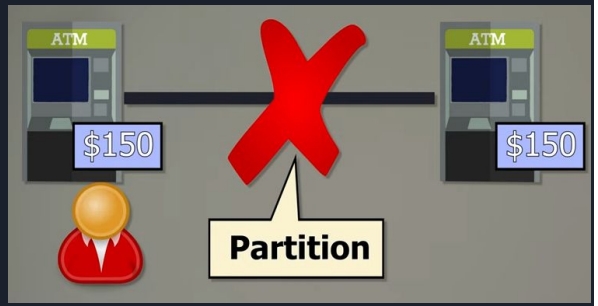
Snapshot Isolation	
Pros	Cons
Increased Concurrency since it uses Row Versioning Not Locking	Increased TempDb Usage for storing Row versions causes overhead

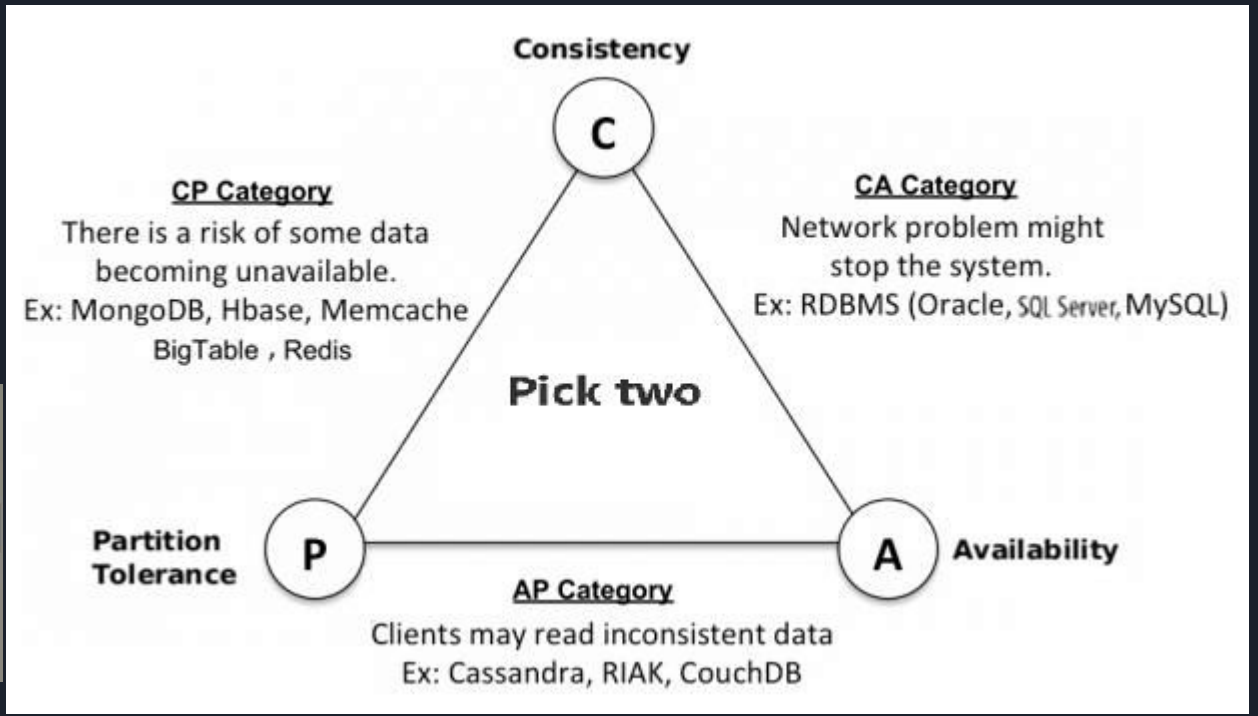


1.4: CAP Theorem

-Formalizes the trade off between consistency and availability in the presence of partitions

