# Database Management Systems

## Module 01: Course Overview

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

*ppd@cse.iitkgp.ac.in*

# Module Objectives



- To understand the importance of database management systems in modern day applications
- To Know Your Course

# Module Outline





- Why Databases?
- KYC: Know Your Course
  - Course Prerequisite
  - Course Outline
  - Course Text Book

# Why Databases?

# Database Management System (DBMS)

- DBMS contains information about a particular **enterprise**
  - Collection of interrelated data
  - Set of programs to access the data
  - An environment that is both *convenient* and *efficient* to use
- Database Applications:
  - Banking: transactions
  - Airlines: reservations, schedules
  - Universities: registration, grades
  - Sales: customers, products, purchases
  - Online retailers: order tracking, customized recommendations
  - Manufacturing: production, inventory, orders, supply chain
  - Human resources: employee records, salaries, tax deductions
  - ...
- Databases can be very large
- Databases touch all aspects of our lives

# University Database Example



- Application program examples
  - Add new students, instructors, and courses
  - Register students for courses, and generate class rosters
  - Assign grades to students, compute grade point averages (GPA) and generate transcripts
- In the early days, database applications were built directly on top of file systems

# Drawbacks of using file systems to store data

- Data **redundancy** and **inconsistency**
  - Multiple file formats, duplication of information in different files
- Difficulty in **accessing data**
  - Need to write a new program to carry out each new task
- Data **isolation**
  - Multiple files and formats
- **Integrity** problems
  - Integrity constraints (e.g., account balance $> 0$) become "buried" in program code rather than being stated explicitly
  - Hard to add new constraints or change existing ones

# Drawbacks of using file systems to store data (2)



- **Atomicity** of updates
  - Failures may leave database in an inconsistent state with partial updates carried out
  - Example: Transfer of funds from one account to another should either complete or not happen at all
- **Concurrent access** by multiple users
  - Concurrent access needed for performance
  - Uncontrolled concurrent accesses can lead to inconsistencies
    - Example: Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time
- **Security** problems
  - Hard to provide user access to some, but not all, data

**Database systems offer solutions to all the above problems**

# Know Your Course





## Know Your Course

# Course Prerequisites: Essential





- **Set Theory**
  - Definition of a Set
    - Intensional Definition
    - Extensional Definition
    - Set-builder Notation
  - Membership, Subset, Superset, Power Set, Universal Set
  - Operations on sets:
    - Union, Intersection, Complement, Difference, Cartesian Product
  - De Morgan's Law
  - **Courses**
    - **MOOCs: Discrete Mathematics:** https://nptel.ac.in/courses/111/106/111106086/
    - **Online Degree Foundational Course: Mathematics for Data Science I** https://onlinedegree.iitm.ac.in/course_pages/BSCMA1001.html

# Course Prerequisites: Essential



- **Relations and Functions**
  - Definition of Relations
  - Ordered Pairs and Binary Relations
    - Domain and Range
    - Image, Preimage, Inverse
    - Properties: Reflexive, Symmetric, Antisymmetric, Transitive, Total
  - Definition of Functions
  - Properties of Functions: Injective, Surjective, Bijective
  - Composition of Functions
  - Inverse of a Function
  - **Courses**
    - **MOOCs: Discrete Mathematics:**
      https://nptel.ac.in/courses/111/106/111106086/
    - **Online Degree Foundational Course: Mathematics for Data Science I**
      https://onlinedegree.iitm.ac.in/course_pages/BSCMA1001.html

# Course Prerequisites: Essential





- **Propositional Logic**
  - Truth Values & Truth Tables
  - Operators: conjunction (and), disjunction (or), negation (not), implication, equivalence
  - Closure under Operations
  - **Courses**
    - **MOOCs: Discrete Mathematics:** https://nptel.ac.in/courses/111/106/111106086/

# Course Prerequisites: Essential





- **Predicate Logic**
  - Predicates
  - Quantification
    - Existential
    - Universal
  - **Courses**
    - **MOOCs: Discrete Mathematics:** https://nptel.ac.in/courses/111/106/111106086/

# Course Prerequisites: Essential





- **Data Structures**
  - Array
  - List
  - Binary Search Tree
    - Balanced Tree
  - B-Tree
  - Hash Table / Map
  - **Courses**
    - **MOOCs: Design and Analysis of Algorithms:** https://nptel.ac.in/courses/106/106/106106131/
    - **MOOCs: Fundamental Algorithms – Design and Analysis:** https://nptel.ac.in/courses/106/105/106105157/

# Course Prerequisites: Essential





- **Programming in Python**
  - **Courses**
    - **Online Degree Foundational Course - Programming in Python**
      https://onlinedegree.iitm.ac.in/course_pages/BSCCS1002.html

# Course Prerequisites: Desirable





- **Algorithms and Programming in C**
  - Sorting
    - Merge Sort
    - Quick Sort
  - Search
    - Linear Search
    - Binary Search
    - Interpolation Search
  - **Courses**
    - **MOOCs: Design and Analysis of Algorithms:** https://nptel.ac.in/courses/106/106/106106131/
    - **MOOCs: Introduction to Programming in C:** https://nptel.ac.in/courses/106/104/106104128/

# Course Prerequisites: Desirable





- **Object-Oriented Analysis and Design**
  - **Courses**
    - **MOOCs: Object-Oriented Analysis and Design:** https://nptel.ac.in/courses/106/105/106105153/

# Course Outline





| Week No. | Topics |
|---|---|
| **Week 1** | Course Overview, Introduction |
| **Week 2** | Basic Structured Query Language |
| **Week 3** | Advanced Structured Query Language |
| **Week 4** | Relational Algebra, Entity Relationship Model |
| **Week 5** | Normal Forms and Functional Dependency |
| **Week 6** | Normal Forms and Functional Dependency |
| **Week 7** | Application Development |
| **Week 8** | Storage Management |
| **Week 9** | Indexing and Hashing |
| **Week 10** | Transactions |
| **Week 11** | Backup and Recovery |
| **Week 12** | Query Optimization, Conclusion |

