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 a 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

![Diagram](image)

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
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 = *Tuples*   <Rows>
Fields = *Attributes* <columns>

One option is to organize into a single extended relation but that would
- lack efficiency by creating duplicated information in tuples dedicated to a single attribute
• risk information loss due to deletion of a tuple that also delete related information that may still be needed;
• reliance on partial tuples 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

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

Fig. 2 Movie
• We can express the schema for this part of the database as follows:
Movie (MovieId:key, Title, Genre, Rating)

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

Fig. 3 Customer

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

<table>
<thead>
<tr>
<th>CustomerId</th>
<th>Movield</th>
<th>DateRented</th>
<th>DateDue</th>
</tr>
</thead>
<tbody>
<tr>
<td>103</td>
<td>104</td>
<td>3-12-2002</td>
<td>3-13-2002</td>
</tr>
<tr>
<td>103</td>
<td>5022</td>
<td>3-12-2002</td>
<td>3-13-2002</td>
</tr>
<tr>
<td>105</td>
<td>107</td>
<td>3-12-2002</td>
<td>3-15-2002</td>
</tr>
</tbody>
</table>

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'
sel ect 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

<table>
<thead>
<tr>
<th>EmpId</th>
<th>JobTitle</th>
<th>Dept</th>
</tr>
</thead>
</table>

Decomposition into 2 Relations

DB1:

<table>
<thead>
<tr>
<th>EmpId</th>
<th>JobTitle</th>
</tr>
</thead>
</table>

DB2:

<table>
<thead>
<tr>
<th>JobTitle</th>
<th>Dept</th>
</tr>
</thead>
</table>
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:

<table>
<thead>
<tr>
<th>EmpId</th>
<th>Name</th>
<th>Address</th>
<th>SIN</th>
</tr>
</thead>
</table>

Job:

<table>
<thead>
<tr>
<th>JobId</th>
<th>JobTitle</th>
<th>SkillCode</th>
<th>Dept</th>
</tr>
</thead>
</table>

Assignment:

<table>
<thead>
<tr>
<th>EmpId</th>
<th>JobId</th>
<th>StartDate</th>
<th>TermDate</th>
</tr>
</thead>
</table>

**Generic Relational Operations:**

1. Select records (tuples) 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 tuples, (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:

```sql
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…

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

Interpretation:

JOIN RA and RB then
SELECT and PROJECT appropriate tuples and attributes as identified in the “Select” and “where” clauses.

Other Examples of Modifying DB:

1. **insert** into RA values ('p', 'q', 'Employee':
2. **delete** from RA where \( x = 'p' \)

3. **update** RA
   - Set \( y = 'q' \)
   - Where \( x = 'p' \)

<table>
<thead>
<tr>
<th>Employee</th>
<th>Address</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>‘422R12’, ‘Lloyd Burt’, 333 Endless Avenue’</td>
<td>‘Jerry Lee’</td>
</tr>
<tr>
<td></td>
<td>‘1812 Napoleon Crt.’</td>
<td>‘Cora Young’</td>
</tr>
</tbody>
</table>