CSCC43: Introduction to Databases
Lecture Notes

Joshua Concon

University of Toronto Scarborough – Fall 2017

Pre-reqs are CSCB63 and STAB52. Instructor is Dr. Marzieh Ahmadzadeh. Check RateMyProf. If you find any problems in these notes, feel free to contact me at conconjoshua@gmail.com. I own nothing, all copyrighted works belong to their respective owners (University of Toronto and Pearson Education).

Contents

1 Thursday, September 7, 2017

1.1 Definitions

Data Objects or events that could be recorded on a computer media. This includes stuff like an email, an address, a student identification number, etc.

However, there are two types of Data, Unstructured and Structured.

Structured Data They are mostly stored in a tabular format, have types including: date, character, numeric, alphanumeric.

Unstructured Data Data that must be analyzed to extract information from, does not already show information. Includes multimedia, maps, documents, pictures, voices.

Database An organized collection of logically related data that can be of any size or complexity. Example:

- Student ID, name, GPA, courses taken
- Patient name, doctor's ID, ward, date of admission
- Personnel ID, base salary, bonus, hours of work

Another Example:

ID	Course	Mark
100	CSCC43	A
100	CSCC44	B
100	CSCC52	A+
200	CSCC43	A+
200	CSCC62	B

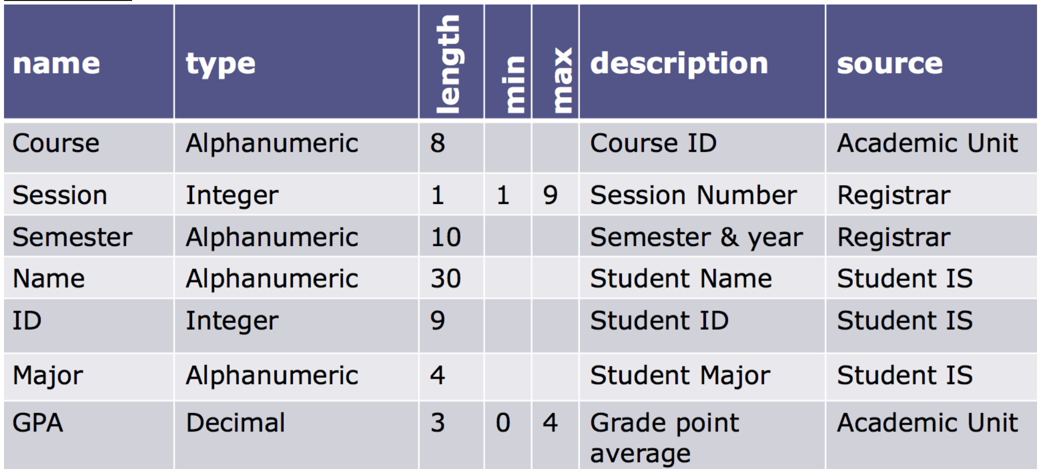
Database Management System (DBMS) A software that sits between users and databases, responsibilities include:

- Allowing users to create, update, sort or retrieve data

- Enforcing the integrity of data (consistency, etc.)
- Improving performance when dealing with huge data or many queries
- Assuring the durability of the data
- Allowing data to be shared among different programs
- managing concurrent access by multiple users/processes

Information Information is data that has been process in order to increase user knowledge, could be data provided in a way that can be analyzed easily (i.e. has labels, maybe graphs showing relationships, etc.)

Metadata "Data about data". Data that describes the properties of end-user data and it's context, it helps the database designer understand what kind of data exists and what it means. Examples include name, type, length, allowable values, ownership (stewardship), source...
Example:



name	type	length	min	max	description	source
Course	Alphanumeric	8			Course ID	Academic Unit
Session	Integer	1	1	9	Session Number	Registrar
Semester	Alphanumeric	10			Semester & year	Registrar
Name	Alphanumeric	30			Student Name	Student IS
ID	Integer	9			Student ID	Student IS
Major	Alphanumeric	4			Student Major	Student IS
GPA	Decimal	3	0	4	Grade point average	Academic Unit

