Database Systems

Introduction	The Purpose & Architecture of Database Management Systems
Data Modelling	Creating a computer model of a real world activity
	The Entity-Relationship and the Extended ER Approach
Relational DBs	Designing a Relational Database using EER
	Data access in RDB's - SQL, Relational Algebra
	Ensuring a Good Database using Normalisation
DBMS Facilities	Storage Structures and the User Interface
	Query Optimisation, Concurrency, Security, Integrity, Recovery
	Distributed Databases
Other Classical DBMS	Network and Hierarchical DBMS
Advanced DBs	Applications built on top of Databases - using Java
	Objects and Databases
	Databases and the Web - E-commerce
	XML and Databases

Core Databases

Internet Technology

What is a Database?

A database is a **structured** collection of **long-lived** and **related data** which represents a body of information

These may be held in paper, on a set of punched cards for instance, but are usually held by computers

The structure helps in the task of locating information for retrieval and for change

Most databases held on a computer are managed by a **Database Management System** (DBMS)

What is a Database Management System?

A DBMS is a complex program which manages databases arising from different applications

It does this by reducing all data to a **common structure**

- For instance **tables**

There are two broad classes of DBMS

- **Personal DBMS** - used on PCs / Macs - single user/ few users - moderate data capacity
  - Examples – Ms Access, IBM Cloudscape, etc.
- **Enterprise DBMS** - require powerful hardware - many users - distributed - massive data capacity
  - Examples – MS SQL Server, IBM DB2, Oracle, MySQL, Postgres

What does a DBMS Provide?

The ability to describe data from a range of **different applications**

A range of relatively simple ways of **accessing the data**

Efficient access to large amounts of data

Maintenance of the **integrity** (i.e. correctness) of the data

Reliable data storage - i.e. the data will be preserved even after the system crashes

Security against data misuse

Concurrent access by multiple users without conflict

Distribution of data and/or users across a network of computers

Why Do We Need Database Systems?

All data processing needs could be met by **programming**.

For instance, the following is the structure of a typical DP task:

1. Define and possibly modify the structure of the data.
2. Enter, modify and remove data.
3. Retrieve the data in various ways.

Consider the data contained in a set of student records.

1. We define the structure, e.g. each student has a matriculation number and some exam results.
   Later we add some new exams.
2. Put in a set of students, change any errors, delete drop-outs.
3. Retrieve: the data for student X
   those with < 30% overall
   those with > 70% overall, etc

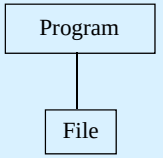
The Traditional Approach

Write programs for each purpose in some high-level programming language (e.g. Java).

The program contains:

- a description of the data structure;
- code to read data from files
- the implementation of any editing or querying operations that will be allowed
- if the data may change, code to write data back to files

The data is held in files, which may be edited by a text editor.



Simple Database Application in Java

```
public class Student
{   String name;
    int matric;
    int exam1;
    int exam2;

    public Student( FileIn theFile )
        {   name = theFile.readString();
            matric = theFile.readInt();
            etc.  }

    public void change()
        { exam11 = ... get new exam from user
    public void printIf()
        {  if (exam1+examp2 > 69)
                    write statements   }
    public void save( FileOut F )
        {  code to write the variables
        }
}
```

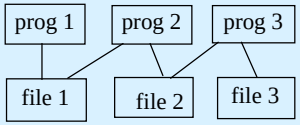
```
public class Database
{   Student[] theStudents = new Student[100];
    int Nstuds, i;

    public void main(String[] args )
    {   FileIn studFile = new FileIn( "stud.in" );
        Nstuds = 0;
        try  { for (;;)
        { Student[Nstuds++] = new Student(studFile); }
        catch (EofX e ) {}

        for (i=0; i<Nstudents; i++ )
                            Student[i].change();
        for (i=0; i<Nstudents; i++ )
                            Student[i].printIf();
        .....code to get an output file
        for (i=0; i<Nstudents; i++ )
                            Student[i].save();
    }
}
```


Weaknesses of this Approach

1. Extending the data structure means modifying the program.
2. It also means editing the data file or having a second file.
3. The operations provided are a fixed set. Extending that set means modifying the program.
4. The program is specific to the application - changing to another application means a new program to do essentially the same job.
5. What emerges is a spaghetti of files and programs:



More Weaknesses of this Approach

6. New facilities, such as duplicating data for security, must be individually added to all programs.

7. Files can be tampered with by others using an editor.

8. Programming is DIFFICULT!

Examples of Changing a Data Processing Program

Adding More Records (Students)

OK until we get to 100 - then we must edit the program.

