

## CHAPTER 2

### DATABASES : ACCESS TO STRUCTURED DATA

This section of the course is meant to make students familiar with the environment in which databases are designed and developed, and an appreciation of the problems associated with the area. A proper DBMS course would take an entire twelve week module or more, and we have less than two hours, so all I expect to cover is an overview and at the end of this I hope that those who have not covered databases formally in the past, will be able to gauge what a database can provide and what it cannot do. For the rest of you, it's a re-run of, for the most part, familiar materials. We discuss relational databases in this chapter, alternatives do exist such as object oriented databases and XML databases.

What is a database ?

Well, it is not just data stored in Access or especially Access itself. Although Access is a DBMS<sup>1</sup>, it is not considered very powerful and would (should) not be used to support any large operations. That said, we can describe a DBMS as:

*A software package that runs on a mainframe, server, desktop, or mobile device which at least provides fast access to structured data.*

... and it is these DBMSs that create and manipulate databases, which themselves are usually a single disk file, or group of files. This is an important difference and one that is often overlooked.

But DBMSs also do much more than this, as we will see... Popular DBMSs include ORACLE (the big one), DB2, SQL Server, MySQL, etc... The second largest software company in the world is a database company. Databases (managed by the DBMS) support information management by supporting the following operations on data:

- **Insertion** of new data
- **Deletion** of existing data
- **Modification** of existing data
- **Retrieval** of existing data, where this retrieval is usually based on some filtering arguments (e.g. retrieval of suppliers where their location is Dublin, ... , ...)

<sup>1</sup> DBMS : DataBase Management System... the name given to database software.

Why use a database in the first place... what do they give us? Could we not just use something else, a disk based file structure for example? Well, no, for a number of reasons:

- **Scale** – dbms's have been carefully written and highly optimised, if a database can be configured to manage a small amount/volume of data effectively and efficiently, then it is usually trivial to zoom up to huge data volumes<sup>2</sup> – need to have a minimal volume of data anyway in order to justify the overhead;
- **Speed** – databases provide fast access, often the fastest possible access, to data for certain types of retrieval requests. Databases do query optimisation which means they can do clever things to improve speed of access. We will briefly look at indexes later;
- **Flexibility** – databases can be used to query underlying data in an enormous number of possible ways – not all these are efficient, but all can be executed... i.e. the sequence of instructions in a database query can have huge impact on query performance;
- **Concurrency** – databases control concurrent use (simultaneous read and write) by large numbers of users, without application developers having to worry about such problems;
- **Backup and recovery** – data can be replicated or backed up and automatically regenerated or recovered in the event of catastrophe or minor errors;
- **Transaction Management** – sometimes we want to do a single logical operation which actually involves more than one physical operation<sup>3</sup> and databases can regard a sequence of separate physical operations as one atomic event... either they all happen, or none happen. This adds robustness in the event of failure in mid-execution or offers the possibility of controlled rollback<sup>4</sup>;

<sup>2</sup> The largest I have ever used is about 4 billion entries in one database table (SQL Server 2000) and it still supported *ms* access times. Most databases will never reach this size.

<sup>3</sup> E.g. adding a new supplier to a database for a store may require a number of different parts of the database to be written to in sequence... for example, the contact details in the contacts section and the details of any products of that supplier in a different section.

<sup>4</sup> Commit and Rollback : If a transaction has completed successfully then it is committed to disk (i.e. transaction completed and the database has moved from one state of integrity to another state of integrity). Otherwise a rollback occurs and the database remains in the state of integrity that existed before the transaction was attempted.

- **Data Integrity** can be maintained and a database can be prevented from having its data resources going into an invalid or impossible status... For example, a user inadvertently types in an incorrect supplier ID when inserting a record of a inbound shipment... the result, a shipment by a supplier who has no contact information... constraints within the database can avoid such scenarios... Or a half completed update of database data (perhaps due to a power-cut) could leave the database in an invalid state if it wasn't for the transaction rollback facility.
- **Distributed databases** – databases can support distributed processing and that's growing in importance;

So, tying all this together, we can say that using a DBMS makes the development of applications / eCommerce ventures faster, less prone to errors, and more flexible. DBMS's exist so why develop your own, which will likely be inferior anyway. The downside, is that a database costs resources (CPU, memory, disk) and sometimes money as well.

