Data Modelling and Databases

Gnkgo

2023-06-21T11:21:37+02:00

Contents

1 Normal Forms

1.1 First Normal Form

1. Using row order to convey information is not permitted
2. Mixing data types within the same column is not permitted
3. Having a table without a primary key is not permitted
4. Repeating groups are not permitted

The Key

- Values in each column should not be tables
- Records in the table should be unique
- Primary key

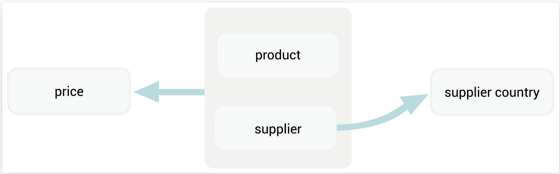

Figure 1: First Normal Form

Higher Form Find the dependencies and split them up

$$AB \to C$$
$$B \to D$$

We see that AB is a superkey, but B is a subset.

$$R1 = \{A, B, C\}$$
$$R2 = \{B, D\}$$

Here's the general algorithm for normalizing a table from 1NF to 2NF.

Suppose you have a table R with scheme S which is in 1NF but not in 2NF. Let $A \to B$ be a functional dependency that violates the rules for 2NF, and suppose that the sets A and B are distinct ($A \cap B = \emptyset$).

Let $C = S - (A \cup B)$. In other words:

- A = attributes on the left-hand side of the functional dependency.
- B = attributes on the right-hand side of the functional dependency.
- C = all other attributes.

We can split R into two parts:

- $R1$, with scheme $C \cup A$.
- $R2$, with scheme $A \cup B$.

The original relation can be recovered as the natural join of $R1$ and $R2$: $R = R1$ NATURAL JOIN $R2$

1.2 Second Normal Form

- Each non-key attribute in the table must be dependent on the entire primary key.
- If you have a strict subset of the key, it is not in 2NF.
- If everything on the right side is a key, then it is in 2NF.

The Whole Key

- No partial dependencies on the candidate keys.
- Columns must be dependent on the whole key.

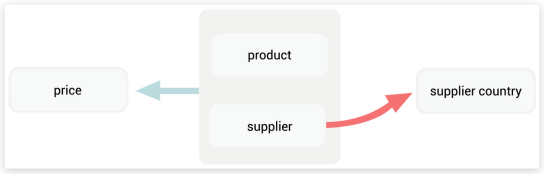

Figure 2: Second Normal Form

Higher Form - Synthesis Algorithm

$$
\begin{aligned}
R1 &= \{A, B, C, D, E, F\} \\
A &\to D \\
B &\to C \\
B &\to D \\
D &\to E
\end{aligned}
$$

ABF is a minimal key Merge

$$
\begin{aligned}
A &\to D \\
B &\to C, D \\
D &\to E
\end{aligned}
$$

Create a relation

$$
\begin{aligned}
R1 &= \{A, D\} \\
R2 &= \{B, C, D\} \\
R3 &= \{D, E\}
\end{aligned}
$$

Does one of these relations contain a key of R? NO, so we add a relation with a minimal key of R:

$$R4 = \{A, B, F\}$$

1.3 Third Normal Form

- Each non-key attribute in the table must depend on the key, the whole key, and nothing but the key.
- If everything on the right side is a key, then it is in 3NF.
- No transitive relations.
- Left-hand side is either a super key or right-hand side is a prime attribute.

Nothing but the Key (Attribute)

- Every non-key attribute is non-transitively (directly) dependent on the candidate key. The columns can only be dependent on the key columns.

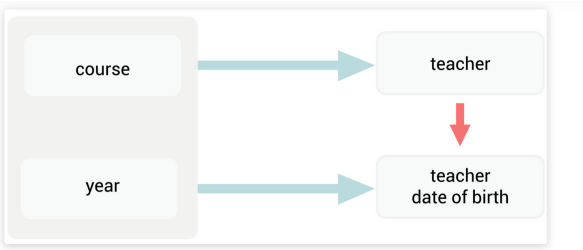

Figure 3: Third Normal Form

Higher Form - Decomposition Algorithm

1. Identify the dependencies that violate the BCNF definition and consider that as $X \rightarrow A$.
2. Decompose the relation R into XA and $R - \{A\}$ (R minus A).
3. Validate if both the decompositions are in BCNF or not. If not, re-apply the algorithm on the decomposition that is not in BCNF.