Adding More Fields (say a Lab mark)

1. Add another variable to Student class - LAB say + read statements.
2. Edit the data file or create a second file.
3. Change program to use the amended file or two files.

Programming a New Application with the Same Operations

Change the variable names.

Simple Database Application in Java

```
public class Ship
{   String name;
    int reg;
    int length;
    int draught;

    public Ship( FileIn theFile )
        {   name = theFile.readString();
            reg = theFile.readInt();
            etc.  }

    public void change()
        {length = ... get new length from user
    public void print()
        {  if (length+draught > 69)
                    write statements   }
    public void save( FileOut F )
        {  code to write the variables
        }
}
```

Data Definition

Querying

```
public class Database
{   Ship[] theShips = new Ship[100];
    int Nships, i;

    public void main(String[] args )
    {   FileIn shipFile = new FileIn( "ship.in" );
        Nships= 0;
        try  { for (;;)
        { Ship[Nships++] = new Ship(shipFile ); }
        catch (EofX e ) {}

        for (i=0; i<Nships; i++ )
                            Ship[i].change();
        for (i=0; i<Nships; i++ )
                    Ship].printIf();
        .....code to get an output file
        for (i=0; i<Nships; i++ )
                    Ship[i].save();
    }
}
```

Data Manipulation

Some More Things a DBMS Provides

The DBMS holds not only the data, but a description of that data - the **meta-data** (a.k.a. the **schema** of the database) - there will be interfaces to define the schema

Programs and data are **independent** of each other:

- e.g. if a LAB mark is added, the program which retrieves matriculation numbers continues to work unchanged

The ability to define multiple **views** of the same data, e.g. just exam marks

The description of the data is at a **higher-level** than in a programming language

The details of the underlying **file organisation** may be hidden from the user

Yet More Things a DBMS Provides

The DBMS is the **only method of accessing the data** - no backdoor tampering

The DBMS limits of **redundancy**; saves file space; prevents inconsistency

The enforcement of **integrity constraints**, such as the matriculation number is six digits or 20 is the maximum number of a tutorial group

The DBMS will carry out **file access** automatically without the need to program it at all

The DBMS describes all of the schemata using a single descriptive device called the **logical data model**

- Thus the DBMS reduces all data from all databases to a common form

Even More Things a DBMS Provides

The DBMS will provide simple ways (such as **query languages**) of specifying the simple aspects of Task 3

- but still allow the user access to a **programming language** if there are more complicated computations to express
- Even in the latter case, however, the DBMS will control how the user coded program accesses the data

The DBMS will support the **facilities** such as efficiency, security and so on, for any database that it manages.

- These facilities, by being programmed as part of the DBMS implementation need only to be coded once to be available to all DBMS users

Constructing a Database Application I

Follow these steps (perhaps cyclically)

1. **Data Investigation (Fact Finding)**

   What kind of data is there and how is it to be used?

   (a) **The Basic Facts**

   A student has a name, a matric number, a faculty, etc.

   A student sits a set of marked exams

   A course is given by one or more lecturers.

   (b) **Constraints**

   A tutorial group is at most 20

   Those getting > 70% on a class exam are exempt from the final exam.

   (c) **Rules**

   When a set of requirements for a course are met, a student becomes eligible for that course.

Constructing a Database Application II

2. **Conceptual or Data Modelling**
   - Generalise the facts into concepts, e.g. notions of course, student, etc.
   - From these produce a model of the data which is independent of the DBMS.