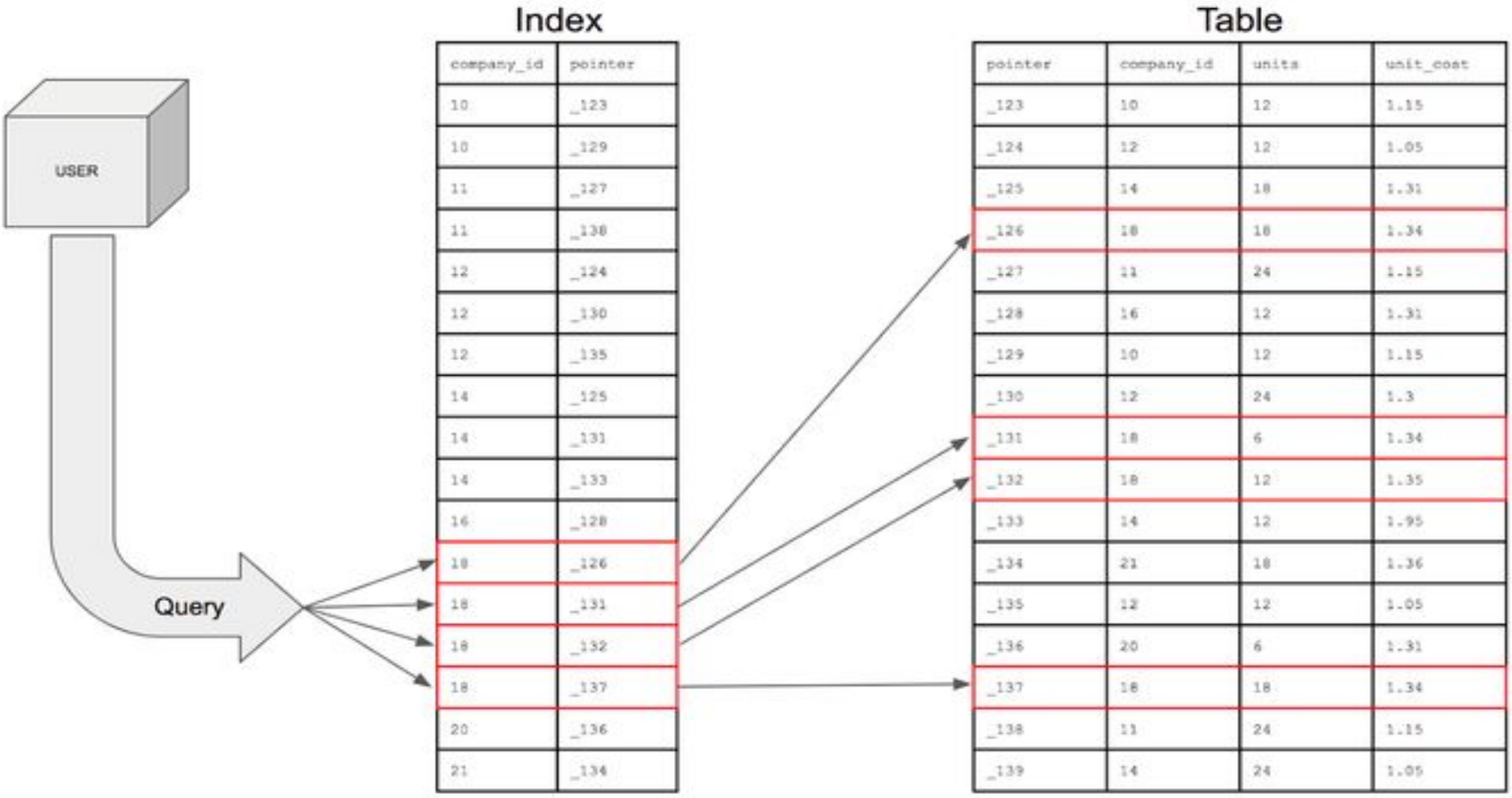
Partially Available Design
- **Deposits: yes**
- **Withdrawls: no**
- **Balance info: no**





1.5: Indexing

- We use indexing to increase the speed at which queries are executed when dealing with large databases
- When we create an Index on a column, we create a separate data structure that orders the data in way that helps search query efficiency
- Indexing Behind the Scenes
- Clustered Index vs Non Clustered Index
- Different types of Index Scans
- Index Fragmentation and How to fix it?


