Chapter 2
Structured Datasets - Relational Databases

Section 1. What is a Database? What is a DBMS?

A ***database*** is a self-describing, integrated collection of data that models relevant aspects of an enterprise. This collection of data and its description are intended to be stored on persistent storage devices and to be shared among multiple users.

- The database's self-description is known as the ***system catalog***, ***data dictionary***, or ***metadata*** (data about data).

Why are databases of interest? For effective decision-making, individuals or organizations maintain data about circumstances related to decisions that it must make. Through this data the individual or organization (which we shall call the *user*) can build an abstraction (or model) of such circumstances. By processing the abstraction the user tries to learn more about the circumstances to guide them in making a decision.[1]

- Spreadsheets can do the same thing? For keeping data when would a person choose a database over a spreadsheet, or vice-versa?

  You will have a better appreciation of this later once you know more about databases, of course, but a quick answer could be obtained by asking yourself what are main data elements of interest? Now imagine that you are keeping data relevant to each of these on index cards. Now consider how you would be typically using these activities:

  a. Would you be frequently be going through the stack of cards and making the same updates to each of these cards, or performing the same calculations using the data on these cards? -- If so, a spreadsheets may be more effective than a database in this case.

  b. Would you commonly be shuffling through the cards, setting aside those cards whose data satisfied a certain condition, or rearranging the cards into groups according to some specified condition? If so, then a database may be more effective than a spreadsheet.

*A **database management system*** (DBMS) is a software system that:

- allows users define a database and to store, use, or modify data in that database,
- controls shared access to a database, and
- provides mechanisms to help ensure the integrity and security of the shared data.

More specifically, among its functions, a DBMS typically has processes to:

a. define the storage structures (in secondary memory) for the data;
b. load the data;
c. accept requests (or *queries*) for data from programs or users,
d. format retrieved data so that it appears in a form appropriate for the person or program that requested it;
e. accept and perform updates to the data; this could include adding new data, deleting data, or changing the values of existing data;
f. perform data backup and recovery;
g. allow concurrent use of the data by several users without having users interfere with one another;
h. provide controlled access to the database.

Among the outcomes of this course are ones to familiarize you with how these functions may be carried out on a widely used DBMS and for you to actually perform them.

[1] The notion that data should be kept because it might prove to be useful dates to the origins of writing. Cuneiform goes back to 8000BC in Sumeria where symbols were used to represent trading goods and livestock using wet clay tablets. This led to the establishment of archives (the archives at Ebla dates to 2500BC) and libraries (the most famous of which was the Library at Alexandria, which was destroyed by fire in 48BC).

Section 2. Relational Databases

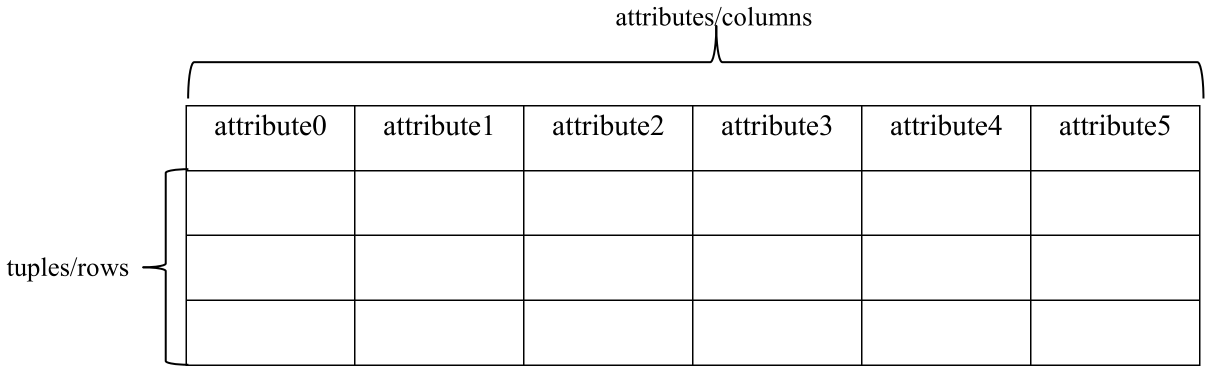
Computerized database systems date back to 1960, when Charles Bachman developed the Integrated Data System using what came to be known as the *network paradigm*, and IBM (not be be outdone by IDS) developed its Information Management System (IMS) based on what was known as the *hierarchical paradigm*.

