CS525-04/05: Advanced Database Organization

Notes 1: Introduction to DBMS Implementation

Yousef M. Elmehdwi

Department of Computer Science

Illinois Institute of Technology

yelmehdwi@iit.edu

January 8th 2024

Slides: adapted from a course taught by Hector Garcia-Molina, Stanford

Core Terminology Review

- Data
  - Data refers to any piece of information that holds value and is worth keeping.
  - It's often stored in electronic form and can range from numbers and text to images and videos.
- Database
  - organized collection of interrelated data that models some aspect of the real-world.
- Query
  - operation that retrieves specific data from a database based on certain criteria or conditions.
  - queries allow users to extract relevant information.
- Relation
  - refers to the organization of data into a two-dimensional table, where rows (tuples) represent basic entities or facts of some sort, and columns (attributes) represent properties of those entities.
- Schema
  - a description of the structure of the data in a database, often called "metadata"
  - it's like a blueprint that outlines how the data is organized, what types of data are stored, and how they are related.

Database Management System (DBMS)

- A DBMS is software that allows applications to store and analyze information in a database.
- A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases.

Advanced Database Organization?

- =Database Implementation
- =How to implement a database system
- and have fun doing it ;-)

What do you want from a DBMS?

- Keep data around (persistent)
- Answer questions (queries) about data
- Update data

Isn't Implementing a Database System Simple?

- Relation $\Rightarrow$ Statements $\Rightarrow$ Results

Introduction the Megatron 3000 Database Management System

- "Imaginary" database System
- The latest from Megatron Labs
- Incorporates latest relational technology
- UNIX compatible
- Lightweight & cheap!

Megatron 3000 Implementation Details

- Megatron 3000 uses the file system to store its relations
- Relations stored in files (ASCII)
- Use a separate file per entity/relation.
- The application has to parse the files each time they want to read/update records.
  - e.g., relation `Students(`*`name`*`,`*`id`*`,`*`dept`*`)` is in `/usr/db/Students`
  - The file `Students` has one line for each tuple.
  - Values of components of a tuple are stored as a character string, separated by special marker character `#`

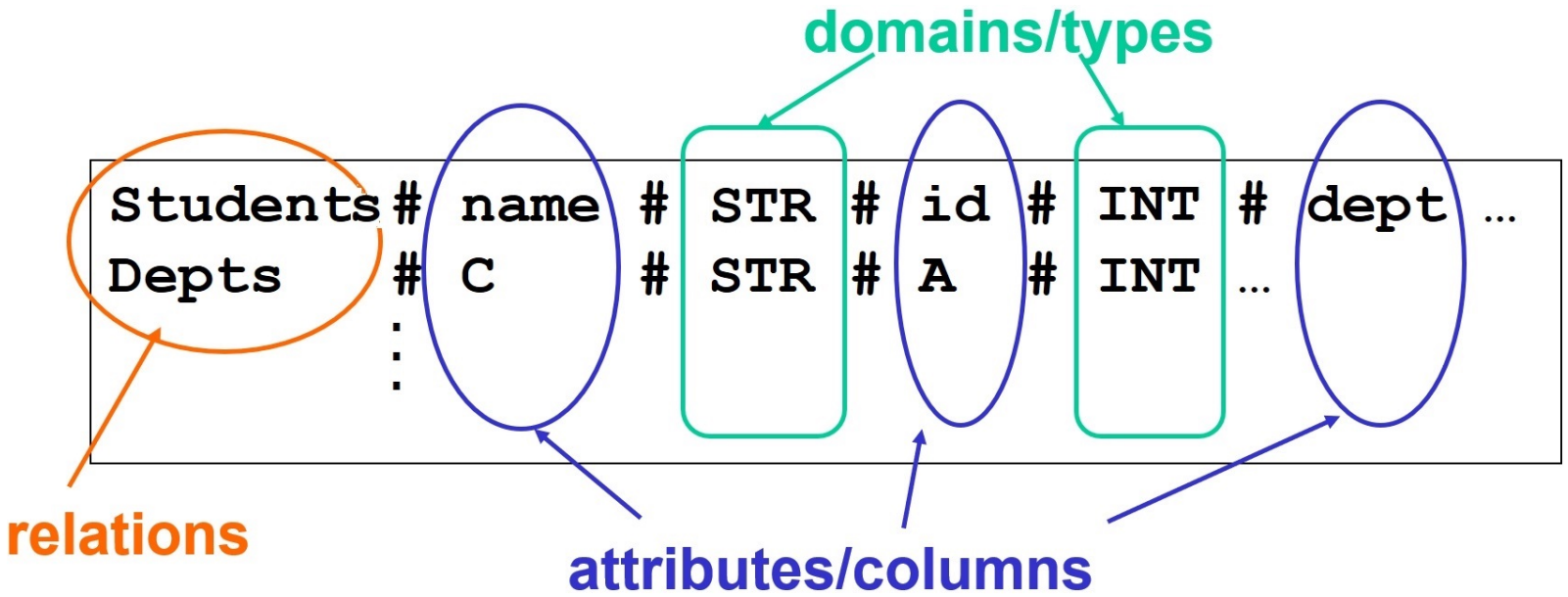
Smith	#	123	#	CS
Jonson	#	522	#	EE
		⋮		

Megatron 3000 Implementation Details

- The database schema is stored in a special file
- Schema file (ASCII) in `/usr/db/schema`
  - For each relation, the file `schema` has a line beginning with that relation name, in which attribute names alternate with types.
  - The character `#` separates elements of these lines.

```
Students  #  name  #  STR  #  id  #  INT  #  dept ...
Depts     #  C     #  STR  #  A   #  INT  ...
                          ⋮
```

