**Data 1**

**Problems with Decentralized Files**

**Separation and Isolation of data**

Isolating data in separate files makes it difficult to cross-reference information.

**Duplication of data**

This is wasteful of both time and space

Uncontrolled duplication of data can occur

Great care must be taken to maintain the consistency of duplicated data

**Data dependence**

Making changes to the structure is difficult since everything are defined in the program code.

Many programs would need to be update

A 3rd party program must be created to convert the existing data to new format

**Incompatible file formats**

Departments might write their programs in different languages which stores information in a different format. This would make it difficult to share information.

**Range of operations available**

Electronic file-based systems were a major improvement over manual system. This led to an increase in the number and types of operations that users wished to perform on their data. However, electronic file-based system requires appropriate programs. Support for an operation requires a dedicated program to process the data files and insert or extract the information as required. Thus, either the available operations are limited to the supported basic set of programs, or a lot of programs must exist to satisfy the information requirements of the organization.

**The Database Approach**

The limitations of the file-based approach can be attributed to two factors:

* The data is used rather than being stored separately and independently
* There is no control over the access and manipulation of data

A more effective approach requires two new concepts:

* The Database
* The database Management System

A database is a shared collection of logically related data designed to meet the information needs of an organization.

**The Database**

A database is a single, large storage of data that is defined once and used simultaneously by many users.

Rather than having disconnected files with extra data, all data is complete with minimum duplications.

No data is owned by a singe department, all data is now a shared corporate resource.

A database hold not only an organization’s operational data, but also a description of that data.

* The description is known as the database schema or meta-data (the data about data)
* This gives independence between programs and data since the structure of the data is separated from the programs and stored in database.

**The Database Management System**

The Database Management System (DBMS) is the software that enables users to define, create and maintain the database and which provides controlled access to this database.

The DBMS comes between the users and the database.

First, the DBMS allows users to define the database, usually through a Data Definition Language (DDL).

* The DDL allows users to specify the data types and structures, and the limitations on the data to be stored in the database.

Second, the SBMS allows users to insert, update, delete, and retrieve data from the database, usually through a Data Manipulation Language (DML)

**The Database Environment**

A database system aims to provide users with an abstract view of data by hiding certain details of how data is stored and manipulated.

The starting point for the design of a database must me an abstract description of the information requirements of the organization. This will be in terms of entities, attributes and relationships.

**An Abstract View**

Abstract example:

* Entities: Staff, Property, Owner, and Renter (maybe others too)
* Attributes describing properties or qualities of each entity (e.g. Stuff have names, addresses etc.)
* Relationships between these entities (e.g. Owners own Properties)

Since database is a shared resource, we may also be concerned to provide different users with different views of the data held in the database.

**Data 2**

**The Relational Data Model**

Allows a high degree of data independence, user’s interactions with the database must not be affected by changes to the internal view of data.

Provides main grounds for dealing with the problems of data. (Normalized relations)

Enables the use of set-oriented data manipulation languages.

One of the most significant implementations of the relational model was System R which was developed by IBM during the late 1970’s.

System R was intended as a “proof of concept” to show that relational database systems could really be build and work efficiently.

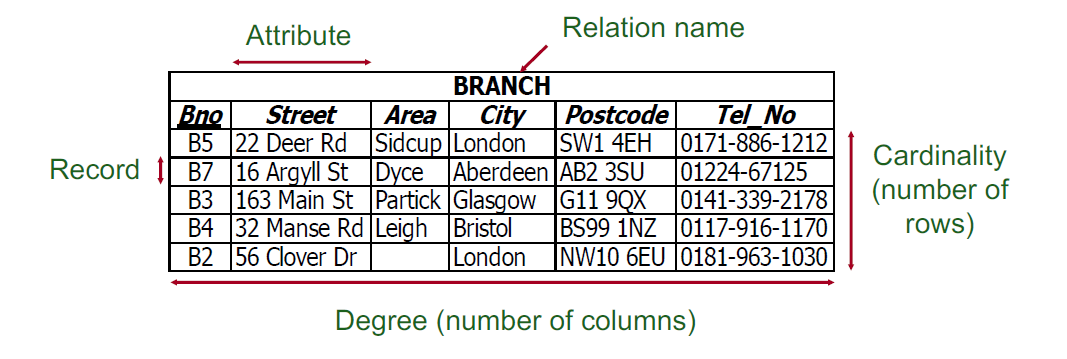
It gave rise to two major developments:

* A structured query language called SQL which has since become an ISO standard.
* The production of various commercial relational DBMS products during the 1980’s such as DB2, SQL/DS, and ORACLE.