So, what can be stored in a database... well anything really, even binary data such as images or even video, usually ascii data/text<sup>5</sup>. Each database vendor will support different types of underlying datatypes, so here's an example of commonly expected datatypes.

#### Text

- Char
- Varchar<sup>6</sup>
- Text

#### Number

- Float
- Integers {int, smallint, tinyint}
- Others {decimal, smallmoney, money, real, numeric}

#### Other

- Binary {binary, varbinary}
- Bit
- Time {smalldatetime, datetime, timestamp}
- image

<sup>5</sup> If an online store (or similar) you will usually store information about objects and dynamically generate the web page, instead of storing whole, pre-generated HTML pages in the database.

<sup>6</sup> Varchar & Char... char is fixed length, varchar is variable length up to a given maximum. Unused space in a char datatype is padded.

---

## DATABASE DESIGN

---

An indirect advantage of using a DBMS is that it helps/forces you to formalise vague and imprecise information organisation by forcing the development of a database schema which is a map of data organisation<sup>7</sup>. That forces us to have to think about what we're doing and what we want from our data. The primary database design mechanism (and underlying DBMS view of data) is governed by a model of database theory called the Relational Model. When we think of databases we naturally assume the relational model and most people only know the relational model so it is essentially what a database is for most people.

## RELATIONAL MODEL

The relational model is the *de facto* standard<sup>8</sup> model for building databases even though it has been around forever (60s). What it supports is the analysis and construction of data in a database in a manner which helps to avoid incorrect data in a database.

The important concepts in this are:

- **Table** – a structured repository of data of a specific type. The table is the underlying data storage device in a relational database. Important here is the rule that the data stored in a table is all of the same type, e.g. a table of suppliers will not contain details of orders. A database will contain many tables.
- **Tuple / row** – the data of a specific type stored in a table. For example, a tuple of a suppliers table will store information about one particular supplier... if there are five suppliers, then there are five tuples. There must be no significance in the order of tuples.
- **Column / attribute** – Rows are composed of columns or attributes. Each column contains a single unit of data, stored in one of the supported datatypes. For example, any row of the suppliers table may contain supplier name, address and phone number, each as columns<sup>9</sup>. There must be no significance in the order of columns (attributes) and each column may only have one value per row.

<sup>7</sup> Well, it is possible to develop databases using just one table and storing everything in this one table... but this is not done... database designers normally plan their database structure, at least to a minimum level.

<sup>8</sup> An alternative is Object Oriented databases, but these are not very popular yet.

<sup>9</sup> Actually it is very important to break data up into columns correctly and address should NEVER be a column... address should be something like (street, town, city,...) This gives you the ability to do more advanced filtering of your query, e.g. suppliers in Dublin, grouped by town...

- **Primary** key – keys, we will get back to these shortly... they are the unique identifiers of each column.
- **Foreign** key – related to the concept of a primary key...
- **Referential integrity** – related to both primary and foreign keys... Tables can have references between data in them. E.g. an orders table will contain the id of (or any unique reference to) the product that the order is for. Hence, if an order is for Product ID 12, then in the orders table, a column called (perhaps) prodID will have an entry 12 for that particular order. Referential integrity ensures that if an entry in one table references another, then this entry must reference an existing entry in the target table... e.g. I can not have an order for a product that does not exist in the product table.
- **Indexes and fast access** – speed is a vital aspect of any DBMS. Data must be retrieved as efficiently as possible. A slow DBMS is unacceptable and in general you will not get a slow DBMS. Any performance issues are to do with the hardware or the actual database, not the DBMS. A badly designed or badly maintained database may be slow, but this is the fault of the database administrator<sup>10</sup>. Indexes provide fast access to data (at the cost of disk space efficiency). Many different types exist, beyond the scope of this overview, but their task is to ensure that the database is structured in such a way as to support fast retrieval performance.
- **SQL** – pronounced sequel, means Structured Query Language and is just that... a query language for structured data in a database. Simple and efficient, it has a number of key benefits. Firstly, it is not proprietary, but every relational DBMS vendor supports SQL (and have their own extensions). Secondly it allows for performing complex and sophisticated database operations... and finally it is fairly easy to understand and use. The basic SQL commands (called statements) are Insert, Select, Delete and Update. There are many other commands, mainly for database maintenance and schema modification, but these four are the most commonly seen. Lets see some example SQL statements (overheads).
- **ER models** – the logical design of a database, called the database schema. Designing using the ER model support the efficient and correct design of database schema. You begin building a new database using ER modelling to achieve the correct structure for your data and then the ER model is converted into tables and relationships. ER models have the following properties:
  - An entity class or set is a group of objects of the same type.
  - An entity is an instance of a physical object (entity set) in the real world.
  - An entity has properties or attributes to describe its characteristics.