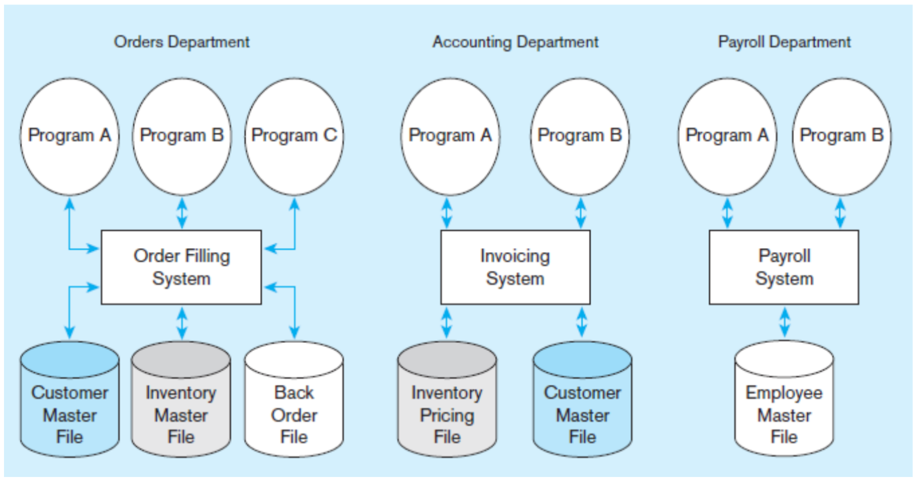
1.2 How do we store data?

There are two ways we can store data, either the Traditional file system or the database approach.

1.2.1 Traditional File System

The Traditional File System includes files that are also collections of related data, was used prior to the invention of databases, but has its own disadvantages.

Example of a file system:



Disadvantages:

Program-Data Dependence If the data is changes, all programs that used the data should be altered to. To use an example from the previous image, if the length of data such as customer name is changed, order filling system programs and invoice system programs are both affected. Using the database approach, this is managed by a Database Management System and no change in order filling or account programs are required, only on the file itself.

Duplication of Data As seen from the last image, the order filling system and the invoice system both have a customer master file. This is a waste of space as it is essentially two sets of the same data, and reliable metadata is difficult to establish since it may be inconsistent. Changes to one copy must also be applied to the other, so this may compromises the files' data integrity.

Limited Data Sharing Since each application has its own data, data sharing between applications is not easy, a system does not have access to another system's data. With the database approach, data sharing is doable.

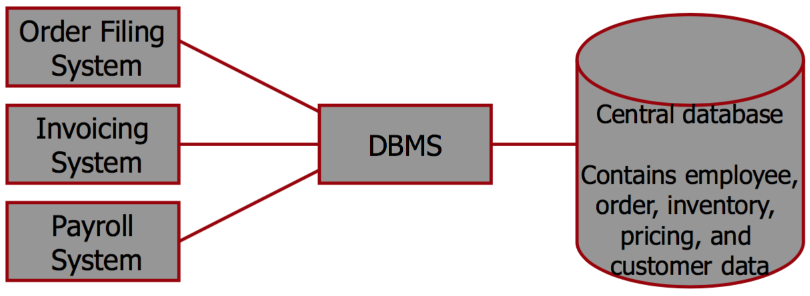
Length Development Times Each new application should be started from scratch by designing a new file format and description, however, in the database approach, you only need to design your database once, and can easily access the data you need.

Excessive Program Maintenance All preceding factors creates heavy program maintenance, so to avoid these disadvantages, the database approach is preferred. Note that if a database is not properly designed, mentioned disadvantages may not disappear.

1.2.2 Database Approach

The database approach creates a repository of shared data, data is managed by a controlling agent (Database Management System), and all data is stored in a standardized convenient form.

Visualization Example:



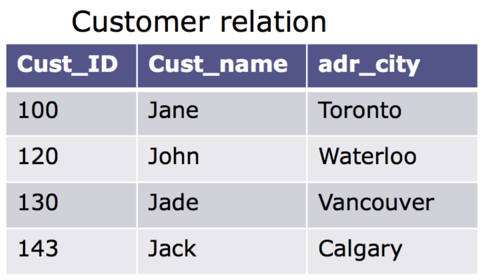
Every Database uses a data model, and this data model is a notation for describing data that specifies the structure of the data, the constrains on the content of the data, and the operations on the data.

Some examples of data models include a relational data model, a semistructured data model, unstructured data, and an Object-oriented model.

1.3 Relational Data Model

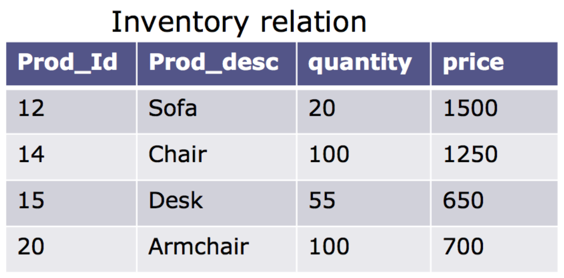
The main concept of the relational data model is a relation, which is based off the concept of relations in math. A relation in a database is like a table consisting of rows and columns.

