Database Management Systems

- A **database** can simply be defined as a structured set of data that…
  - is any collection of data stored in mass storage that can serve as the data source for a variety of applications
  - has the ability to emulate a variety of organizational forms depending on the needs of the application
  - the result of combining a variety of data collections into single integrated collection – a "consolidated" data system

- A **database management system** (DBMS) is a combination of software and data made up of:
  - Physical database—a collection of files that contain the data
  - Database engine—software that supports access to and modification of the database contents
  - Database schema—a specification of the logical structure of the data stored in the database

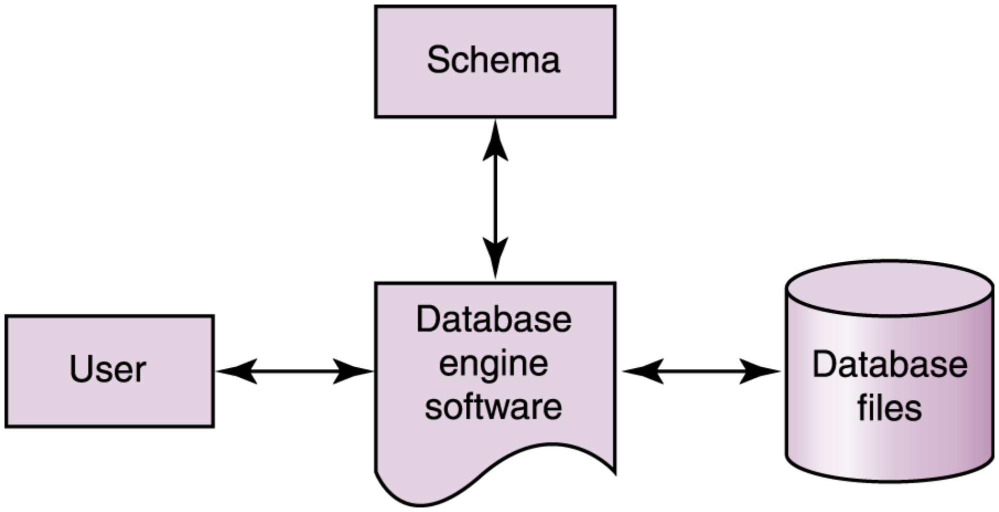


Fig. 1 **The elements of a database management system**

Major concerns in Data Consolidation are:

1. **Security**: control of access to sensitive data is as important an issue as the ability to share it.

   - To provide for this distinction of access privileges, databases often rely on schemas and subschemas.
   - A schema is a description of the entire database structure, a systematic organized framework, used by the database software to maintain the database.

2. **Manageability**: today, many databases are extremely large and continue to grow.

   - Along with this growth come problems of misinformation and misapplication of information.

3. **Privacy and Rights:** It is difficult to determine who has the right to collect what kinds of information.

The Layered Approach to DB Implementation:

Expectation: Usually the "end user" should be allowed to concentrate on using the application for the job at hand.

Duty of the Database System: To present its information in terms of the application NOT in computer scientific terms.

Solution: The layered approach to DB implementation

Layers of DBMS

1. **End User:** machine seen in terms of the application
2. **Application Software:** machine seen in terms of a High-level language
3. **Translator/Interpreter:** machine seen in terms of the Machine language
4. **Actual Machine Hardware:**

DBMS Access

End User Layer:

- The user requests information to the application software

Application Software Layer:

- The application software passes the request to the DBMS

DBMS Layer:

- The DBMS converts the request into a form understood by the routines that actually manipulate the data in mass storage.

Advantages of the Layered Approach –

By separating the functions of the application and the DBMS we can achieve…

1. **Simplified design** process for the application programmer
    - In the context of a distributed database (DB spread over several machines in a network) the work of the application program becomes extremely cumbersome
    - The DBMS allows the application software to be written as though the database was stored on a single machine

2. Provide means for **controlling access** to the database

    - The DBMS can use the entire schema for its internal use but require that each user remain within the bounds of his/her subschema

3. **Data independence**: the ability to change the database organization without changing the application software
    - To implement a change required by a particular user, one needs only to change (a) the schema used by the central system, or (b) subschema of the users involved in the change
    - Allows the application to be written in terms of simplified conceptual view of the database – the database model
    - The actual complexities of data manipulation can be allocated to the DBMS
    - The host language or the general purpose language enhance the usability of the system by adding to the original language

- Specialized database languages allow the user to specify the structure of data; add, modify, and delete data; and **query** the database to retrieve specific stored data
- The database **schema** provides the logical view of the data in the database