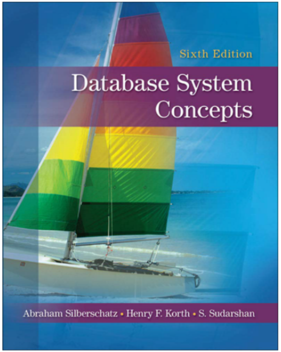
Application Programmer

DBA / Designer

# Course Textbook








**Database System Concepts**,
*Sixth Edition*,

Abraham Silberschatz,
Henry Korth,
S. Sudarshan,

Publisher: McGraw Hill Education
ISBN: 0073523321

Website: http://db-book.com/

*7th Edition will also do*

# Module Summary





- Elucidates the importance of database management systems in modern day applications
- Introduced various aspects of the Course

**Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.**

**Edited and new slides are marked with "PPD".**

# Database Management Systems

## Module 02: Why DBMS?/1

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

*ppd@cse.iitkgp.ac.in*

# Module Objectives





- To understand the need for a DBMS from historical perspective

# Module Outline





- Evolution of data management practices
- History of DBMS

# Evolution of Data Management

# Data Management





Management of Data or Records is a basic need for human society:

- Storage
- Retrieval
- Transaction
- Audit
- Archival

For:

- Individual
- Small / Big Enterprise
- Global

There have been two major approaches in this practice:

- Physical
- Electronic

# Data Management: Physical





Physical Data or Records management, more formally known as *Book Keeping*, has been using physical ledgers and journals for centuries.

The most significant development happened when Henry Brown, an American inventor, patented a "receptacle for storing and preserving papers" on November 2, 1886.

Herman Hollerith adapted the punch cards used for weaving looms to act as the memory for a mechanical tabulating machine, in 1890.

# Data Management: Electronic





Electronic Data or Records management moves with the advances in technology - especially of memory, storage, computing, and networking.

- 1950s: Computer Programming started
- 1960s: Data Management with punch card / tapes and magnetic tapes
- 1970s:
  - COBOL and CODASYL approach was introduced in 1971
  - On October 14 in 1979, Apple II platform shipped VisiCalc, marking the birth of the spreadsheet
  - Magnetic disks became prevalent
- 1980s: RDBMS changed the face of data management
- 1990s: With Internet data management started becoming global
- 2000s: e-Commerce boomed, NoSQL was introduced for unstructured data management
- 2010s: Data Science started riding high

# Electronic Data Management Parameters





Electronic Data or Records management depends on various parameters including:

- **Durability**
- **Scalability**
- **Security**
- **Retrieval**
- **Ease of Use**
- **Consistency**
- **Efficiency**
- **Cost**
- ...

# Book Keeping





Recall how shop owners used to maintain their accounts.
A book register was maintained on which the shop owner wrote the amount received from customers, the amount due for any customer, inventory details and so on.

Problems with such an approach of book-keeping:

- **Durability**: Physical damage to these registers is a possibility due to rodents, humidity, wear and tear
- **Scalability**: Very difficult to maintain for many years, some shops have numerous registers spanning over years
- **Security**: Susceptible to tampering by outsiders
- **Retrieval**: Time consuming process to search for a previous entry
- **Consistency**: Prone to human errors

Not only small shops but large organizations also used to maintain their transaction details in book registers.

# Spreadsheet Files - A better solution





**Spreadsheet Softwares like Google Sheets:** Due to the disadvantages of maintaining ledger registers, organizations dealing with huge amount of data shifted to using spreadsheet softwares for maintaining their records in files.

- **Durability**: These are computer applications and hence data is less prone to physical damage.
- **Scalability**: Easier to search, insert and modify records as compared to book ledgers
- **Security**: Can be password-protected
- **Easy of Use**: Computer applications are used to search and manipulate records in the spreadsheets leading to reduction in manpower needed to perform routine computations
- **Consistency**: Not guaranteed but spreadsheets are less prone to mistakes than registers.

**Mostly useful for single user or small enterprise applications**

# Why leave filesystems?



## Lack of efficiency in meeting growing needs



- With rapid scale up of data, there has been considerable increase in the time required to perform most operations.
- A typical spreadsheet file may have an upper limit on the number of rows.
- Ensuring consistency of data is a big challenge.
- No means to check violations of constraints in the face of concurrent processing.
- Unable to give different permissions to different people in a centralized manner.
- A system crash could be catastrophic.

The above limitations of filesystems paved the way for a comprehensive platform dedicated to management of data - the **Database Management Systems**.

# History of DBMS

# History of Database Systems

- 1950s and early 1960s:
  - Data processing using magnetic tapes for storage
    - Tapes provided only sequential access
  - Punched cards for input
- Late 1960s and 1970s:
  - Hard disks allowed direct access to data
  - Network and hierarchical data models in widespread use
  - Ted Codd defines the relational data model
    - Would win the ACM Turing Award for this work
    - IBM Research begins System R prototype
    - UC Berkeley begins Ingres prototype
  - High-performance (for the era) transaction processing