Index
company_id
pointer
10 _123
10 _129
11 _127
11 _138
12 _124
12 _130
12 _135
14 _125
14 _131
14 _133
16 _128
18 _126
18 _131
18 _132
18 _137
20 _136
21 _134
Table
pointer
company_id
units
unit_cost
_123 10 12 1.15
_124 12 12 1.05
_125 14 18 1.31
_126 18 18 1.34
_127 11 24 1.15
_128 16 12 1.31
_129 10 12 1.15
_130 12 24 1.3
_131 18 6 1.34
_132 18 12 1.35
_133 14 12 1.95
_134 21 18 1.36
_135 12 12 1.05
_136 20 6 1.31
_137 18 18 1.34
_138 11 24 1.15
_139 14 24 1.05
USER
Query

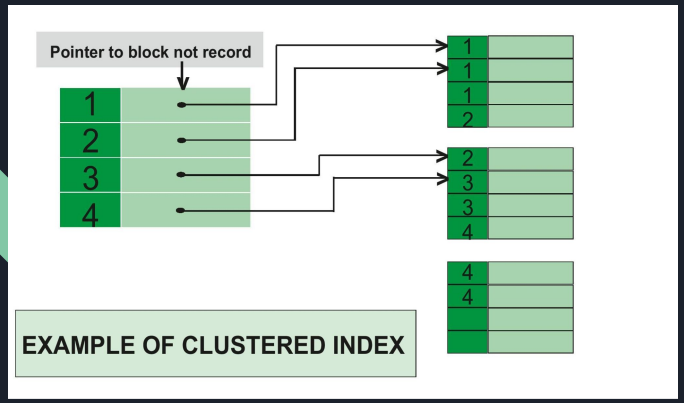
Clustered Index vs Non-Clustered Index

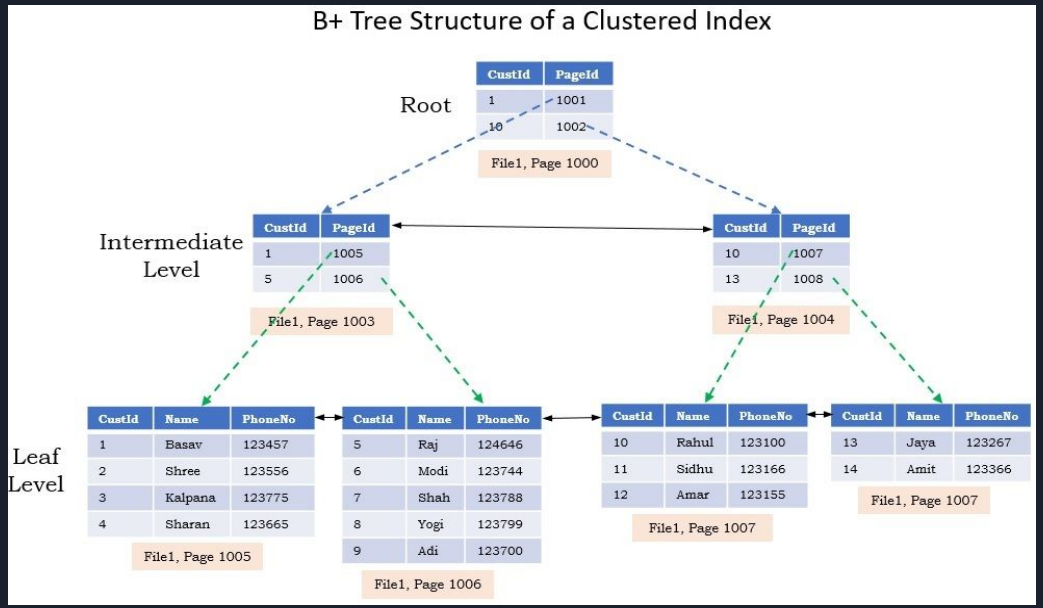
Clustered Index:

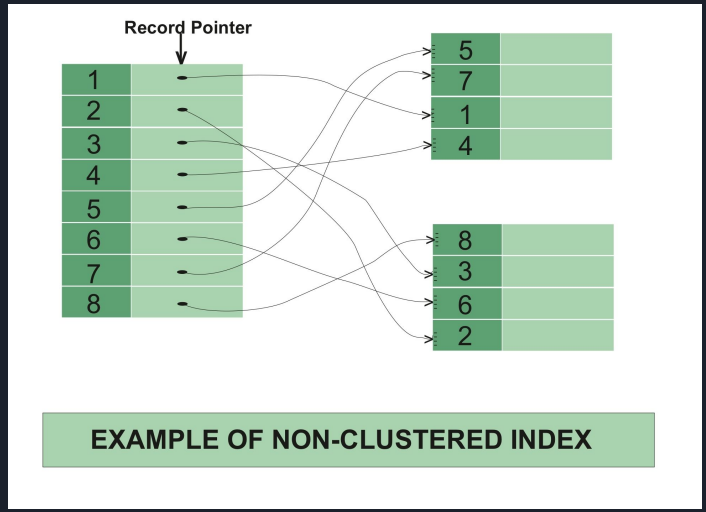
- An index made off the primary key of the table that physically reorders the table to match index.
- Leaf Nodes contain the row data itself