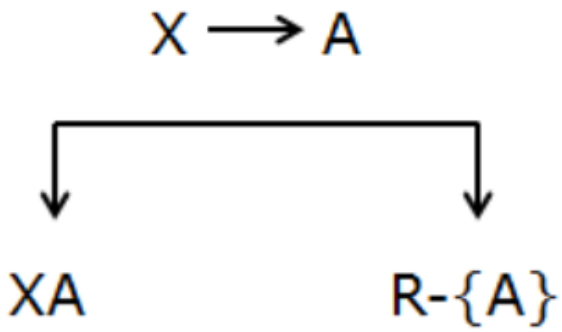

Figure 4: BCNF Decomposition

$$R1 = \{A, B, C, D, E\}$$
$$AB \rightarrow CD$$
$$D \rightarrow E$$
$$A \rightarrow C$$
$$B \rightarrow D$$

Candidate Key:AB
Prime Attributes: A, B
Non-Prime Attributes: C, D, E

$AB \rightarrow CD$ (Full Dependency - CD is dependent on the candidate key)
$D \rightarrow E$ (Transitive Dependency: non-prime derives non-prime)
$A \rightarrow C$ (Partial Dependency: Prime derives non-prime)
$B \rightarrow D$ (Partial Dependency: Prime derives non-prime)

Hence the dependencies that violate BCNF are D -¿ E, A -¿ C, B -¿ D. So, we will take $D \rightarrow E$ first as $X \rightarrow A$ (not the A listed in relation as attributes). So $X = D$ and $A = E$. XA will be DE and $R - \{A\}$ will be $ABCD$.

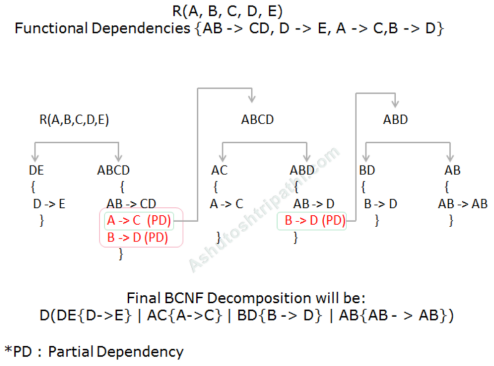

Figure 5: BCNF

1.4 Boyce-Codd Normal Form

- Each attribute in the table must depend on the key, the whole key, and nothing but the key.
- If everything on the right is a full key, it is fine (but check again).

Nothing but the Key

- All arrows must be out of candidate keys.
- Check if you have a super key with transitivity for each key.

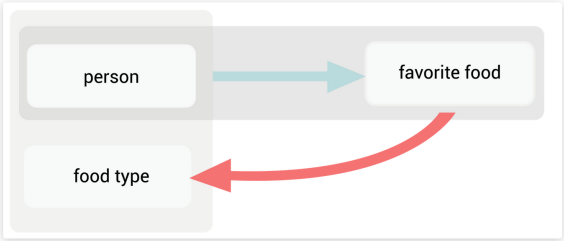

Figure 6: Boyce-Codd Normal Form

1.5 Fourth Normal Form

The only kinds of multivalued dependency allowed in a table are multivalued dependencies on the key.

1.6 Fifth Normal Form

It must not be possible to describe the table as being the logical result of joining some other tables together.

2 Candidate Key

- You have a candidate key if all its attributes appear only on the left side of the dependencies.
- To get the candidate key, look at the closure. If you can reach all the important values, it is a super key for sure. If it is the only one, then it is even a candidate key.

3 Minimal Basis

$$A \to BC$$
$$B \to C$$
$$A \to B$$
$$AB \to C$$

1. Convert right-hand-side attributes into singleton attributes

$$A \to B$$
$$A \to C$$
$$B \to C$$
$$A \to B$$
$$AB \to C$$

2. Remove the extra left-hand-side attribute

   Find the closure of A

   $A^+ = \{A, B, C\}$

So, $AB \rightarrow C$ can be converted into $A \rightarrow C$

$$A \rightarrow B$$
$$A \rightarrow C$$
$$B \rightarrow C$$
$$A \rightarrow B$$
$$A \rightarrow C$$

Remove redundant functional dependencies

$$A \rightarrow B$$
$$B \rightarrow C$$

Now, we will convert the above set of FDs into the canonical cover.