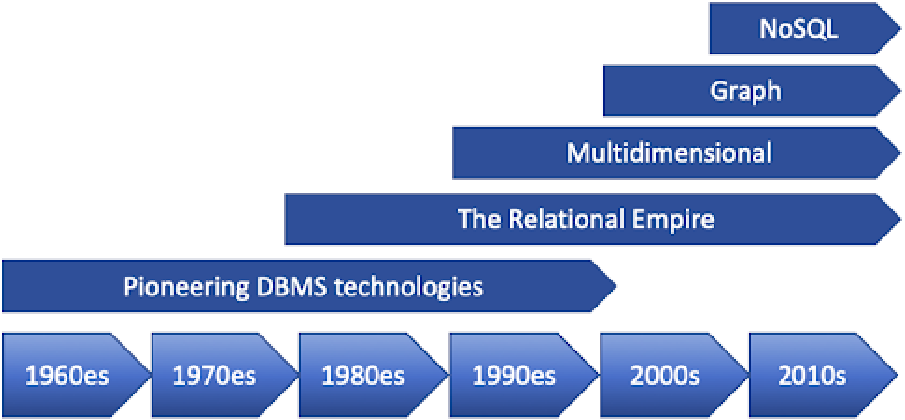
# History (2)



- 1980s:
  - Research relational prototypes evolve into commercial systems - SQL becomes industrial standard
  - Parallel and distributed database systems
  - Object-oriented database systems
- 1990s:
  - Large decision support and data-mining applications
  - Large multi-terabyte data warehouses
  - Emergence of Web commerce
- Early 2000s:
  - XML and XQuery standards
  - Automated database administration
- Later 2000s:
  - Giant data storage systems - Google BigTable, Yahoo PNuts, Amazon, ...

# History (3): Evolution of Data Models

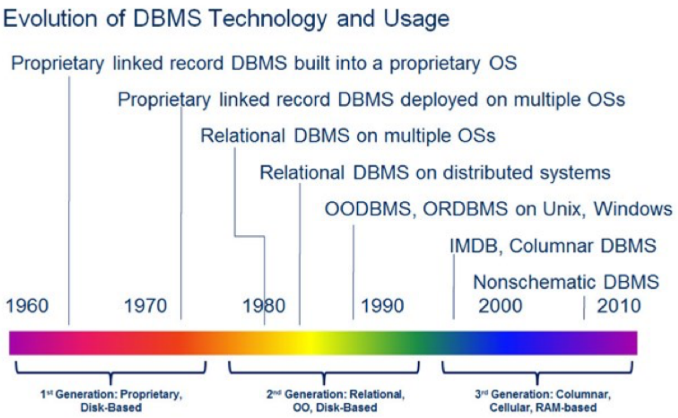




# History (4): Evolution of DB Technology



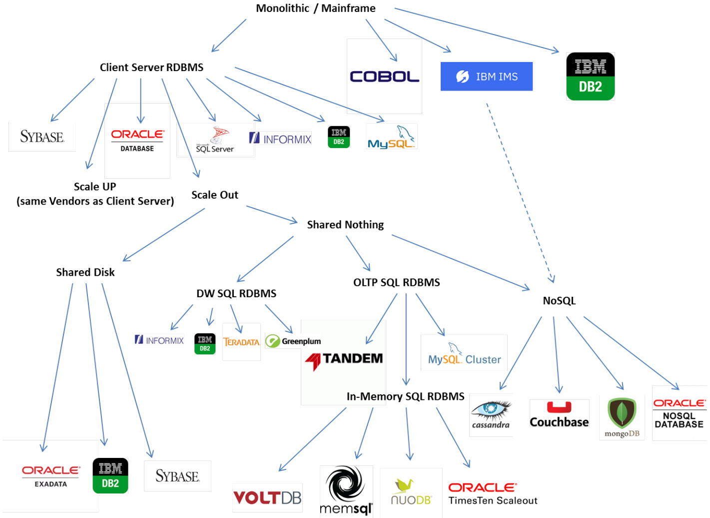




# History (5): Evolution of DB Architecture







# Module Summary





- Walk through of evolution of Data and Records Management
- History of DBMS

**Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.**
**Edited and new slides are marked with "PPD".**

# Database Management Systems

## Module 03: Why DBMS?/2

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

*ppd@cse.iitkgp.ac.in*

# Module Recap





- Evolution of Data and Records Management
- History of DBMS

# Module Objectives





- Comparison of File based data management and DBMS

# Module Outline





- File handling by Python viz-a-viz DBMS - Bank Transaction example
- Parameterized Comparison

# Case Study of Bank Transaction

## Banking Transaction System

Consider a simple banking system where a person can open a new account, transfer fund to an existing account and check the history of all her transactions till date.

The application performs the following checks:

- If the account balance is not enough, it will not allow the fund transfer
- If the account numbers are not correct, it will flash a message and terminate the transaction.
- If a transaction is successful, it prints a confirmation message.

# Case study: A bank transaction (2)





We will use this banking transaction system to compare various features of a file-based (spreadsheet/.csv files) implementation viz-a-viz a DBMS-based implementation

- Account details are stored in
  - *Accounts.csv* for file-based implementation
  - *Accounts table* for DBMS implementation
- The transaction details are stored in
  - *Ledger.csv* file for file-based implementation
  - *Ledger table* for DBMS implementation

In the following slides we discuss a fund transfer transaction.