Non-Clustered Index:

- An index that has a separate data structure in a different order than the table with reference points to the rows in the table
- Leaf Nodes contain index value and pointer to row


Pointer to block not record
1
2
3
4
1
1
1
2
2
3
3
4
4
4
EXAMPLE OF CLUSTERED INDEX


B+ Tree Structure of a Clustered Index
Root
CustId
PageId
1
1001
10
1002
File1, Page 1000
Intermediate Level
CustId
PageId
1
1005
5
1006
File1, Page 1003
CustId
PageId
10
1007
13
1008
File1, Page 1004
Leaf Level
CustId
Name
PhoneNo
1
Basav
123457
2
Shree
123556
3
Kalpana
123775
4
Sharan
123665
File1, Page 1005
CustId
Name
PhoneNo
5
Raj
124646
6
Modi
123744
7
Shah
123788
8
Yogi
123799
9
Adi
123700
File1, Page 1006
CustId
Name
PhoneNo
10
Rahul
123100
11
Sidhu
123166
12
Amar
123155
File1, Page 1007
CustId
Name
PhoneNo
13
Jaya
123267
14
Amit
123366
File1, Page 1007


Record Pointer
1
2
3
4
5
6
7
8
5
7
1
4
8
3
6
2
EXAMPLE OF NON-CLUSTERED INDEX

Different Types of Index Scans

- Sequential Scan: Regular in order table scan, fast for fetching row from small table, or high amount of rows from large table.
- Index Scan: Scans index for pages that match query then uses reference point to fetch matching rows
- Index Only Scan: Scans index for pages that match query but data needed is already in index so no need to jump to table (faster than index)

SQL Copy

```sql
USE AdventureWorks2012;
GO
-- Creates a nonclustered index on the Person.Address table with four included (nonkey) columns.
-- index key column is PostalCode and the nonkey columns are
-- AddressLine1, AddressLine2, City, and StateProvinceID.
CREATE NONCLUSTERED INDEX IX_Address_PostalCode
ON Person.Address (PostalCode)
INCLUDE (AddressLine1, AddressLine2, City, StateProvinceID);
GO
```

Bitmap Heap and BitmapAND scans

Bitmap for heap pages

Fetch page 0 and page 1

Fetch page 1 and page 11

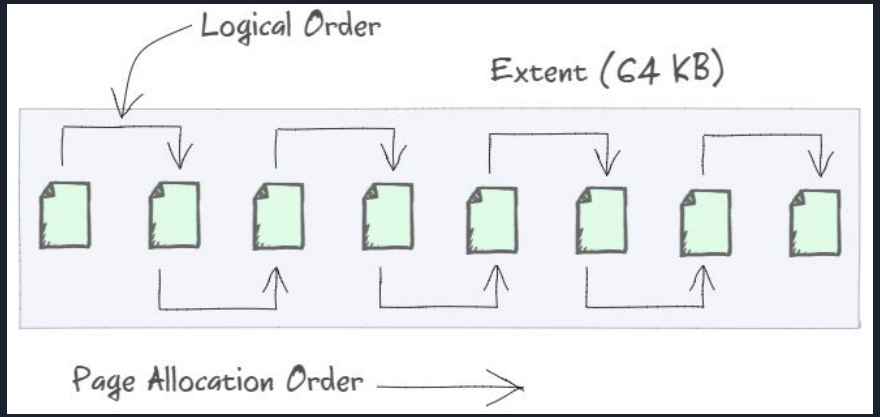
Bitmap Index A Scan (page 0, page 11)

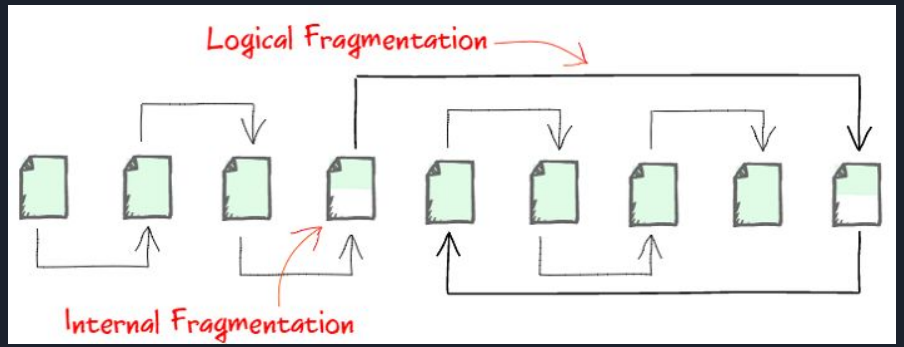
Bitmap Index B Scan (page 0 page 1)