Megatron 3000 Implementation Details



Megatron 3000 Sample Sessions

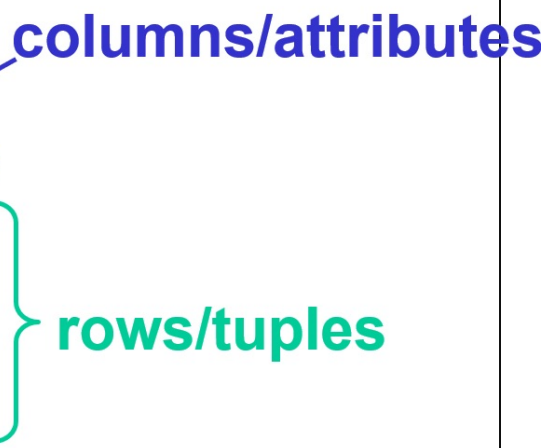
```
% MEGATRON3000
   Welcome to MEGATRON 3000!
&
   :
   :
& quit
%
```

- We are now talking to the Megatron 3000 user interface, to which we can type SQL queries in response to the Megatron prompt (`&`).

Megatron 3000 Sample Sessions

```
& select * from Students #

    Relation Students
  name      id    dept
  Smith     123   CS
  Johnson   522   EE
  ...
&
```



- A # ends a query

Megatron 3000 Sample Sessions

- Execute a query and send the result to printer

```
& select *
  from Students | LPR #
&
```

- Result sent to LPR (printer).

Megatron 3000 Sample Sessions

- Execute a query and store the result in a new file

```
& select *
  from Students
  where id < 100 | LowId #
&
```

- New relation `LowId` created.

How Megatron 3000 Executes Queries

- To execute

```
SELECT * FROM R WHERE <condition>
```

1. Read schema to get attributes of `R`
2. Check validity of condition
3. Display attributes of `R` as the header
4. Read file `R`; for each line:
   a. Check condition
   b. If TRUE, display the line as tuple

Megatron 3000 Query Execution

- To execute

```
SELECT * FROM R WHERE <condition> | T
```

1. Process select as before but omit `Step 3`
2. Write results to new file `T`
3. Append new line to dictionary `usr/db/schema`

Megatron 3000 Query Execution

- Consider a more complicated query, one involving a join of two relations `R`, `S`
- To execute

```
SELECT A,B FROM R,S WHERE <condition>
```

1. Read schema to get `R,S` attributes
2. Read `R` file, for each line `r`:
   a. Read `S` file, for each line `s`:
      1. Create join tuple `r` & `s`
      2. Check condition
      3. If TRUE, Display `r,s[A,B]`

What's wrong with Megatron 3000 DBMS?

- DBMS is not implemented like our imaginary Megatron 3000
- Described implementation is inadequate for applications involving significant amount of data or multiple users of data
- Partial list of problems follows

What's wrong with Megatron 3000 DBMS?

- Tuple layout on disk is inadequate with no flexibility when the database is modified
- e.g., change String from *CS* to *CSDept* in one `Students` tuple, we have to rewrite the entire file
  - ASCII storage is expensive
  - Deletions are expensive

What's wrong with Megatron 3000 DBMS?

- Search expensive; no indexes
  - e.g., cannot find tuple with given key quickly
  - Always have to read full relation

What's wrong with Megatron 3000 DBMS?

- Brute force query processing
- e.g.,

```
SELECT * FROM R,S WHERE R.A = S.A and S.B > 1000
```

  - Much better if use index to select tuples that satisfy condition (Do select using `S.B` >1000 first)
  - More efficient join (sort both relations on `A` and merge)

What's wrong with Megatron 3000 DBMS?

- No buffer manager
  - There is no way for useful data to be buffered in main memory; all data comes off the disk, all the time
  - e.g., need caching.

What's wrong with Megatron 3000 DBMS?

- No concurrency control
  - Several users can modify a file at the same time with unpredictable results.

What's wrong with Megatron 3000 DBMS?

- No reliability
- e.g., in case of error/crash, say, power failure or leave operations half done
  - Can lose data

What's wrong with Megatron 3000 DBMS?

- No security
- e.g., file system security is coarse
  - Unable to restrict access, say, to some fields of a relation and not others

What's wrong with Megatron 3000 DBMS?

- No application program interface (API)
  - e.g., how can a payroll program get at the data?

What's wrong with Megatron 3000 DBMS?

- Cannot interact with other DBMSs.

What's wrong with Megatron 3000 DBMS?

- No GUI

This Course

- Introduce students to better way of building a database management systems.

Reading assignment

- Refresh your memory about basics of the relational model and SQL
  - from your earlier course notes
  - from some textbook
  - http://cs.iit.edu/~cs425/schedule.html

Reading

- Course Blackboard: Assignments\Reading subfolder
  - Chapter 1: "Introduction to DBMS Implementation"

Next

Notes 2: Hardware