The canonical cover for the above set of FDs will be as follows:

$$A \rightarrow BC$$
$$B \rightarrow C$$

4 Example

To get the minimal cover, you have to make two steps. To demonstrate, I'll first split the dependencies into multiple (only one attribute on the right side) to make it more clean:

$$A \rightarrow B$$
$$ABCD \rightarrow E$$
$$EF \rightarrow G$$
$$EF \rightarrow H$$
$$ACDF \rightarrow E$$
$$ACDF \rightarrow G$$

The following steps must be done in this order (#1 and then #2), otherwise you can get an incorrect result. **Step 1: Get rid of redundant attributes (reduce left sides):** Take each left side and try to remove one each attribute one at a time, then try to deduce the right side (which is now only one attribute for all dependencies). If you succeed, you can then remove that letter from the left side, then continue. Note that there might be more than one correct result; it depends on the order in which you do the reduction. You will find out that you can remove B from the dependency $ABCD \rightarrow E$ because $ACD \rightarrow ABCD$ (use the first dep.) and from $ABCD \rightarrow E$. You can use the full dep. you are currently reducing; this is sometimes confusing at first, but if you think about it, it will become clear that you can do that. Similarly, you can remove F from $ACDF \rightarrow E$ because $ACD \rightarrow ABCD \rightarrow ABCDE \rightarrow E$ (you can obviously deduce a single letter from the letter itself).

After this step, you get:

$$\begin{aligned} A &\to B \\ ACD &\to E \\ EF &\to G \\ EF &\to H \\ ACD &\to E \\ ACD &\to G \end{aligned}$$

These rules still represent the same dependencies as the original. Note that now we have a duplicate rule $ACD \to E$. If you look at the whole thing as a set (in the mathematical sense), then of course, you can't have the same element twice in one set. For now, I'm just leaving it twice here because the next step will get rid of it anyway. **Step 2: Get rid of redundant dependencies** Now for each rule, try to remove it and see if you deduce the same rule by only using others. In this step, you, of course, cannot use the dep. you're currently trying to remove (you could in the previous step). If you take the left side of the first rule $A \to B$, hide it for now, you see you can't deduce anything from A alone. Therefore, this rule is not redundant. Do the same for all others. You'll find out that you can (obviously) remove one of the duplicate rules $ACD \to E$, but strictly speaking, you can use the algorithm also. Hide only one of the two same rules, then take the left side (ACD), and use the other to deduce the right side. Therefore, you can remove $ACD \to E$ (only once, of course). You'll also see you can remove $ACDF \to G$, because $ACDF \to ACDFE \to G$. Now the result is:

$$\begin{aligned} A &\to B \\ EF &\to G \\ EF &\to H \\ ACD &\to E \end{aligned}$$

Which is the minimal cover of the original set.

5 SQL Difference On, Where

The information is from [**?**].

- Does not matter for inner joins
- Matters for outer joins

a. `WHERE` clause: After joining. Records will be filtered after the join has taken place. b. `ON` clause - Before joining. Records (from the right table) will be filtered before joining. This may end up as null in the result (since OUTER join).

6 SQL `WITH`

- Referencing a temporary table multiple times in a single query
- Performing multi-level aggregations, such as finding the average of maximums
- Performing an identical calculation multiple times over within the context of a larger query
- Using it as an alternative to creating a view in the database

OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40

Table 1: Sample OrderDetails Table

The objective is to return the average quantity ordered per ProductID:

```
WITH cte_quantity AS
        SELECT
        SUM(Quantity) as Total
        FROM OrderDetails
        GROUP BY ProductID

    SELECT
        AVG(Total) average_product_quantity
        FROM cte_quantity;
```

7 SQL `OVER`

sale_day	sale_month	sale_time	branch	article	quantity	revenue
2021-08-11	AUG	11:00	New York	Rolex P1	1	3000.00
2021-08-14	AUG	11:20	New York	Rolex P1	2	6000.00
2021-08-17	AUG	10:00	Paris	Omega 100	3	4000.00
2021-08-19	AUG	10:00	London	Omega 100	1	1300.00
2021-07-17	JUL	09:30	Paris	Cartier A1	1	2000.00
2021-07-11	JUL	10:10	New York	Cartier A1	1	2000.00
2021-07-10	JUL	11:40	London	Omega 100	2	2600.00
2021-07-15	JUL	10:30	London	Omega 100	3	4000.00