There are now several hundred commercial relational database systems for both mainframes and microcomputers.

**Relations**

The relational model is based on the mathematical concept of a relation. A relation is represented as a two-dimensional table containing rows and columns. Relations are used to hold information about the entities to be represented in the database. The rows correspond to individual records. The columns correspond to attributes or fields. The order of the attributes is unimportant. They can appear in any order and the relation will remain the same.



**Properties of Relations -I**

The name of a relation is unique. No two relations may have the same name.

The name of an attribute is unique only within its relation. So, we can have two attributes called Name in separate relations, but not in the same relation.

The values of an attribute are all from the same domain. We should not allow a postcode to appear in a salary column for example.

The order of attributes within a relation has no significance. If we re-order the columns of a relation it does not become a different relation.

The order of rows within a relation has no significance. If we re-arrange the rows of a relation, it does not become a different relation.

Each cell of a relation should contain at most one value. For example, we cannot store two phone numbers in the same cell.

The records within a relation should all be distinct. No duplicates. Note that Microsoft Access and MySQL allow relations to contain duplicate records.

We need to be able to identify uniquely each row in a relation by the values of its attributes. We use relational keys for this purpose.

**DATA 3**

**The Entity – Relationship (ER) Model**

The ER Model is a high-level conceptual data model. It was originally developed in the 1970s to facilitate database design. An example of a basic ER diagram is shown below:



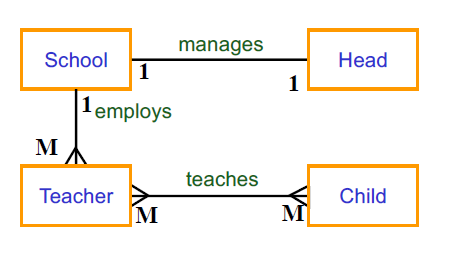
The entities Student and School are shown as rectangles.

The relationship “attends” is shown as a labelled connection between the entities.

We can read this diagram in two ways:

- One student “attends” one School (1:1 relationship from point of view of Student)

- One School “is attended by” one Student (1:1 relationship from point of view of School)



1:1 One to one e.g. **head** manages **school**

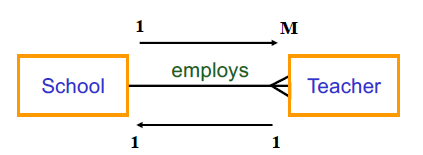
1:M One to many e.g. **school** employs **teacher**

M:M Many to many e.g. **teacher** teaches **child**

-One **head** manages only one **school** (1:1 from POV of head) and each **school** has only one **head** (1:1 from POV of school)

-One **school** employs many **teachers** (1:M from POV of school) but one **teacher** is employed by only one **school** (1:1 from POV of teacher)

-One **teacher** teaches many **children** (1:M from POV of teacher) and each **child** is taught by many **teachers** (1:M from POV of child) – hence this relationship is M:M overall (both POVs)

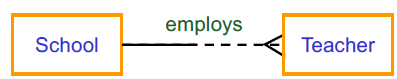


We define this as:

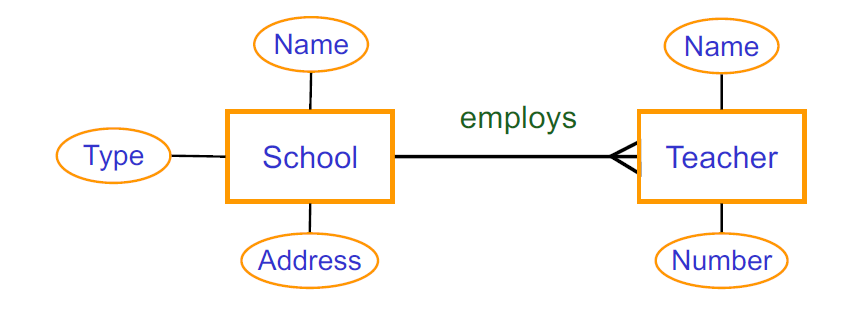
-One school employs one or more teachers. (1:M relationship from POV of one school)

-One teacher is employed by only one school. (1:1 relationship from POV of one teacher)

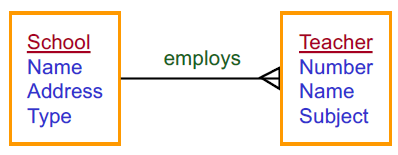
However, it may be that some teachers are not employed by any school. We say that the “employed by” relationship is optional from the POV of the teacher. That is a teacher may or may not be employed by a school.