The Relational Model

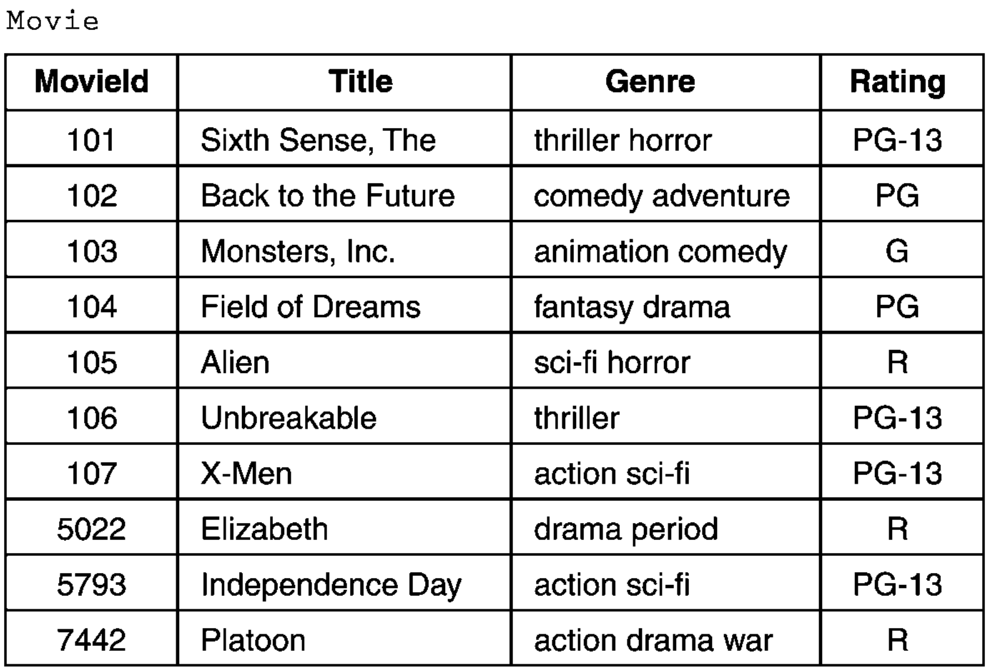
Tables = ***Relations*** <Stored Data>
Records = ***Tupples*** <Rows>
Fields = ***Attributes*** <columns>

One option is to organize into a single extended relation but that would

- lack efficiency by creating duplicated information in tupples dedicated to a single attribute

- risk information loss due to deletion of a tupple that also delete related information that may still be needed;
- reliance on partial tupples may point to a design flaw in the application

- In a relational DBMS, the data items and the relationships among them are organized into **tables**
  - A table is a collection of **records**
  - A record is a collection of related **fields**
  - Each field of a database table contains a single data value
  - Each record in a table contains the same fields



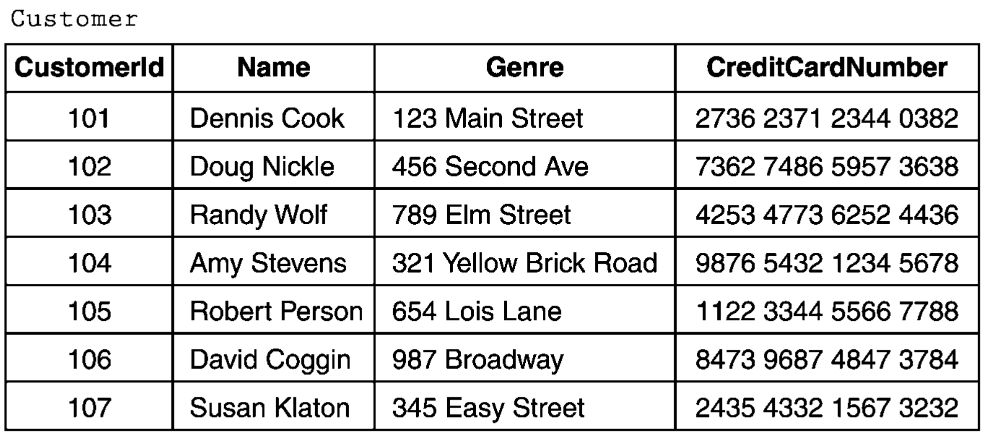
Movie

MovieId	Title	Genre	Rating
101	Sixth Sense, The	thriller horror	PG-13
102	Back to the Future	comedy adventure	PG
103	Monsters, Inc.	animation comedy	G
104	Field of Dreams	fantasy drama	PG
105	Alien	sci-fi horror	R
106	Unbreakable	thriller	PG-13
107	X-Men	action sci-fi	PG-13
5022	Elizabeth	drama period	R
5793	Independence Day	action sci-fi	PG-13
7442	Platoon	action drama war	R

Fig. 2 Movie

- We can express the schema for this part of the database as follows:

Movie (MovieId:key, Title, Genre, Rating)