Table 2: Sample Sales Table

The window frame is a set of rows that depends on the current row; thus, the set of rows could change for each row processed by the query. We define window frames using the `OVER` clause. The syntax is:

OVER ([PARTITION BY columns] [ORDER BY columns])

The `PARTITION BY` subclause defines the criteria that the records must satisfy to be part of the window frame. In other words, `PARTITION BY` defines the groups into which the rows are divided; this will be clearer in our next example query. Finally, the `ORDER BY` clause defines the order of the records in the window frame. Let's see the SQL `OVER` clause in action. Here's a simple query that returns the total quantity of units sold for each article.

```
SELECT sale_day, sale_time,branch, article, quantity, revenue,SUM(quantity) OVER (PARTITION BY article) AS total
```

This query will show all the records of the `sales` table with a new column displaying the total number of units sold for the relevant article. We can obtain the quantity of units sold using the `SUM` aggregation function, but then we couldn't show the individual records. In this query, the `OVER PARTITION BY` article subclause indicates that the window frame is determined by the values in the article column; all records with the same article value will be in one group.

8 View

An `UPDATE` statement against a View can only affect one target table at a time, and your `UPDATE` statement cannot update data in a derived column.

9 Referential Constraints

9.1 Cascade

- Propagate update or delete

9.2 Restrict

- Prevent deletion of the primary key before trying to do the change, cause an error
- Throw error immediately

9.3 No Action

- Prevent modifications after attempting the change cause an error
- Throw error after trying

9.4 Set Default, Set Null

- Set referenced to NULL or to a default value

10 Keys

10.1 Super Key

- Like a superset
- Uniquely identify the tuple
- ID, Name; ID, Phone; Name, Phone, etc.
- May contain extraneous attributes

10.2 Candidate Key

- Minimal super keys are called candidate keys

10.3 Primary Key

- Unique
- Should not have null values (email, phone number is not that good)

11 Database Systems

11.1 Disk Manager

- Allocates, deletes, fetches pages
- No other layer has to interact with the disk directly

11.2 Buffer Pool Manager

- Maintains an in-memory buffer
- Upper layers have the illusion that the entire data is in memory and not on disk
- Provides functionality to fetch and update pages

11.3 Access Methods

- Sequential Scan
- B-Tree Index
- Hash Table
- Sort

It provides a higher-level abstraction to access information in a table without interacting with the buffer or disk.

11.4 Operator Execution

Executes a relational algebra:

- Join
- Projection
- Select

11.5 Query Optimization

Generates a good execution plan.

11.6 Heap File

A heap file is an unordered collection of pages where tuples are stored in random order.

11.7 Record ID

$(PageID, SlotID)$

11.8 Pages

A page is a fixed-size block of data:

- Contain tuples, metadata, indexes, log records, etc.

11.9 Different Joins

11.9.1 Sort Merge Join (equi-join)

- Efficient: Sorted attribute (e.g., clustered index)
- Inefficient: Unsorted attribute

11.9.2 Nested Loop Join

- Efficient: Smaller relation fits into memory
- Inefficient: Both relations do not fit into memory
- $B(R) + |R| \cdot B(S)$

11.9.3 Block Nested Loop Join (equi-join)

11.9.4 Index Nested Loop Join (equi-join)

- Efficient: Low selectivity (few reads per disk)
- Inefficient: High selectivity (loads of reads per disk)

11.9.5 Hash Join (equi-join)

- If the Hash table fits in DRAM: $B(S) + B(R)$
- Very efficient
- Hash join algorithms are in general only applicable to equi joins and natural joins
- Efficient: Result of join fits into memory
- Inefficient: Result doesn't fit into memory
- Solution: Grace Hash Join

11.9.6 Grace Hash Join

$3(B(R) + B(S))$ All data ends up in the same partition or all data ends up in the same hash bucket, then the Grace Hash algorithm performs worse [**?**].

11.10 Lossless

$$S = S_1 \cup S_2$$
$$S = S_1 \bowtie S_2$$

11.11 Quiz Questions and Answers

11.11.1 While adding tuples to a page, both the slot array and the data of the tuples will grow from the beginning to the end:

The slot array will grow from the beginning to the end, whereas the data of the tuples will grow from the end to the beginning. When they meet, the page becomes full [**?**].

