NOTES ON DATABASE DESIGN
OPIM 3103

<u>1. Key terms and ideas</u>

What is database?

Data Terminology and Concepts

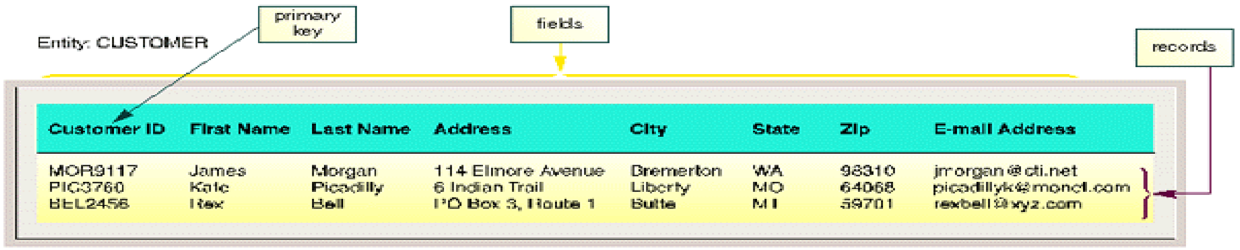
– Field (Attribute): A single characteristic or fact about an entity
– Record (Tuple): A collection of fields that describes one instance of an entity
– Table (Entity): A collection of records

(A ***table*** may be also called an ***entity*** or a ***relation*** (hence the name "relational databases" – please do not confuse this with a *relationship* between entities)

A ***column*** is also called a ***field*** or an ***attribute***

A ***row*** is also called a ***tuple*** or a ***record***)

– Database: A collection of **related** tables


Entity: CUSTOMER
primary key
fields
records

Customer ID	First Name	Last Name	Address	City	State	Zip	E-mail Address
MOR9117	James	Morgan	114 Elmore Avenue	Bremerton	WA	98310	jmorgan@cti.net
PIC3760	Kate	Picadilly	6 Indian Trail	Liberty	MO	64068	picadillyk@moncl.com
BEL2456	Rex	Bell	PO Box 3, Route 1	Butte	MI	59701	rexbell@xyz.com

First off, why do we study the database design?

Databases are at the core of virtually all business processes these days. Without databases and database management systems, it would be virtually impossible to implement the types of business decisions that are required from modern-day managers

The apps, platforms (Amazon, Facebook,…) use database to store their data.

Ok, but can't we use Excel to store data?

We could, but we should not. There are three reasons for that:

- Excel is **limited in the amount of information** it can store in a table. If you are curious, go ahead and find the maximum number of rows and columns that may be in a single Excel spreadsheet. The storage capacities of database management systems are virtually unlimited and are restricted only by the physical capacities of hard drives and other media
- We should use the tools for the purposes they were designed for. Excel is designed for **handling calculations**. It is not good for handling the **storage and access** to data

- Probably most importantly, storing data in a spreadsheet-type manner results in enormous difficulties **when parts of the data needs to be updated**. For ease of management and to minimize errors, data should be split in logically independent but connected units. This is the principle behind the relational database design

So, what are the components of a relational database and why is it so good for data management?

A relational database stores data in several tables that are connected to each other. Usually, tables correspond to logical structures or real-world objects. Such objects are usually described by some attributes, which are represented by columns of the table. Information about a particular instance of the object is represented by a row of the table. Some terms are used interchangeably, depending on the perspective from which people come to the database design world.

Relational databases are good for data management, because they **minimize the difficulty of updating the information and probability of creating mistakes.** At the same time, they also **store data efficiently (minimum redundancy) and preserve important relationships in data.**

How is data redundancy minimized?

This is the role of the primary key in a table. A ***primary key (PK)*** is a column or several columns in the table, which allow to uniquely identify each record in that table. By having a PK, we make sure that there are no cases when two or more exact same records are present. Consequently, we need to make sure that the values of the PK in each table are unique (non-repeating) and never null (non-empty). There must be one and only one PK in each table. The idea of PK is related to the concept of functional dependency (see below)

How are the relationships between data preserved?

This is the role of foreign keys. A ***foreign key (FK)*** is a column or several columns in a table, which are, in turn, a PK in some other table. This way, the table which includes a FK is logically connected to the table that gave its PK. There may be multiple FKs (or no FK at all) in a table.

