What is Transaction Processing?

#3 IN A SERIES

One of the most common types of data processing is referred to as "**transaction processing**". Most individuals have used "transaction processing" at one time or another, even though they might not have recognized or referred to it as such. Here are a few common examples of "transaction processing":

- Withdrawing money from an ATM machine
- Making a hotel, car, or airline reservation
- Making a credit card purchase
- Using a home banking application to transfer funds between bank accounts
- Order entry performed by a call center clerk from a catalog sale customer

If we examine these examples in detail, we can compile a list of characteristics that will help us understand transaction processing and be able to identify transaction processing in general.

Let's use the ATM example above since most people are familiar with it. Let's examine an ATM withdrawal from the customer's perspective. While at the ATM, it appears as though you are the only one interacting with the bank's computers. But in reality, the bank may have hundreds or possibly thousands of ATM terminals available for use. So the bank's computer systems have to be ready for the possibility of potentially thousands of customers wanting to concurrently do banking at one of their ATM terminals. Next, if you think about what you can do at an ATM, you realize that there are a relatively small number of actions that can be requested. For example, you can make a deposit, make a withdrawal, transfer funds, or request a balance. Anything beyond these few actions requires that you see a teller in the bank. And these actions are the same for any of the potentially thousands of customers wanting to use the bank's ATMs. So, at this point, we have identified a couple of key characteristics of transaction processing: a large number of users and a relatively small number of actions (or "transactions") that the users can request.
Now think about the transaction you request at the ATM. Let's say it is a withdrawal. Do you recall any instance when you became impatient waiting for your cash to be dispensed? How long was it before you became impatient? Right, not very long. So, another important characteristic of transaction processing is that the request (or transaction) needs to be completed very quickly – usually within a matter of a couple of seconds (or less) or else the end user becomes impatient and begins to suspect something is wrong. The fact that the transactions are predefined (deposit, withdraw, transfer, etc.) and that every bank customer is repeatedly using the same ones helps make it possible to process transactions quickly, even though we cannot accurately predict the volume or sequence of the transactions that will be entered.

And finally, when you request a withdrawal transaction, you expect that the request will be processed against the current value in your account. That is, you are expecting the current account balance to be accurate and that it reflects <u>all</u> prior account activity (deposits and withdrawals). (More on this below)

Now let's look at the ATM transaction processing requests from the bank's perspective. To manage the potentially large number of incoming requests, each of which can come from any customer, the bank's computers need to be able to access the data representing any of the customer's account information. This data is stored in databases and/or files that are shared – that is, all of the bank's customers need access to the same files and/or databases. But customers cannot get to all of the data within those files and/or databases – they can only "see" and affect their own account information. The banking applications and transaction processing software (commonly called a "transaction server" or "application server") are responsible for making sure the integrity of the data is maintained. So, customer

A cannot affect customer B's data and vice versa. This is both a security and a data integrity concern. Further, any possible "collisions" need to be detected and managed. For example, suppose two people named on a single bank account went to two different ATMs and each entered a transaction at exactly the same time. If one transaction was for a deposit and the other was for a withdrawal, what would happen to the data representing the bank account balance? The transaction processing software needs to ensure that one transaction doesn't overlay the other causing data corruption. The two transactions can be processed serially, but they cannot be processed simultaneously. This is accomplished using "locks" or "semaphores" and is the responsibility of the application and transaction server software.

If you read the computer science literature regarding transaction servers and transaction processing, you will find the acronym "ACID". It represents the four properties that a transaction server must have to do transaction processing. ACID stands for Atomicity, Consistency, Isolation, and Durability. In the paragraph above, we discussed an example of a transaction server that managed data integrity and ensured that data remains "consistent" under all circumstances. The data is always taken from one consistent state to another. The other ACID properties are equally important. Atomicity refers to the update process – the update(s) for a transaction must all occur or none of them can occur. That is, an "all" or "nothing" approach to updates is used. If some part of a transaction fails, the entire transaction fails and the data is left unchanged. Isolation refers to the property that ensures that changes from a transaction that is not yet complete, cannot be seen by other transactions. In general, transactions are not aware of other transactions that are running concurrently. Durability refers to the property that ensures that once a change is made to the data (i.e. committed), that change is "durable" or persistent no matter what error or failure occurs afterwards – the results of the completed transaction cannot be lost. The ACID properties are covered in more detail in most computer science texts.

Transaction processing is generally not CPU intensive. That is, it usually involves little "number crunching" (think about the update to a bank account as a result of a deposit or withdrawal – the update is done with just one add or subtract instruction). But it can, and usually does, involve a lot of I/O (to authenticate the customer, bring into memory the customer information, etc.). An application workload that involves little CPU processing and lots of I/O is an ideal match for an enterprise server[1] with a robust I/O subsystem. Consequently you will often times find transaction processing systems running on an enterprise server and thus the enterprise server "owning" or managing the key data for the business.

So there you have it. Transaction processing involves a handful of predefined requests coming from a potentially large number of end users and each of those requests (transactions) must be processed quickly while maintaining data integrity for all of the shared data. From a data processing perspective, transactions represent a discrete unit of processing (or unit of work) to the application. From a business perspective, a computer transaction usually corresponds directly to a business transaction (buy, sell, deposit, reserve, etc.). Or it may even correspond to a collection of business transactions (for example, reserving a hotel, flight, and car all at once for a trip). It is not unusual for businesses to process millions of transactions per day so another business characteristic of a transaction is that is must be extremely cost effective. The cost to run any transaction needs to be measured in pennies or fractions thereof, not nickels and dimes.

We will explore other aspects of Enterprise Computing in subsequent articles.

[1] See ECI No. 5