Example:



Customer relation

Cust_ID	Cust_name	adr_city
100	Jane	Toronto
120	John	Waterloo
130	Jade	Vancouver
143	Jack	Calgary



Inventory relation

Prod_Id	Prod_desc	quantity	price
12	Sofa	20	1500
14	Chair	100	1250
15	Desk	55	650
20	Armchair	100	700

1.3.1 Advantages

Program-data independence Metadata is stored in a central location called a repository, and data could be changed independent of what program uses it.

Planned data redundancy Data is only duplicated when we want it to be duplicated, so we can control redundancy.

Improved data consistency Which is implied since data redundancy is eliminated.

Increased productivity of application development Since the application does not deal with the data management, since it is taken care of by the Database Management System, so time and cost of development is saved.

Enforcement of standards Standards are enforced by the uniform procedures for accessing and updating data.

Improved data quality By defining constraints that the data must follow.

Improved data accessibility and responsiveness Can be done by using a query system to access data.

Reduced program maintenance Because of data-program independence, changing the data does not affect any program.

Improved data sharing By creating a user view and since all applications share the same data already, data sharing is doable.

1.3.2 Cost and Risk

New specialized personnel Companies need to hire new people for database design and database administration services.

Installation and Management of Cost and Complexity Implementation of the database and scaling it will be a new concern for companies

Conversion Costs Costs for converting legacy system to a modern database

Need for backup and recovery New costs and concerns for backing up the database and new services companies must worry about for recovery.

Organizational Conflict On data definition, data format and right to update data.

1.4 Database Development Process

Model the rule of the organization Such as, "each student at UofT can take up to 5 courses in one semester"

Explore a relation between various entities Such as, "A course might be taken by many students", "A student must take at least one course", etc.

Create your relation Creating the tables with relations between the entities and data associated with each object

Normalize your relation We will learn what this is later in the course

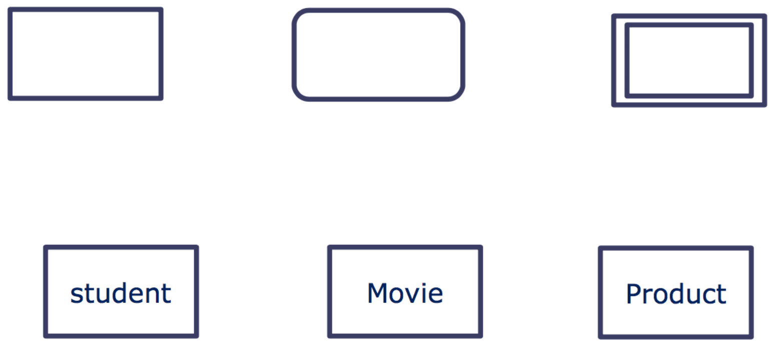
Implementation Actual Implementation of the database

2 Thursday, September 14, 2017

2.1 Relational Data Model

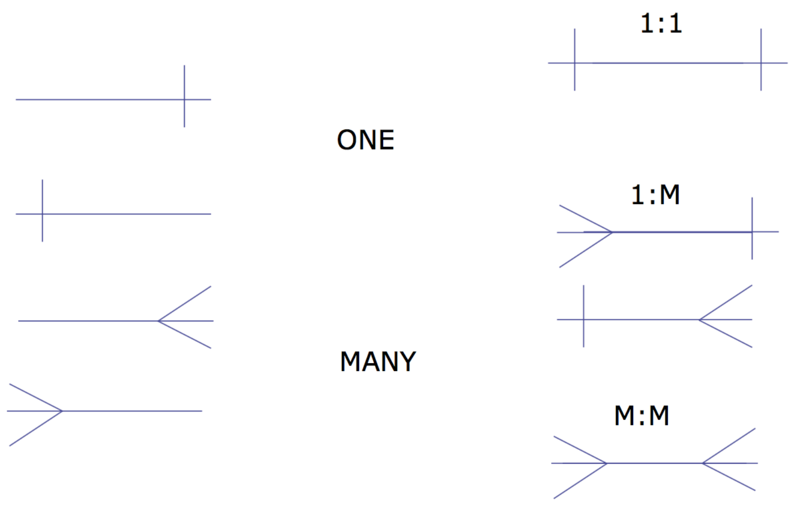
A relational data model has 3 components, one of which is an **Entity**, which contains the Entity itself: which is a concept that we maintain data about, an **Entity Type**: which corresponds to a table and what the information in the table refers to, and an **Entity Instance**, which is an instance of an entity, such as (100, Jane, 3.5), which refers to a student named Jane with a Student ID of 100 and a cGPA of 3.5.