3. **Database Design**
   - Map the Conceptual Model onto the specific style of the DBMS
   - Define which external views will be available

4. **Database Physical Design and Implementation**
   - Specify the internal model using the DDL of the DBMS
   - Select storage structures if known

Constructing a Database Application III

5. **Database Application Construction**
   - Building the application program which makes the database usable to end-users:
     - may use a programming language (COBOL, Visual Basic, Java)
     - may only need to use the in-built DBMS tools – e.g. Form Builders, Switchboard Manager
6. **Database Monitoring and Tuning**
   - Improve implementation in the light of experience

Database Interfaces/Languages

Divides into three parts for the three tasks:

1. **Data Definition**

   Specifying and modifying the schema

   This is a privileged task usually done by a Database Administrator (DBA)

   It is carried out using a **Data Definition Language** (DDL)

2. **Adding, Deleting and Changing Data**

   Uses a **Data Manipulation Language** (DML)

3. **Retrieving Data**

   Specified using a **Query Language** (QL)

Notes

The 'Languages' may be **graphical**

They should be **well-matched**, particularly the DML & QL

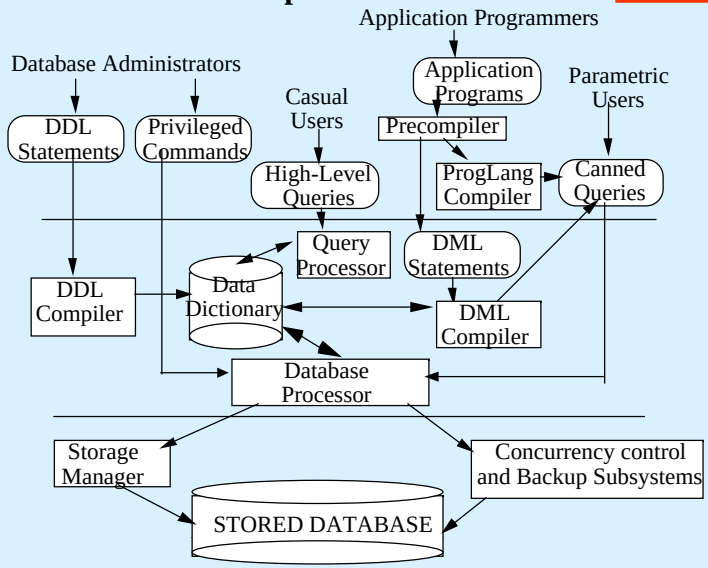
DBMS Users

A DBMS is intended to be usable by a wide range of users:

- A **database administrator** (DBA) is a database expert who will be responsible for setting up databases as well as monitoring and tuning performance
  - needs a powerful interface to change DBMS settings
- An **application programmer** uses a programming language interface to the DBMS to create applications for others to use
  - cannot change the settings of the DBMS, but can construct complex software on top of the DBMS.
- A **casual user** uses SQL or a form interface to carry out simple querying tasks.
  - does not have programming skills, but should know a bit about the logical database model and the metadata.
- A **parametric** or **naïve user** uses a pre-prepared or "canned" interface to invoke regularly performed tasks.
  - A **canned interface** requires the user only to invoke an operation and to provide parameters - e.g. cash point, library checkout.

DBMS Components and Users

Key Slide



The Layering Approach to Manage the Complexity

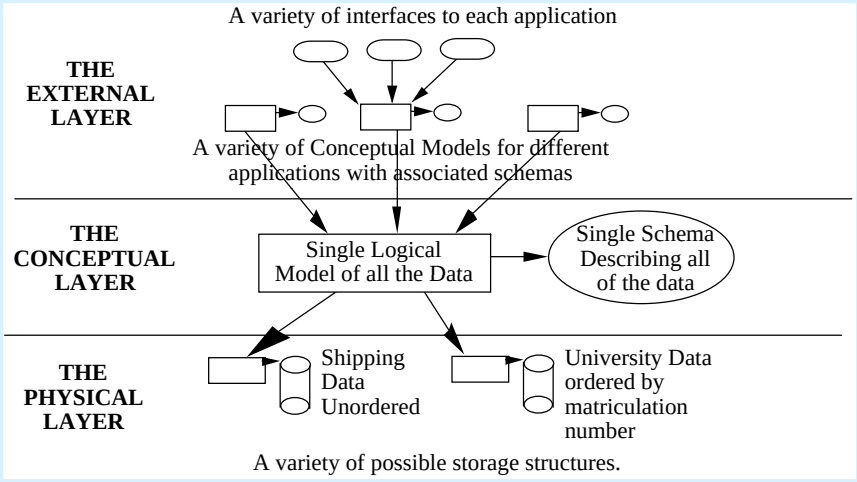
The consequence of the wide range of users and potential database applications is complexity