**Source**: *https://github.com/bhaskariitm/transition-from-files-to-db/tree/main*

# Python viz-a-viz SQL

# Bank Transaction: Python viz-a-viz SQL





## Python

```
def begin_Transaction(creditAcc,
debitAcc, amount):

temp = []
success = 0

# Open file handles to retrieve and
   store transaction data

f_obj_Account1 =
    open('Accounts.csv', 'r')
f_reader1 =
    csv.DictReader(f_obj_Account1)
f_obj_Account2 =
    open('Accounts.csv', 'r')
f_reader2 =
    csv.DictReader(f_obj_Account2)
f_obj_Ledger =
    open('Ledger.csv', 'a+')
f_writer =
    csv.DictWriter(f_obj_Ledger,
    fieldnames=col_name_Ledger)
```

## SQL

// Handled implicitly by the DBMS

# Bank Transaction: Python viz-a-viz SQL (2)



## Python

```
try:
  for sRec in f_reader1:
   #CONDITION CHECK FOR ENOUGH BALANCE
   if sRec["AcctNo"] == debitAcc and
   int(sRec["Balance"]) > int(amt):
...
```

## SQL

```
do $$
begin
amt = 5000;
sendVal = '1800090';
recVal = '1800100';
select balance from accounts
into sbalance
where account_no = sendVal;
if sbalance < amt then
...
 $$
```

# Bank Transaction: Python viz-a-viz SQL (3)





## Python

```python
try:
  for sRec in f_reader1:
   #CONDITION CHECK FOR ENOUGH BALANCE
   if sRec["AcctNo"] == debitAcc and
   int(sRec["Balance"]) > int(amt):
       for rRec in f_reader2:
           if rRec["AcctNo"] == creditAcc:
             sRec["Balance"] =                 #DEBIT
                str(int(sRec["Balance"])-int(amt))
             temp.append(sRec)
...
```

## SQL

```sql
do $$
begin
amt = 5000;
sendVal = '1800090';
recVal = '1800100';
select balance from accounts
into sbalance
where account_no = sendVal;
if sbalance < amt then
 raise notice "Insufficient balance";
else
update accounts
    set balance =
     balance - amt
    where account_no = sendVal;
...
$$
```

# Bank Transaction: Python viz-a-viz SQL (4)





## Python

```
try:
  for sRec in f_reader1:
   #CONDITION CHECK FOR ENOUGH BALANCE
   if sRec["AcctNo"] == debitAcc and
   int(sRec["Balance"]) > int(amt):
       for rRec in f_reader2:
           if rRec["AcctNo"] == creditAcc:
             sRec["Balance"] =                #DEBIT
                str(int(sRec["Balance"]) - int(amt))
             temp.append(sRec)
         # Critical point
             f_writer.writerow({
                "Acct1":sRec["AcctNo"],
                "Acct2":rRec["AcctNo"],
                "Amount":amt, "D/C":"D"})
             rRec["Balance"] =                #CREDIT
                str(int(rRec["Balance"]) + int(amt))
             temp.append(rRec)
...
```

## SQL

```
do $$
begin
amt = 5000;
sendVal = '1800090';
recVal = '1800100';
select balance from accounts
into sbalance
where account_no = sendVal;
if sbalance < amt then
 raise notice "Insufficient balance";
else
update accounts
    set balance =
     balance - amt
    where account_no = sendVal;
insert into
  ledger(sendAc, recAc, amnt, ttype)
  values(sendVal, recVal, amt, 'D');
update accounts
    set balance =
     balance + amt
    where account_no = recVal;
...
```

# Bank Transaction: Python viz-a-viz SQL (5)





## Python

```python
try:
  for sRec in f_reader1:
   #CONDITION CHECK FOR ENOUGH BALANCE
   if sRec["AcctNo"] == debitAcc and
   int(sRec["Balance"]) > int(amt):
       for rRec in f_reader2:
           if rRec["AcctNo"] == creditAcc:
             sRec["Balance"] =                #DEBIT
                    str(int(sRec["Balance"]) - int(amt))
             temp.append(sRec)
         # Critical point
             f_writer.writerow({"Acct1":sRec["AcctNo"],
                "Acct2":rRec["AcctNo"],
                "Amount":amt, "D/C":"D"})
             rRec["Balance"] =                #CREDIT
                str(int(rRec["Balance"]) + int(amt))
             temp.append(rRec)
             f_writer.writerow({"Acct1":rRec["AcctNo"],
                "Acct2":sRec["AcctNo"],
                "Amount":amt,"D/C": "C"})
             success = success + 1
             break
   f_obj_Account1.seek(0)
   next(f_obj_Account1)
   for record in f_reader1:
       if record["AcctNo"] != temp[0]["AcctNo"] and
           record["AcctNo"] != temp[1]["AcctNo"]:
              temp.append(record)
except:
   print("Wrong input entered !!!")
```

## SQL

```sql
do $$
begin
amt = 5000;
sendVal = '1800090';
recVal = '1800100';
select balance from accounts
into sbalance
where account_no = sendVal;
if sbalance < amt then
 raise notice "Insufficient balance";
else
update accounts
    set balance =
     balance - amt
    where account_no = sendVal;
insert into
  ledger(sendAc, recAc, amnt, ttype)
  values(sendVal, recVal, amt, 'D');
update accounts
    set balance =
     balance + amt
    where account_no = recVal;
insert into
  ledger(sendAc, recAc, amnt, ttype)
  values(recVal, sendVal, amt, 'C');
commit;
raise notice "Successful";
end if;
end; $$
```

# Bank Transaction: Python viz-a-viz SQL (6)





## Python

```
#Writing back to the file

 f_obj_Account1.close()
 f_obj_Account2.close()
 f_obj_Ledger.close()

 if success == 1:
    f_obj_Account = open('Accounts.csv', 'w+', newline='')
    f_writer = csv.DictWriter(f_obj_Account,
         fieldnames=col_name_Account)
    f_writer.writeheader()
    for data in temp:
        f_writer.writerow(data)

    f_obj_Account.close()
    print("Transaction is successful !!")

 else:
    print('Transaction failed : Confirm Account details')
```

