ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**Database Management System**

**Theory Assignment**

**SUBMITTED BY:**

Ronak Agrawal

013BSCCSIT033

**SUBMITTED TO**

|  |  |
| --- | --- |
| **Er. Sanjay Kr. Yadav**  **( Lecturer )** |  |
| **Department of Computer Science** | |

**Transaction Management**

**Transaction**

A transaction is a logical unit of work that contains one or more SQL statements. A transaction is an atomic unit. The effects of all the SQL statements in a transaction can be either all committed (applied to the database) or all rolled back (undone from the database).

A transaction begins with the first executable SQL statement. A transaction ends when it is committed or rolled back, either explicitly with a COMMIT or ROLLBACK statement or implicitly when a DDL statement is issued.

To illustrate the concept of a transaction, consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction can consist of three separate operations:

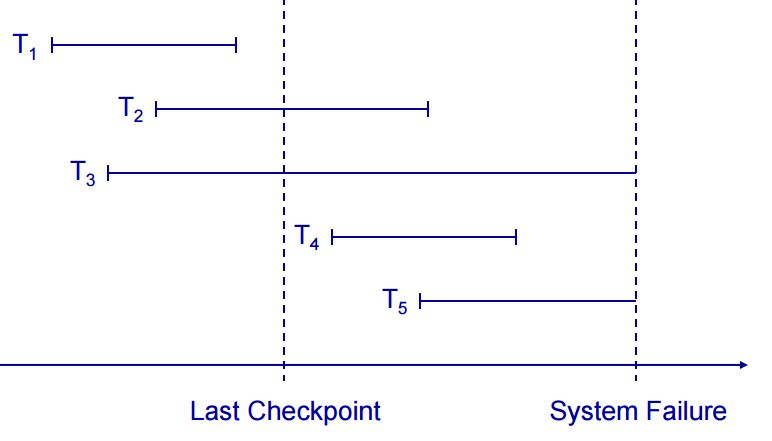
* Decrement the savings account
* Increment the checking account
* Record the transaction in the transaction journal

**Properties of Transactions**

Every transaction, for whatever purpose it is being used, has the following four properties. Taking the initial letters of these four properties we collectively call them the **ACID Properties**. Here we try to describe them and explain them.

1. **Atomicity:** This means that either all of the instructions within the transaction will be reflected in the database, or none of them will be reflected.
2. **Consistency:** If we execute a particular transaction in isolation or together with other transaction, (i.e. presumably in a multi-programming environment), the transaction will yield the same expected result.
3. **Isolation:** In case multiple transactions are executing concurrently and trying to access a sharable resource at the same time, the system should create an ordering in their execution so that they should not create any anomaly in the value stored at the sharable resource.
4. **Durability:** It states that once a transaction has been complete the changes it has made should be permanent.

**Types of Transaction:**



**States of a transaction**:

* **Active:** Initial state and during the execution
* **Paritally committed:** After the final statement has been executed
* **Committed:** After successful completion
* **Failed:** After the discovery that normal execution can no longer proceed
* **Aborted:** After the transaction has been rolled back and the DB restored to its state prior to the start of the transaction. Restart it again or kill it.

**System Recovery:**

* Any transaction that was running at the time of failure needs to be undone and restarted
* Any transactions that committed since the last checkpoint need to be redone
* Transactions of type T1 need no recovery
* Transactions of type T3 or T5 need to be undone and restarted
* Transactions of type T2 or T4 need to be redone

**Transaction Recovery:**

* **Backward Recovery:**
  + We need to undo some transactions
  + Working backwards through the log we undo any operation by a transaction on the UNDO list
  + This returns the database to a consistent state
* **Forward Recovery:**
  + Some transactions need to be redone
  + Working forwards through the log we redo any operation by a transaction on the REDO list
  + This brings the database up to date

**Media Recovery:**

* Restore the database from the last backup
* Use the transaction log to redo any changes made since the last backup
* If the transaction log is damaged you can’t do step 2
  + Store the log on a separate physical device to the database
  + The risk of losing both is then reduced

**Two Phase Commit**

A feature of [transaction processing](http://www.webopedia.com/TERM/T/transaction_processing.html) systems that enables [databases](http://www.webopedia.com/TERM/D/database.html) to be returned to the pre-transaction state if some error condition occurs. A single transaction can update many different databases. The two-phase commit strategy is designed to ensure that either all the databases are updated or none of them, so that the databases remain synchronized.

Database changes required by a transaction are initially stored temporarily by each database. The transaction monitor then issues a "pre-commit" command to each database which requires an acknowledgment. If the monitor receives the appropriate response from each database, the monitor issues the "commit" command, which causes all databases to simultaneously make the transaction changes permanent.