CS 4604: Introduction to Database Management Systems

B. Aditya Prakash

Lecture #1: Introduction
Course Information

- **Instructor**
  B. Aditya Prakash, Torg 3160 F, badityap@cs.vt.edu
  - Office Hours: 2:30-3:45pm Mondays and Wednesdays
  - And by appointment
  - Include string **CS 4604** in subject

- **Teaching Assistant**
  Elaheh Raisi, McBryde 106, elaheh@vt.edu
  - Office Hours: TBD

- **Class Meeting Time**
  Monday and Wednesday, 4:00PM-5:15PM, GOODW 115

- **Keeping in Touch**
  Course web site [http://courses.cs.vt.edu/~cs4604](http://courses.cs.vt.edu/~cs4604)
  updated regularly through the semester
  - *Piazza link on the website*
Textbook

- **Required**
  

  Web page for the book (with errata)
  

- **Optional:**
  
  - Garcia-Molina, Ullman and Widom, 3rd Ed.
  - Silberschatz, Korth and Sudarshan, 6th Ed.
Pre-reqs and Force-adds

- Prerequisites: a grade of C or better in CS 3114, senior standing

- Force-add requests:
  - Go to: https://www.cs.vt.edu/S15Force-Adds
  - Password: 4604bap$
  - Survey link will work during the entire class period, for the first and second lectures
  - If you miss both lectures, go to McB 114 and fill paper form, and find me to get a signature.
### Course Grading

<table>
<thead>
<tr>
<th></th>
<th>Weight</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homework</td>
<td>30%</td>
<td>6-7</td>
</tr>
<tr>
<td>Midterm exam</td>
<td>20%</td>
<td>(Tentative) March 4, Wed., in class</td>
</tr>
<tr>
<td>Final exam</td>
<td>30%</td>
<td>May 11, Mon., 7:45am-9:45am</td>
</tr>
<tr>
<td>Course project</td>
<td>20%</td>
<td>3 assignments</td>
</tr>
</tbody>
</table>

- Project is spread over 3 deliverables
- Submit hard copies of homeworks and project assignments at the start of class on the due date
- Each class has required reading (on course web page)
- No Pop-Quizzes 😊
Course Project

- We will put project overview later (first project assignment)
- 2, or 3 persons per project.
- Project runs the entire semester with regular assignments and a final implementation assignment.
**Class Policies**

- Make sure you go through the detailed policies on website:
  
  [http://courses.cs.vt.edu/~cs4604/Spring15/policies.html](http://courses.cs.vt.edu/~cs4604/Spring15/policies.html)

- Late policy: 4 ‘slip’ days (to be used only for HWs not project)

- Exams: no aids allowed, except:
  - 1 page with your notes (both sides), for the midterm
  - 2 such pages, for the final
Why Study Databases?

- **Academic**
  - Databases involve many aspects of computer science
  - Fertile area of research
  - Three Turing awards in databases

- **Programmer**
  - A plethora of applications involve using and accessing databases

- **Businessman**
  - Everybody needs databases => lots of money to be made

- **Student**
  - Get those last three credits and I don’t have to come back to Blacksburg ever again!
  - Google, Oracle, Microsoft, Facebook etc. will hire me!
  - Databases sound cool!
  - ???
What Will You Learn in CS 4604?

- **Implementation**
  - What is under-the-hood of a DB like Oracle/MySQL?

- **Design**
  - How do you model your data and structure your information in a database?

- **Programming**
  - How do you use the capabilities of a DBMS?

- **CS 4604 achieves a balance between**
  - a firm theoretical foundation to designing moderate-sized databases
  - creating, querying, and implementing realistic databases and connecting them to applications
Course Outline

- **Weeks 1–4: Query/Manipulation Languages and Data Modeling**
  - Relational Algebra
  - Data definition
  - Programming with SQL
  - Entity-Relationship (E/R) approach
  - Specifying Constraints
  - Good E/R design

- **Weeks 5–8: Indexes, Processing and Optimization**
  - Storing
  - Hashing/Sorting
  - Query Optimization
  - NoSQL and Hadoop

- **Week 9-10: Relational Design**
  - Functional Dependencies
  - Normalization to avoid redundancy

- **Week 11-12: Concurrency Control**
  - Transactions
  - Logging and Recovery

- **Week 13–14: Students’ choice**
  - Practice Problems
  - XML
  - Data mining and warehousing
What is the goal of a DBMS?