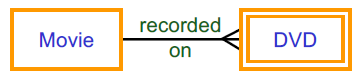
In this diagram the DASHED portion of the (half) line signifies that a teacher **may optionally be** employed by a school. However, from the POV of a school, it is vital that the school employs at least one teacher. Thus, schools **must** always employ teachers. This is indicated by the solid half of the connector attached to School above.



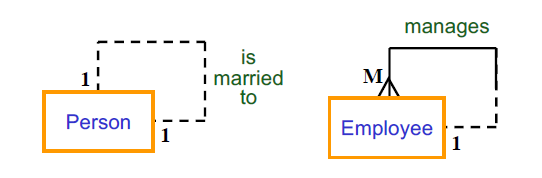
Name, Address, Type of School, Name and Number of Teacher are the attributes of each entity. An alternative notation for displaying attributes is shown below:



A weak entity is one which cannot exist without the existence of some other entity. For example, in a video hire shop, we may have the following ER diagram:



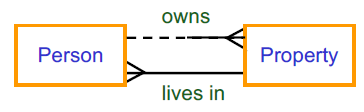
This represents the fact that a given movie must be recorded on one or more DVDs and given DVD must contain a film. So, we cannot have DVDs if there is no Movies. If we delete a movie from out database, we would also like to delete all the information about the DVDs containing that movie. Thus, DVD is a weak entity since it depends on a corresponding movie.



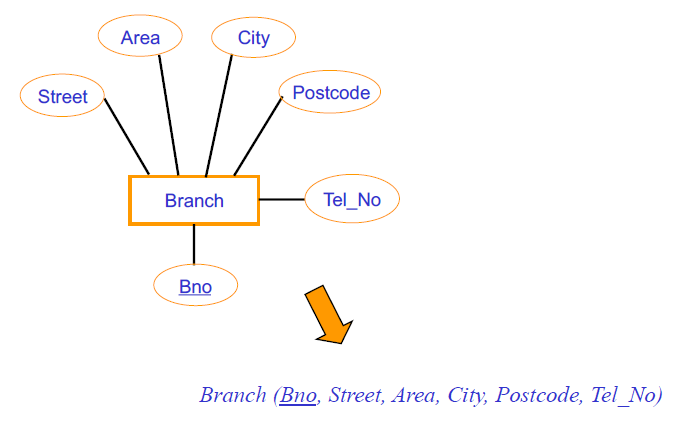
Left: A person is optionally married to one other person.

Right: An employee manages zero or more other employees. Each employee is managed by one other employee.

Two entities can be related to each for different reasons simultaneously.



Here we consider that owning a property is a separate concept from living in it. Thus, we represent this using separate relationships.

**Converting Entities into Relations**

**Superkeys (SQL)**

A superkey is any attribute or set of attributes that uniquely identifies a row within a relation. Using this definition and the BRANCH relation as an example, the following would qualify as superkeys:

-(Bno)

-(Bno, Street, Area)

-(Bno, Postcode, Tel\_No)

Any combination of attributes that contains Bno would be a superkey. Each superkey gives us a set of attributes that we could possibly use as a key for the BRANCH relation. The problem with superkeys is that they ay contain attributes that are not strictly required for unique identification. For example:

-(Bno, Street, Area) Street and Area are not required

-(Bno, Postcode, Tel\_No) Postcode and Tel\_No are not required

In this case the single attribute Bno is enough to identify any record. We are interested in the superkeys that contain only the attributes necessary for unique identification.

**Candidate Keys**

If a superkey does not contain any unnecessary attributes, we say that it is minimal, and it no longer provides unique identification.

A candidate key, K, for a relation R has the following properties:

* Uniqueness:
  + In each row of R, the values of K uniquely identify that row.
  + In other words: no two rows of R can have the same values for K
* Irreducibility:
  + No subset of K has the uniqueness property
  + Therefore, K cannot consist of fewer attributes

Some relations may have several candidate keys.