## SQL

// Handled implicitly by the DBMS

# Comparison





| Parameter | File Handling via Python | DBMS |
|---|---|---|
| Scalability with respect to amount of data | Very difficult to handle insert, update and querying of records | In-built features to provide high scalability for a large number of records |
| Scalability with respect to changes in structure | Extremely difficult to change the structure of records as in the case of adding or removing attributes | Adding or removing attributes can be done seamlessly using simple SQL queries |
| Time of execution | In seconds | In milliseconds |
| Persistence | Data processed using temporary data structures have to be manually updated to the file | Data persistence is ensured via automatic, system induced mechanisms |
| Robustness | Ensuring robustness of data has to be done manually | Backup, recovery and restore need minimum manual intervention |
| Security | Difficult to implement in Python (Security at OS level) | User-specific access at database level |
| Programmer's productivity | Most file access operations involve extensive coding to ensure persistence, robustness and security of data | Standard and simple built-in queries reduce the effort involved in coding thereby increasing a programmer's throughput |
| Arithmetic operations | Easy to do arithmetic computations | Limited set of arithmetic operations are available |
| Costs | Low costs for hardware, software and human resources | High costs for hardware, software and human resources |

# Parameterized Comparison

# Scalability

## File handling via Python

- **Number of records:** As the # of records increases, the efficiency of flat files reduces:
  - the time spent in searching for the right records
  - the limitations of the OS in handling huge files
- **Structural Change:** To add an attribute, initializing the new attribute of each record with a default value has to be done by program. It is very difficult to detect and maintain relationships between entities if and when an attribute has to be removed.

## DBMS

- **Number of records:** Databases are built to efficiently scale up when the # of records increase drastically.
  - In-built mechanisms, like indexing, for quick access of right data.
- **Structural Change:** During adding an attribute, a default value can be defined that holds for all existing records - the new attribute gets initialized with the default value. During deletion, constraints are used either not to allow the removal or ensure its safe removal

# Time and Efficiency





| File handling via Python | DBMS |
|---|---|
| • The effort needed to implement a file handler is quite less in Python<br>• In order to process a 1GB file, a program in Python would typically take few seconds. | • The effort to install and configure a DB in a DB server is expensive & time consuming<br>• In order to process a 1GB file, an SQL query would typically take few milliseconds. |

- If the number of records is very small, the overhead in installing and configuring a database will be much more than the time advantage obtained from executing the queries.
- However, if the number of records is really large, then the time required in the initialization process of a database will be negligible as compared to the time saved in using SQL queries.

# Persistence, Robustness, Security





## File handling via Python

- **Persistence**: Data processed using in-memory data structures stay in the memory during processing. After updates, these are manually updated to the file on disk
- **Robustness**: Ensuring consistency, reliability and sanity is manual via multiple checks. On a system crash, a transaction may cause inconsistency or loss of data.
- **Security**: Extremely difficult to implement granular security in file systems. Authentication is at the OS level.

## DBMS

- **Persistence**: Data persistence is ensured via automatic, system mechanisms. The programmer does not have to worry about the data getting lost due to manual errors
- **Robustness**: Backup, recovery & restore need minimum manual intervention. The backup and recovery plan can be devised for automatic recovery on a crash
- **Security**: DBMS provides user-specific access at the database level with restriction for to view only access

# Programmer's Productivity





## File handling via Python

- **Building the file handler**: Since the constraints within and across entities have to be enforced manually, the effort involved in building a file handling application is huge
- **Maintenance**: To maintain the consistency of data, one must regularly check for sanity of data and the relationships between entities during inserts, updates and deletes
- **Handling huge data**: As the data grows beyond the capacity of the file handler, more efforts are needed

## DBMS

- **Configuring the database**: The installation and configuration of a database is specialized job of a DBA. A programmer, on the other hand, is saved the trouble
- **Maintenance**: DBMS has in-built mechanisms to ensure consistency and sanity of data being inserted, updated or deleted. The programmer does not need to do such checks
- **Handling huge data**: DBMS can handle even terabytes of data - Programmer does not have to worry

# Arithmetic Operations





## File handling via Python

- **Extensive support for arithmetic and logical operations**: Extensive arithmetic and logical operations can be performed on data using Python. These include complex numerical calculations and recursive computations.

## DBMS

- **Limited support for arithmetic and logical operations**: SQL provides limited arithmetic and logical operations. Any other complex computation has to be done outside the SQL.

# Costs and Complexity





## File handling via Python

- File systems are cheaper to install and use. No specialized hardware, software or personnel are required to maintain filesystems.

## DBMS

- Large databases are served by dedicated database servers need large storage and processing power
- DBMSs are expensive software that have to be installed and regularly updated
- Databases are inherently complex and need specialized people to work on it - like DBA
- The above factors lead to huge costs in implementing and maintaining database management systems

# Module Summary





- Elucidated the difference between File handling by Python viz-a-viz DBMS through an Bank Transaction example
- Parameterized Comparison

**Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.**
**Edited and new slides are marked with "PPD".**

# Database Management Systems

## Module 04: Introduction to DBMS/1

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

*ppd@cse.iitkgp.ac.in*

# Module Recap





- Comparison of data management using Python & files and DBMS
- Efficacy and Efficient DBMS highlighted

# Module Objectives





- To familiarize with the basic notions and terminology of database management systems
- To understand the role of data models and languages
- To understand the approaches to database design

# Module Outline





- Levels of Abstraction
- Schema & Instance
- Data Models
    - Relational Databases
- DDL & DML
- SQL
- Database Design

# Levels of Abstraction

# Levels of Abstraction

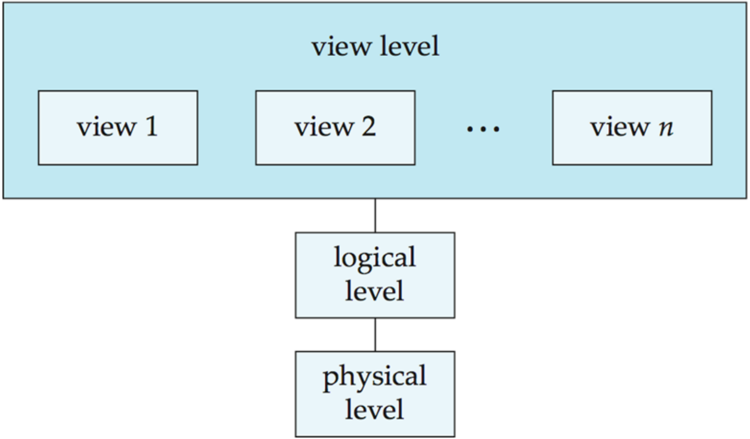


- **Physical level:** describes how a record (for example, instructor) is stored
- **Logical level:** describes data stored in database, and the relationships among the data fields