- Electronic record-keeping
  
  *Fast* and *convenient* access to information

- DBMS == database management system
  
  - `Relational’ in this class
  
  - data + set of instructions to access/manipulate data
What is a DBMS?

- **Features of a DBMS**
  - Support massive amounts of data
  - Persistent storage
  - Efficient and convenient access
  - Secure, concurrent, and atomic access

- **Examples?**
  - Search engines, banking systems, airline reservations, corporate records, payrolls, sales inventories.
  - New applications: Wikis, social/biological/multimedia/scientific/geographic data, heterogeneous data.
Features of a DBMS

• **Support massive** amounts of data
  – Giga/tera/petabytes
  – Far too big for main memory

• **Persistent** storage
  – Programs update, query, manipulate data.
  – Data continues to live long after program finishes.

• **Efficient** and **convenient** access
  – Efficient: do not search entire database to answer a query.
  – Convenient: allow users to query the data as easily as possible.

• **Secure, concurrent, and atomic** access
  – Allow multiple users to access database simultaneously.
  – Allow a user access to only to authorized data.
  – Provide some guarantee of reliability against system failures.
Example Scenario

- Students, taking classes, obtaining grades
  - Find my GPA
  - <and other ad-hoc queries>
Obvious solution 1: Folders

- Advantages?
  - Cheap; Easy-to-use

- Disadvantages?
  - No ad-hoc queries
  - No sharing
  - Large Physical foot-print
Obvious Solution++

- Flat files and C (C++, Java...) programs
  - E.g. one (or more) UNIX/DOS files, with student records and their courses
Obvious Solution++

- Layout for student records?
  - CSV (‘comma-separated-values’)
    
    Hermione Grainger,123,Potions,A
    Draco Malfoy,111,Potions,B
    Harry Potter,234,Potions,A
    Ron Weasley,345,Potions,C
Obvious Solution++

Layout for student records?

– Other possibilities like

Hermione Grainger, 123
Draco Malfoy, 111
Harry Potter, 234
Ron Weasley, 345

123, Potions, A
111, Potions, B
234, Potions, A
345, Potions, C
Problems?

- inconvenient access to data (need ‘C++’ expertise, plus knowledge of file-layout)
  - data isolation
- data redundancy (and inconsistencies)
- integrity problems
- atomicity problems
- concurrent-access problems
- security problems
- .......
Problems-Why?

- Two main reasons:
  - file-layout description is buried within the C programs and
  - there is no support for transactions (concurrency and recovery)

DBMSs handle exactly these two problems
Example Scenario

- RDBMS = “Relational” DBMS
- The relational model uses relations or tables to structure data
- ClassList relation:

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hermione Grainger</td>
<td>Potions</td>
<td>A</td>
</tr>
<tr>
<td>Draco Malfoy</td>
<td>Potions</td>
<td>B</td>
</tr>
<tr>
<td>Harry Potter</td>
<td>Potions</td>
<td>A</td>
</tr>
<tr>
<td>Ron Weasley</td>
<td>Potions</td>
<td>C</td>
</tr>
</tbody>
</table>

- Relation separates the logical view (externals) from the physical view (internals)
- Simple query languages (SQL) for accessing/modifying data
  - Find all students whose grades are better than B.
  - SELECT Student FROM ClassList WHERE Grade > “B”
DBMS Architecture

- Schema Modifications
- Queries
- Modifications

“Query” Processor

Storage Manager

Data

Metadata

Transaction Manager
Transaction Processing

- One or more database operations are grouped into a “transaction”
- Transactions should meet the “ACID test”
  - Atomicity: All-or-nothing execution of transactions.
  - Consistency: Databases have consistency rules (e.g. what data is valid). A transaction should NOT violate the database’s consistency. If it does, it needs to be rolled back.
  - Isolation: Each transaction must appear to be executed as if no other transaction is executing at the same time.
  - Durability: Any change a transaction makes to the database should persist and not be lost.
Disadvantages over (flat) files?
Disadvantages over (flat) files

- Price
- additional expertise (SQL/DBA)

(hence: over-kill for small, single-user data sets
But: mobile phones (eg., android) use sqlite)
A Brief History of DBMS

- The earliest databases (1960s) evolved from file systems
  - File systems
    - Allow storage of large amounts of data over a long period of time
    - File systems do not support:
      - Efficient access of data items whose location in a particular file is not known
      - Logical structure of data is limited to creation of directory structures
      - Concurrent access: Multiple users modifying a single file generate non-uniform results
    - Navigational and hierarchical
    - User programmed the queries by walking from node to node in the DBMS.

