Recovery and Backup

Recovery and Backup

Every system can fail; similarly computer system can fail also and result in loss of data, program, software and work. There are several reasons for the failure of computer system: disk crash, power outage, software error, fire, natural disaster, cyber attack. Failure can be of different types, some failure just result in system restart (because of system crash or some problem), some failure may cause system (here system can be any system like: operating system, database management system etc.) to stop function and some failure can cause only database management system to stop functioning. Whatever kind of failure it may be great precaution must be taken to prevent data, programs and work. Sometimes the failures may result in unnecessary waste of time of users, as they may have to rework lots of things which are lost because of system failure.

Thus, it is extremely important to take all the possible measures to maintain data intact even in case of system failure, natural disaster, disk crash etc. So, there must be backup of database stored in places from where data can be restored. Sometime active transaction may fail which may make database inconsistent. To maintain transaction integrity there must be recovery strategy/scheme to ensure atomicity and durability properties of transactions to restore database to the consistent state that existed before the database failure. The recovery strategy/scheme must ensure high availability of the data; that is, it database must be up and running quickly after the system failure.

Backup of database

It is necessary to create backup of database as database system and computer system are prone to failure. If backup is available even in case of computer failure or database system failure data can be recovered from the backup. Backup can be made in

- same hard disk,
- different hard disk in same computer,
- in different hard disk in different computers in same location or
- in computers in different locations.

The nature of service, organization and seriousness of database matters while designing database backup strategy.

Failure Classification

Let us talk about database system failures. There may be different kinds of database system failures and each of the failures must be dealt in a different manner.

- **Transaction Failure**. Transaction may fail because of:
  - **Logical error**. The program used may have some kind of logical errors like: AND operator is used where OR operator has to be used; sometimes bad input is provided, data may be missing etc.
  - **System error**. Transaction may not continue because some undesirable situation like occurrence of deadlock in the system, system not able to respond as some complex query is running in background, so, transaction on database can't continue.
- **System crash**. Operating system may hang, database system may also hang and result in restarting the system; Hardware may malfunction, or some malicious program may make

system nonfunctional and result in the loss of data of volatile storage brings transaction processing to halt.

- **Disk failure**. Data lost because disk crashed or read or write head of disk crashed.

Data needs to be written to non volatile storage device to make is secure from failures; data in non-volatile storage devices are subject to loss. Data may loss from stable storage devices also. So, it is important to know methods to protect data stored in different types of storage media; it is also necessary to know how to protect storage media from failure during data during data transfer.

Data in blocks is transferred between main memory and stable storage device (like hard disk) during processing of the data. Block transfer of data between memory and disk storage can result in:

- Successful completion. The processed data is transferred and stored safely at its destination in the disk storage.
- Partial failure. The partial failure of transaction means data has not be transferred and stored in its destination correctly.
- Total failure. Failure occurred in the beginning of transfer that means no data has been transferred to the destination and there is no error in the block.

The stage of the transaction enables to know the data transfer status and helps to recover data also. System must maintain two separate physical blocks either in same place (mirrored disks) or in remote backup. That is replica of the database must be made. An output operation should be executed as follows:

I. The data is written to first physical block
II. After the successfully writing data on the first block same information is written on the second physical block.
III. The process of writing (output) is said to be completed if write on the second physical block is completed successfully.

The system may also fail while making replication. It must be ensured that duplicate copy is created successfully. By examining the first and second block (replica) it can be known if the replication is successful or not.

- If data in first and second blocks are same with no detachable error then replication is successful and no need to take any further actions.
- If both blocks have no detectable error, but have difference in content then replication is not successful.

Data access and Blocks

Database is stored on nonvolatile storage and only part of database is stored in main memory at the time of data processing. The database is divided into fixed-length storage units known as blocks. (Blocks are the unit of data transfer to and from disk.) The blocks of nonvolatile storage are called **physical blocks** and the blocks of main memory is called **buffer blocks**. Disk buffer is the area of main memory where blocks reside temporarily.

Two operations are used to move block between disk and main memory:

- Input(B): transfers the physical block B to main memory
- Output(B): transfers the buffer block B to the disk

When a transaction, Ti, started, conceptually it has its own private working area where copies of all data items required by Ti are stored. That private working area is created as transaction is initiated and removed after the transaction either commits or aborts. Data item Xi stored in the working area of transaction Ti is denoted by xi. Transfer of data occurs between the working area of transaction and the system buffer. Two different operations are used while transferring data between system buffer and working area of transaction:

- **Read (X)**: value of data item X is assigned to the local variable xi.
    - If block Bx on which X resides is not in buffer block main memory, then input(Bx) is issued
    - The value of X is assigned to xi from the buffer block
- Write(X): value of local variable xi is assigned to data item X in the buffer block.
    - If block Bx on which X resides is not in buffer block, then input (Bx) is issued
    - Value of xi is assigned to X in buffer Bx.