11.11.2 Sequential Scan or B-Tree

When nearly all the tuples fulfill the requirement, scan is fast, B - Tree is slow. When only a few or only one fulfills the requirement, B - Tree is fast, scan is slow.

$$T_{\text{scan}} = T_{\text{access}} + \frac{(\text{pageSize} * \text{m})}{\text{Bandwidth}}$$

$$T_{\text{index}}(k) = (T_{\text{access}} \frac{\text{page size}}{\dots})$$

12 Recoverability

12.1 Recoverable (RC)

If T_i reads from T_j then $c_j < c_i$. Each transaction commits only after each transaction from which it has read has committed. No need to undo a committed transaction.

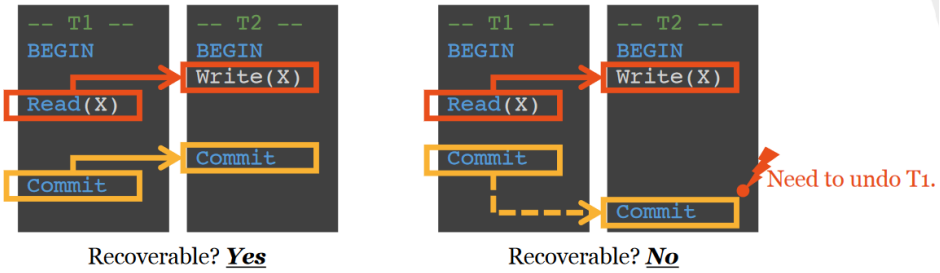

Figure 7: Recoverable Schedule [?]

12.2 Avoids Cascading Aborts (ACA)

If T_i reads X from T_j and commits then $c_j < r_i[x]$. $r_i[X]$ is the time T_i reads X. Avoids cascading rollback if transactions may read only values written by committed transactions. Aborting a transaction does not cause aborting others.

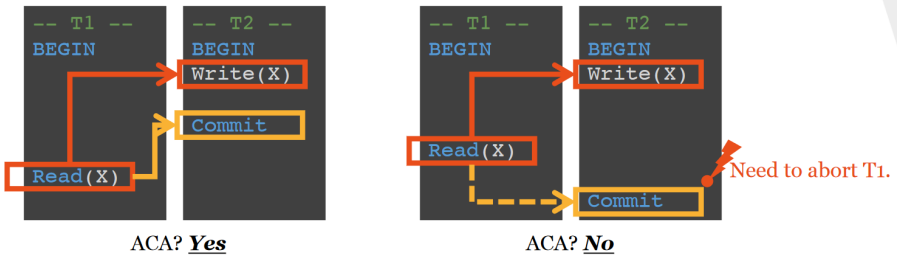

Figure 8: Avoids Cascading Aborts Schedule [?]

Strict (ST)

- If T_i reads from or overwrites a value written by T_j, then ($c_j < r_i[X]$ AND $c_j < w_i[X]$) or ($a_j < r_i[X]$ AND $a_j < w_i[X]$)
- a_j is the abort time of T_j
- Transaction must not release any exclusive locks until the transaction has either committed or aborted, and the commit or abort log record has been flushed to disk.
  - A schedule of transactions that follow the strict-locking rule is called a strict schedule.
  - Undoing a transaction does not undo the changes of other transactions.

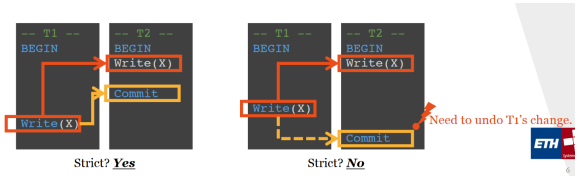

Figure 9: Strict Schedule Diagram

Serializable

Defined by the equivalence of result.

Conflict Serializable

Defined by swap adjacent, non-conflicting, operations. Conflict Serializable is a **subset** of Serializable.

How to decide Conflict-Serializability?

- Go from definition – do the swap.
- Dependency Graph.
  - You can ignore the aborts.
  - No reads, then it is automatically recoverable and ACA → just need to check strict.