- Relational DBMS (1970s to now)
  - View database in terms of relations or tables
  - High-level query and definition languages such as SQL
  - Allow user to specify what (s)he wants, not how to get what (s)he wants

- Object-oriented DBMS (1980s)
  - Inspired by object-oriented languages
  - Object-relational DBMS
The DBMS Industry

- A DBMS is a software system.

- Major DBMS vendors: Oracle, Microsoft, IBM, Sybase

- Free/Open-source DBMS: MySQL, PostgreSQL, Firebird.  
  - Used by companies such as Google, Yahoo, Lycos, BASF…. 

- All are “relational” (or “object-relational”) DBMS.

- A multi-billion dollar industry
Fundamental concepts

- 3-level architecture
- logical data independence
- physical data independence
3-level architecture

- view level
- logical level
- physical level
3-level architecture

- **view level**
- **logical level:** eg., tables
  - STUDENT(ssn, name)
  - TAKES (ssn, cid, grade)
- **physical level:**
  - how are these tables stored, how many bytes / attribute etc
3-level architecture

- view level, eg:
  - v1: select ssn from student
  - v2: select ssn, c-id from takes

- logical level

- physical level
3-level architecture

- hence, **physical** and **logical** data independence:
  - logical D.I.:
    - ???
  - physical D.I.:
    - ???
3-level architecture

- hence, **physical** and **logical** data independence:
  - **logical** D.I.:
    - can add (drop) column; add/drop table
  - **physical** D.I.:
    - can add index; change record order
Database users

- ‘naive’ users
- casual users
- application programmers
- [ DBA (Data base administrator)]
Casual users

```
select * from student
```

DBMS

data

and meta-data = catalog
``Naive’’ users

Pictorially:

DBMS

app. (eg., report generator)

and meta-data = catalog

data
App. programmers

- those who write the applications (like the ‘report generator’ )
DB Administrator (DBA)

- Duties?
DB Administrator (DBA)

- schema definition (‘logical’ level)
- physical schema (storage structure, access methods)
- schemas modifications
- granting authorizations
- integrity constraint specification
Overall system architecture

- [Users]
  - DBMS
    - query processor
    - storage manager
    - transaction manager
- [Files]
naive  app. pgmr  casual  DBA

users

emb. DML  DML proc.  DDL int.

app. pgm(o)  query eval.

trans. mgr  buff. mgr  file mgr

query proc.

storage mgr.

data  meta-data
Overall system architecture

- query processor
  - DML compiler
  - embedded DML pre-compiler
  - DDL interpreter
  - Query evaluation engine
Overall system architecture (cont’d)

- storage manager
  - authorization and integrity manager
  - transaction manager
  - buffer manager
  - file manager
Overall system architecture (cont’ d)

- Files
  - data files
  - data dictionary = catalog (= meta-data)
  - indices
  - statistical data
Some examples:

- DBA doing a DDL (data definition language) operation, eg.,
  create table student ...
DDL int.  

int.  

DML  

proc.  

query  

perf.  

app. pgm(o)  

trans. mgr  

buff. mgr  

file mgr  

meta-data  

data  

query proc.  

storage mgr.  

naive  

app. pgmr  

casual  

DBA  

users
Some examples:

- casual user, asking for an update, eg.:
  
  update student
  set name to ‘smith’
  where ssn = ‘345’
naive  app. pgmr  casual  DBA

emb. DML  DML proc.  DDL int.

app. pgm(o)  query eval.

trans. mgr  buff. mgr  file mgr

query proc.

data

storage mgr.

meta-data
naive  app. pgmr  casual  DBA

users

emb. DML  DML proc.  DDL int.

app. pgm(o)  query eval.

query proc.

trans. mgr  buff. mgr  file mgr

storage mgr.

data  meta-data
Some examples:

- app. programmer, creating a report, eg
  
  ```
  main()
  {
  ....
  exec sql "select * from student"
  ...
  }
  ```
Some examples:

- ‘naive’ user, running the previous app.
Conclusions

- (relational) DBMSs: electronic record keepers
- customize them with `create table` commands
- ask SQL queries to retrieve info
Conclusions contd

main advantages over (flat) files & scripts:

- **logical + physical data independence** (ie., flexibility of adding new attributes, new tables and indices)

- **concurrency control and recovery**