**Finding Candidate Keys**

We cannot examine a relation and decide, for example, that Postcode is a candidate key simply because no two rows share the same post code. The fact that there are no duplicates at ta moment in time does not guarantee that duplicates are not possible. However, the presence of duplicates may be used to show that a certain attribute is NOT a candidate key.

Therefore, to correctly identify a candidate key, we need to be aware of the meanings of the attributes in the real world and think about whether duplicates could arise for a given choice of key.

For each relation in the database, we must choose one of its candidate keys to be its primary key.

**Foreign Keys**

When an attribute appears in more than one relation, its appearance usually represents a relationship between records of the relations. When the primary key of one relation appears as an attribute in another relation it is called a foreign key.

**Entity Integrity Rule:**

In a relation, no attribute of a primary key can be null.

**Referential Integrity Rule:**

If a relation contains a foreign key, either the foreign key value must match the value of a candidate key of a record in the home relation, or the foreign key value must be wholly null.

**Introduction to SQL**

SQL is a declarative language for manipulating a relational database. Use it to issue commands to the database for tasks such as Creating and managing tables, inserting data into tables, searching for and retrieving data from tables, Deleting data and tables.

Names of database elements such as tables and field are enclosed in backward `quotes`.

Statements are separated by semi-colons;

**SELECT `name from `mytable` WHERE `name` = ‘John’**

**Creating a Table**

The simplest form of SQL CREATE TABLE looks like:

**CREATE TABLE IF NOT EXISTS tablename**

**(colname datatype,**

**. . . )**

**CREATE TABLE Staff**

**(Sno INT,**

**Sname VARCHAR (20),**

**Dept VARCHAR (20),**

**Grade VARCHAR (7))**

**Column constraints**

**[NOT NULL | NULL]**

**[DEFAULT default\_value]**

**[AUTO\_INCREMENT]**

**[UNIQUE [KEY] | [PRIMARY] KEY]**

**[COMMENT ‘string’]**

**PRIMARY KEY (only one per table)**

**FOREIGN KEY REFERENCES table (column)**

Eg. **CREATE TABLE Staff**

**(Sno INT PRIMARY KEY,**

**Sname VARCHAR (20), NOT NULL,**

**. . .)**

Eg.2 **CREATE TABLE Staff**

**( . . .**

**Dept VARCHAR (20) FOREIGN KEY REFERENCES Depts (Dname),**

**Grade VARCHAR (20) FOREIGN KEY REFERENCES Paytable**

**)**

**Data Integrity: Deadly Embrace**

Notice that we cannot declare the Staff table until we have declared Depts and Paytable. Also, we cannot insert any data into Staff until we have matching data in Depts and Paytable.

After the last field, we can add table-constraints. These look like column-constraints, but they can reference more than one column.

**CREATE TABLE HTR**

**(Hour char 6),**

**Teacher char (3),**

**Room char (4),**

**PRIMARY KEY (Hour, Teacher));**

This is how to declare composite primary keys. We can declare foreign keys in the same sort of way.

**Getting Data into Tables**

There are two ways of using SQL to get data into tables. Firstly, with the values in the SQL statement:

**INSERT INTO Staff VALUES (123, ‘Lee’, ‘CompSci’, ‘II.7’);**

If we are not loading all the columns, use this form:

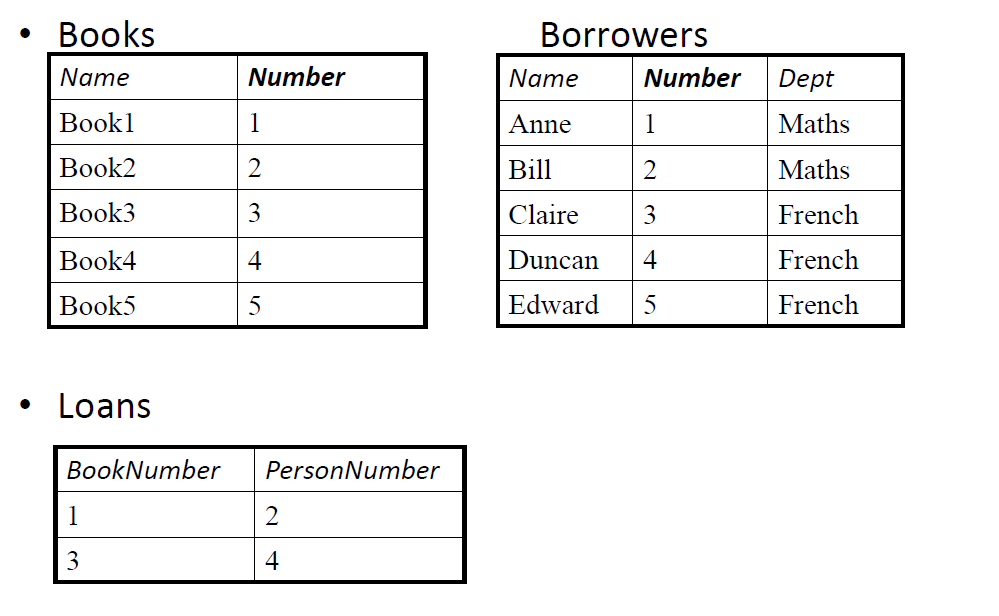
**INSERT INTO Staff (Sno, Sname) VALUES (456, `Waldenstein`);**