- Entities can be associated via relationships where there are relationship classes and relationship instances, and each class can have properties.
- Entities become tables and relationships become tables.
- **Joins** – the ability to join two or more tables together to create a short-lived (life of the query) temporary virtual table for the purpose of complex SQL queries. For example, a database has a table for suppliers (address etc...) and one for products sold (type, cost, etc...). A join will allow, for example, a query for all products sold by suppliers in Dublin that cost less than €10.
- **Views** – a virtual table based on an underlying table or number of tables (or subset of both). Views are often used to limit access for certain users to certain data... instead of giving them access to the whole table (incl. salary details for example) you can give them access to a view, which may contain all employee information except the salary details.

#### INTEGRITY OF A DATABASE

It is essential that the database remains in a state of integrity. Keys provide an important mechanism for maintaining database integrity. Every tuple in a table should have an attribute, or group of attributes that uniquely identifies it. Although, it is strictly not essential to select a row as a key row in most DBMSs it is good practice to do so and helps in ensuring that the database schema is acceptably designed.

- A **Key** is a unique identifier of any row in a table... e.g. phone number, SSN,...
- A **Candidate Key** is an attribute or combination of attributes which is a unique identifier within a given table.
- One candidate key is chosen as the primary key and the others are alternate keys.
- **Primary keys** provide the sole guaranteed row-level addressing mechanism and are a fundamental part of the relational model.
- A **Foreign key** is a (combination of) attribute(s) in one table whose values are required to match those of the primary key of another table.
- Foreign keys are not necessarily part of the primary key and foreign-to-primary matches represent references.
- A **Compound key** is a concatenated key... a concatenation of attributes is required for uniqueness of the key... e.g. first name & last name and age
- A **Simple key** is a key that is not a compound key, if a single row is sufficient to identify a row.

A relational database will implement rules in order to ensure that the database remains in a state of integrity. In the relational model there are 2 integrity rules:

<sup>10</sup> DataBase Administrator (DBA), a privileged user of the database that may make schema modifications (add new tables, change integrity constraints, add new users to the database...) and perform other administrator tasks.

- **Entity Integrity:** No attribute forming part of the primary key of a base table is allowed to have NULL<sup>11</sup> values.
- **Referential Integrity:** If a table T2 includes a foreign key FK matching the primary key PK of some base table T1, then every value of FK in T2 must:
  - be equal to the value of the PK in some tuple of T1; or
  - be wholly NULL, i.e. each attribute in that FK must be NULL.

#### A SHORT NOTE ON INDEXES

They are NOT created by default and must be created by a privileged database user such as the administrator. Similar to the back of a book index, they maintain a sorted list to support fast retrieval. As a default, tables are sorted by the primary key, which allows for fast and efficient retrieval based on the primary key. Searching for other columns is usually not as efficient (if you do not have an index specified for these columns)... e.g. if you want to retrieve all suppliers in a certain city, the DBMS must read each supplier sequentially and extract only the suppliers with the correct city... Hence, the use of indexes to generate a sorted list<sup>12</sup> over a column, or group of columns.

Once an index is defined for a column, the DBMS searches the index to locate the required data and retrieves only those specific tuples (rows)... i.e. it does not have to read each row to locate the data, it knows where to look.