Controlling the complexity is managed by layering the data

- At the **lowest level**, efficient storage of data is vital to achieve speed
  - This might imply complicated and hard to understand storage structures
- At a **middle level**, all data must be reduced to the same form
  - or we would have to write a different program for each application
- At the **user level**, the interface must be able to provide a number of sufficient and manageable interfaces:
  - one for the DBA to determine the storage structures
  - another for programmers to construct programs
  - and several for end users appropriate to their skills and needs

The 3-Level Architecture 'ANSI-SPARC' Key Slide

This is designed to separate out the major issues of database maintenance so that they can be handled separately.



ANSI-SPARC - The External Level

is a collection of **external views**

includes facilities for the three tasks (**data definition**, **data manipulation**, **querying**)

each view may use a different **interface/language** and be at a different level of abstraction

requires a **mapping process** to the Conceptual level

the end user doesn't need to know what is happening underneath

ANSI-SPARC - The Conceptual Level

describes all data in a single uniform manner -

- the **logical model** of the data;

holds all data communally;

deals principally with the **logical structure**;

- thus provides **independence** from the actual manner of storage;

the **major facilities** (security, concurrency, etc.) are programmed against the logical model

- thus making them available to all databases

the language/model with which this level describes all data is the **distinguishing feature** between various kinds of DBMS.

- network, hierarchical, **relational**, object-oriented, object-relational
  - we will use relational DBMS

ANSI-SPARC - The Internal Level

is concerned with **physical storage**

controls which data is in which **files**

controls how data is **structured** in files (e.g. indexes, hashing, etc.)

allows storage to be chosen for **efficiency** rather than simplicity or intelligibility

must support a **mapping** from the logical model into files

Data Models

A **data model** is a language or a set of concepts that can be used to describe the structure of a database

- i.e. the data types relationships and constraints in the DB.

There are models at each of the three levels :

- **Low-level** or **Physical Data Models** describe how data is stored in the computer - files, storage structures, etc.
- **Implementation Data Models** describe data at the Conceptual Level - as relations and records,for instance
- **High-level** or **Conceptual** or **Semantic Data Models** describe data in a way closer to their real world meaning - as entity types, attributes and relationships

So it is possible to talk of several different schemata for the same relational database:

- the **ER schema** - a semantic view of the database
- the **Relational schema** - this describes the set of tables
- the **Physical schema** - this describes the file structure

Some Easily Confused Terms

Key Slide

A **schema** (plural **schemata**) is the description of a database in terms of the constructs provided by the data model

A **data model** is the language with which schemata are described.

- Note - the language could be graphical or textual

Meta-data is the information contained in the schemata - literally data about data

The **catalogue** or **data dictionary** is the set of files which hold the meta-data

Informal Introduction to Relational Databases

Key Slide

A relational database consists of a set of tables such as:

Person

name	address	age	worksFor
Jim	19, Jill St.	23	1
Jill	19, Jim St.	27	1

Company

number	name	business
1	X&Y	Medical
2	A&B	Trucking

The tables are **rectangular**

The **columns** have **names** and **types** (integer or string in this case)

The rows are **records** and are identified and connected by their content (**keys**).

For instance, the 1 in *worksFor* tells us that the two people work for "X&Y"

No two records in a table can have exactly the same data

The cells are **atomic** - i.e. all cells hold single values, no records or sets

There are **no other data structures**!