Recovery and Atomicity

Let us consider example of transferring 50 units of product from store A to store B. Initially, Store A has 90 units of product and store B has 110 units of same product. Suppose transaction, Ti, which is transferring 50 units from store A to store B crashed during the execution, after output(Ba) has taken place, but before output (Bb) was executed. Here Ba and Bb denote the buffer blocks on which A and B reside.

As the system restarts, the quantity of product in store A will be 40 as deducting product has been done, and that of store b will be 110 as adding failed. This resulted in an inconsistent state of database; here the atomicity property of transaction has been violated; not all operations of transaction Ti have been successfully committed.

Log Records

Log is a data structure which records the all the steps performed during database modifications. In the log sequence of steps taken while modifying database is written. The sequence of log is called log records. All the update activities performed in the database are recorded in the log records. Update log record is one type of log record which describes a single database write. It has following fields:

- **Transaction identifier**: identifier of the transaction that performed write operation.
- **Data-item identifier**: identifier of the data item which is written; it is the location of data item on disk; it includes block identifier of the block on which data item resides, and an offset within the block.
- **Old value**: the value of data item prior to the write
- **New value**: the value of data item will have after the write.

An update log record can be represented as <Ti, Xj, V1, V2> where

- Ti is transaction performing write on data item Xj
- V1 is old value of Xj, that is before write

- V2 is new value of Xj, that should be after write

There exists other type of log records also:

- <Ti start>. Transaction Ti has started.
- <Ti commit>. Transaction Ti has committed.
- <Ti abort>. Transaction Ti has aborted.

It is necessary for a transaction to create a log record for the write operation, add write operations to the log before the database is modified. Log record enables to output the modification to the database if it is all right and also enables to undo a modification that has already been done.

Log records must be stored in stable storage to make it useful for recovery from system and disk failures.

Database Modification

How to decide if a transaction has to be redone or undone? If a transaction aborts then any changes made by it must be undone and if a transaction commits successfully then it must be redone. It is log record with the help of which it can be decided which changes made by aborted transactions are to be undo and which changes made by committed transactions are to be redone.

It is necessary to know the steps taken by transaction while modifying a data item:

I. Some computations are performed in own private part of transaction in the main memory
II. Data block residing in disk buffer is modified by transaction
III. Output operations is executed to write data block to disk by database system

Database is modified only if an update on a disk buffer or on the disk itself is performed. Modification can be:

- Deferred-modification: transaction does not modify database until transaction is committed
- Immediate-modification: transaction modifies database when it is active.

The log records are created as transaction is started, with the log records system can perform undo and redo operations as appropriate:

- Undo: data item is set to old value
- Redo: data item is set to new value

Transaction Commit

If a commit log record of a transaction (the last log records of the transaction) is output to stable storage then the transaction is said to be committed. Transaction is rolled back if system crashed before the log record <Ti commit> is output to stable storage.

Let us see how log can be used to recover from a system crash. Consider our example of stores. Let T0 be a transaction that transfers 50 units of product from store A to store B:

<T0 start>
<T0, A, 90, 40> 90 is old value of A and 40 is new value of A
<T0, B, 110, 150> 110 is old value of B and 150 is new value of B

<T0 commit>

Using a log, lots of system failures, which don't result in loss of information in nonvolatile storage can be handled. The recovery scheme uses two recovery procedures to find data items that are updated; old and new values of the updated data items from the log records.

- Redo(Ti) sets the value of all data items updated by transaction Ti to the new values

Log	**Database**
<T0 start>	
<T0, A, 90, 40>	
<T0, B, 110, 150>	
	A = 40
	B = 150
<T0 commit>	

Figure 5. State of system log and database

- Undo(Ti) restores the value of all data items updated by transaction Ti, to the old values.

The system consults log record to determine which transactions need to be redone and which need to be undone in the case of system crash.

- Transaction Ti is undone if the log contains the record <Ti start>, but doesn't contain either record <Ti commit> or <Ti abort>.
- Transaction Ti is redone if the log contains <Ti start> and either <Ti commit> or <Ti abort>.

Let us illustrate the recovery process with example

Log		
<T0 start>	<T0 start>	<T0 start>
<T0, A, 90, 40>	<T0, A, 90, 40>	<T0, A, 90, 40>
<T0, B, 110, 150>	<T0, B, 110, 150>	<T0, B, 110, 150>
	<T0 commit>	<T0 commit>
	<T1 start>	<T1 start>
	<T1, C, 900, 800>	<T1, C, 900, 800>
		<T1 commit>
Situation (a)	Situation (b)	Situation (c)
Figure 6 The same log, shown at two different times		

Let us consider three different situations:

Situation (A)

Let us assume that system crash occurred just after the following step is written to the log record:

Write(B)

of transaction T0 has been written to stable storage (Fig 6a). when the system restarts, if finds <T0 start> in the log, but no corresponding <T0 commit> or <T0 abort> records. Thus, transaction T0 must be undone, undo(T0) is performed and the values of quantity of product in store A and B (on the disk) will be 90 and 110 respectively.