Since the mid 1980s the dominant paradigm for structuring databases has been the *relational* model.[1] The term "relational" comes from the mathematical concept of a *relation*, which comprises the theoretical foundation on which this approach to database design rests. In practice relations are realized as tables and one will find the terms "table" and "relation" used interchangeably.

The columns of a relation (table) are known as the relation's ***attributes***, while each row of a relation is known as a ***tuple***. The number of rows in a table at any one time is the ***cardinality*** of the table.

- Each attribute must have a name, preferably one that describes what the attribute represents. No two attributes in the same table can have the same name.

- All of the tuples of a relation are expected to be distinct. That is, for any pair of tuples in a relation, there must be at least one attribute where the values of these tuples for that attribute are different.



attribute0	attribute1	attribute2	attribute3	attribute4	attribute5

Example*:* Consider the following relation, where the attribute names (admittedly not descriptive) are given in bold

att0	**att1**	**att2**	**att3**
X	aaa	1	R2D2
Y	bbb	3	C3PO
X	cc	5	OB1
Y	bbb	3	C3PO
W	aaa	2	C3PO

cause the table to be ill-formed as a relation since they are identical

You will notice that the second and fourth tuples in this relation have identical values across all of their attributes. This relation is therefore inappropriate insofar as being a relation in a relational database since there is no way to distinguish the second tuple from the fourth tuple.

[1] Although it wasn't until the mid-80s that the relational database started to become the predominant database model, the idea for the relational model was proposed in a 1970 paper by E.F. Codd "A Relational Model of Data for Large Shared Data Banks" that was published in the *Communications of the ACM*, Volume 13, Number 6, June 1970, pp 377-387. In 1981 Codd was awarded ACM's A.M. Turing Award, the "Nobel Prize of Computing."

Examples Of Simple Relational Databases.

1. **`Employee-Project-AssignedTo`** database (EPA): This database attempts to model a company's employees, the projects the company is working on, and who is working on each project (or what projects each employee is working on).

Employee	
EmpNo	**EmpName**
101	Jones
103	Smith
104	Clark
106	Byron
107	Evans
110	Drew
112	Smith

Project		
ProjNo	**ProjDesc**	**Supervisor**
COMP231	Mobile App	107
COMP278	Web Service	110
COMP353	Database	104
COMP354	OS	110
COMP453	Database	101

AssignedTo	
ProjNo	**EmpNo**
COMP453	101
COMP354	103
COMP353	104
COMP453	104
COMP231	106
COMP278	106
COMP353	106
COMP354	106
COMP231	107
COMP353	107
COMP278	110
COMP354	110
COMP354	112
COMP453	112

2. **`Student-Class-EnrolledIn database (SCE):`** This database attempts to model a university's students, the courses the university offers, and who is enrolled in each course (or what courses each student is taking).

Student			
STID	**Name**	**Major**	**Level**
100	Jones	HIST	JR
150	Parks	DATA	SO
200	Baker	CSCI	JR
250	Glass	HIST	SR
300	Baker	ACCT	SR
350	Russell	CSCI	JR
400	Rye	CSCI	FR
450	Jones	DATA	SR

Class		
Name	**Time**	**Room**
BA200	TR9	SC110
DA210	MWF3	SC213
BF410	MWF8	SC213
CS150	MWF2	EA304
CS250	MWF12	EA304

EnrolledIn	
STID	**ClassName**
100	DA210
150	BA200
200	DA210
200	CS250
300	CS150
400	BA200
400	BF410
400	CS250
450	DA210