2. Relationship

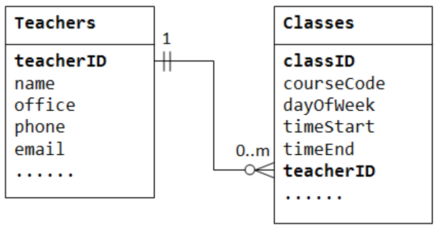
http://www.ntu.edu.sg/home/ehchua/programming/sql/relational_database_design.html

A database consisting of independent and unrelated tables serves little purpose (you may consider to use a spreadsheet instead). The power of relational database lies in the relationship that can be defined between tables. The most crucial aspect in designing a relational database is to identify the relationships among tables. The types of relationship include:

1. one-to-many
2. many-to-many
3. one-to-one

One-to-Many

Question: Design a database that can store the information about teachers and courses.
Constraint: One class must be taught by one teacher. One teacher can teach multiple classes or zero class.
You can determine how many tables you need to use in this database.
In a "class roster" database, a teacher may teach zero or more classes, while a class is taught by one (and only one) teacher. In a "company" database, a manager manages zero or more employees, while an employee is managed by one (and only one) manager. In a "product sales" database, a customer may place many orders; while an order is placed by one particular customer. This kind of relationship is known as *one-to-many*.
One-to-many relationship cannot be represented in a single table. For example, in a "class roster" database, we may begin with a table called Teachers, which stores information about teachers (such as name, office, phone and email). To store the classes taught by each teacher, we could create columns class1, class2, class3, but faces a problem immediately on how many columns to create. On the other hand, if we begin with a table called Classes, which stores information about a class (courseCode, dayOfWeek, timeStart and timeEnd); we could create additional columns to store information about the (one) teacher (such
as name, office, phone and email). However, since a teacher may teach many classes, its data would be duplicated in many rows in table Classes.
To support a one-to-many relationship, we need to design two tables: **a table Classes to store information about the classes with classID as the primary key**; and **a table Teachers to store information about teachers with teacherID as the primary key**. We can then create the one-to-many relationship by storing the primary key of the table Teacher (i.e., teacherID) (the "one"-end or the *parent table*) in the table classes (the "many"-end or the *child table*), as illustrated below.



The column teacherID in the child table Classes is known as the *foreign key*. A foreign key of a child table is a primary key of a parent table, used to reference the parent table.

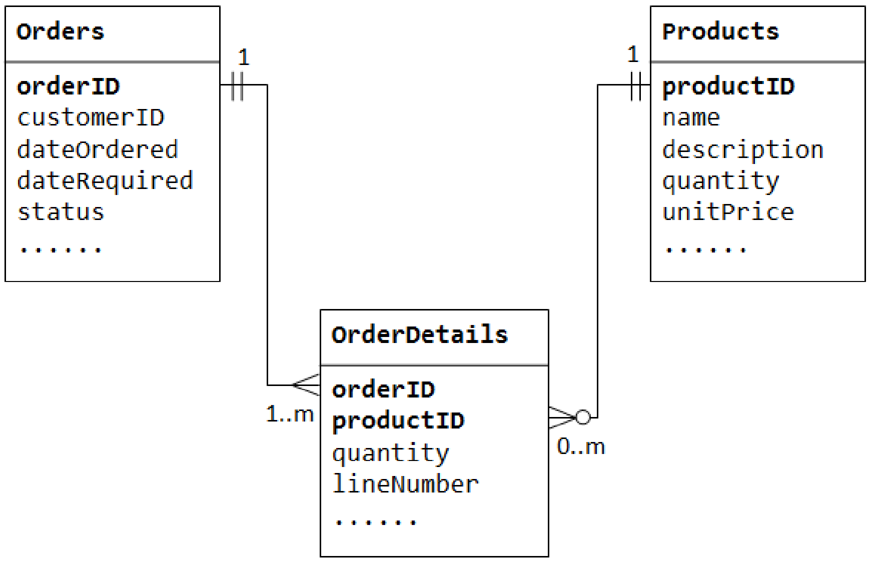
Take note that for every value in the parent table, there could be zero, one, or more rows in the child table. For every value in the child table, there is one and only one row in the parent table.