BitmapAnd Index A & B (Page 0)

Index Fragmentation

- A fragmented index can occur from numerous data modification operations.
- More free space on pages because of page splits, so reading more pages to than we need to get the same data.
- Index size increased because of blank spaces





Internal Fragmentation:

- Free Space cause by Inserts or Deletes that make the index store more data and results in more IO operations to read.

Logical Fragmentation:

- Order of pages does not match physical ordering of pages making the searching not sequential

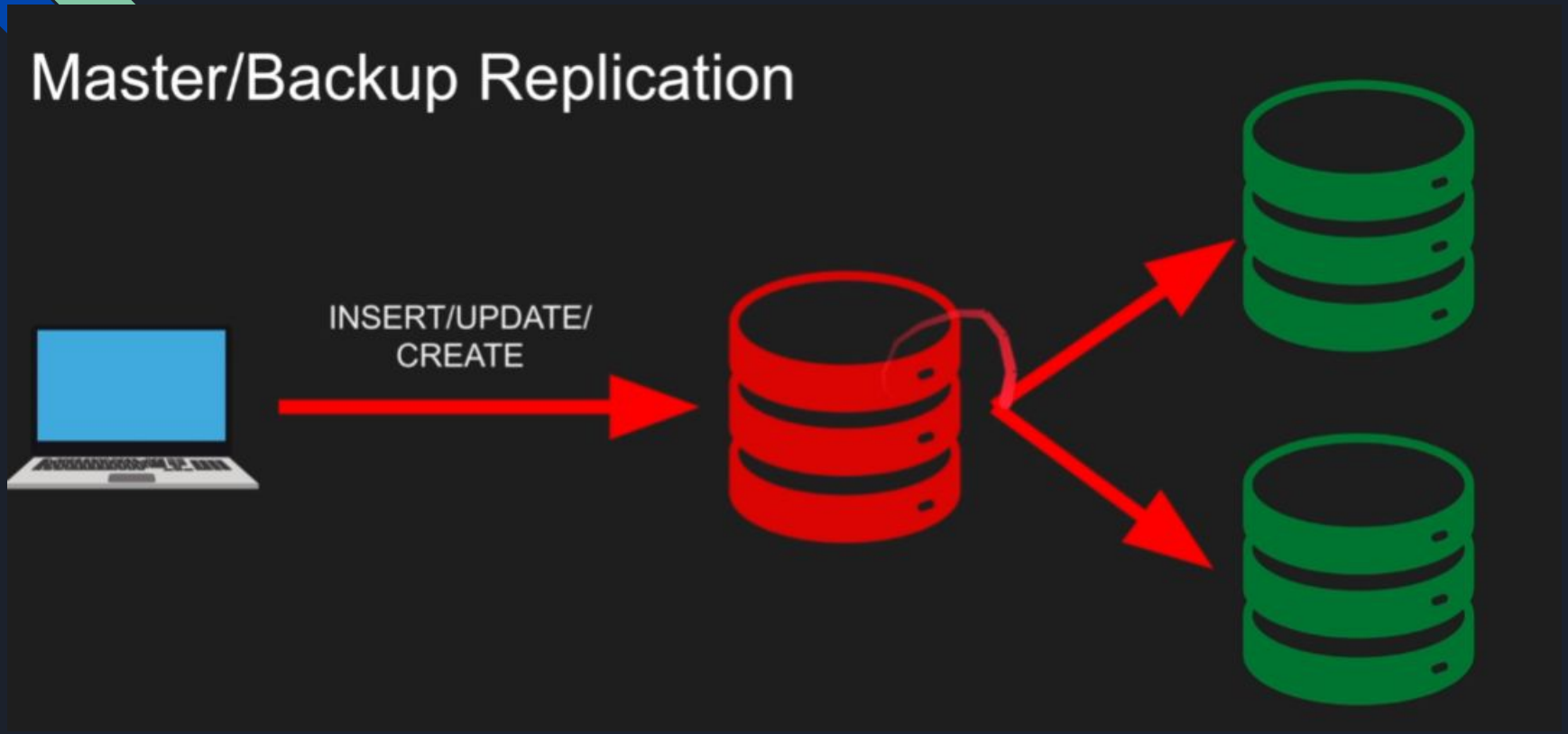
How to maintain an Index?