Great, so lets make indexes for every column and group of columns! Well not exactly, while they aid retrieval performance (immensely for large tables) you should remember that:

- Indexes degrade performance of data insertion, modification and deletion.
- Indexes are memory and disk space hungry.
- Not all data is suitable for indexing, data must be sufficiently unique.

Most DBMSs provide tools or wizards to help tune indexes.

#### A REALLY SHORT NOTE ON NORMALISATION

One of those words that people in computer science often fear... but all it is is a formalism of simple ideas with a practical application in logical database schema design... for the ER model. Essentially the process of Normalisation ensures that tables do not contain repeating groups/data (e.g. a persons address entered a

<sup>11</sup> A NULL value is a value of nothing (e.g. no name entered for a person).

<sup>12</sup> In reality, this will usually be a tree of some sort, but we can assume a sorted list for simplicity.

number of times in a number of different tables in a single database, leads to likely errors if one address is updated and others not).

No repetition means less chance of erroneous data being entered. There are many stages of the normalisation process, called normal forms... they extend from unnormalised form, through 1<sup>st</sup>, 2<sup>nd</sup>, 3<sup>rd</sup>, ..., 7<sup>th</sup>. Unnormalised form is basically a collection of data that will need to be stored in the database.

Tables in relational databases are always normalised as underlying domains (cells) contain atomic values only (1NF). 3<sup>rd</sup> (3NF) is the basic goal, because in this form, tables do not contain repeating groups.

Essentially, normalisation theory should allow us to recognise relations with undesirable properties, tell us what is "wrong" and how to "correct" it. So to recap why Normalisation is a good thing:

- Helps to keep data accurate by reducing redundancy (duplication). Duplication may lead to errors as we will see.
- Saves space by limiting the amount of redundant information stored. It is obvious that storing data many times is less desirable than storing data only once.
- Reducing redundancy allows for faster processing of data and having to update data once is faster than many times.
- Provides a framework within which design of Relational Databases should be structured.

So, a really quick example... overheads.

#### A EVEN SHORTER NOTE ON ODBC

ODBC, Open DataBase Connectivity is a standard that is used to enable applications to interact with different back-end databases, without having to worry specifically about how each operates or which form of SQL they use<sup>13</sup>. For example, the same code written could interact (assuming use of ODBC) with SQL Server, DB2, and Oracle databases without the programmer having to examine how each works. So what is ODBC?

It is a wrapper around databases that makes all databases operate in a clearly defined and consistent fashion. It accomplishes this using software drivers. The developer interacts with ODBC which does all the required translation for various backend databases and hides this from the programmer.

<sup>13</sup> Remember on page 3 we stated that different vendors support different data types, well different vendors can also implement different SQL standards. Luckily there is one core set of standards that all Relational Databases (at a minimum) adheres to.

---

**DATABASES IN ACTION**


---

Here's another example, apart from the others in the overgeads. Tables are suppliers, products and shipments.

Perhaps the obvious option is to have one table to store each shipment of products.

## Shipments

S-Name	P-Name	Volume

But this is wrong... It will work, but look for the problems. The relational model gives us the power to address these issues and build an optimal database schema. What would be more correct is.

## Suppliers

S#	S-Name

## Products

P#	P-Name

## Shipments

S#	P#	Volume

Suppliers are ... DELL, Compaq, IBM, HP, Sony

Products are ... Desktop, Notebook, Server, Mainframe

Shipments are ...

What columns would we create indexes on ?

What kinds of questions can we ask, apart from the obvious table lookup?

What volume of shipments come from supplier number X	SELECT SUM(VOLUME) FROM SHIPMENTS WHERE S#=X;
--	---

How many shipments come from supplied number X	SELECT COUNT(*) FROM SHIPMENTS WHERE S#=X;
How many shipments come from any suppliers in IRELAND	SELECT COUNT(*) FROM SHIPMENTS, SUPPLIERS WHERE SHIPMENTS.S#=SUPPLIERS.S# AND SUPPLIERS.COUNTRY = "IRELAND";