Secondly, by extracting the data from existing tables

**INSERT INTO Loan SELECT DISTINCT Sno, Bno, Date\_out FROM Staff\_Borrower;**

We can also use a bulk-loader utility (phpMyAdmin has one):

**INSERT INTO `books` (`Name`, `Number`) VALUES (‘book1’, ‘1’), (‘book2’, ‘2’), (‘book3’, ‘3’);**



**SELECT \* FROM Borrowers** Show whole table

**SELECT Name FROM Borrowers** List Borrowers names only

**SELECT Number, Dept FROM Borrowers WHERE Name = “Anne”** Get Anne’s number and department

**SELECT MAX(Number) FROM Borrowers** Shows largest borrower number (Min is also available)

**SELECT Number+1 FROM Borrowers** Adds 1 to each borrower number and reports it

**SELECT sin (45)** Calculates the sine of 45 – no need to reference a table at all

**SELECT Borrowers.Name, Books.Name FROM Borrowers, Books** Pairs every borrower with every book

**SELECT Number FROM Books WHERE Name = “Book1”** Tell us that book1 has the ID number 1

**SELECT PersonNumber FROM Loans WHERE BookNumber=1** Tell us that person 2 has the book

**SELECT Name FROM Borrowers WHERE Number=2** Tell us that person number 2 is Bill, so Bill has Book1

**Put it all Together**

**SELECT Borrowers.Name FROM Borrowers, Books, Loans WHERE Books.Name = “Book1” AND**

**Books.Number = Loans.BookNumber AND Borrowers.Number = Loans.PersonNumber**

-Returns Bill

-Note the use of **AND,** we can also use **OR** and **NOT,** and comparisons: **> , < , <= , >=, =, != (or <>)**

When comparing NULL, use **IS NULL,** or **IS NOT NULL**

We can search for partial strings using **LIKE:**

**SELECT Name FROM Borrowers WHERE Name LIKE “%e”**

The % character is a wild card, it looks for anything ending in “e”

**SELECT Name FROM Borrowers WHERE Name LIKE “\_e”**

The \_ is a single character wild card, it looks for names with two characters ending in “e”

**SELECT \* FROM table WHERE x BETWEEN 1 and 3**

**SELECT \* FROM table WHERE x IN (‘A’,’B’,’C’)**

**SELECT \* FROM table WHERE x NOT IN (‘A’,’B’,’C’)**

Use brackets to enforce operator order:

**WHERE (a=1) AND (b=2 OR b=3)**

**!=**

**WHERE (a=1 AND b=2) OR (b=3)**

Here a few things you can do to the selected list:

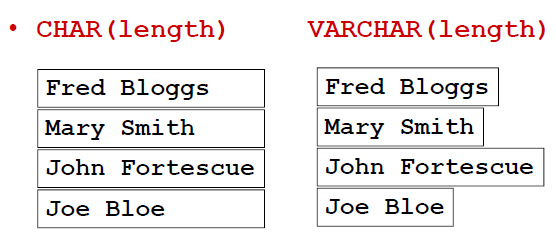
* Sort the results by one or more fields
  + **ORDER BY field, field, . . . [DESC]**
* Force a query to contain unique entries only
  + **SELECT DISTINCT**
* Select a given number of entries starting at a given offset
  + **SELECT \* FROM table LIMIT (offset, count)**