- Handling Index Fragmentation:
- Less than 10% fragmentation no need to fix index
- 10%-30% fragmentation, reorganize index (logical reordering)
- >30% fragmentation, rebuild index

1.6: Replication

- Involves writing or copying data same data to different locations
- It can be from between hosts in different locations or storage devices on same host or cloud based host.
- Replication Techniques:
- Master/Standby
- Synchronous vs Asynchronous

Master/Standby



Synchronous vs Asynchronous Replication

- Synchronous: Writing transactions to the master is blocked until data is also written to standby nodes
- Asynchronous: Transaction is considered complete when data is written to master and master writes to standby asynchronously

Sync Pros and Cons:

- Pro: Data written instantly to standby
- Con: Will suffer if connection between master and standby degrades

Async Pros and Cons:

- Pro: If there is a crash, data can be lost if they were not replicated.
- Con: Can tolerate some degrading in connection

1.7: What is Partitioning?

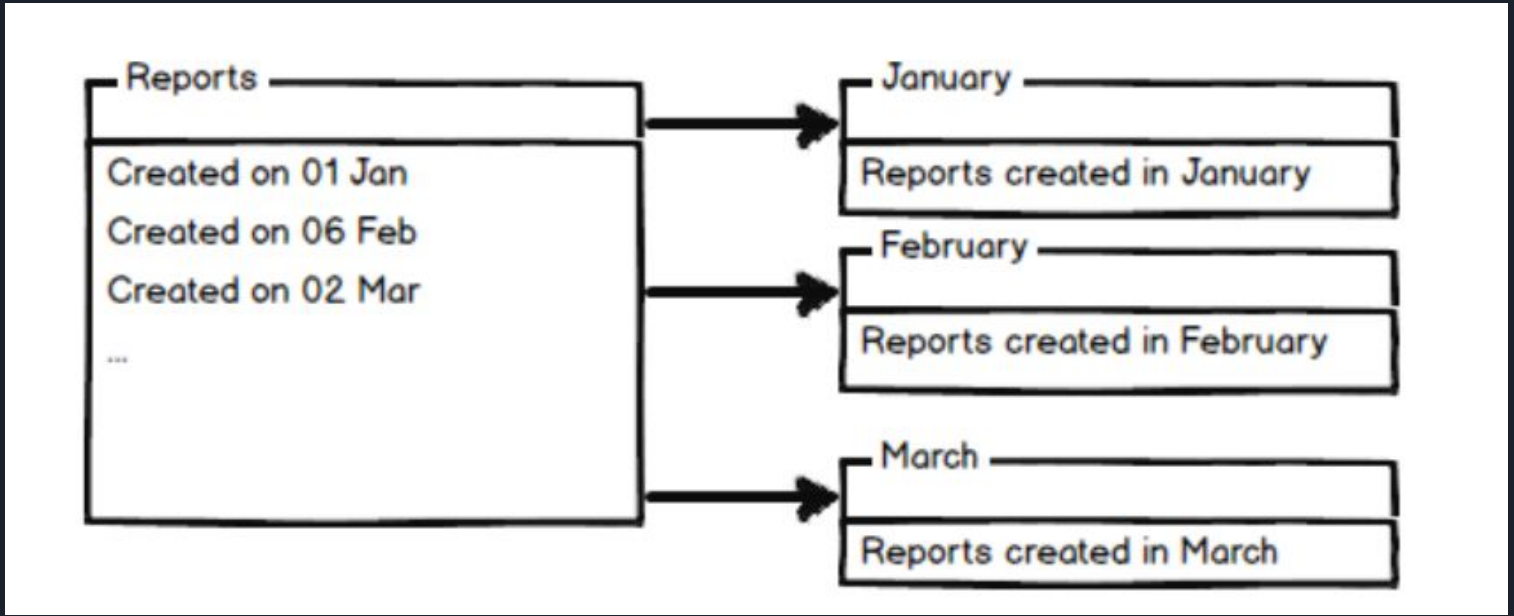
- Partitioning is the process of splitting a database into multiple tables in order for queries to execute faster since there is less data to scan.
- Goal: To decrease read and load data response time.

