# Databases, The Introduction

*"I have traveled the length and breadth of this country and talked with the best people, and I can assure you that data processing is a fad that won't last out the year."* —The editor in charge of business books for Prentice Hall, 1957

## What is a database?

Pretty much anything that stores/organizes data is a 'database' in some sense. WordNet[1] defines a database as: *an organized body of related information*.

Thus, the most important detail that we're concerned with is "organized" or more precisely, *structured* data.

For example: a book stores data, a web-page stores data, etc., but they don't have well defined structure. Databases (as we use the term) store data according to the defined *metadata*—which is the *structure* of the database.

The difference is that databases are not defined by their data, but by the structure. For example, a book without text is pointless, but a database with no data is still a database; it has *table structure* defined, even though those tables may not contain any data.

## What is data?

Well, we've defined what is a database. But what exactly are we storing in this database? As the name implies, we are storing *data*. Data (by itself) is a set of *entities*. This is really a broad definition—anything is an entity. Considering object oriented perspective, you can also think of entities as *objects*.

Entities generally have various *attributes* (also commonly known as *properties*). These are what differentiages one entity from another. Attributes have values (a person may have an 'age' attribute). Attributes may also reference other entities (a person may have a 'parent' attribute that points to another person).

## What is information?

Ignoring the dictionary definition, 'information' is not a synonym of 'data'. In fact, you can have gigabytes of data, yet have no information. Just consider a random sequence.

The whole purpose of databases (and computers in general) is to work with information. Often times, computers are used to sift through lots of data to extract information. Generally, data has little or no value. Information on the other hand can be priceless.

[1]WordNet is a machine-readable lexical database organized by meanings (i.e.: a dictionary); developed at Princeton University.

## Broad Field

While we will be mostly concerned with stand-alone databases, many programs have relatively sophisticated data storage and processing capabilities. Just consider how much data a typical computer game handles—without some organization these things wouldn't be possible. So to paraphrase a quote I've seen once: *computer games are just databases with pretty interfaces*.

## Tuples

A *tuple* is structured data—a row in a table. It is something that we store in a database as a single 'record'. Considering the object oriented perspective, these are *instances* of entity objects.

## Tables

Tables—sometimes known as *relations*—are basically just lists of *tuples*. To define a table, we need: table name, table fields, and a primary key.

### Name

The *name* of the table is quite easy. It it usually a single word name of some entity. For example `Person` may be the name of table that has 'person' records.

There is a bit of a confusion as to whether table names should be singular or plural. ie: should you name a table `User` or `Users`. It is upto you to decide which way is best.

### Fields/Attributes

Every table needs to have one or more fields (also known as attributes). A field needs to have a name, and more often than not, a *type* as well.

The name should be picked with great care: it should be relatively descriptive, and appropriate to the type of information it represents. For example, using `FIRSTNAME` to represent 'first name'.

A field 'type' is the data format that the field holds. For example, an `INT` field type would hold integer values, while a `CHAR` field type would host character data.

Fields often have many other parameters, like comments, indicators if the field is a primary key, default values, valid/invalid values, etc.

## Primary Key

A primary key is one or more fields that uniquely identify each individual row (record, tuple, etc.) in the database.

For example, a person's SS# (social security number) uniquely identifies each individual, etc.[2]

## Relational Databases

The idea of *relational* databases isn't really anything special. It just means that our data is stored as several tables. For example, we can have a table with *company* information, and another table with *employee* information. Those two tables may be related to each other (details to be discussed later) to indicate which employee works for each company.

## Operations on Tables

Once we have tables, what can we do with them? Well, there are five basic operations that can be performed: *select*, *project*, *union*, *difference*, and *product*. The following describes each one in some detail.

### Select

The *select* operation retrieves data from a table on some condition. For example, we can imagine a statement that says: *"select all persons whose name is John Doe"*. Note that this example is a bit trivialized.

We can use various operations in conditions, for example: *"select all persons whose first name is John and who are 21 or older"*.

### Project

Projection operation extracts one or more columns from a table. For example, the `Person` table may have 20 various fields, but the *"project first name and last name from Person table"* only gets us first names and last names.

The result has just as many records as went in; just with possibly fewer columns.

One important concept is that we can combine operations, so for example, we can have *"project first name and last name from (select all persons whose first name is John and who are 21 or older)"*.

### Union

Union just combines all records of two tables. For this to work, the two tables must have the same number of columns (of matching data type).

### Difference

The difference is similar to a *Union*, except it has the effect of *removing* records. For example: *"Table1 - Table2"* would result in all records that are in *Table1* but are *not* in *Table2*.

[2]As we'll discuss later, it is usually a very bad idea to use SS# as a primary key.

## Product

The product operation (or $\times$) is a Cartesian product of two tables. For example, if we have a table $A = (a, b, c)$, and table $B = (d, e, f)$, the *product* would be:

$$A \times B = \{(a,d),(a,e),(a,f),(b,d),(b,e),(b,f),(c,d),(c,e),(c,f)\}$$

Notice that $A$ and $B$ both have 3 elements, and that the product has 9.

## Join

While *join* isn't exactly a 'basic' table operation[3] it is useful enough to be considered on its own.

Join (written $\bowtie$) is very much like the *product* operation, except often it doesn't produce all possible pairs. When the fields happen to have the same name, the *join* operation tries to match on those fields. When the join is only on the equality condition, then it is called a *natural* join.

Note that we can also specify our own conditions for join. For example, using $A$ and $B$ as defined above, the: "$A \bowtie B$ where both are vowels", will produce just $\{(a, e)\}$.

# Structured Query Language

The table operations discussed so far are mostly for abstract thinking about databases. For every-day database manipulation, most databases have settled on a standard languaged called SQL, for Structured Query Language.

There are two primary parts to SQL: The DDL and DML.

## DDL - Data Definition Language

This is a standard subset of SQL that is used to define tables (database structure), and other metadata related things. The few basic commands include: `CREATE DATABASE`, `CREATE TABLE`, `DROP TABLE`, and `ALTER TABLE`.

There are many other statements, but those are the ones most commonly used.

## DML - Data Manipulation Language

This is a standard subset of SQL that is used for data manipulation. Intuitively, we need to first inset data into the database. Once it's there, we can retrieve it, modify it, and delete it. These directly correspond to: `INSERT`, `SELECT`, `UPDATE`, and `DELETE` statements.

[3]Join can be defined using the five basic operations defined above.

## Flavors of SQL

Basically every database implements its own version of SQL. The basic statements (like the ones listed above) are almost always there and are almost always exactly the same on all databases.

Extensions come in when you're trying to do something fancy, like create stored procedures, or work with a specific database environment. Microsoft SQL Server use a variant called T-SQL, for Transact-SQL. To contract, Oracle uses PL/SQL, for Procedural Language SQL.