The notation for the entities is as follows (It's mostly just a rectangle tbh):



Another one of the 3 components is a **Relationship**, which contains a **Relationship Instance**: which is an association between entity instances itself, such as a Student named Jane taking 3 courses (which is a 1:M relationship as 1 student can take many courses) and a **Relationship Type**: which was the type of relationship, such as (1:M) as described in the previous example.

The notation for relationship types is as follows:



The last of the 3 components are **attributes**, which are properties or characteristics of an entity, which often corresponds to a field in a table. For example, for a student entity, an example attribute for this student entity would be ”cgpa” or ”student ID”.

2.2 Database Analysis

Before the database can be created, you must first gather user requirements for the data, what exactly we need this data to do and what kind of constraints we need. We'll need to know the entities we would have to make for the database and what relationships these entities have with each other.

Afterwards, we must develop a data model, more specifically, an Entity Relationship (ER) model.

2.3 Business Rules

Business Rules are statements that define or constrain some aspect of the business. For Example: ”at UofT, a student must take at least one course in each semester.” These statements are usually derived from policies, procedures, events, and functions, for example: ”at UofT, a student should have passed the prerequisites to be able to take a course.” These constraints and definitions must also be familiar to the end users and must be automatically enforced by the Database Management Software.

We gather these business rules by speaking to people involved in the business that this database is being created for and by investigating business documents.

2.3.1 Cardinality Constraints

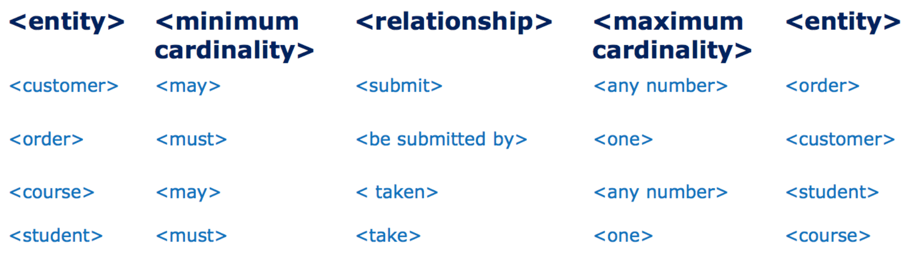
Cardinality Constraints are rules that specifies the number of instances of one entity that can be associated with each instance of another entity. Examples include:

- "Each student at UofT can take at most 5 courses each semester"
- "Each person can have at most 1 family doctor"
- "A movie can be screened at most 6 times a day"

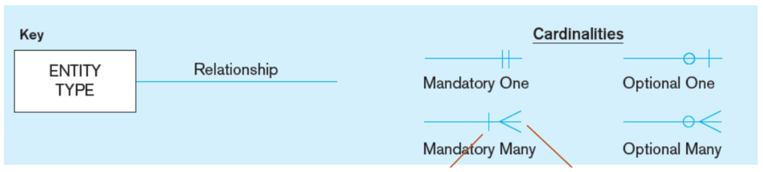
Minimum Cardinality is the minimum number of instances of one entity that can be related to the other instance.

Maximum Cardinality is the maximum number of instances of one entity that may be related to each instance of another entity.

The notation for these cardinality constraints are as such:



<entity>	**<minimum cardinality>**	**<relationship>**	**<maximum cardinality>**	**<entity>**
<customer>	<may>	<submit>	<any number>	<order>
<order>	<must>	<be submitted by>	<one>	<customer>
<course>	<may>	< taken>	<any number>	<student>
<student>	<must>	<take>	<one>	<course>



2.4 Entity Relationship (ER) Model

After business rules have been collected, from these rules, relationships between entities can be modelled, and for this, we will use the Entity Relationship Model. This model is a logical representation of the data for an organization, the entities are to represent the data, and the relationships are to show the associations between entities.

2.4.1 Entity

An entity is the building block of the Entity Relationship Model, such examples include:

- Person: *employee, student, patient*
- Place: *store, warehouse, state, drugstore*
- Object: *machine, building, automobile*

An entity is supposed to have a noun name and is meant to be general.