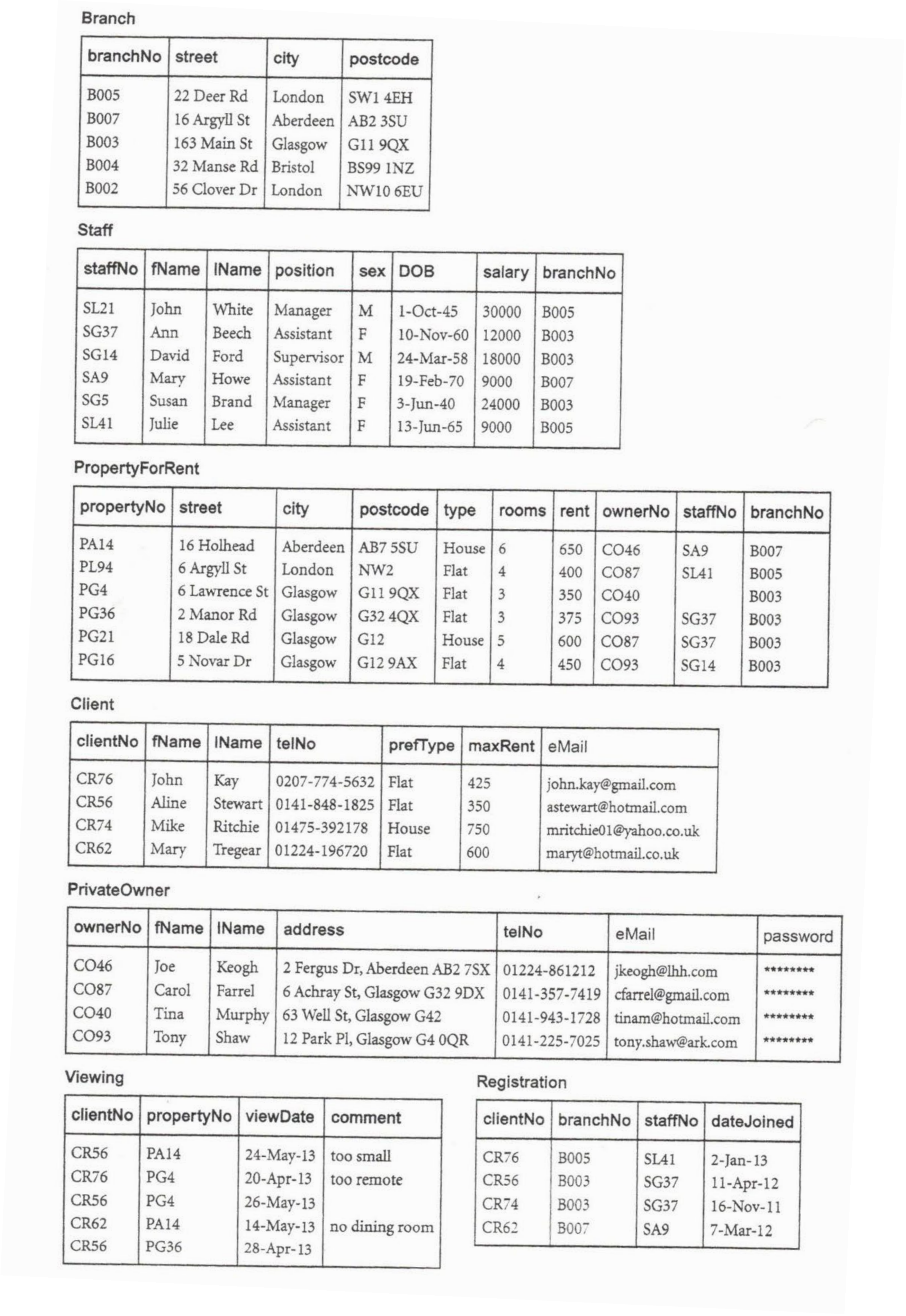
3. **Salesperson-Customer-Order database (SCO):** This database attempts to model a sales-based company's salespeople, customers, and the orders those customers have placed.