Partitioning Techniques

- Horizontal Partitioning
- Vertical Partitioning
- Sharding

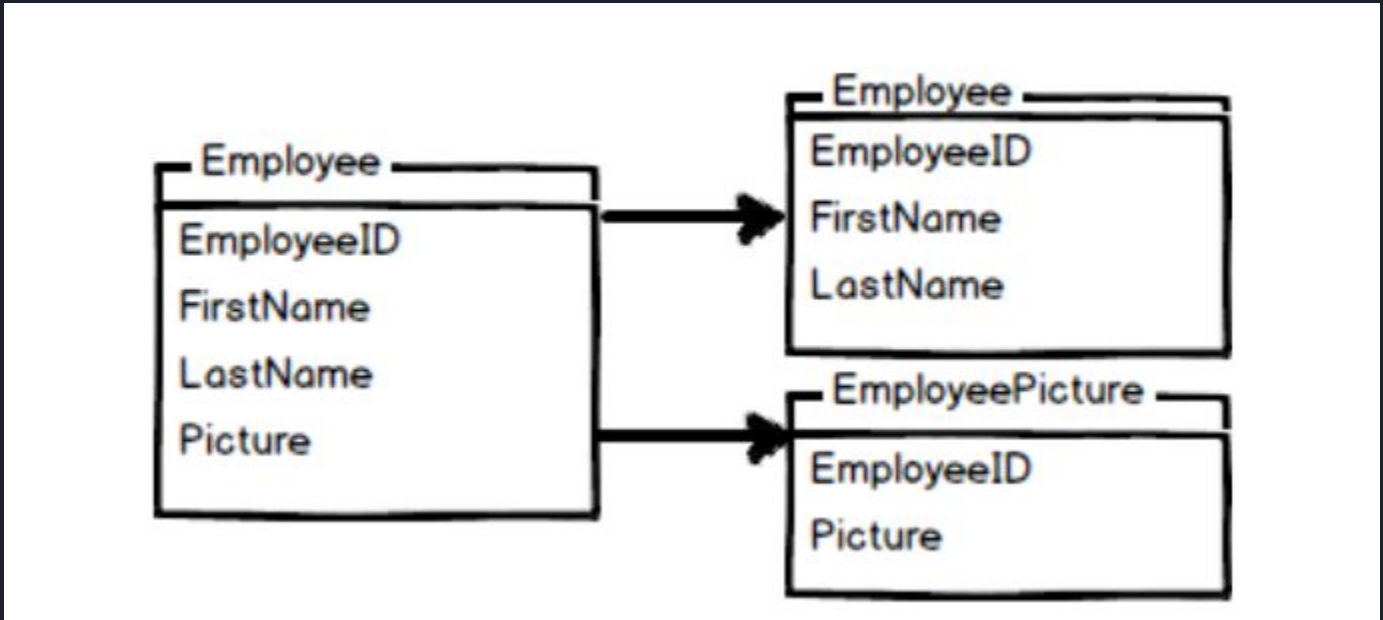
Horizontal Partitioning

Splitting rows into multiple partitions



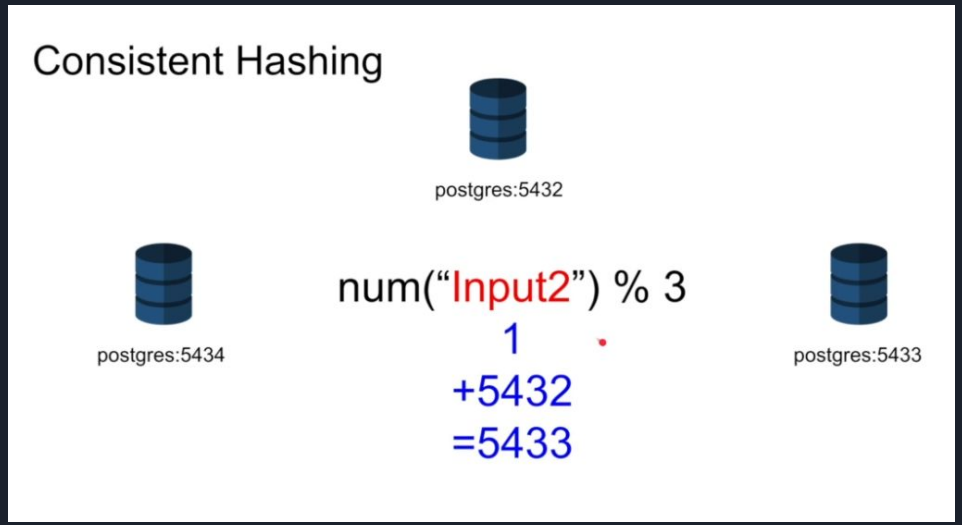
Vertical Partitioning

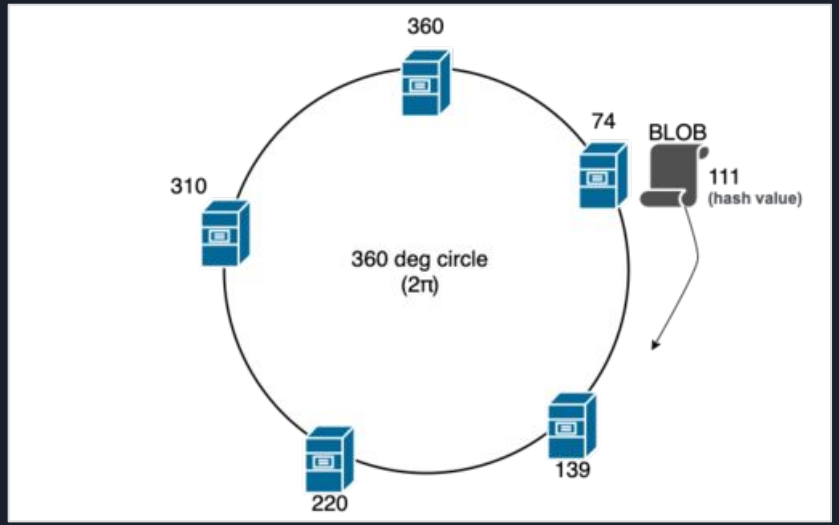
Splits Columns into partitions. Particularly used when one column contains many rows but isn't being queried as much.



Sharding

- Partitioning database into multiple database servers with the same schema.
- Use Consistent Hashing to determine which server to query on





Sharding: Advantages and Disadvantages

Advantages:

- Spreading the load of data and memory onto different servers
- Better Security (Controlling what shards are accessed by what clients)
- Since database is spread into shards, we get smaller indexes

Disadvantages:

- Cannot maintain ACID properties when Sharding
- Hard to do maintenance(ex: Adding new column)

Partitioning: Pros and Cons

Pros:

- Improving Query Performance
- Makes decision making easier for database when choosing between index scans
- Partition data that isn't accessed as much.

Cons:

- Moving Data from one partition to another is slow
- maintaining multiple partitions can be tiresome

References

Big Resource: Fundamental of Database Engines by Hussein Nasser

(https://www.udemy.com/course/database-engines-crash-course/)