Examples of Keys

No duplication of numbers:

Numbers1 : relation (number : integer; word : string)

Numbers2

word	number
string	integer
"dozen"	12
"five"	5
"halfdozen"	6
"one"	1
"six"	6
"twelve"	12
"two"	2
"zero"	0

The keys

No duplication of names

Numbers1

number	word
integer	string
0	"zero"
1	"one"
2	"two"
3	"three"
4	"four"
5	"five"
6	"six"

Numbers2 : relation (word : string ; number : integer))

Designing and Creating Relational Databases

Getting the right set of tables is quite difficult for complex applications

Therefore, design is usually done using a higher level model

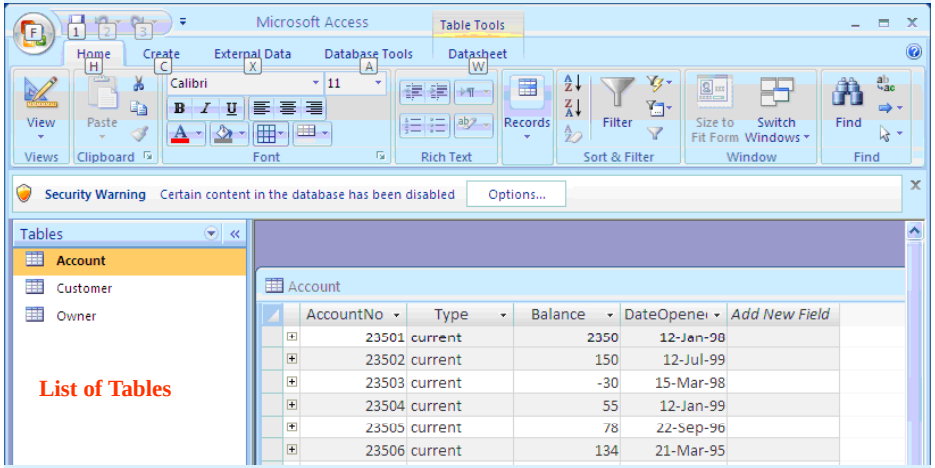
- the Entity Relationship model is the most important
- UML is of growing importance
- these model the things we want to store data about
- and can then be transformed into a set of tables

Tables are then created by specifying their names and the names and types of their columns

- this can be done using a GUI – e.g. Microsoft Access Design View
- or using SQL, e.g.

```
create table Person( name Text Primary Key,
                    address Text,
                    age Number )
```