  **type** *instructor* = **record**
  
  *ID* : string;
  
  *name* : string;
  
  *dept_name* : string;
  
  *salary* : integer;
  
  **end;**
- **View level:** application programs hide details of data types
  - Views can also hide information (such as an employee's salary) for security purposes

# View of Data

An architecture for a database system



# Schema and Instance

# Schemas and Instances





- Similar to type of a variable and value of the variable at run-time in programming languages
- **Schema**
  - **Logical Schema** – the overall logical structure of the database
    - ▷ Analogous to type information of a variable in a program
    - ▷ Example: The database consists of information about a set of customers and accounts in a bank and the relationship between them
    - ▷ Customer Schema

| Name | Customer ID | Account # | Aadhaar ID | Mobile # |
|---|---|---|---|---|

    - ▷ Account Schema

| Account # | Account Type | Interest Rate | Min. Bal. | Balance |
|---|---|---|---|---|

  - **Physical Schema**– the overall physical structure of the database

# Schemas and Instances





- **Instance**
  - The actual content of the database at a particular point in time
  - Analogous to the value of a variable
  - Customer Instance

| Name | Customer ID | Account # | Aadhaar ID | Mobile # |
|---|---|---|---|---|
| Pavan Laha | 6728 | 917322 | 182719289372 | 9830100291 |
| Lata Kala | 8912 | 827183 | 918291204829 | 7189203928 |
| Nand Prabhu | 6617 | 372912 | 127837291021 | 8892021892 |

  - Account Instance

| Account # | Account Type | Interest Rate | Min. Bal. | Balance |
|---|---|---|---|---|
| 917322 | Savings | 4.0% | 5000 | 7812 |
| 372912 | Current | 0.0% | 0 | 291820 |
| 827183 | Term Deposit | 6.75% | 10000 | 100000 |

# Schema and Instances



- **Physical Data Independence** – the ability to modify the physical schema without changing the logical schema
  - Analogous to independence of *Interface* and *Implementation* in Object-Oriented Systems
  - Applications depend on the logical schema
  - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

# Data Models

# Data Models

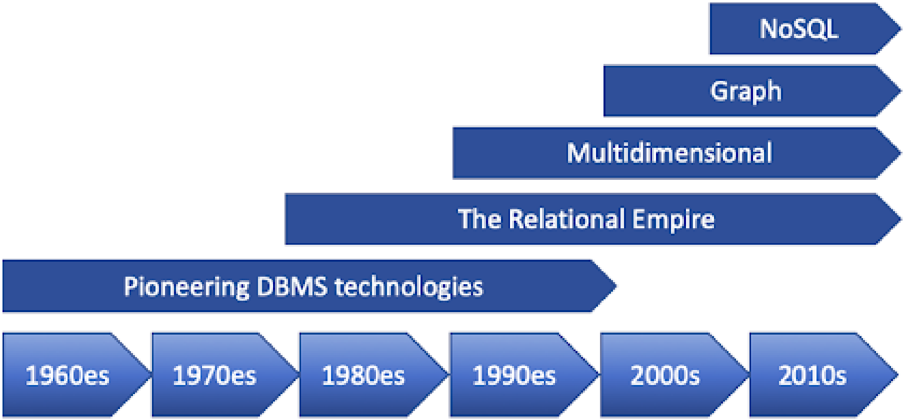


- A collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- **Relational model** (we focus in this course)
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Other older models
  - Network model
  - Hierarchical model
- Recent models for Semi-structured or Unstructured data
  - Converted to easily manageable formats
  - Content Addressable Storage (CAS) with metadata descriptors
  - XML format.
  - RDBMS which supports BLOBs

# Data Models (2)







# Relational Model



- All the data is stored in various tables
- Example of tabular data in the relational model

Columns

| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 22222 | Einstein | Physics | 95000 |
| 12121 | Wu | Finance | 90000 |
| 32343 | El Said | History | 60000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 98345 | Kim | Elec. Eng. | 80000 |
| 76766 | Crick | Biology | 72000 |
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 58583 | Califieri | History | 62000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 15151 | Mozart | Music | 40000 |
| 33456 | Gold | Physics | 87000 |
| 76543 | Singh | Finance | 80000 |

Rows

(a) The *instructor* table

# A Sample Relational Database



| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 22222 | Einstein | Physics | 95000 |
| 12121 | Wu | Finance | 90000 |
| 32343 | El Said | History | 60000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 98345 | Kim | Elec. Eng. | 80000 |
| 76766 | Crick | Biology | 72000 |
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 58583 | Califieri | History | 62000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 15151 | Mozart | Music | 40000 |
| 33456 | Gold | Physics | 87000 |
| 76543 | Singh | Finance | 80000 |

(a) The *instructor* table

| *dept_name* | *building* | *budget* |
|---|---|---|
| Comp. Sci. | Taylor | 100000 |
| Biology | Watson | 90000 |
| Elec. Eng. | Taylor | 85000 |
| Music | Packard | 80000 |
| Finance | Painter | 120000 |
| History | Painter | 50000 |
| Physics | Watson | 70000 |

(b) The *department* table

# DDL and DML

# Data Definition Language (DDL)



- Specification notation for defining the database schema
  - Example:

    **create table** *instructor* (
    *ID* **char**(5),
    *name* **varchar**(20),
    *dept_name* **varchar**(20),
    *salary* **numeric**(8,2))
- DDL compiler generates a set of table templates stored in a *data dictionary*
- Data dictionary contains metadata (that is, data about data)
  - Database schema
  - Integrity constraints
    - Primary key (ID uniquely identifies instructors)
  - Authorization
    - Who can access what

# Data Manipulation Language (DML)



- Language for accessing and manipulating the data organized by the appropriate data model
  - DML: also known as **Query Language**
- Two classes of languages
  - **Pure** – used for proving properties about computational power and for optimization
    - ▷ **Relational Algebra** (we focus in this course)
    - ▷ Tuple relational calculus
    - ▷ Domain relational calculus
  - **Commercial** – used in commercial systems
    - ▷ SQL is the most widely used commercial language

# SQL

# SQL



- The most widely used commercial language
- *SQL is NOT a Turing Machine equivalent language*
  - Cannot be used to solve all problems that a C program, for example, can solve
- To be able to compute complex functions, SQL is usually embedded in some higher-level language
- Application programs generally access databases through one of
  - Language extensions to allow embedded SQL
  - Application Programming Interface or API (for example, ODBC/JDBC) which allow SQL queries to be sent to a database

# Database Design

# Database Design



The process of designing the general structure of the database:

- **Logical Design** – Deciding on the database schema. Database design requires that we find a **good** collection of relation schema
  - Business decision
    - What attributes should we record in the database?
  - Computer Science decision
    - What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- **Physical Design** – Deciding on the physical layout of the database

# Database Design (2)



- Is there any problem with this relation?

| *ID* | *name* | *salary* | *dept_name* | *building* | *budget* |
|---|---|---|---|---|---|
| 22222 | Einstein | 95000 | Physics | Watson | 70000 |
| 12121 | Wu | 90000 | Finance | Painter | 120000 |
| 32343 | El Said | 60000 | History | Painter | 50000 |
| 45565 | Katz | 75000 | Comp. Sci. | Taylor | 100000 |
| 98345 | Kim | 80000 | Elec. Eng. | Taylor | 85000 |
| 76766 | Crick | 72000 | Biology | Watson | 90000 |
| 10101 | Srinivasan | 65000 | Comp. Sci. | Taylor | 100000 |
| 58583 | Califieri | 62000 | History | Painter | 50000 |
| 83821 | Brandt | 92000 | Comp. Sci | Taylor | 100000 |
| 15151 | Mozart | 40000 | Music | Packard | 80000 |
| 33456 | Gold | 87000 | Physics | Watson | 70000 |
| 76543 | Singh | 80000 | Finance | Painter | 120000 |

# Module Summary





- Familiarized with the basic notions and terminology of database management systems
- Introduced the role of data models and languages
- Introduced the approaches to database design

**Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.**
**Edited and new slides are marked with "PPD".**

# Database Management Systems

## Module 05: Introduction to DBMS/2

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

*ppd@cse.iitkgp.ac.in*

# Module Recap





- Basic notions and terminology of database management systems
- Role of data models and languages
- Approaches to database design

# Module Objectives



- To understand models of database management systems
- To familiarize with major components of a database engine
- To familiarize with database internals and architecture
- To understand the historical perspective

# Module Outline





- Database Design
- OO Relational Model
- XML
- Database Engine
  - Storage Management
  - Query Processing
  - Transaction Management
- Database Internals and Architecture
- Database Users and Administrators

# Database Design

# Database Design



The process of designing the general structure of the database:

- **Logical Design**
  - Deciding on the database schema. Database design requires that we find a **good** collection of relation schema
  - Business decision
    - What attributes should we record in the database?
  - Computer Science decision
    - What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- **Physical Design**
  - Deciding on the physical layout of the database

# Database Design (2)





- Is there any problem with this relation?

| *ID* | *name* | *salary* | *dept_name* | *building* | *budget* |
|---|---|---|---|---|---|
| 22222 | Einstein | 95000 | Physics | Watson | 70000 |
| 12121 | Wu | 90000 | Finance | Painter | 120000 |
| 32343 | El Said | 60000 | History | Painter | 50000 |
| 45565 | Katz | 75000 | Comp. Sci. | Taylor | 100000 |
| 98345 | Kim | 80000 | Elec. Eng. | Taylor | 85000 |
| 76766 | Crick | 72000 | Biology | Watson | 90000 |
| 10101 | Srinivasan | 65000 | Comp. Sci. | Taylor | 100000 |
| 58583 | Califieri | 62000 | History | Painter | 50000 |
| 83821 | Brandt | 92000 | Comp. Sci | Taylor | 100000 |
| 15151 | Mozart | 40000 | Music | Packard | 80000 |
| 33456 | Gold | 87000 | Physics | Watson | 70000 |
| 76543 | Singh | 80000 | Finance | Painter | 120000 |

# Database Design (3)





| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 22222 | Einstein | Physics | 95000 |
| 12121 | Wu | Finance | 90000 |
| 32343 | El Said | History | 60000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 98345 | Kim | Elec. Eng. | 80000 |
| 76766 | Crick | Biology | 72000 |
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 58583 | Califieri | History | 62000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 15151 | Mozart | Music | 40000 |
| 33456 | Gold | Physics | 87000 |
| 76543 | Singh | Finance | 80000 |

(a) The *instructor* table

| *dept_name* | *building* | *budget* |
|---|---|---|
| Comp. Sci. | Taylor | 100000 |
| Biology | Watson | 90000 |
| Elec. Eng. | Taylor | 85000 |
| Music | Packard | 80000 |
| Finance | Painter | 120000 |
| History | Painter | 50000 |
| Physics | Watson | 70000 |