Transactions:

https://www.tutorialspoint.com/dbms/dbms_transaction.htm

https://fauna.com/blog/introduction-to-transaction-isolation-levels

https://youtu.be/CTCAo89fcQw

ACID:

https://www.geeksforgeeks.org/acid-properties-in-dbms/?ref=lbp

https://levelup.gitconnected.com/transaction-isolation-levels-in-ms-sql-guide-for-backend-developers-6a5998e34f6c

https://sqlite.org/wal.html

CAP: https://www.geeksforgeeks.org/the-cap-theorem-in-dbms/

https://youtu.be/k-Yaq8AHlFA

Indexing:

https://chartio.com/learn/databases/how-does-indexing-work/

https://www.geeksforgeeks.org/difference-between-clustered-and-non-clustered-index/#:~:text=A

https://www.pgmustard.com/docs/explain/sequential-scan

https://www.spotlightcloud.io/blog/tips-for-fixing-sql-server-index-fragmentation#:~:text=External

https://blog.devart.com/sql-server-index-fragmentation-in-depth.html

https://www.sqlservercentral.com/forums/topic/index-fragmentation-and-ssds#:~:text=Index

https://www.beyondtrust.com/docs/privileged-identity/faqs/reorganize-and-rebuild-indexes-in-database.htm#:~:text=Reorganizing

Sharding:

https://youtu.be/iHNovZUZM3A

Replication:

https://www.stitchdata.com/resources/data-replication

Q&A


IF YOU HAVE ANY QUESTIONS
PLEASE ASK THEM MEOW
memegenerator.net


WHAT DO YOU MEAN
YOU HAVE QUESTIONS

Thank you for your attention!