Situation (B)

Suppose system crashed just after writing following step to the log records:

Write(C) of transaction T1 (Fig b). When the system restarts, two recovery actions are taken. One is undo(T1) is performed and another is redo(T0) is performed. Undo(T1) is performed as T1 has <T1 start> but no corresponding <T1 commit> or <T1 abort> records; Redo(T0) is performed as T0 has <T0 start> and corresponding <T0 commit>. So, at the end of whole recovery procedure, the values at Store A will be 40, B will be 150 and C will be 900.

Situation (C)

Suppose system crashed just after <T1 commit> has been written to stable storage (Fig C). When the system restarts, both T0 and T1 will be redone, since <T0 start> and <T0 commit> and <T1 start> and <T1 commit> records appear in the log. After the system recovery, values at Stores A, B and C will be 40, 150, and 800 respectively.

Checkpoints

When system crash occurs whole log file is checked to find which transaction to redo and which to undo. This is a time consuming process and most of the transactions have to be redone even they are already written their updates into the database. This overhead can be reduce by using the concept of checkpoints.

Suppose if <Ti commit> record (or <Ti abort> record) of transaction Ti appears before <checkpoint L> in the log record, then it should be understood that any database modification made by Ti must have been written to the database prior to the checkpoint or as part of checkpoint itself.

Creating backup of database in MySQL Database Server using phpMYAdmin

Here are the steps to back up of database in MySQL:

1. Select the database of which backup has to be created
2. Click the EXPORT tab in the top horizontal menu
3. Specify format of the backup file as SQL in the combobox
4. Click Go.

Making backup of selected tables only from a Database

1. Open the database of which tables are to be backed up

2. List the tables of selected database by clicking the tables option of selected database from the left menu
3. Select the tables of which back have to be created
4. Select the Export option in the comboBox found below the list of tables
5. Specify format of tables are SQL in the ComboBox
6. Click Go.

Restoring database from backup

1. Open the phpMyAdmin panel
2. Select IMPORT from the top horizontal menu
3. In the File to import section
    a. Specify the backup file
    b. Specify the format as SQL in the format ComboBox
    c. Click go.

Making backup of MySQL database from command line using mysqldump

Backup of database from MySQL console is created using **mysqldump** command.

```
$ mysqldump –u [user name] –p[password] [database name] > [name of backup file.sql];
```

Suppose, backup of database named library2017 has to be created, user name of database user is root and password is blank. Then following command is used

```
$ mysqldump –u root –p  library2017 > library2017_2.sql;
```

Creating backup of required tables only from a database

```
$ mysqldump –u root –p library2017 member  books > tables_library.sql;
```

Here, library2017 is database; member and books are the tables of library2017 database and the backup of the tables will be created in the tables_library.sql file.

Restoring database from backup files

To retore database from backup files, two things have to be done.

1. Create a database with appropriate name
2. Load file from backup using **mysql** command.

For example, let library2017_2.sql is backup of database. Then using following command the database can be restored from the backup.

```
$ mysql –u root –p library2017_1 < library2017_2.sql;
```

Here, library2017_1 is new database created and library2017_2.sql is backup file of a database.

Recovery and Backup

Creating backup of MySQL database using Program

Here is a small program written in PHP to create backup of database: library2017.

```
<?php

$con = mysqli_connect("localhost","root","","library2017") or die("Connection can't be set");
$myfile = fopen("librarybackup.sql", "w");
$r = mysqli_query($con,"select * from member") or die("query problem: ".mysqli_error($con));
$str = "";
fwrite($myfile,"--Records of member--\n");
while($row=mysqli_fetch_assoc($r)){
        $str="insert into member
values('".$row["memberid"]."','".$row["m_name"]."','".$row["level"]."','".$row["semester"]."','".$row["
program"]."');\n";
        fwrite($myfile,$str);
        echo $str;
}
$r =  mysqli_query($con,"select * from books") or die("query problem: ".mysqli_error($con));
$str = "";
fwrite($myfile,"--Records of book--\n");
while($row=mysqli_fetch_assoc($r)){
        $str="insert into books
values('".$row["bookid"]."','".$row["b_name"]."','".$row["quantity"]."');\n";
        fwrite($myfile,$str);
        echo $str;
}

$r =  mysqli_query($con,"select * from issue") or die("query problem: ".mysqli_error($con));
$str = "";
fwrite($myfile,"--Records of issue--\n");
while($row=mysqli_fetch_assoc($r)){
$str="insert into books
        values('".$row["issue_no"]."','".$row["member_id"]."','".$row["book_id"]."','".$row["issue_date
"]."','".$row["return_date"]."');\n";
        fwrite($myfile,$str);
        echo $str;
}
?>
```

Here,

1. A file librarybackup.sql is created in write mode.
2. Connection with Database library2017 is made
3. Records from table of the database library2017 is extracted, insert statement is formed for each table and each insert statement is written to the librarybackup.sql file.