Salesperson		
Name	**Age**	**Salary**
Abel	63	120000
Baker	38	42000
Jones	22	36000
Murphy	42	50000
Smith	59	118000
Abernathy	22	36000

Customer	
Name	**City**
Abernathy	Miami
Broadway	Charlotte
Office Pro	Charleston
Amalgamated	Charlotte

Order			
Number	**Salesperson**	**Customer**	**Amount**
100	Abernathy	Abernathy	560
200	Jones	Abernathy	2500
300	Abel	Broadway	480
400	Abel	Office Pro	2500
500	Murphy	Amalgamated	6000
600	Jones	Abernathy	7000
700	Jones	Abernathy	2500

4. Database for Managing Rental Property In Multiple Cities



Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Client

clientNo	fName	lName	telNo	prefType	maxRent	eMail
CR76	John	Kay	0207-774-5632	Flat	425	john.kay@gmail.com
CR56	Aline	Stewart	0141-848-1825	Flat	350	astewart@hotmail.com
CR74	Mike	Ritchie	01475-392178	House	750	mritchie01@yahoo.co.uk
CR62	Mary	Tregear	01224-196720	Flat	600	maryt@hotmail.co.uk

PrivateOwner

ownerNo	fName	lName	address	telNo	eMail	password
CO46	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212	jkeogh@lhh.com	********
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419	cfarrel@gmail.com	********
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728	tinam@hotmail.com	********
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025	tony.shaw@ark.com	********

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-13	too small
CR76	PG4	20-Apr-13	too remote
CR56	PG4	26-May-13	
CR62	PA14	14-May-13	no dining room
CR56	PG36	28-Apr-13	

Registration

clientNo	branchNo	staffNo	dateJoined
CR76	B005	SL41	2-Jan-13
CR56	B003	SG37	11-Apr-12
CR74	B003	SG37	16-Nov-11
CR62	B007	SA9	7-Mar-12

Properties of Relations

1. ***Each relation within a database must have a name.***
   - Two relations in the same database cannot have the same name, although the same relation name can be used in different databases.

   Examples: ***Done in class***

2. ***Each attribute of each relation must have a name***.
   - While within a relation the names used for attributes must be distinct, two different relations in the same database can have one or more attributes with the same names.

   Examples: ***Done in class***

3. ***Each attribute must have an associated "type".***
   - The ***type*** defines the structure of the values that will be associated with that attribute. Among the most common types of values used in relational databases are[1]:
     + integers, commonly represented as **INTEGER** or **INT**;
     + real numbers -- which may be further classified as decimal, floating point (float), or double precision (double). For our purposes we will use a specification **DECIMAL(size,d)**, where *size* is the total number of digits and *d* is the number of digits after the decimal point;
     + fixed-length sequences (or strings) of characters, commonly represented as **CHAR(length)**, where length is length of the string
     + variable-length character strings, represented as **VARCHAR(maxLen)**, where *maxLen* is the maximum lenth that a string can have.;
     + dates, which we shall simply represent as **DATE**.

   - The values of these types must be atomic - that is, when one wants to retrieve a value from, or store a value in, an attribute of this type, the entire value is stored or retrieved. For further clarification, if one had a variable-length string representing a person's name, one cannot retrieve just the person's first name in one operation. Instead one must retrieve the entire name and then use a separate string operation to get the first name.

   - Generally when one merely looks at examples of relations one can only *infer* the type of an attribute from the appearance of its values. The actual type of an attribute must be documented somewhere, however.

   Examples: ***Done in class***

4. ***Each relation must have one or more attributes whose combined values are (always) sufficient to distinguish (or identify) each tuple in the relation***. Such a group of attributes is known as the ***primary key*** of the relation.

   - It is usually easiest to work with single-attribute keys. Multi-attribute keys are permitted, however, and are known as ***composite keys*** (or ***compound keys***).

   - It is not uncommon for there to be several attributes or groups of attributes that could serve as a primary key. Each such attribute or attribute group is known as a ***candidate key***.