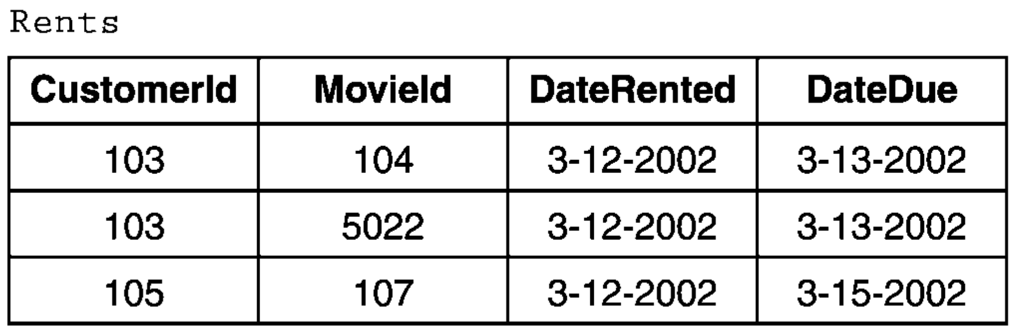


Customer

CustomerId	Name	Genre	CreditCardNumber
101	Dennis Cook	123 Main Street	2736 2371 2344 0382
102	Doug Nickle	456 Second Ave	7362 7486 5957 3638
103	Randy Wolf	789 Elm Street	4253 4773 6252 4436
104	Amy Stevens	321 Yellow Brick Road	9876 5432 1234 5678
105	Robert Person	654 Lois Lane	1122 3344 5566 7788
106	David Coggin	987 Broadway	8473 9687 4847 3784
107	Susan Klaton	345 Easy Street	2435 4332 1567 3232

Fig. 3 Customer

We can use a table to represent a collection of relationships between objects



Rents

CustomerId	MovieId	DateRented	DateDue
103	104	3-12-2002	3-13-2002
103	5022	3-12-2002	3-13-2002
105	107	3-12-2002	3-15-2002

Fig. 4 Rents

- The **Structured Query Language (SQL)** is a comprehensive database language for managing relational databases

E.G. Movie database SQL query:

Syntax: select *attribute-list* from *table-list* where *condition*

```
select Title from Movie where Rating = 'PG'
select Name, Address from Customer
select * from Movie where Genre like '%action%'
select * from Movie where Rating = 'R' order by Title
```

E.G. A database may contain

1. info about employees
2. info about jobs available in the company
3. info about relationship between employees and jobs such as job history

The second option is to redesign the system by keeping the discrete aspects of the relations (new and old) through data decomposition

Employee w/ 3 Attributes

EmpId	JobTitle	Dept

Decomposition into 2 Relations

DB1:

EmpId	JobTitle

DB2:

JobTitle	Dept

Here, information is lost: can find the job title of the target employee and a department having such a job but this does not necessarily mean that the target employee works in that department because several departments may have identical job titles

In other cases, NON_LOSS decomposition is possible.

See:

- 3 Relations Database (R3)

3 Relations Example:
Employee:

EmpId	Name	Address	SIN

Job:

JobId	JobTitle	SkillCode	Dept

Assignment:

EmpId	JobId	StartDate	TermDate

Generic Relational Operations:

1. Select **records** (**tupples**) having certain characteristics and place them in a new relation.   The SELECT operation extracts Rows from a relation.

NEW-1 <- SELECT *from* Employee *where* EmpId = "34Y70"

-

2. Having SELECTed the tupples, (NEW-1), create a new relation that contains the column of values from a certain **attribute** using the **PROJECT** operation.

NEW-2 <- PROJECT JobTitle *from* NEW-1

Another example: when a mailing list is desired you might query:

MAIL <- PROJECT Name, Address *from* Customers

3. Combine different relations into one new relation using the **JOIN** operation.

For Example: Suppose we want to list all EmpId and corresponding Depts…

Using the generic operations require 3 steps:

NEW-1 <- JOIN Assignment *and* Job
**   *where* Assignment.JobId = Job.JobId**

NEW-2 <- SELECT *from* NEW-1
**   *where* Assignment .TermDate = "*"**

FINAL_LIST <- PROJECT Assignment.EmpId, Job.Dept *from* NEW-2

SQL: Structured Query Language

- **Standardized data manipulation language**
- **Lets programmers learn one powerful query language and use it on systems ranging from PCs to the largest mainframe computers**

The above example expressed in SQL looks like this:

Select EmpId, Dept **from** Assignment, Job **where** Assignment.JobId = Job.JobId **and** Assignment.TermDate = "*"

Abstractly in SQL…

1. "**Select** x, y **from** RA" is the ***PROJECT*** operation

2. "**Select** x, y, z **from** RA **where** y = "p" is the ***SELECT*** operation

For example consider…

Select RA. Y, RB. Z
from RA, RB
where RA. x = RB. x

Interpretation:

JOIN RA and RB then
SELECT and **PROJECT** appropriate *tupples* and *attributes* as identified in the "Select" and "where" clauses.

Other Examples of Modifying DB:

1. **insert** into RA values ('p', 'q',	*Employee*:

‘r’)	‘422R12’, ‘Lloyd Burt’, 333 Endless Avenue’
2. **delete** from RA where x = ‘p’	*Employee*: Name = ‘Jerry Lee’

3. **update** RA
Set y = ‘q’
Where x = ‘p’
	Employee:
	Address = ‘1812 Napoleon Crt.’
	Name = ‘Cora Young’