Introduction to Databases

Reading / References

- Concise Guide to Databases: A Practical Introduction[1] chapters 1-3
- Introduction to SQL: Mastering the Relational Database Language[2]
- MySQL[3]

Practice questions on the reading

- What are the data-related challenges of corporate takeovers?
- What are some privacy issues that have arisen related to data in the modern world?
- What is a *transaction*?
- What are the fundamental operations in a data management system?
- What does the acronym ACID stand for?
- How does a data warehouse differ from a database?
- Modern database systems offer two-phase and three-phase commits. Explain what this achieves.
- Most databases these days are *relational*. Explain what that means.

Notes

Data management systems require a diverse set of operations, but there are four fundamental operations that may be desired of any such system. These are affectionally known via their acronym, CRUD:

Create We need the ability to create new entries and records in the system. For instance a new product for our company, a new course to be offered, and so on.
Read We need the ability to access stored information, preferably in flexible ways and with efficiency.
Update We often need to update existing entries or records, e.g. when cash is withdrawn from a bank account, when a book returns from a loan, when a course changes name.
Delete On occasion we may need to delete entries, often for example as a result of erroneous entry or because the record is no longer relevant.

Different management systems exist depending on the importance and use of these four operations on the current system.

[1] http://learning.acm.org/books/book_detail.cfm?id=2560109&type=24
[2] http://learning.acm.org/books/book_detail.cfm?id=1208031&type=safari
[3] http://learning.acm.org/books/book_detail.cfm?id=2484635&type=safari

Data Warehouse Data warehouses are specifically designed to only *create* and *read* data, as well as facilitate the *analysis* of the data, and does not allow its modification. Warehouses are essential pipelines in the processing of data from an operational system to some other processing center. For instance the wireless access points around campus could be constantly sending information to a data warehouse about the different contact they have with the various computers, tablets and phones that use them. This information is effectively streaming constantly, and it needs to be stored somewhere and quickly retrieved for analysis, but it never has to be updated.

Similarly we could imagine a network of sensors recording temperature information from various locations, which may be fed into some analytical process. A data warehouse would be well suited to manage such a situation.

Databases Databases are meant to support all CRUD operations. They are therefore best suited for record-keeping working with data that requires updates. A bank's account information would be a good example of the need for a database. While information about the various transactions requested at a bank's ATMs or the internet could be stored well in a warehouse, the actual account information might best be served in a database that allows the update of those accounts.

Transactions and ACID

Databases are built around the idea of processing transactions. A **transaction** is one or more operations that make up a single task, considered together as a unit. For example a money transfer requires a number of things:

- To read the source account's information and ensure there is enough money to withdraw.
- Withdrawing that money from the source account.
- Depositing that money in the target account.

These things need to happen as a unit: We don't want the target account to be credited some money without also withdrawing the money from the source account. We also would not want to withdraw the money from the source account without also ensuring that it get deposited to the target. If we do these steps in order and the power goes out halfway through, we don't want to be left in the middle of it. So *the tasks in a transaction must either all happen or all not happen.* If something goes wrong halfway through the process, we need to have the ability to do a **rollback**.

Four principles guide this behavior:

Atomicity Atomicity refers exactly to the idea that either the entire transaction happens, or none of it happens. We need to have the ability to return to the state before the transaction started. A successfully processed transaction is said to have undergone a **commit**, while a failed would undergo a **rollback**.

Consistency Consistency refers to rules that may be in place about the state of the database, and ensuring that any committed transaction leaves the database in a valid state. For instance we should not be allowed to remove a course from the database and at the same time leave students somehow marked as “enrolled” in that course. Or we shouldn’t be allowed to delete a course without also deleting the sections of it. Or we might have a rule that says that we cannot have a course without a least one instructor assigned to it.

Isolation Transactions must be independent of each other. Or to be more precise, changes to a transaction must not be visible to other transactions until the transaction has completed. Consider for example the Read-Withdraw-Update money transfer example we considered, and imagine two different transactions like that both reading from the same account. They may both “read” that there is enough funds there for *their* transaction, and then they would both attempt to withdraw, where there might have only been enough funds for one transaction. The system must be such as to not allow this to happen: A transaction should not be able to update a value that is in the process of being updated by another transaction, if those transactions happen in parallel.

Durability The effects of a transaction should persist in the system. An account withdrawal shouldn’t simply vanish if a power loss occurs.

Relational Databases

One of the most popular types of databases are the so-called **relational databases**. Data is arranged in tables, each table consisting of records with predetermined fields. For instance in our student evaluation database there is a students table, whose records contain three fields, for login, first name, and last name.

Tables are then linked to each other via joins that use **foreign keys**. For example section enrollments are a separate table, whose records consist of a foreign key pointing to a student and a foreign key pointing to a section, and the presence of these two together suggests that the student is enrolled in the section.

An important feature that led to the popularity of relational databases is the existence of a powerful query language, known as SQL. We will be learning more about relational databases and SQL in the coming days.

NoSQL

NoSQL, standing for “Not only SQL”, is a term used to refer to a number of recent database models that do not rely on the rigid relational table structure, but store information in other more flexible formats. A number of popular systems exist and we will be looking at them all in the near future.