   Examples: ***Done in class***

5. Some relations may have an attribute whose values are expected to match those of the primary key from another relation. Such an attribute is known as a ***foreign key***.
   - Note again, foreign keys only reference primary key attributes. A relation may have a foreign key that references its own primary key.

   Examples: ***Done in class***

[1] The data types we described here represent some of the data types originally used with relations. As other types of data emerged in popularity, such as images, sounds, video, objects, etc. many vendors of relational databases extended their supported data types to accommodate these.

Section 3. Database Schemas

A database's ***schema*** refers to the organization of the database, including the set of relations, and for each relation the names of the attributes and their types, the relation's primary key, and any foreign keys. For our purposes, we give the organization of each table in one of two formats:

1. In the first format, we give a complete specification that includes the name of each relation/table and then for each table the name and type of each of its attributes. In addition, for each attribute we indicate whether, or not, a tuple must have a value specified for that attribute. Finally, we give the attribute(s) constituting the primary key; we also list all foreign key attributes and the table and attribute it references.

   The notation we use here is our own. Text in bold must be present, values in italics are ones the database designer provides

```
Table TableName1
      Attribute attribute1
         Type dataType1
      Attribute attribute2
         Type dataType2
      Attribute attribute3
         Type dataType3
      ...
      Attribute attributen
         Type dataTypen

      Primary Key (comma-separated list of attribute names)
      Foreign Key attributeA References tableName(attributeName)
      Foreign Key attributeB References tableName(attributeName)
      ...
      Foreign Key attributeK References tableName(attributeName)

Table TableName2
   similar structure to above
...
Table TableNameM
   similar structure to above
```

   Examples: Illustration of database schemas based on the example databases given. ***Done in class***

2. Later on we will be working in situations where it is most important to know just the names of relations and their attributes, and each relation's primary key and foreign keys. For this purpose we can use a much more compact notation based on the following notation

```
TableName(att1, att2, att3,…attn)
```

   The primary key attribute(s) will be underlined and any foreign keys will be circled or placed in bold-italics. In the latter case, we shall assume that the attribute being referenced by a foreign key is apparent from the foreign key's name, although of course this may not always be the case.

   We give no type information in the schema, nor any information as to whether attribute values must be present in a tuple. Full details on each table's structure, including attribute types, whether they are required, and a precise accounting of foreign key references, must come from a more detailed specification such as the first one we specified.

Examples: Illustration of database schemas based on the four databases given. ***Done in class***

Example: We give a schema based on approach 2 above for a simple relational database, UNIVERSITY, to represent some data commonly used for colleges or universities. This is a more elaborate, and more realistic, structure than the one we saw earlier. As such, it involves more relations and more attributes per relation.

DEPARTMENT(DID, DeptName, OfficeLocation, ***ChairID***)

FACULTY(FID, LastName, FirstName, Rank, Tenured, ***DepartmentID***)

STUDENT(STID, LastName, FirstName, Major, Level, GPA, ***AdvisorID***)

COURSE(CID, Title, DeptOffering, CreditHours)

SECTION(SectionID, ***CourseID***, SectionNumber, Semester, Year, ***InstructorID***)

CLASS(***SectID***, ***StudentID***, Grade)

Note that in the relation CLASS the attributes *SectID* and *StudentID* are both components of a compound key, and each is a foreign key as well.

Example: Now let us specify this same schema using the more expansive, and precise, format in our first approach. We designate:

integers with the notation **INTEGER**,

decimal values with the notation **DECIMAL(p,s)** where p denotes the maximum number of digits for the number and s denotes the number of digits to use to the right of the decimal point.

fixed-length character strings with the notation **CHAR(n)**, where n is the fixed size of the character string.

variable length character strings with the notation **VARCHAR(n)**, where n represents the maximum length of such a string can attain. For each case we cannot use a symbol n as we've done here, but must provide an actual value. Deciding on such values is an important design decision that must be made when defining a database.

This example will be completed in class.