## Chapter 6.1 Recoverable Operation Problem and Model

*Introduction:*

In this chapter, we discuss the possible problem and how DBMS help solve these problems and start from where, and discuss how to deal with these malfunctions.

*At first, we mainly discuss ‘System Malfunction’ or ‘Crash’, and the design of Logging and Recovery is used to recover these kind of errors. Also we will introduce the Buffer Management Model, it is the fundamental of the DBMS Recovery.* Also in the next chapter, we still need this model for several transactions to access the Database.

### Chapter 6.1.1 Malfunction Model

When the database has been visited or updated, there would has a lot of problems. The problem’s range from the wrong input error data from Keyboard to Explosion which happened in the room where the Disk stored.

*Wrong Data Input*

Some wrong data input would not be detected. What we need to do is to write the constraint and trigger to find out the wrong data input.

*Medium Malfunction*

* *Partial Malfunction* - one or several bytes malfunction in the disk, and normally *Parity Check* can be used to check this issue.
* *Disk Head Malfunction* - the whole Disk can not be accessed. Then there have one or two methods to solve:
* Using *RAID model, then lost Disk can be recovered*.
* *Maintain one Backup*, which is to say that there would have one copy on tape or disk. *Create Backup periodically*, just completely or incremental, stored in safety distance far from Database.
* We can not use the Backup method, but to *save the redundancy copy online*. These copy can be distributed on several points. Later, we will discuss how to maintain the database consistency.

*Disaster Malfunction*

This kind of malfunction includes several situations that range from the location explosion or fire or malicious damage to DBMS.

Under such situation, the RAID Model can not provide any help, since all Data Disk and Parity Check would lose all their usage. But, other methods such as *Backup, Redundancy and Distributed Copy can be used to prevent the Disaster Malfunction.*

*System Malfunction*

*Transaction:*

The processor to Query and Database Modification are called transaction. Transactions can be just as all other function procedures to execute a series of steps, normally, some of which are used to modify the database.

*Each Transaction includes the status which stands for the specific step that the transaction has processed, and there have several things included in the status:*

* *The current location that transaction executes.*
* *All temporary variables which are needed later.*

*System Malfunction is the main reason that causes the loss of status.* Typical system Malfunctions are *Power Loss* and *Software Error*. Since the main memory is Volatile Storage, then Power Loss would cause the loss of all contents in main memory, the result that saved in the main memory for Transaction Steps have all disappeared, however this situation is totally different from the disk.

*However, the similar situation can also happen when there has software error that may cover parts of content in the main memory, and it may include the value of Program Status.*

When the main memory lost, then the status of the transaction is also lost, which means that we can not make sure which part of Transaction has been finished. Also re-do all transactions can not repair the problem.

*Example:*

When the value in the database should add by 1 by using the transaction, but we do not know whether we need to repeat the process.

The solution is to update all things by using the divorced, and non-volatile logging to update the database, and recover the database if necessary. But we need to ensure that the logging would be recorded under the non-distributed method, it is very complex, which would be checked in the future.

### Chapter 6.1.2 Further Discussion about Transaction

### *Definition:*

* Transaction is the Execution Unit of Database Operation, each Query or Database Update Statement is one Transaction.
* When we use Embedded SQL Interface, the range of transaction is controlled by programmer, it includes several query and update, also it can be included in the Host Language.
* In the classical Embedded SQL system, once the transaction starts, then the transaction starts, however, the end of Transaction would by ‘COMMIT’ or ‘ROLLBACK’ command.

*Ensure the rightness execution of transaction would be the work of Transaction Management, the sub - system function includes:*

* Send message to Logger Management, and make sure the necessary message has been sent and stored into the Logging.
* Ensure that currency transaction would not be blocked by each other.