**Numeric Types**

* **Integer Types of various sizes:**
  + **Tinyint =** 1 byte
  + **Small int =** 2 bytes
  + **Mediumint =** 3 bytes
  + **Int =** 4 bytes
  + **Bigint =** 8 bytes
* **Floating Point:**
  + **Float =** 4 bytes
  + **Double =** 8 bytes
* **Fixed Point:**
  + **Numeric or Decimal (size, d)**
  + **D =** number of digits to right of decimal point
* **Bool** is implemented as Tinyint
  + **Boolean:** True = 1 and False = 0

**String Types**

* **CHAR (length)**
  + Stores string with “length” characters – smaller strings are right padded with blanks
    - Up to maximum of 255 characters
  + Blanks removed on retrieval
  + Indexed searching is faster
* **VARCHAR (length)**
  + Stores variable size (up to “length”) strings with no padding ie only natural characters stored.
    - Blanks may or may not be removed
  + Stores prefix (1 or 2 bytes) that contains the length of the string
  + Uses less storage
  + If they are always the same length, uses more storage.



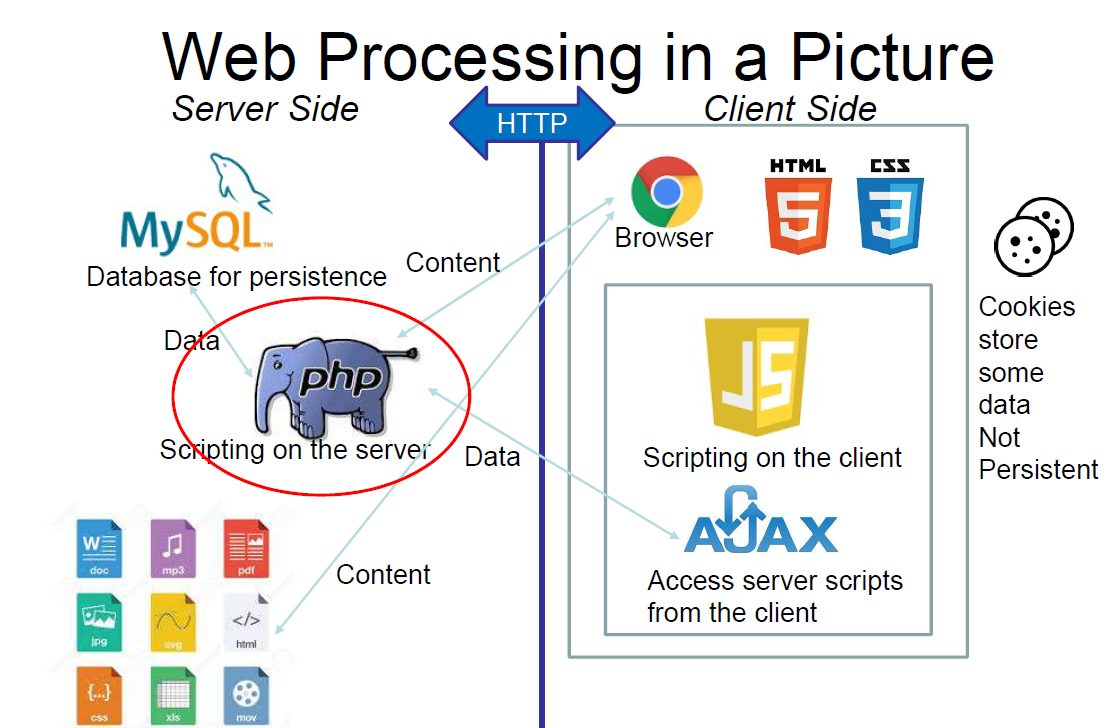
**TEXT Types**

* **TEXT** types hold variable length character strings
  + **TINYTEXT** (up to 255 characters)
  + **TEXT** (up to 65,535 characters)
  + **MEDIUMTEXT** (up to 16,777,215 characters)
  + **LONGTEXT** (up to 4,294,967,295 characters)
* **TEXT** has no trailing space removal on INSERT or SELECT
* **TEXT** has padding added in comparisons to fit the compared object

**Date and Time Types**

* **DATETIME** stores a date and time: YYYY-MM-DD HH:MM:SS
* **DATE** stores just a date: YYYY-MM-DD
* You can enter values in reasonable number of formats:
  + Yyyy-mm-dd as a string
  + Yyyy/mm/dd as a string
  + Yyyymmdd as a string or number
  + Yymmdd as a string or number
* **TIME** as HH:MM:SS
* **YEAR** as YY or YYYY
* Part extraction:
  + **Hour(), Month() etc**
* Current time:
  + **Now()**
* Counting:
  + **Dayofmonth(), Dayofyear()**
* Adding:
  + **Addtime()**
* You can also use standard comparisons
  + **WHERE date1 > date2**
  + **WHERE date BETWEEN date1 AND date2**

**PHP**



In a client browser the PHP program is run on the server, no code is downloaded to the client. Only the output from the PHP program is sent to the client. The easiest way to run a PHP program to se its output is to put it on a web server and open it in a browser.