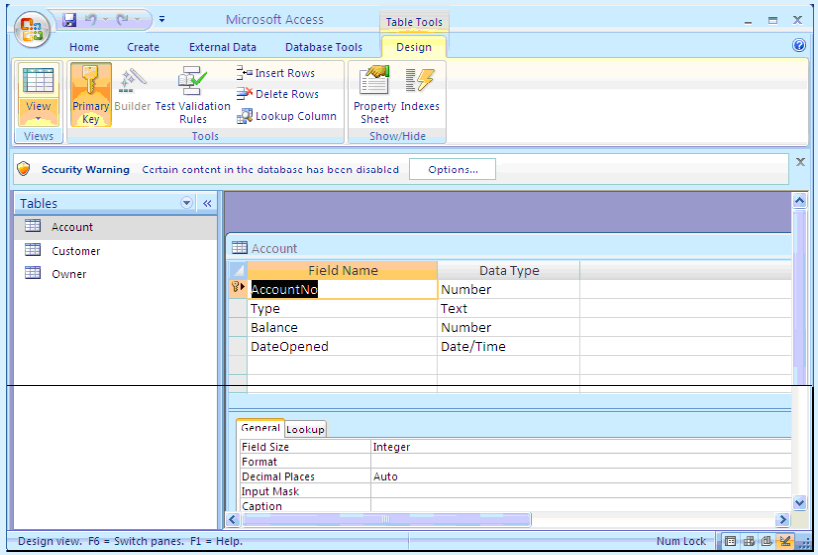

Microsoft Access 2007



List of Tables

Account Table in Datasheet View

Design View – Editing the Schema



The Objects Found in Access

Microsoft Access is a personal database which is to be used for the next lab

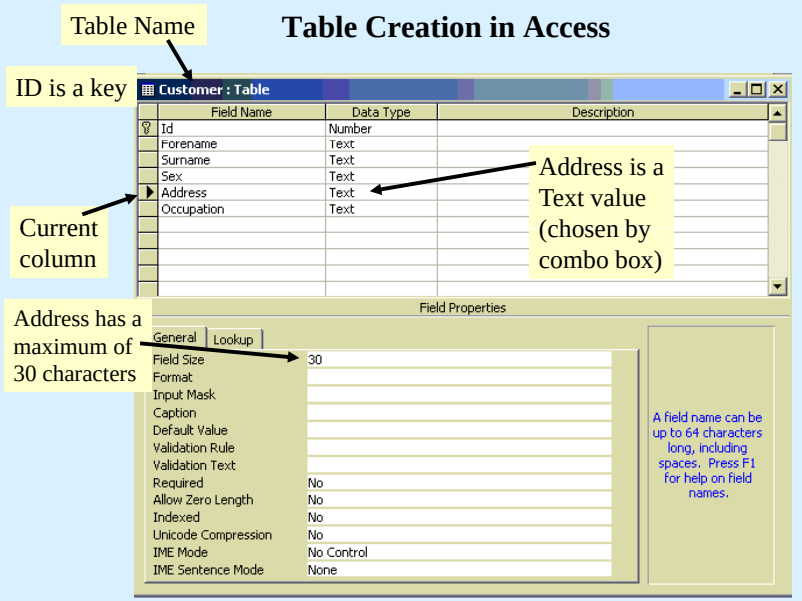
It manages the following kinds of object:

- Tables
- Queries – ways of generating tables as a response to a question
- Forms – mechanisms for displaying and entering data
- Reports
- Macros
- Modules

For each kind of object it supplies a number of views

- e.g. a table can be displayed as one row for each record (datasheet view) or as one row for each field description (design view)
- a query can be displayed as its resulting data, in SQL, or in a graphical form

Table Creation in Access

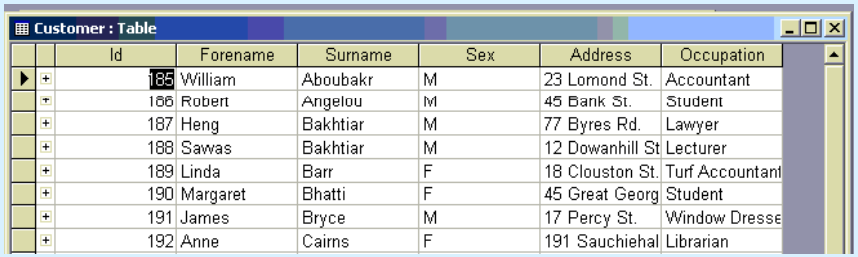


Managing Data

There are several ways that the data gets entered and updated in a relational database:

- importing from a file
- entry through an application interface
- entry through the use of SQL

  insert into Person **values** (“John”, “15, The Buildings”, 23)
- entry through a database facility – e.g. Access Datasheet View:



Id	Forename	Surname	Sex	Address	Occupation
185	William	Aboubakr	M	23 Lomond St.	Accountant
186	Robert	Angelou	M	45 Bank St.	Student
187	Heng	Bakhtiar	M	77 Byres Rd.	Lawyer
188	Sawas	Bakhtiar	M	12 Dowanhill St	Lecturer
189	Linda	Barr	F	18 Clouston St.	Turf Accountant
190	Margaret	Bhatti	F	45 Great Georg	Student
191	James	Bryce	M	17 Percy St.	Window Dresse
192	Anne	Cairns	F	191 Sauchiehal	Librarian

Relating Data in a Relational Database

Two records are related by a **foreign key**

- e.g. the primary key of one record is copied into a field of the other
  - example the key of *Company* is *number*
  - therefore company numbers are put into a column (*worksFor*) in the *Person* table

