# COMP 4754
# Introduction

Fall 2024

Instructor: Hafez Seliem

# Database in Software

## Three Layers Architecture

**User Interface (UI)**

- Web
- Mobile APP
- GUI

**Business logic/Application**

- Python
- Java
- C#

**Persistence / database**

- MySQL
- MS SQL
- Postgres
- Mongo

# Traditional Database Application

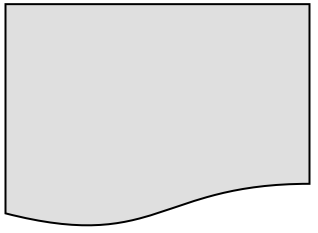
Suppose we are building a system  to store the information about:

- students
- courses
- professors
- who takes what, who teaches what

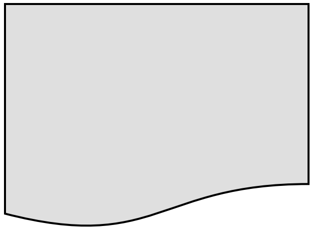
# Can we do it without a DBMS ?

Sure we can!  Start by storing the data in files:

students.txt



professors.txt



Now write Python or Java programs to implement specific tasks

# Doing it without a DBMS...

- Enroll “student1” in “COMP 4754”:

Write a Python /Java program to do the following:

Read ‘students.txt’
Find&update the record “student1”
Write “students.txt”

# Problems

- **Data Organization**
  - **Redundancy and inconsistency**
    - Duplication of information in different files, multiple file formats,

Name, Course, Email, Grade

student1, s1@mun.ca, COMP 4754, B

student2, s2@mun.ca, COMP 4754, A

- **Data Retrieval**
  - **For every query we need to write a program!**

Find the students registered for COMP 4754

Find the students with GPA > 3.5

- **Data Integrity**
  - **No Security**
  - **No coping mechanisms for system crashes**
  - **No means of Preventing Data Entry Errors (checks must be hard-coded in the programs)**

# Benefits of the Database Approach

- Data can be shared
- Redundancy can be reduced
- Transaction support possible
- Integrity can be maintained
- Security can be enforced
- Enforced standards
- Data independence:  Physical and logical

# The Database Approach

- Organizations must have access to operational data that is
  - accurate
    - timely
    - convenient
  - up-to-date
  - secure but available
- As control decentralizes in an organization, there is a danger that data management decentralizes as well.

# Definition of DB and DBMS

- A database (DB) is a collection of interrelated computer files, whose data contents and structure are described in a *data* dictionary, and which is under the control of a database management system (DBMS)

- A **Database Management System (DBMS)** is a software package designed to store and manage databases.
  - setting up storage structures
  - loading data
  - accepting and performing updates
  - accepting data requests from users and programs.

# Functions of a DBMS

- A good DBMS performs the following functions
  - Maintain data dictionary
  - Support multiple views of data
  - Enforce integrity constraints
  - Enforce access constraints
  - Support concurrency control
  - Support backup and recovery procedures
  - Support logical transactions

# Relational Model

- Example of tabular data in the relational model

Attributes

| *id* | *name* | *street* | *city* | *account-#* |
|---|---|---|---|---|
| 192-83-7465 | Johnson | Alma | Palo Alto | A-101 |
| 019-28-3746 | Smith | North | Rye | A-215 |
| 192-83-7465 | Johnson | Alma | Palo Alto | A-201 |

- ***Relational database*** – a collection of relations
- ***Relation*** – a set of *tuples*
- ***Tuple*** -- a record in the set
- **Relation scheme** – structure of tuples in a relation

# Entities and Relationships

- All database models must implement the following two concepts
  - *Entity* – real or abstract “things”
  - *Relationships* between entities
- Relational model represents both entities and relationships via *tables*.

# Structure Query Language (SQL)

- SQL: widely used (declarative) non-procedural language
  - E.g. find the name of the customer with customer-id 192-83-7465

    **select** *customer.customer-name*
    **from** *customer*
    **where** *customer.customer-id* = ‘192-83-7465’

  - E.g. find the balances of all accounts held by the customer with customer-id 192-83-7465

    **select** *account.balance*
    **from** *depositor*, *account*
    **where** *depositor.customer-id* = ‘192-83-7465’ **and**
    *depositor.account-number* = *account.account-number*

# Relational Database Design

Design 1

| InstDep | | | | | |
|---|---|---|---|---|---|
| iID | name | salary | depName | bldng | budget |
| 111 | Alice | 5000 | CS | DC | 20000 |
| 222 | Bob | 4000 | Physics | PHY | 30000 |
| 333 | Carl | 5200 | CS | DC | 20000 |
| 444 | Diana | 5500 | CS | DC | 20000 |
| … | … | … | … | … | … |

*If each department identified by depName has as associated (bldng, budget)*
*Design 1, intuitively, is a bad design with redundancy.*

Design 2

| Inst | | | |
|---|---|---|---|
| iID | name | salary | depName |
| 111 | Alice | 5000 | CS |
| 222 | Bob | 4000 | Physics |
| … | … | …. | … |

| Dep | | |
|---|---|---|
| depName | bldng | budget |
| CS | DC | 20000 |
| Physics | PHY | 30000 |

# Data retrieval: Indexing

- How to answer fast the query: “Find the student with SID = 101”?

- One approach is to scan the student table, check every student, retrurn the one with id=101... very slow for large databases

- Any better idea?

  1st keep student record over the SID. Do a binary search…. Updates…
  2nd Use a hash table. Much faster for exact match queries… but cannot support Range queries. (Also, special hashing schemes are needed for dynamic data)

# Data Integrity: *Transaction processing*

- Why Concurrent Access to Data must be Managed?

John and Jane withdraw $50 and $100 from a common account…

John:

1. get balance
2. if balance > $50
3. balance = balance - $50
4. update balance

Jane:

1. get balance
2. if balance > $100
3. balance = balance - $100
4. update balance

Initial balance $300. Final balance=?

It depends…

# Data Integrity: *Recovery*

Transfer $50 from account A ($100) to account B ($200)

1. get balance for A
2. If balanceA > $50
3. $\text{balance}_A = \text{balance}_A - 50$
4. Update $\text{balance}_A$ in database
5. Get balance for B
6. $\text{balance}_B = \text{balance}_B + 50$
7. Update $\text{balance}_B$ in database

System crashes….

Recovery management

# Steps in Database Design

- Requirements Analysis
  - user needs; what must database do?
- Conceptual Design
  - high level description (often done w/ER model)
- Logical Design
  - translate ER into DBMS data model
- Schema Refinement
  - consistency, normalization
- Physical Design - indexes, disk layout
- Security Design - who accesses what, and how