**<?php . . .**

**. . .**

**?>**

Anything inside these delimiters is interpreted as php, anything outside them is sent straight to the client browser. (Usually HTML/CSS/Javascript code)

PHP ends each line with a semi-colon; and uses { } braces to delimit code blocks. PHP can be used in either a procedural or object-oriented way. PHP denotes variables with the dollar sign: $a.

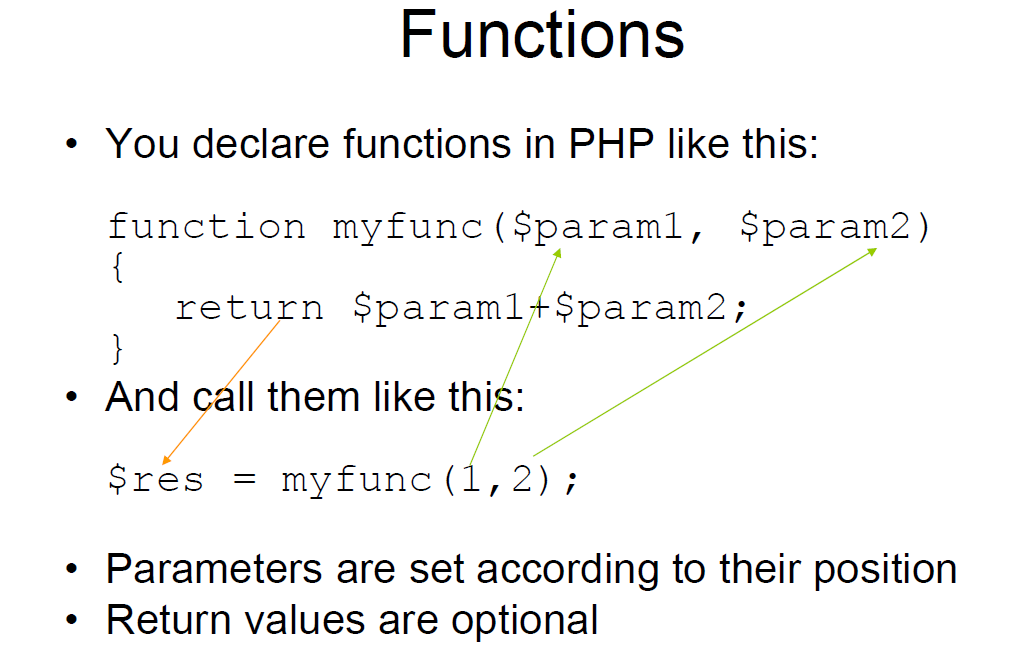
Eg You assign variables like this: **$a = 5;**

And compare them like this: **if ($a ==5) . . .**

You can used them in strings like this: **print(“$a is the value of variable a”);**

You don’t have to declare a variable before you use it, but you should. **$a = $b** // $a never declared

PHP supports two kinds of array: Numeric indexed, where the intex is a number and Associative where the index is a string.



Strings are concatenated with a dot.

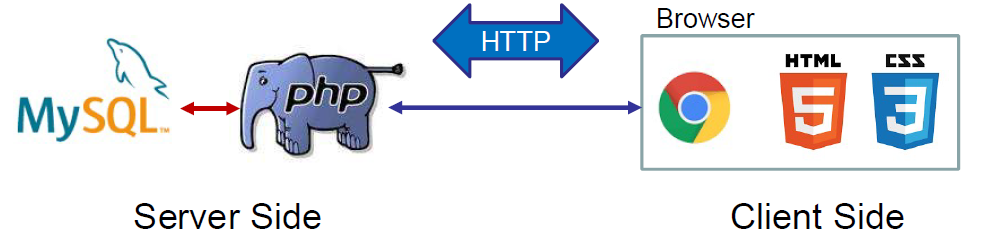
$a = “Hello”;

$b= “There”;

$c= $a.$b;

$c becomes “Hello There”

You can also use $a. = “something”; to tag a string onto the end of $a



The MySQL database and the PHP code sit on the server. The PHP runs, makes queries from the database, and creates HTML. The HTML is sent to the browser and displayed. The PHP code is not readable by people using the browser – important security. To run a query, the user requests a new web page either by clicking a link or submitting a form.