*Transaction Manager Interaction With other Modules:*

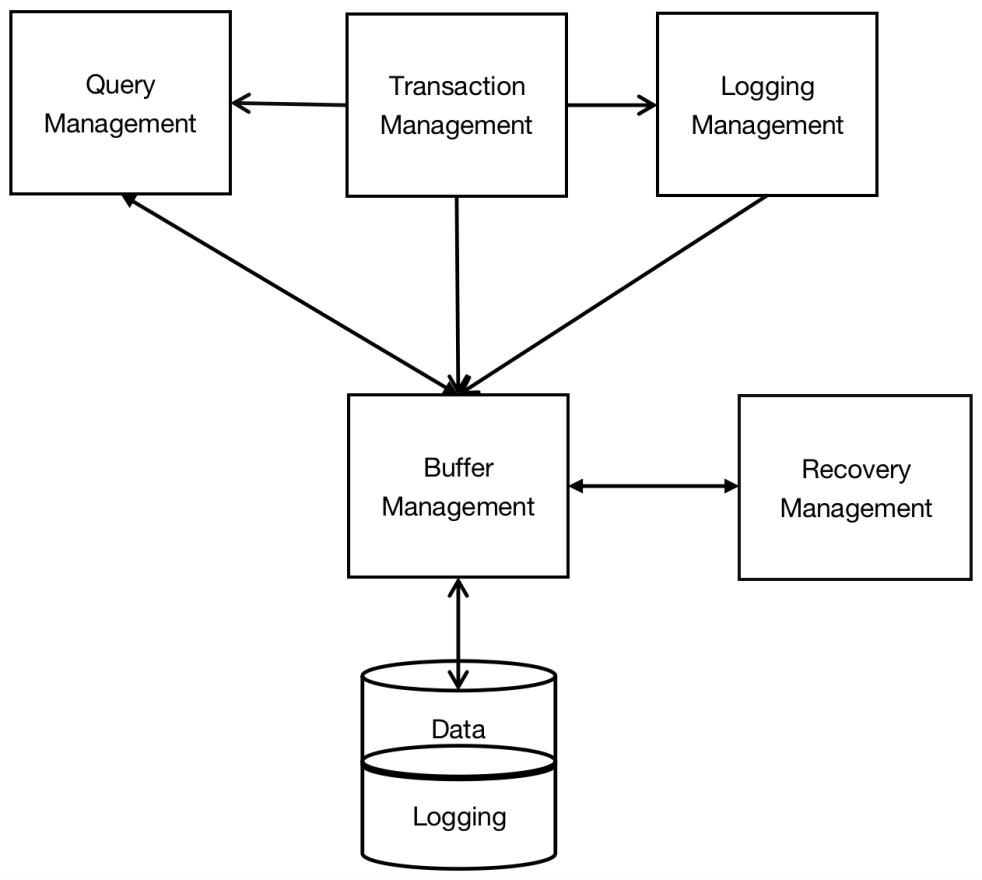
1. Sends Transaction Execution results to Logging Manager.
2. Sends how and when to copy message from Buffer Area back to the Disk to the Buffer Manager.
3. Send the message to Query Processor and enable it to execute Query and other Database Operations that consists in other Transactions.

*Main Task of Logging Manager:*

1. Logging Manager manages the Logging Management.
2. Connect with the Buffer Manager, since the Logging Space stays in the Main Memory Buffer and at the specific time, it needs to copy to the Disk. Logging is the same as Data, it occupies the Disk Space.

*Main Task of Recovery Manager:*

* When the crash happens, the Recovery Manager would be activated. It helps check Logging and would recover the data when necessary.
* Just as the normal time, the access to Disk is through Buffer Manager.



6 - 1 Logging Management and Transaction Management

### Chapter 6.1.3 Right Execution of Transaction

*Introduction:*

Before we discuss how to correct the error of the Database System, we need to figure out what the meaning of the rightness of Transaction Execution. Here, we assume that the Database System consists of *‘Element’, this Element can be accessed or modified by Transaction.* However we do not want to figure out what the element really is. *Different Elements have different Definitions, but normally it is included in one or more below:*

1. *Relation*
2. *Disk Block or Page*
3. *Single Tuple or Object in Relation.*

Below, we can image the tuple as Database Element, in many instances, you can even image it as the integer. *However select Disk or Pages as the Database Tuple has several advantages, the things in Buffer can be single element, which can avoid several serious logging and transaction problems, these problems can be discovered during learning different technology.*

Also it can help avoid the situation that the element is bigger than Database Element and avoid the situation that crash happens and parts of the elements have already been put into Non-Volatile Storage.

*The database has its status, which is corresponding to the value of each Element.* Normally, we think that some status are consistent, while others are inconsistent. *However the consistent status satisfy all constraints of the Database Module, such as the constraint on Key and on Value.*

*The consistent status should also satisfy the hidden constraints in the Designer’s intention.* However, the Hidden Constraints may be designed and maintained as part of trigger, but it also maybe shown as the strategy instruction or information warning.

*Assumption about the Transaction:*

* *Correctness Principle: If transaction can be executed with no other Transaction and System Error, and at the start of the Transaction, the database is at the consistent status, and at the end of the Transaction, the database is still at the consistent status.*

The other expression about the Correctness Principle has constructed the *Motivation of Logging technology and Concurrency Control*, which is:

1. *The transaction is Atomic; which means that the Transaction has to be executed as a whole or with nothing. If only parts of the Transaction has been executed rightly, then the generated Database should be inconsistent.*
2. *Concurrent execution of the Transaction can cause the inconsistent of database, unless we can control the mutual influence between the transactions.*

### Chapter 6.1.4 Primitive Operation about Transaction

In this Chapter, we would consider the interaction between two Transactions. Here, we have *Three Address Space*:

1. *Disks Space that are used to save all database elements.*
2. *Buffer Management that manages Virtual Memory and Main Memory Address Space.*
3. *Partial Address Space that is used in the Transaction Space.*

*Procedure:*

1. The transaction needs to read and write the element of Database, however these elements need to be stored in one or multi - buffer areas in the main memory.
2. The Data Information can be written into Partial Address Space.
3. The process that the Transaction writes data back to Disk is totally reverted. Transaction needs to create the new data in its own space and then the data can be written back to the Buffer Area.
4. The data in Buffer Area can not be written into Disk Directly, but it is the task of Buffer Management.