Snapshot Isolation

- When a transaction T starts, it receives a timestamp $TS(T)$.
- All reads are carried out as of the DB version of $TS(T)$.
- All writes are carried out in a separate buffer.
- When a transaction commits, DBMS checks for conflicts - Abort T_1 if there exists T_2 such that T_2 committed after $TS(T_1)$ and before T_1 commits, and T_1 and T_2 updated the same object.
- Writes and readers do not block each other.

- Concurrency and availability.
  - No read or write of a transaction is ever blocked.
- Overhead.
  - Need to keep the write-set of a transaction only.
  - Very efficient way to implement aborts.
- No deadlocks, but unnecessary rollbacks.

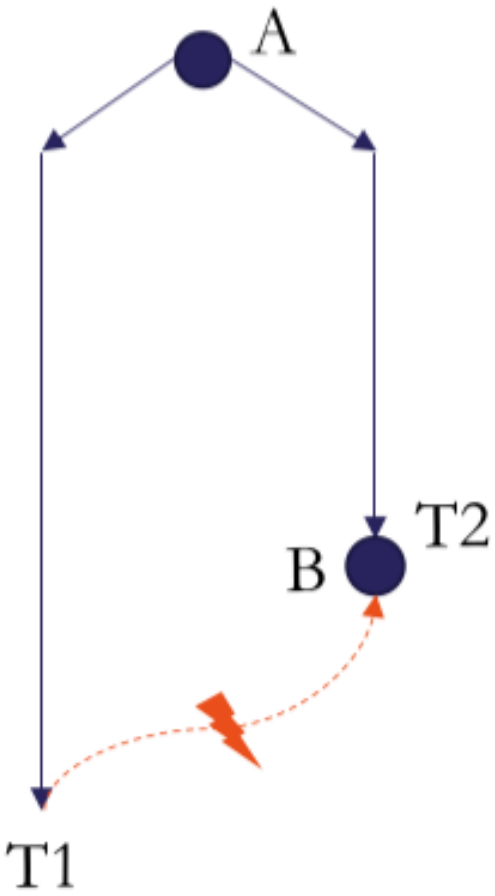

Figure 10: Snapshot Isolation Diagram

Two Phase Locking

Phase 1: Growing

- Each transaction requests the lock that it needs from the DBMS's lock manager.

- It cannot release locks in phase 1.

Phase 2: Shrinking

- Transaction is only allowed to release locks that it previously required. It cannot acquire new locks.
- Guarantees conflict serializability.
  - Only generates schedules whose dependency graph is acyclic.

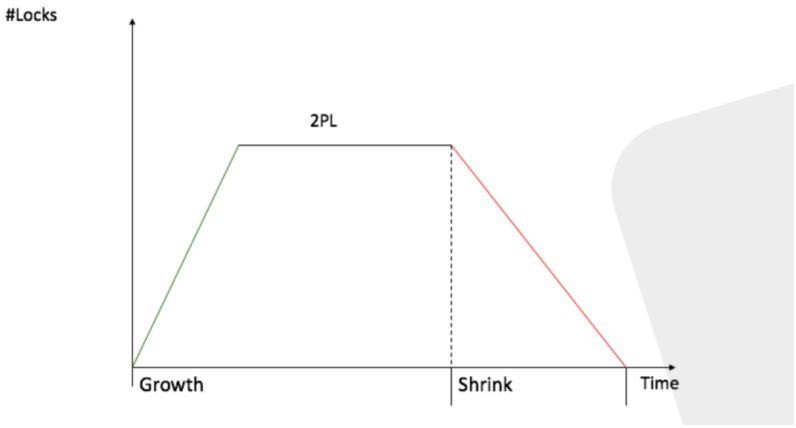

Figure 11: Two Phase Locking Diagram

Strict Two Phase Locking

At phase 2: all locks are kept until the end of the transaction (commit or abort).

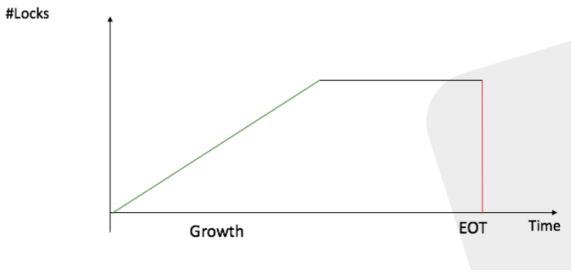

Figure 12: Strict Two Phase Locking Diagram