Three Step Process:

* Web pages send data to the server in several ways:
  + Data from a form filled in by the user
  + Data stored in a cookie on the user’s computer
  + Session data
  + Data appended to the <URL:go.php?user=fred>
  + The name of the .php file to run
  + When the form is submitted, data is sent to the PHP program using method specified:
    - ‘get’ appends the data to the URL
    - ‘post’ sends the data in the HTTP request
  + Data is accessed in the PHP program using arrays $\_GET and $POST
* Run PHP
  + PHP program connects to the database and sends queries
  + Results are read from the database by the PHP program
  + These results are used to decide what to show in the web browser
* Produce Output
  + The most usual output is for the PHP program to produce HTML to be sent to the browser
  + It can also write to the database
  + Or carry out other actions based on the results of its processing
    - Send emails
    - Process orders or money etc.

There are three main methods of connecting to MySQL from PHP:

1. Mysqli\_procedural set of commands
2. Mysqli\_object-oriented set of commands
3. Use PHP Data Objects(PDO)

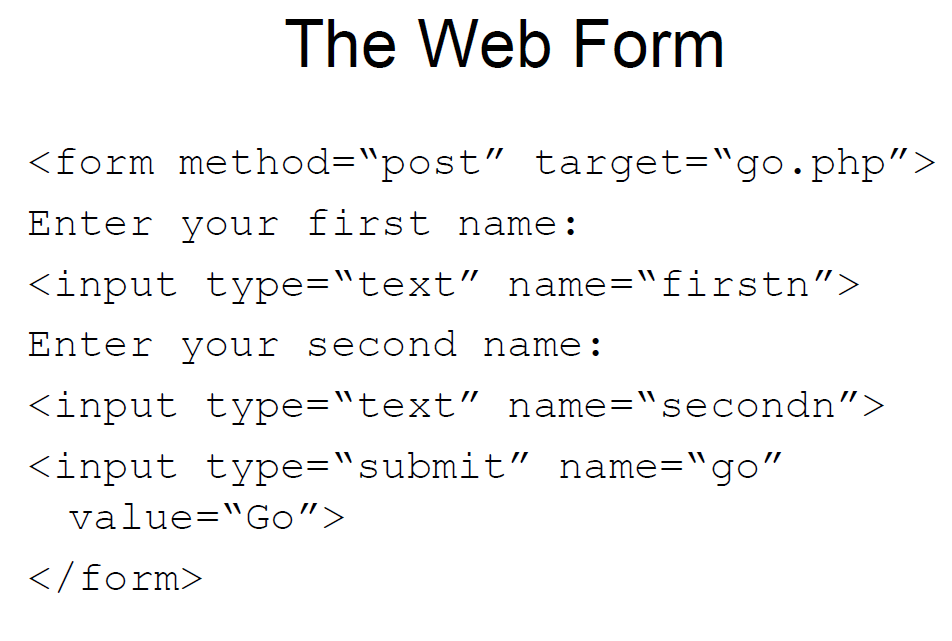
Mysqli\_:

* Connecting and selecting databases
* Running queries
* Stepping through the results of queries
* Example object:
  + **$conn = new mysqli($servername, $username, $password);**
* Example function:
  + **$conn = mysqli\_connect($servername, $username, $password)**

Regardless of how a web page sends data to the PHP program, a malicious hacker could write their own code to send whatever they want to your script. And you can’t prevent this. All you can do is be very careful at the PHP stage to check what was sent from the form before sending anything to the database.

A simple worked Example

1. Build a web page to ask for a user’s first and second name
2. Run a HP script to access that data from HTTP Post
3. Use the data to build a MySQL statement to search a table for that name and return the phone number that goes with it
4. Display the phone number in HTML



1. Connect to a database using mysqli\_connect
2. Need server name, user name, password and name of database to connect to

**$d = mysqli\_connect($server, $user, $password, $db) if (!$d) {die(“Failed: " . mysqli\_connect\_error());}**

1. Returns a link to the database to pass to other mysqli\_calls