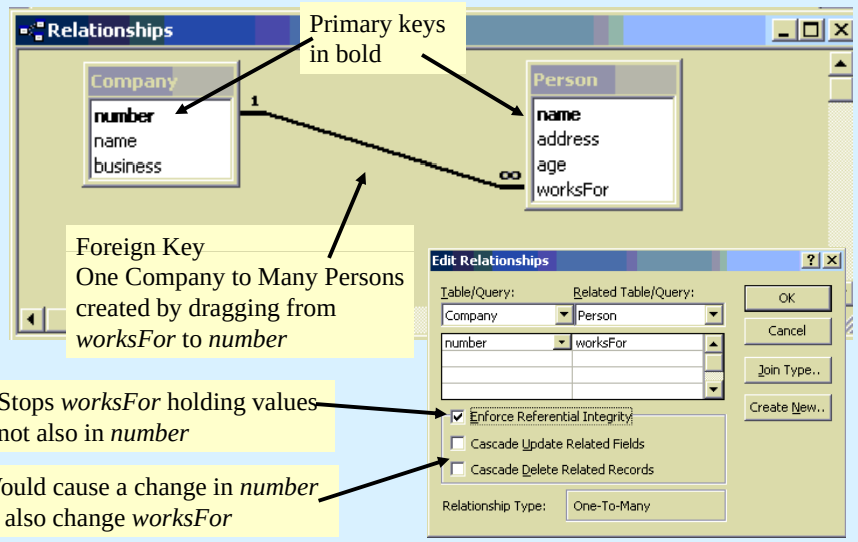
To specify a foreign key we again have a variety of mechanisms:

- in SQL this can be done in create table as in:

```
create table Person( name Text Primary Key,
                     address Text,
                     age Number
                     worksFor Number references Company(number) )
```

- in Access the Relationship View can be used - this button calls it 

Relationship View in Access



Choosing Foreign Keys

Where to put foreign keys depends on how many records are related

If a record is related to only one other record (e.g. *worksFor* in *Person*) you create a column as shown

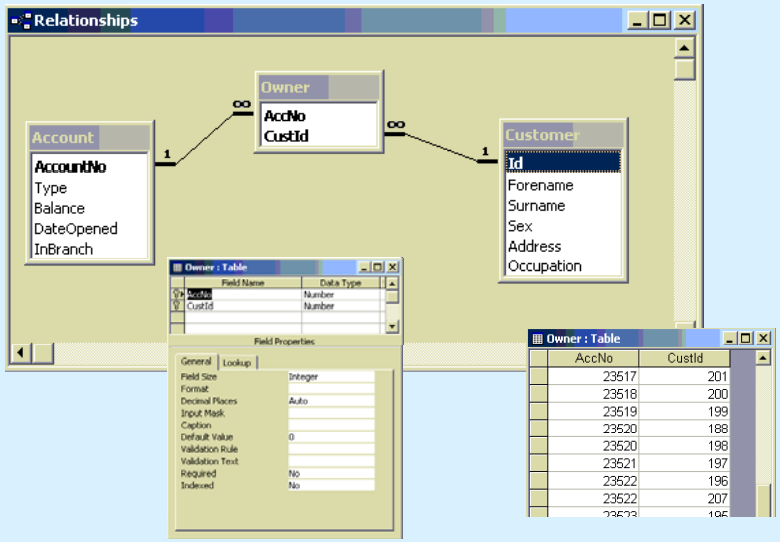
If a record is related to more than one other record, you create a link in the other record, (e.g. you don't put a link for *worksFor* in *Customer*)

If there are many records related in both directions, you must create a separate table (This is called a **many-to-many relationship**)

Example (from exercise)

- bank accounts may have many owners
- each owner may own many accounts
- so create a separate table Owner with
  - one foreign key column for each related table
  - the primary key is both columns jointly

Many-to-many Relationships in Access



Other Things Access Provides

In addition to setting up a database with tables, Access allows you to create and store:

- Queries – *ad hoc* queries which retrieve sections of the database
- Forms – for data entry and display
- Reports – for presenting summary data
- Pages – web pages automatically generated for you
- the Switchboard – which provides a start point for calling up forms, reports and so on and allows you to create a whole application
- macros – the use of Visual Basic to add functionality