Many-to-Many

In a "product sales" database, a customer's order may contain one or more products; and a product can appear in many orders. In a "bookstore" database, a book is written by one or more authors; while an author may write zero or more books. This kind of relationship is known as *many-to-many*.

Let's illustrate with a "product sales" database. We begin with two tables: Products and Orders. The table products contains information about the products (such as name, description and quantityInStock) with productID as its primary key. The table orders contains customer's orders (customerID, dateOrdered, dateRequired and status). Again, we cannot store the items ordered inside the Orders table, as we do not know how many columns to reserve for the items. We also cannot store the order information in the Products table.

To support many-to-many relationship, we need to create a third table (known as a *junction table*), say OrderDetails (or OrderLines), where each row represents an item of a particular order. For the OrderDetails table, the primary key consists of two columns: orderID and productID, that uniquely identify each row. The columns orderID and productID in OrderDetails table are used to reference Orders and Products tables, hence, they are also the foreign keys in the OrderDetails table.



The many-to-many relationship is, in fact, implemented as two one-to-many relationships, with the introduction of the junction table.

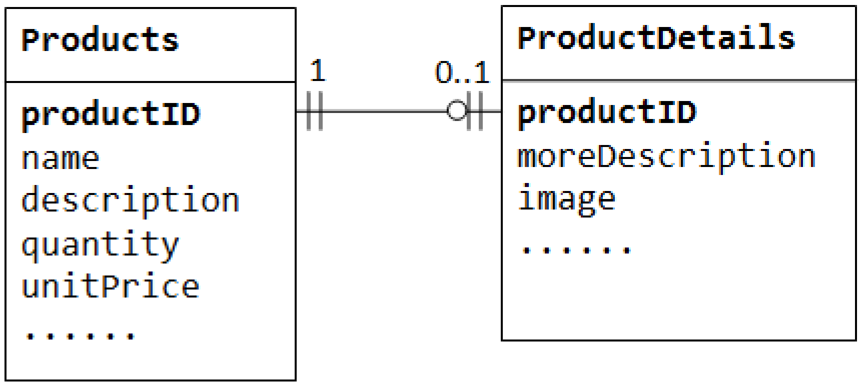
1. An order has many items in OrderDetails. An OrderDetails item belongs to one particular order.

2. A product may appears in many OrderDetails. Each OrderDetails item specified one product.

One-to-One

In a "product sales" database, a product may have optional supplementary information such as image, moreDescription and comment. Keeping them inside the Products table results in many empty spaces (in those records without these optional data). Furthermore, these large data may degrade the performance of the database.
Instead, we can create another table (say ProductDetails, ProductLines or ProductExtras) to store the optional data. A record will only be created for those products with optional data. The two tables, Products and ProductDetails, exhibit a *one-to-one relationship*. That is, for every row in the parent table, there is at most one row (possibly zero) in the child table. The same column productID should be used as the primary key for both tables.

Some databases limit the number of columns that can be created inside a table. You could use a one-to-one relationship to split the data into two tables. One-to-one relationship is also useful for storing certain sensitive data in a secure table, while the non-sensitive ones in the main table.



It seems we never use one-to-one relationship, because we can always put everything into one single table instead of using two tables and at the same time building a one-to-one relationship.

https://stackoverflow.com/questions/517417/is-there-ever-a-time-where-using-a-database-11-relationship-makes-sense

A deep understanding on the difference between one-to-one and one-to-many.
https://stackoverflow.com/questions/4751923/difference-between-one-to-one-and-one-to-many-relationship-in-database

If two tables are many-to-many, it will be difficult to create a “foreign key” to connect these two tables.

Table 1: Students

Record	**<u>StudentID</u>**	**StudentName**	**Major**
1	S1	Joe	Finance
2	S2	Amy	MIS
3	S3	Tim	Accounting
4	S4	Chen	Management
5	S5	Kumar	Marketing
6	S6	Carla	MIS
7	S7	Shawn	Accounting

Table 2: Courses

Record	**<u>CourseID</u>**	**CourseName**
1	FIN 101	Finance fundamentals
2	OPIM 203	Business info systems
3	MKTG 201	Marketing management
4	ACCT 202	Financial Accounting
5	MGMT 204	Managing in organizations

If we add a constraint that one student can choose at most one course, then the relationship between Course and Student will be one to many. In this case, we can create a foreign key on the "many side"