(b) The *department* table

# Design Approaches

- Need to come up with a methodology to ensure that each relations in the database is **good**
- Two ways of doing so:
  - Entity Relationship Model (Chapter 7)
    - Models an enterprise as a collection of entities and relationships
    - Represented diagrammatically by an entity-relationship diagram
  - Normalization Theory (Chapter 8)
    - Formalize what designs are bad, and test for them

# Object-Relational Data Models

# Object-Relational Data Models



- Relational model: flat, **atomic** values
- Object Relational Data Models
  - Extend the relational data model by including object orientation and constructs to deal with added data types
  - Allow attributes of tuples to have complex types, including non-atomic values such as nested relations
  - Preserve relational foundations, in particular the declarative access to data, while extending modeling power
  - Provide upward compatibility with existing relational languages

# XML: Extensible Markup Language

# XML: Extensible Markup Language



- Defined by the *WWW Consortium (W3C)*
- Originally intended as a document markup language not a database language
- The ability to specify new tags, and to create nested tag structures made XML a great way to exchange data, not just documents
- XML has become the basis for all new generation data interchange formats
- A wide variety of tools is available for parsing, browsing and querying XML documents/data

# **Database Engine**

# Database Engine





- Storage manager
- Query processing
- Transaction manager

# Storage Management

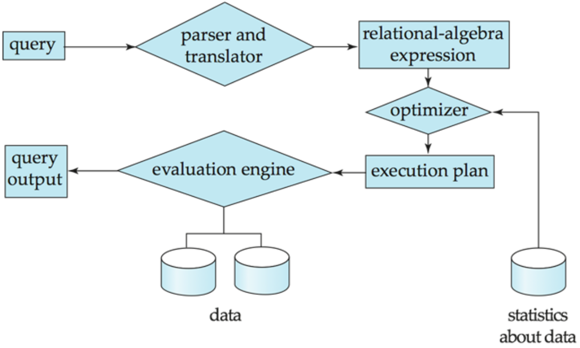


- **Storage manager** is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system
- The storage manager is responsible to the following tasks:
  - Interaction with the OS file manager
  - Efficient storing, retrieving and updating of data
- Issues:
  - Storage access
  - File organization
  - Indexing and hashing

# Query Processing



a) Parsing and translation
b) Optimization
c) Evaluation



# Query Processing (2)



- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation
- Cost difference between a good and a bad way of evaluating a query can be enormous
- Need to estimate the cost of operations
  - Depends critically on statistical information about relations which the database must maintain
  - Need to estimate statistics for intermediate results to compute cost of complex expressions

# Transaction Management

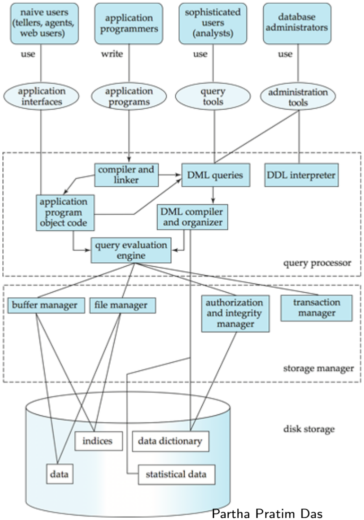


- What if the system fails?
- What if more than one user is concurrently updating the same data?
- A **transaction** is a collection of operations that performs a single logical function in a database application
- **Transaction-management component** ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- **Concurrency-control manager** controls the interaction among the concurrent transactions, to ensure the consistency of the database.

# Database System Internals

# Database System Internals





# Database Architecture

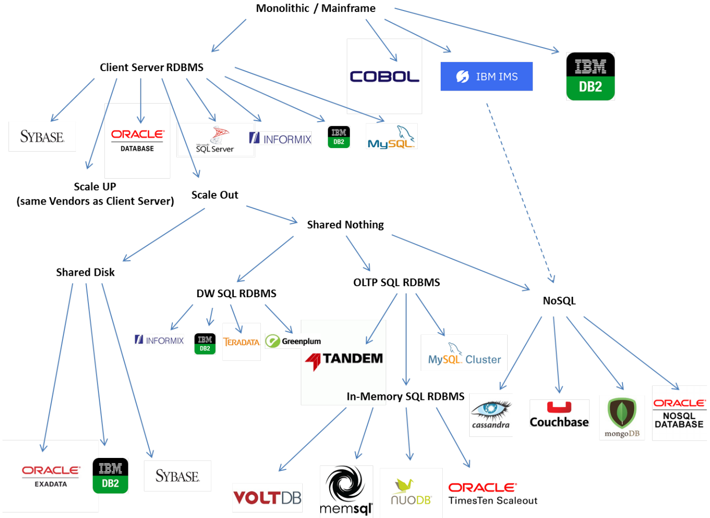
The architecture of a database system is greatly influenced by the underlying computer system on which the database is running:

- Centralized
- Client-server
- Parallel (multi-processor)
- Distributed
- Cloud

# Database Architecture (2)



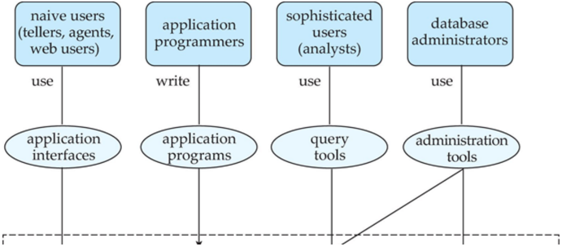




# Database Users and Administrators

# Database Users and Administrators






**Database**

# Module Summary





- Introduced models of database management systems
- Familiarized with major components of a database engine
- Familiarized with database internals and architecture

**Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.**
**Edited and new slides are marked with "PPD".**