Each entity usually consists of several **attributes.**

An entity should be an object that will have many instances in the database, and object that will be composed of multiple attributes, and/or an object that we are trying to model. However, it should also not be a user of the database system or an output of the database system.

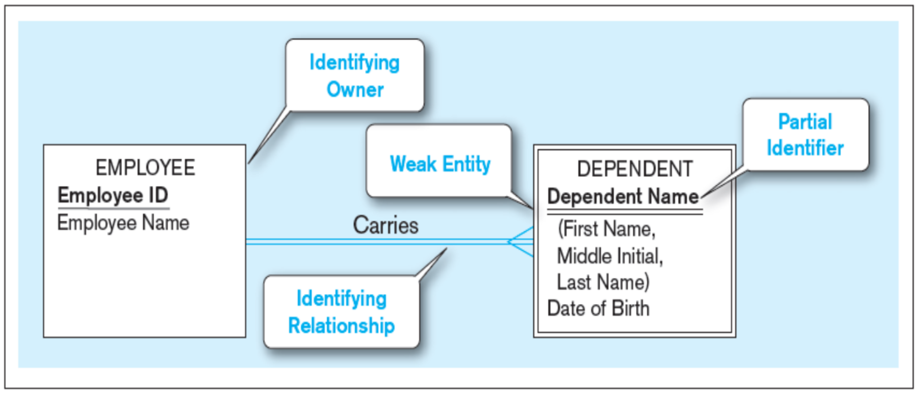
2.4.2 The Weak shall fear the Strong

There are two different types of Entities, **Strong** and **Weak** entities.

Strong entities exist independently of other types of entities and have their own unique identifier (e.g students, employee, course, surgeon, independent black woman who need no man).

Weak entities depend on a strong entity, which may or may not be their identifying owner, and cannot exist on it's own. They also do not have a unique identifier, only a partial identifier, and the entity box and partial identifier have double lines.

The following is an example of a weak entity:



2.4.3 Attributes

Each entity type has a set of **attributes** associated with it, and by default, the name of an attribute starts with a capital.
s
An attribute has different types, including: required, optional, simple, composite, single values, multivalued, stored, derived, and identifier.

Required Attributes Required Attributes should have a value. Must be boldfaced or represented by a special character in E-R Diagrams.

Optional Attributes Optional Attributes can be null. May be represented by an 'o' in E-R Diagrams.

Composite Attributes An attribute that has meaningful component parts. For example: an address can be broken down to city, street, etc. It is up to the user as how to define these attributes. If you might use only part of an attribute, you might want to separate it as its own attribute.

Simple Attributes An attribute that cannot be broken down into smaller components, such as a StudentID.

Multivalued Attributes An attribute that can have more that one value. For example, skill can be 'programmer', 'analyst', etc. This type of attribute is represented with curly braces in ER diagrams.

Single Valued Attributes An attribute that can only have one value, such as an attribute that can only be a boolean (True or False).

Derived Attributes An attribute that is derived from other stored attributes. For example, years employed can be derived from today's date minus date employed. This type of attribute is represented with square brackets [] in ER diagrams.

Stored Attributes An attribute that is not derived from other stored attributes, and is just stored.

Identifier Attributes An attribute whose value distinguishes instances of an entity type, so no two instances are referred to by one identifier, such as an EmployeeID for example.

Composite Identifier An identifier that consists of a composite attribute. For naming these attributes, it should be a unique singular noun or noun phrase such as 'Customer ID' or 'Product Minimum Price'. The name should also follow a standard for uniqueness, such as [Entity type name [Qualifier]] Class, where the Qualifier is a phrase to constrain the class, and the Class is the properties of the entities. An example of an attribute under this template is 'Customer hourly wage'. Similar attributes of different entity types should use the same qualifiers and classes, such as 'Faculty Residence City' and 'Student Residence City'.

2.4.4 Criteria for Identifiers

Identifiers must be values that will never be null and will not change in value. It is ideal to substitute new, simple keys for long, composite keys if possible.

2.4.5 Modelling Relationships

A relationship in E-R diagrams are modelled by verbs and is represented by a line labelled with the name of the relationship.

Degree of Relationships The degree of a relationship is the number of entity types that participate in it, so for example, a **Unary** relationship would refer to one entity related to another of the same type, a **Binary** relationship would refer to two different entities, a **Ternary** relationship would refer to three different entity types.