*(The Basic Job of Recovery is to force Buffer Management to write the Buffer back to the Disk.)*

1. Database System enables any updates to be stored into the Non-Volatile Memory, under some condition combinations.

In order to do research on Logging Algorithm and other Transaction Management details, we need to use some methods to record operations that can be used to move data between *Address Space*, here *all Atomic Operations* include:

1. *INPUT(X)*: Copy the Database Block that include the Database Element X to the Main Memory Buffer Area.
2. *READ(X, t)*: Copy the Database element X into the Transaction Temporary Variable t. More Precisely, If the block that include the database element X does not exist in the Main Memory Space, then we need to execute the INPUT(X) first, and then assign the variable X to Local Variable t.
3. *WRITE(X, t)*: Copy the Local Variable value t into the Database Element X in the Main Memory Buffer Area. More precisely, if the block that includes Database Element X does not exist in the main memory, then we need first execute INPUT(X), then copy the value t into the Buffer Area X.
4. *OUTPUT(X)*: Copy the Buffer Area that includes Data X into Disk.

As long as the Database Element exists in the single Disk Block, then the Operation above is meaningful. If one Database Element occupies multi-Blocks, then we need to treat the block size as one element. However, we also need to ensure that the logging needs to be finished when treated WRITE(X, t) as Atomic Operation, which means all Blocks that includes X write back to the Disk, otherwise have not update to the Disk at all. Assume that:

* *The size of Database Element would not exceed one Block.*

*All different DBMS would send all commands that introduced before. We can tell that READ and WRITE Atomic Operations are all sent by Transaction Management. INPUT and OUTPUT Operations are normally sent by Buffer Management, in some situation, OUTPUT Operations can be sent by Logging Management.*

*Example:*

In order to figure out how to connect Transaction to the things by Atomic Operation, we need to figure out the elements A and B in the Database, the constraint that in the database is to ensure that at any time, the value of A and B are equal.

The Transaction T consists by two steps:

1. A = A \* 2
2. B = B \* 2

If Transaction T starts from one consistent status A = B, during the process, there has no disturbance of Transactions and System Malfunctions, then the final status should be consistent, which means Transaction T would add two elements which are equal, and get another two equal elements.

The procedure of the execution T includes six Key steps:

1. *READ(A, t);*
2. *t := t\*2;*
3. *WRITE(A, t);*
4. *READ(B, t);*
5. *t := t\*2;*
6. *WRITE(B, t);*

Buffer Management would execute OUTPUT step and write these buffer areas back to Disk.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Action | t | Main Memory A | Main Memory B | Disk A | Disk B |
| READ(A, t) | 8 | 8 | 8 | 8 | 8 |
| t := t\*2 | 16 | 8 | 8 | 8 | 8 |
| WRITE(A, t) | 16 | 16 | 8 | 8 | 8 |
| READ(B, t) | 8 | 16 | 8 | 8 | 8 |
| t := t\*2 | 16 | 16 | 8 | 8 | 8 |
| WRITE(B, t) | 16 | 16 | 16 | 8 | 8 |
| OUTPUT(A) | 16 | 16 | 16 | 16 | 8 |
| OUTPUT(B) | 16 | 16 | 16 | 16 | 16 |

* The First Step, read A by Transaction T, If the block includes A doesn’t stay in Buffer Area, then this would generate INPUT(A) command called by Buffer Management. READ command would read value A from Buffer Area into Transaction Address Space.
* The Second Step, double A, this would not cause any influence on A, no matter it is on the Buffer Area or Disk.
* The Third Step it to write t back to Buffer Area A, this will not influence the parameter A in Disk.
* Next Three Steps would do the same thing to B. Finally, write back A and B to Disk.

*Analysis:*

1. We find that before A and B are written back to Disk, that is to say, before executing OUTPUT(A) and OUTPUT(B), if there has System Malfunction, then the Database that stored in the Disk would not be influenced, just as had never happened, then here can ensure that Consistency has been proved.
2. But, If the System Malfunction happens during OUTPUT(A) and OUTPUT(B), then the constraint that A should be equal to B has been violated because of the irresistible reason, since value A doesn’t equal to value B.

*Solution:*

In order to solve this kind of situation, then we can repair this kind of problem by reset A and B to 8 or just update both to 16.